Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/070607

FPGA Design for Pseudorandom Number Generator Based on Chaotic Iteration used in Information Hiding Application

Jacques M. Bahi, Xiaole Fang, Christophe Guyeux and Laurent Larger*

Femto-St Institute, University of Franche-Comté, France

Received: 20 Feb. 2013, Revised: 18 Jun. 2013, Accepted: 21 Jun. 2013
Published online: 1 Nov. 2013

Abstract: Lots of researches indicate that the inefficient generation of random numbers is a significant bottleneck for information communication applications. Therefore, Field Programmable Gate Array (FPGA) is developed to process a scalable fixed-point method for random streams generation. In our previous researches, we have proposed a technique by applying some well-defined discrete chaotic iterations that satisfy the reputed Devaney's definition of chaos, namely chaotic iterations (CI). We have formerly proven that the generator with CI can provide qualified chaotic random numbers. In this paper, this generator based on chaotic iterations is optimally redesigned for FPGA device. By doing so, the generation rate can be largely improved. Analyses show that these hardware generators can also provide good statistical chaotic random bits and can be cryptographically secure too. An application in the information hiding security field is finally given as an illustrative example.

Keywords: Information security, Pseudorandom number generator, Discrete chaotic iteration, Cryptographical security, FPGA.

1 Introduction

The extremely rapid development of the Internet brings more and more attention to the information security techniques, such as text, image, or video encryption, etc. As a result, highly qualified random sequences, as an inseparable part of encryption techniques, are urgently required. There are two kinds of random sequences: real random sequences generated by physical methods and pseudorandom sequences generated by algorithm simulations, which are in accordance with some kind of probability distributions. The implementation methods for different classes of random number generators are visualized in Figure 1. However, the constructions of the real random sequences are usually poor in speed and efficiency, and require considerably more storage space as well, and these defects restrict their usage in modern cryptography. On the one hand, field programmable gate arrays (FPGAs) have been successfully used for realizing the speed requirement in pseudorandom sequence generation, due to their high parallelization capability [1–3]. Advantages of such physical generation way encompass performance, design time, power consumption flexibility, and cost. On the other hand, there is a growing interest to use chaotic dynamical systems as PRNGs, among other things due to the unpredictability and distorted-like properties of such systems ([4–6]). Nowadays, such chaos-based generators have also been successfully used to strengthen optical communications [7].

A short overview of our previous researches is given thereafter. It has firstly been stated that a tool called chaotic iterations (CIs), used in distributed computing, satisfies the chaotic property as it is defined by Devaney [8]. The chaotic behavior of CIs has then been exploited to obtain a class of unpredictable PRNGs [9]. This class receives two given, potentially defective, generators as input and mix them with chaotic iterations, producing by doing so a sequence having a better random profile than the two inputs taken alone [10]. Then, in [11], two new versions of such "CIPRNGs" have been proposed, involving respectively two logistic maps and two XORshifts.

In this paper, we continue the works initiated in [9–12]: the two approaches introduced before are merged by proposing a discrete chaos-based generator designed on FPGA. The idea is to improve the efficiency of our formerly proposed generators, without any lack of chaos properties. To do so, a new model of CIPRNG Version 1 [9] on Field Programmable Gate Array is introduced and its security is proven in some cases.

[*] Corresponding author e-mail: xiaole.fang@univ-fcomte.fr

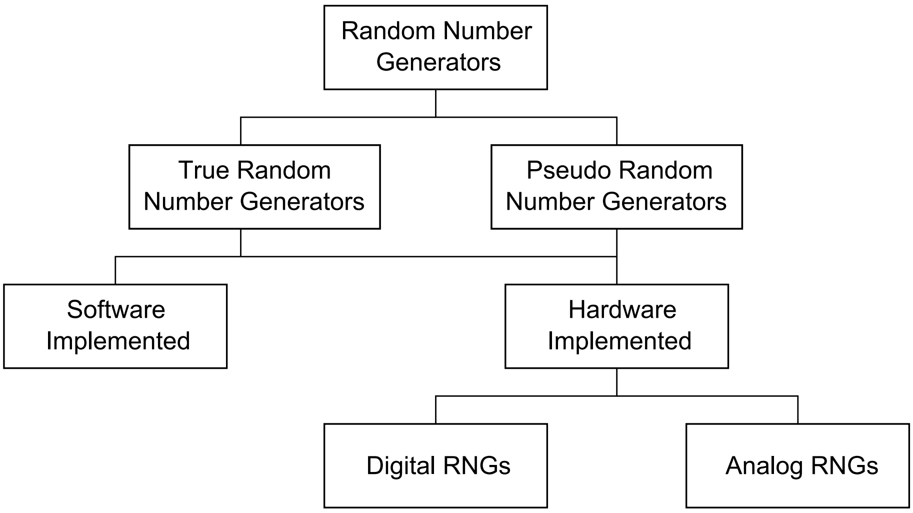


Fig. 1: Implementations of random number generator classes

Additionally, the randomness of this novel proposal is evaluated by the famous NIST test suite (widely used as a randomness standard battery of tests [13]). Last but not the least, a potential usage of this generator in a cryptographic application is presented.

2 Definitions and terminologies

2.1 Notations

$[\![1;N]\!]$	$\rightarrow \{1,2,\ldots,N\}$
S^n	$\rightarrow$ the n^{th} term of a sequence $S = (S^1, S^2, \ldots)$
v_i	$\rightarrow$ the i^{th} component of a vector: $v = (v_1, v_2, \ldots, v_n)$
strategy	$\rightarrow$ a sequence which elements belong in $[\![1;N]\!]$
$\mathbb{S}$	$\rightarrow$ the set of all strategies
$X^{\mathbb{N}}$	$\rightarrow$ the set of sequences belonging into X
$\mathbf{C}_n^k$	$\rightarrow$ the binomial coefficient $\binom{n}{k} = \frac{n!}{k!(n-k)!}$
$+$	$\rightarrow$ the integer addition
$\ll$ and $\gg$	$\rightarrow$ the usual shift operators
$\mathbb{N}^*$	$\rightarrow$ the set of positive integers {1,2,3,...}
&	$\rightarrow$ the bitwise AND
$\oplus$	$\rightarrow$ the bitwise exclusive or between two integers.

2.2 Blum Blum Shub and XORshift

The Blum Blum Shub generator [14] (usually denoted by BBS) takes the form:

$$x^{n+1} = (x^n)^2 \ mod\ m, \quad y^{n+1} = x^{n+1} \ mod\ log(log(m)),$$

where m is the product of two prime numbers (these prime numbers need to be congruent to 3 modulus 4), and y^n is the returned binary sequence.

Algorithm 1 XORshift algorithm

Input: x (a 64-bit word)
Output: r (a 64-bit word)
Parameters: a, b, c (integers)

```
$x \leftarrow x \oplus (x \ll a)$;
$x \leftarrow x \oplus (x \gg b)$;
$x \leftarrow x \oplus (x \ll c)$;
$r \leftarrow x$;
An arbitrary round of XORshift
```

XORshift, on its part, is a category of very fast PRNGs designed by George Marsaglia [15]. Algorithm 1 shows its working procedure. The values of a, b, c decide the offsets of shifting.

2.3 Chaotic iterations

Definition 1.*The set* $\mathbb{B}$ *denoting* $\{0,1\}$*, let* $f : \mathbb{B}^N \longrightarrow \mathbb{B}^N$ *be an "iteration" function and* $S \in \mathbb{S}$ *be a strategy. Then, the so-called* chaotic iterations *are defined by [16]:*

$$\begin{cases} x^0 \in \mathbb{B}^N, \\ \forall n \in \mathbb{N}^*, \forall i \in [\![1;N]\!], x_i^n = \begin{cases} x_i^{n-1} & if\ S^n \neq i \\ f(x^{n-1})_{S^n} & if\ S^n = i. \end{cases} \end{cases} \tag{1}$$

In other words, at the n^{th} iteration, only the S^n-th cell is "iterated". Note that in a more general formulation, S^n can be a subset of components and $f(x^{n-1})_{S^n}$ can be replaced by $f(x^k)_{S^n}$, where $k < n$, describing for example delays transmission. For the general definition of such chaotic iterations, see, e.g., [16].

Chaotic iterations generate a set of vectors (Boolean vectors in this paper), they are defined by an initial state x^0, an iteration function f, and a strategy S said to be a "chaotic strategy". Being an iterative process producing binary vectors given a "seed" x^0, such chaotic iterations can be used as pseudorandom number generators. The mathematical fundations of such a contruction is recalled in the next section.

2.4 Chaotic iterations as PRNG

Our generator denoted by $CI_f(PRNG1, PRNG2)$ is designed by the following process.

Let $\mathsf{N} \in \mathbb{N}^*, \mathsf{N} \geqslant 2$. Some chaotic iterations are fulfilled, with f as iteration function and $PRNG1$ for strategy, to generate a sequence $(x^n)_{n\in\mathbb{N}} \in \left(\mathbb{B}^{\mathsf{N}}\right)^{\mathbb{N}}$ of Boolean vectors: the successive states of the iterated system. Some of these vectors are randomly extracted using $PRNG2$, and their components constitute our pseudorandom bit flow.

Chaotic iterations are realized as follows. Initial state $x^0 \in \mathbb{B}^{\mathsf{N}}$ is a Boolean vector taken as a seed and chaotic strategy $(S^n)_{n\in\mathbb{N}} \in [\![1,\mathsf{N}]\!]^{\mathbb{N}}$ is constructed with PRNG2. Lastly, iterate function f is the vectorial Boolean negation

$$f_0 : (x_1,...,x_{\mathsf{N}}) \in \mathbb{B}^{\mathsf{N}} \longmapsto (\overline{x_1},...,\overline{x_{\mathsf{N}}}) \in \mathbb{B}^{\mathsf{N}}.$$

To sum up, at each iteration only S^i-th component of state X^n is updated, as follows

$$x_i^n = \begin{cases} x_i^{n-1} \text{ if } i \neq S^i, \\ \overline{x_i^{n-1}} \text{ if } i = S^i. \end{cases} \quad (2)$$

Finally, let $\mathcal{M}$ be a finite subset of $\mathbb{N}^*$. Some x^n are selected by a sequence m^n as the pseudorandom bit sequence of our generator, $(m^n)_{n\in\mathbb{N}} \in \mathcal{M}^{\mathbb{N}}$. So, the generator returns the following values: the components of x^{m^0}, followed by the components of $x^{m^0+m^1}$, followed by the components of $x^{m^0+m^1+m^2}$, *etc.* In other words, the generator returns the following bits:

$$x_1^{m_0} x_2^{m_0} x_3^{m_0} \ldots x_{\mathsf{N}}^{m_0} x_1^{m_0+m_1} x_2^{m_0+m_1} \ldots x_{\mathsf{N}}^{m_0+m_1} x_1^{m_0+m_1+m_2} \ldots$$

or the following integers:

$$x^{m_0} x^{m_0+m_1} x^{m_0+m_1+m_2} \ldots$$

In details, when considering the Boolean negation and two integer sequences p and q, we obtain the CIPRNG(p,q) version 1 published in [17]: p is S and the output of the generator is the subsequence $\left(x^{\sigma(n)}\right)_{n\in\mathbb{N}}$, where $\sigma(0) = q^0$ and $\sigma(n+1) = \sigma(n) + q^n$. Reason to be of the sequence q is that, between two iterates of chaotic iterations, at most 1 bit will change in the vector, and thus the sequence (x^n) cannot pass any statistical test: we must extract a subsequence $(x^{\sigma(n)})$ of (x^n) to produce the outputs. CIPRNG(p,q) version 2, for its part, will extract a subsequence from the strategy $S = p$ to prevent from negating several times a same position between two outputs.

*Example 1.*If we consider the Boolean negation for f, then chaotic iterations of Definition 1 can be rewritten as: $x^{n+1} = x^n \oplus s^n$, where $s^n \in [\![0, 2^{\mathsf{N}-1}]\!]$ is such that its k−th binary digit is 1 if and only if $k \in S^n$. Such a particular chaotic iterations will be our generator called XOR CIPRNG [18].

2.5 PRNGs based on chaotic iterations

Let us now recall with more details some previous works in the field of CIPRNGs: chaotic iteration based pseudorandom number generators.

2.5.1 CIPRNG(PRNG1,PRNG2): Version 1

Let PRNG1 and PRNG2 be two given generators provided as input, or "entropy sources". The objective of the CIPRNG approach is to mix them together using chaotic iterations, in such a way that chaos improve their statistics against well-known batteries of tests, while the speed of the resulted mixed PRNGs is of the same order than the slowest input. Additionally, we will show in a further section that if the PRNG1 is cryptographically secure, then it is the case too for the mixed CIPRNG(PRNG1,PRNG2). Thus expected properties of entropy sources could be, for instance, speed for PRNG2 and security or good statistical properties for PRNG1, even though, theoretically speaking, nothing is required for these inputs except that they must not be totally defective (chaos cannot correct constant inputs for instance).

Some chaotic iterations are fulfilled (see Flow chart 2) to generate a sequence $(x^n)_{n\in\mathbb{N}} \in \left(\mathbb{B}^N\right)^{\mathbb{N}}$ of Boolean vectors, which are the successive states of the iterated system. Some of these vectors are randomly extracted and their components constitute the pseudorandom bit flow [9]. Chaotic iterations are realized as follows. The initial state $x^0 \in \mathbb{B}^N$ is a Boolean vector taken as a seed and the chaotic strategy $(S^n)_{n\in\mathbb{N}} \in [\![1,N]\!]^{\mathbb{N}}$ is constructed with PRNG2. At each iteration, only the S^i-th component of state x^n is updated. Finally, some x^n are selected by a sequence m^n, obtained using the PRNG1, as the pseudorandom bit sequence of our generator.

The basic design procedure of the first version of the CIPRNG generator is summed up in Algorithm 2. The internal state is x, whereas a and b are computed by PRNG1 and PRNG2. See Table 2 for a run example of this CIPRNG version 1.

Table 1: Running example of CIPRNG version 1

$m:$	4					5						4				
S	2	4	2	2		5	1	1	5	5		3	2	3	3	
x^0					x^4						x^9					x^{13}
1					1		$\xrightarrow{1} 0$	$\xrightarrow{1} 1$			1					1
0	$\xrightarrow{2} 1$		$\xrightarrow{2} 0$	$\xrightarrow{2} 1$	1						1		$\xrightarrow{2} 0$			0
1					1						1	$\xrightarrow{3} 0$		$\xrightarrow{3} 1$	$\xrightarrow{3} 0$	0
0		$\xrightarrow{4} 1$			1						1					1
0					0	$\xrightarrow{5} 1$			$\xrightarrow{5} 0$	$\xrightarrow{5} 1$	1					1

Output: $x_1^0x_2^0x_3^0x_4^0x_5^0x_1^4x_2^4x_3^4x_4^4x_5^4x_1^9x_2^9x_3^9x_4^9\ x_5^9x_1^{13}x_2^{13}x_3^{13}x_4^{13}x_5^{13}... = 10100111101111110011...$

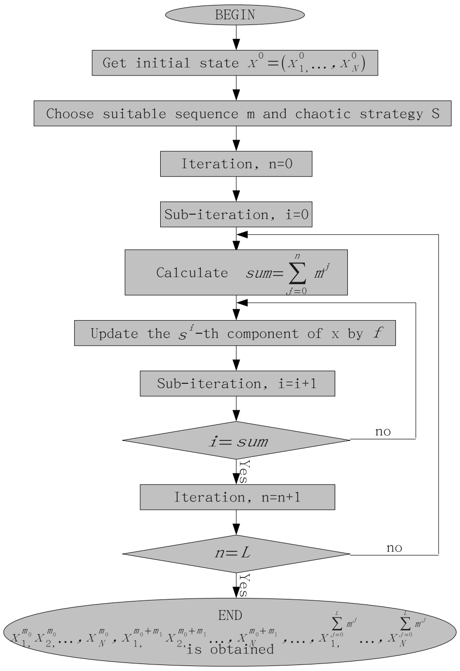

BEGIN
Get initial state $x^0=(x_1^0,\ldots,x_N^0)$
Choose suitable sequence m and chaotic strategy S
Iteration, n=0
Sub-iteration, i=0
Calculate $sum=\sum_{j=0}^{n} m^j$
Update the s^i-th component of x by f
Sub-iteration, i=i+1
$i=sum$
no
Yes
Iteration, n=n+1
$n=L$
no
Yes
END
$x_1^{m_0}, x_2^{m_0},\ldots, x_N^{m_0}, x_1^{m_0+m_1}, x_2^{m_0+m_1},\ldots, x_N^{m_0+m_1},\ldots, x_1^{\sum_{j=0}^{L} m^j},\ldots, x_N^{\sum_{j=0}^{L} m^j}$ is obtained

Fig. 2: Flow chart of CIPRNG version 1

2.5.2 CIPRNG(PRNG1,PRNG2): Version 2

The second version of the CI-based generators is designed by the following process [11]. First of all, some chaotic iterations have to be done to generate a sequence $(x^n)_{n\in\mathbb{N}} \in \left(\mathbb{B}^{32}\right)^{\mathbb{N}}$ of Boolean vectors, which are the successive states of the iterated system. Some of these vectors will be randomly extracted and the pseudorandom bit flow will be constituted by their components. Such chaotic iterations are realized as follows.

–Initial state $x^0 \in \mathbb{B}^N$ is a Boolean vector taken as a seed.

Algorithm 2 An arbitrary round of the CIPRNG Version 1

Input: the internal state x (an array of N 1-bit words)
Output: an array r of N 1-bit words

```
$a \leftarrow PRNG1()$;
$m \leftarrow a \bmod 2 + c$;
while $i = 0,\ldots,m$ do
    $b \leftarrow PRNG2()$;
    $S \leftarrow b \bmod$ N;
    $x_S \leftarrow \overline{x_S}$;
end while
$r \leftarrow x$;
return $r$;
```

–Chaotic strategy $(S^n)_{n\in\mathbb{N}} \in [\![1,N]\!]^{\mathbb{N}}$ is an irregular decimation of the PRNG2 sequence.

At each iteration, only the S^i-th component of state x^n is updated using the vectorial negation, as follows: $x_i^n = x_i^{n-1}$ if $i \neq S^i$, else $x_i^n = \overline{x_i^{n-1}}$. Finally, some x^n are selected by a sequence m^n as the pseudorandom bit sequence of our generator, where $(m^n)_{n\in\mathbb{N}} \in \mathcal{M}^{\mathbb{N}}$ is computed from PRNG1. The basic design procedure of this CIPRNG Version 2 generator is summarized in Algorithm 3. The internal state is x. a and b are those computed by the two inputted PRNGs. Finally, the value m is the integers sequence defined in Eq.(3).

$$m^n = g_1(S^n) = \begin{cases} 0 \text{ if } 0 \leqslant S^n < C_{32}^0, \\ 1 \text{ if } C_{32}^0 \leqslant S^n < \sum_{i=0}^{1} C_{32}^i, \\ 2 \text{ if } \sum_{i=0}^{1} C_{32}^i \leqslant S^n < \sum_{i=0}^{2} C_{32}^i, \\ \vdots \quad \vdots \\ N \text{ if } \sum_{i=0}^{N-1} C_{32}^i \leqslant S^n < 1. \end{cases} \tag{3}$$

3 Security Analysis of CIPRNG Version 1

In this section the concatenation of two strings u and v is classically denoted by uv. In a cryptographic context,

Algorithm 3 An arbitrary round of the CIPRNG Version 2

Input: the internal state x (N bits)
Output: a state r of N bits

```
for $i = 0,\dots,N$ do
    $d_i \leftarrow 0$
end for
$a \leftarrow PRNG1()$
$m \leftarrow g_1(a)$
$k \leftarrow m$
while $i = 0,\dots,k$ do
    $b \leftarrow PRNG2()\ mod$ N
    $S \leftarrow b$
    if $d_S = 0$ then
        $x_S \leftarrow \overline{x_S}$
        $d_S \leftarrow 1$
    else if $d_S = 1$ then
        $k \leftarrow k+1$
    end if
end while
$r \leftarrow x$
return $r$
```

a pseudorandom generator is a deterministic algorithm G transforming strings into strings and such that, for any seed s of length m, $G(s)$ (the output of G on the input s) has size $l_G(m)$ with $l_G(m) > m$. The notion of secure PRNGs can now be defined as follows.

3.1 Algorithm expression conversion

For the convenience of security analysis, CIPRNG Version 1 detailed in Algorithm 2 is converted as in Eq.(4), where internal state is x, S and T are those computed by PRNG1 and PRNG2, whereas at each round, x^{n-1} is updated to x^n.

$$\begin{cases} x^0 \in [\![0, 2^{\mathsf{N}}-1]\!], S \in [\![0, 2^{\mathsf{N}}-1]\!]^{\mathbb{N}}, T \in [\![0, 2^{\mathsf{N}}-1]\!]^{\mathbb{N}} \\ C = S^n \& 1 + 3 * N \\ w^0 = T^m \ mod\ N, w^1 = T^{m+1} \& 3, \dots w^{C-1} = T^{m+C-1} \& 3 \\ d^n = (1 \ll w^0) \oplus (1 \ll w^1) \oplus \dots (1 \ll w^{C-1}) \\ \forall n \in \mathbb{N}^*, x^n = x^{n-1} \oplus d^n. \end{cases} \tag{4}$$

3.2 Security notion

Definition 2. *A cryptographic PRNG G is secure if for any probabilistic polynomial time algorithm D, for any polynomial p, and for all sufficiently large m's,*

$$|Pr[D(G(U_m)) = 1] - Pr[D(U_{l_G(m)}) = 1] < \frac{1}{p(m)}, \tag{5}$$

where U_r is the uniform distribution over $\{0,1\}^r$ and the probabilities are taken over U_m, $U_{l_G(m)}$ as well as over the internal coin tosses of D.

Intuitively, it means that there is no polynomial time algorithm that can distinguish a perfect uniform random generator from G with a non negligible probability. Note that it is quite easily possible to change the function l into any polynomial function l' satisfying $l'(m) > m$.

The generation schema developed in Eq.4 is based on two pseudorandom generators. Let H be the "PRNG1" and I be the "PRNG2". We may assume, without loss of generality, that for any string S_0 of size L, the size of $H(S_0)$ is kL, then for any string T_0 of size M, it has $I(T_0)$ with kN, $k > 2$. It means that $l_H(N) = kL$ and $l_I(N) = kM$. Let $S_1,...,S_k$ be the string of length L such that $H(S_0) = S_1...S_k$ and $T_1,...,T_k$ be the string of length M s.t. $H(S_0) = T_1...T_k$ ($H(S_0)$ and $I(T_0)$ are the concatenations of S_i's and T_i's).

The generator X defined in Algorithm 4 is mapping any string $x_0 S_0 T_0$, of length $L + M + N$, into the string $x_0 \oplus d^1, x_0 \oplus d^1 \oplus d^2, ...(x_0 \bigoplus_{i=0}^{i=k} d^i)$, c.f. Eq.(4). One in particular has $l_X(L+M+N) = kN = l_H(N)$ and $k > M+L+N$. We announce that if the inputted generator H is cryptographically secure, then the new one defined in Eq.(4) is secured too.

Proposition 1. *If PRNG1 is a secure cryptographic generator, then for all PRNG2, we can have that X is a secure cryptographic PRNG too.*

Proof. The proposition is proven by contraposition. Assume that X is not secure. By definition, there exists a polynomial time probabilistic algorithm D, a positive polynomial p, such that for all k_0 there exists $L+M+N \geq k_0$ satisfying

$$|\Pr[D(X(U_{L+M+N})) = 1] - \Pr[D(U_{kN} = 1)]| \geq \frac{1}{p(L+M+N)}.$$

Consider a word w of size kL.

1.Decompose w into $w = w_1...w_k$.
2.Pick a string y of size N uniformly at random.
3.Pick a string of size $(3kN + \sum_{j=1}^{j=k}(w_j \& 1))M$: u.
4.Decompose u into $u = u_1...u_{3kN+\sum_{j=1}^{j=k}(w_j\&1)}$.
5.Define $t_i = (\bigoplus_{l=3N(i-1)+(\sum_{l=1}^{l=i-1}(w_j\&1))+1}^{j=3N(i)+(\sum_{j=1}^{j=i}(w_j\&1))} (1 << u_l))$.
6.Compute $z = (y \oplus t_1)(y \oplus t_1 \oplus t_2)...(y \bigoplus_{i=1}^{i=k}(t_i))$.
7.Return $D(z)$.

On one hand, consider for each $y \in \mathbb{B}^{kN}$ the function φ_y from $\mathbb{B}^{kN}$ into $\mathbb{B}^{kN}$ mapping $t = t_1 \dots t_k$ (each t_i has length N) to $(y \oplus t_1)(y \oplus t_1 \oplus t_2) \dots (y \bigoplus_{i=1}^{i=k} t_i)$. On the other hand, treat each $u_l \in \mathbb{B}^{(3Nk+\sum_{j=0}^{j=k}(w_j\&1))M}$ by the function ϕ_u from $\mathbb{B}^{(3kN+\sum_{j=0}^{j=k}(w_i\&1))M}$ into $\mathbb{B}^{kN}$ mapping $w = w_1 \dots w_k$ (each w_i has length L) to:
$(\bigoplus_{l=1}^{l=3N+(w_1\&1)}(1 << u_l))((\bigoplus_{l=1+3N+(w_1\&1)}^{l=6N+(w_1\&1)+(w_1\&1)}(1 << u_l))$
$\dots(\bigoplus_{l=3N(k-1)+\sum_{j=1}^{j=k-1}(w_j\&1)}^{l=3Nk+\sum_{j=1}^{j=k}(w_j\&1)}(1 << u_l))$.

By construction, one has for every w,

$$D'(w) = D(\varphi_y(\phi_u(w))). \qquad (6)$$

Therefore, and using Eq.(6), one has $\Pr[D'(U_{kL}) = 1] = \Pr[D(\varphi_y(\phi_u(U_{kL}))) = 1]$ and, therefore,

$$\Pr[D'(U_{kL}) = 1] = \Pr[D(U_{kN}) = 1]. \qquad (7)$$

Now, using Eq.(6) again, one has for every x,

$$\Pr[D'(U_{H(x)}) = 1] = \Pr[D(\varphi_y(\phi_u(U_{H(x)}))) = 1]. \qquad (8)$$

Since where y and u_j are randomly generated. By construction, $\varphi_y(\phi_u(x)) = X(yu_1w)$, hence

$$\Pr[D'(H(U_{kL})) = 1] = \Pr[D(X(U_{N+M+L})) = 1]. \qquad (9)$$

Compute the difference of Eq.(9) and Eq.(8), one can deduce that there exists a polynomial time probabilistic algorithm D', a positive polynomial p, such that for all k_0 there exists $L+M+N \geq k_0$ satisfying

$$|\Pr[D'(H(U_{KL})) = 1] - \Pr[D(U_{kL}) = 1]| \geq \frac{1}{p(L+M+N)},$$

proving that H is not secure, which is a contradiction.

Compared to stream ciphers, which are symmetric key ciphers where plaintext digits are combined with a pseudorandom cipher digit stream (keystream), the CIPRNG method can be described as a post-treatment on two inputted PRNGs, that:

1. add chaotic properties to these generators,
2. by doing so, improve their statistical properties when the inputs are defective,
3. while preserving their security, for instance when one of the input is cryptographically secure.

If PRNG1 is already used as a keystream in a stream cipher, because it is cryptographically secure, then the combined CIPRNG(PRNG1,XORshift), which runs potentially faster than PRNG1, can be used too as a keystream. The security comparison between CIPRNG and other designs is thus summarized in Proposition 1: the security of CIPRNG(PRNG1,PRNG2) is directly related to the one of PRNG1, meaning that if PRNG1 is secure, then the resulted CIPRNG is secure too.

4 CIPRNG Version 1 Designed for FPGA

4.1 An efficient and cryptographically secure PRNG based on CIPRNG Version 1

In Algorithm 4 is given an efficient and cryptographically secure generator suitable for FPGA applications. It is based on CIPRNG Version 1 and thus presents a good random statistical profile.

Algorithm 4 An efficient and cryptographically secure generator based on CIPRNG version 1

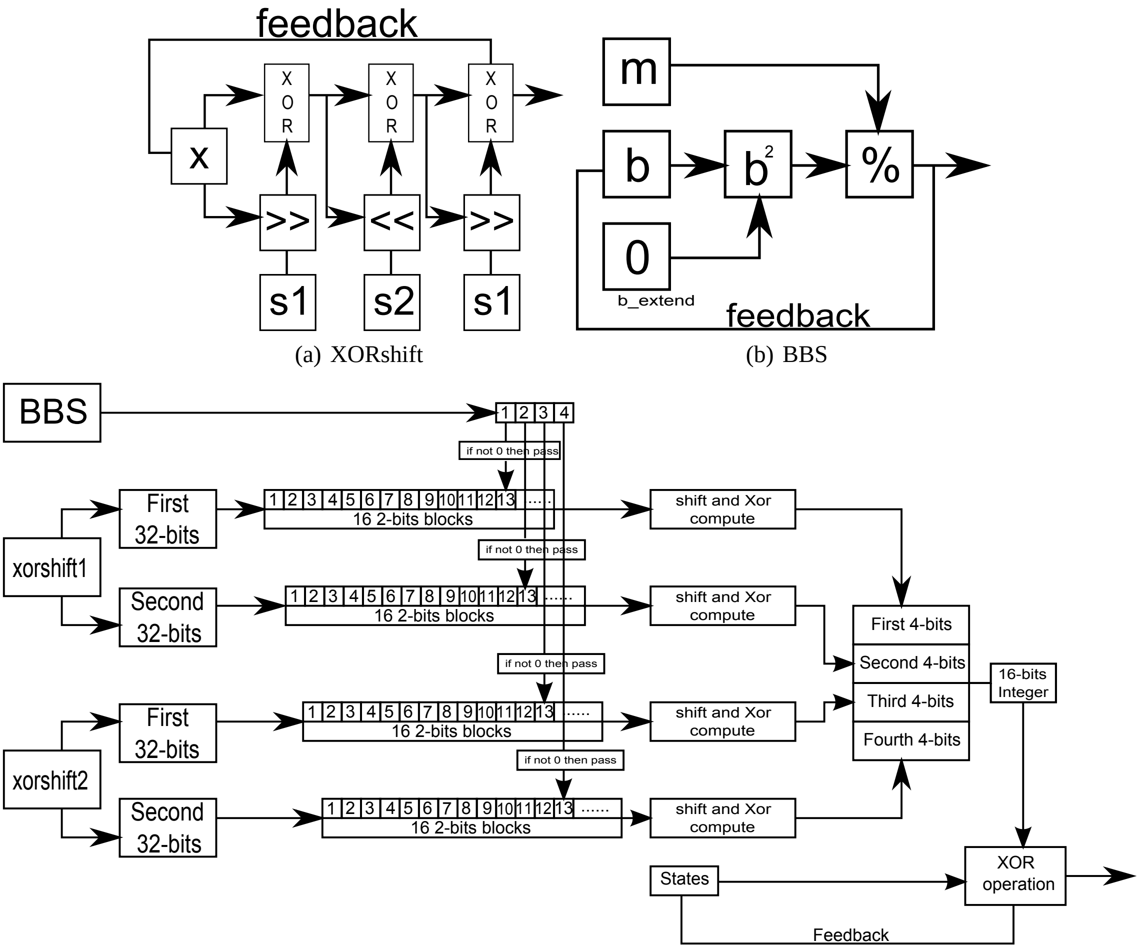
Notice: xorshift1, xorshift2 (64-bit XORshift generators)
Input: z (a 16-bit word)
Output: r (a 16-bit word)

```
$x \leftarrow xorshift1();$
$y \leftarrow xorshift2();$
$z1 \leftarrow x\&0xffffffff$
$z2 \leftarrow (x >> 32)\&0xffffffff$
$z3 \leftarrow y\&0xffffffff$
$z4 \leftarrow (y >> 32)\&0xffffffff$
$t \leftarrow bbs();$
$t1 \leftarrow t\&1;$
$t2 \leftarrow t\&2;$
$t3 \leftarrow t\&4;$
$t4 \leftarrow t\&8;$
$w1 \leftarrow 0;$
$w2 \leftarrow 0;$
$w3 \leftarrow 0;$
$w4 \leftarrow 0;$
while $i = 0, \ldots, 11$ do
    $w1 \leftarrow (w1 \oplus (1 \ll ((z1 \gg (i \times 2))\&3)));$
    $w2 \leftarrow (w2 \oplus (1 \ll ((z2 \gg (i \times 2))\&3)));$
    $w3 \leftarrow (w3 \oplus (1 \ll ((z3 \gg (i \times 2))\&3)));$
    $w4 \leftarrow (w4 \oplus (1 \ll ((z4 \gg (i \times 2))\&3)));$
end while
if $(t1 \neq 0)$ then $w1 \leftarrow (w1 \oplus (1 \ll ((z1 \gg 24)\&3)));$
if $(t2 \neq 0)$ then $w2 \leftarrow (w2 \oplus (1 \ll ((z2 \gg 24)\&3)));$
if $(t3 \neq 0)$ then $w3 \leftarrow (w3 \oplus (1 \ll ((z3 \gg 24)\&3)));$
if $(t4 \neq 0)$ then $w4 \leftarrow (w4 \oplus (1 \ll ((z4 \gg 24)\&3)));$
$z \leftarrow z \oplus w1 \oplus (w2 \ll 4) \oplus (w3 \ll 8) \oplus (w4 \ll 12);$
$r \leftarrow z;$
return $r$;
```

The internal state x is a vector of 16 bits, whereas two 64-bit XORshift generators ($xorshift1(), xorshift2()$) are provided as entropy sources. As it can be seen in the algorithm, the two outputs of XORshift generators are spread into four 32-bit integers. Then for each integer, there are 16 2−bits components that can be found; every 12 of these components are used to update the states. Lastly, the 4 least significant bits (LSBs) of the output $bbs()$ of the Blum Blum Shub generator decide if the state must be updated with the considered 13-bits block or not.

According to Section 3, this generator based on CIPRNG version 1 can turn to be cryptographically secure, if the PRNG1 entropy source is cryptographically secure. Here, this inputted generator is the well known BBS, which is believed to be the most secured PRNG method currently available [19]. The t value is computed by a BBS with a modulo m equal to 32 bits. Then the $log(log(m))$ LSBs of t can be treated as secure, this is why we only considerate 4 LSBs in this algorithm.

Following the approach detailed in [10], we thus have used chaotic iterations in order to improve the statistical



(a) XORshift

(b) BBS

(c) The proposed CIPRNG

Fig. 3: The processing structure for BBS in FPGA (per clock step)

behavior of the inputted generators. Here, two coupled 64 bits XORshift generators together with one BBS are applied. By doing so, we obtain in Algorithm 4 a generator being both chaotic and cryptographically secure [18].

Table 2 shows the test results of the proposed CIPRNG against the NIST battery [13]. Results of XORshift and BBS are provided too. According to NIST test suite, the sole BBS generator algorithm cannot produce a statistically perfect output. This is not contradictory with Prop. 1, as the cryptographically secure property is an asymptotic one: even though the Blum Blum Shum generator is cryptographically secure (which is a property independent from the chosen modulo *m*), the very small value chosen for *m* makes it unable to pass the NIST battery. Obviously, best statistical performances are obtained using the proposed CIPRNG.

4.2 FPGA Design

In order to take benefits from the computing power of FPGA, a whole processing needs to spread into several independent blocks of threads that can be computed simultaneously. In general, the larger the number of threads is, the more logistic elements of FPGA are used, and the less branching instructions are used (if, while, ...), the better the performances on FPGA are. Obviously, having these requirements in mind, it is possible to build a program similar to the algorithm presented in Algorithm 4, which produces pseudorandom numbers with chaotic properties on FPGA. To do so, Verilog-HDL [20] has been used to help programming. In this generator, there are three PRNG objects that use the exclusive or operation, two XORshifts, and a BBS, their processing are described thereafter.

Table 2: NIST SP 800-22 test results ($\mathbb{P}_T$)

Method	CIPRNG	XORshift	BBS
Frequency (Monobit) Test	0.073128	0.145326	0.32435
Frequency Test within a Block	0.719128	0.028817	0.000000
Runs Test	0.314992	0.739918	0.000000
Longest Run of Ones in a Block Test	0.445121	0.554420	0.000000
Binary Matrix Rank Test	0.888124	0.236810	0.000000
Discrete Fourier Transform (Spectral) Test	0.912003	0.514124	0.000000
Non-overlapping Template Matching Test*	0.500459	0.512363	0.000000
Overlapping Template Matching Test	0.702445	0.595549	0.000000
Universal Statistical Test	0.666230	0.122325	0.000000
Linear Complexity Test	0.475761	0.249284	0.000000
Serial Test* (m=10)	0.780099	0.495847	0.043355
Approximate Entropy Test (m=10)	0.679102	0.000000	0.000000
Cumulative Sums (Cusum) Test*	0.819200	0.074404	0.000000
Random Excursions Test*	0.697803	0.507812	0.000000
Random Excursions Variant Test*	0.338243	0.289594	0.000000
Success	15/15	14/15	2/15

4.2.1 Design of XORshift

The structure of XORshift designed in Verilog-HDL is shown in Figure 3(a). There are four inputs:

- The first one is the initial state, which costs 64 bits of register units,
- the other three ones are used to define the shift operations.

Let us remark that, in FPGA, this shift operation costs nothing, as it simply consists in using different bit cells of the input. We can thus conclude that there are $64-s1+64-s2+64-s3 = 192-s1-s2-s3$ logic gates elements that are required for the XORshifts processing.

4.2.2 Design of BBS

Figure 3(b) gives the proposed design of the BBS generator in FPGAs. There are two inputs of 32 bits, namely b and m. Register b stores the state of the system at each time (after the square computation). m is also a register that saves the value of M, which must not change. Another register *b_extend* is used to combine b to a data having 64 bits, with a view to avoid overflow. After the last computation, the three LSBs from the output of % are taken as output. Let us notice that a BBS is performed at each time unit.

4.2.3 Design of CI

Two XORshifts and one BBS are connected to work together, in order to compose the proposed CIPRNG (see Figure 3(c)). As it can be shown, the four bits of the BBS output are switches for the corresponding 32 bits outputs from XORshift. Every round of the processing costs two time units to be performed: in the first clock, the three PRNGs are processed in parallel, whereas in the second one, the results of these generators are combined with the current state of the system, in order to produce the output of 16 bits.

In our experiments, the type $EP2C8Q208C8$ from Altera company's CYCLONE II FPGA series has been used. By default, its working frequency is equal to 50 MHz. However, it is possible to increase it until 400 MHz by using the phase-lock loop (PLL) device. In that situation, the CIPRNG designed on this FPGA can produce over 6000 Mbits per second (that is, $400(MHz)\times 16(bits)$, see Figure 4), while using 6114 of the 8256 logic elements in $EP2C8Q208C8$. This is nearly 30 times faster than when it is processed in continuous method.

In the next section, an application of this CSPRNG designed on FPGA in the information hiding security fields is detailed, to show that this hardware pseudorandom generator is ready to use.

5 An Information Hiding Application

Information hiding has recently become a major information security technology, especially with the increasing importance and widespread distribution of digital media th-rough the Internet [21]. It includes several techniques like digital watermarking. The aim of

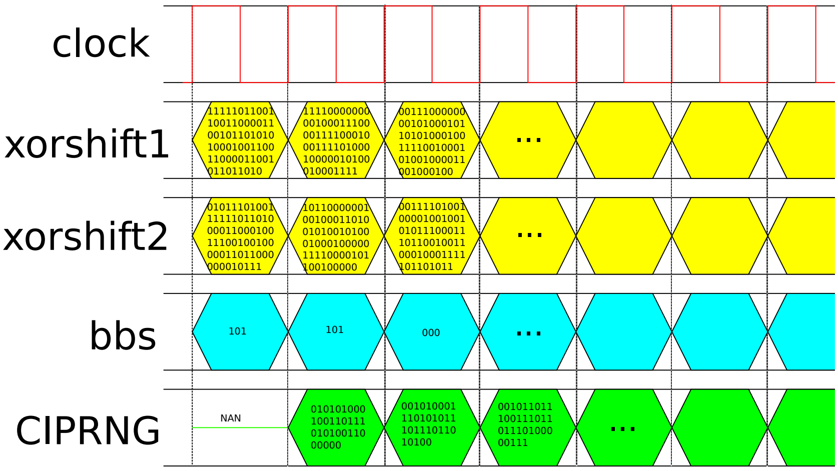


Fig. 4: Outputs of each component in clock step unit

digital watermarking is to embed a piece of information into digital documents, such as pictures or movies. This is for a large panel of reasons, such as: copyright protection, control utilization, data description, content authentication, and data integrity. For these reasons, many different watermarking schemes have been proposed in recent years. Digital watermarking must have essential characteristics, including: security, imperceptibility, and robustness. Chaotic methods have been proposed to encrypt the watermark before embedding it in the carrier image for these security reasons. In this paper, a watermarking algorithm based on the chaotic PRNG presented above is given, as an illustration of use of this PRNG based on CI.

5.1 Most and least significant coefficients

The definitions of most and least significant coefficients are shown at first, as they have been formerly introduced in [22, 23].

Definition 3.*For a given image, the most significant coefficients (in short MSCs), are coefficients that allow the description of the relevant part of the image,* i.e., *its most rich part (in terms of embedding information), through a sequence of bits.*

Definition 4.*By least significant coefficients (LSCs), we mean a translation of some insignificant parts of a medium in a sequence of bits (insignificant can be understand as: "which can be altered without sensitive damages").*

These LSCs can be for example, the last three bits of the gray level of each pixel, in the case of a spatial domain watermarking of a gray-scale image.

In the proposed application, LSCs are used during the embedding stage: some of the least significant coefficients of the carrier image will be chaotically chosen and replaced by the bits of the mixed watermark. With a large number of LSCs, the watermark can be inserted more than once and thus the embedding will be more secure and robust, but also more detectable. The MSCs are only useful in the case of authentication: encryption and embedding stages depend on them. Hence, a coefficient should not be defined at the same time, as a MSC and a LSC; the last can be altered, while the first is needed to extract the watermark. For a more rigorous definition of such LSCs and MSCs see, *e.g.*, [24].

5.2 Stages of the algorithm

We recall now a formerly introduced watermarking scheme, which consists of two stages: (1) mixture of the watermark and (2) its embedding [25].

5.2.1 Watermark mixture

Firstly, for safety reasons, the watermark can be mixed before its embedding into the image. A common way to achieve this stage is to use the bitwise exclusive or (XOR), for example, between the watermark and the above PRNG. In this paper and similarly to [25], we will use another mixture scheme based on chaotic iterations. Its chaotic strategy, defined with our PRNG, will be highly sensitive to the MSCs, in the case of an authenticated watermark, as stated in [12].

5.2.2 Watermark embedding

Some LSCs will be substituted by all bits of the possibly mixed watermark. To choose the sequence of LSCs to be altered, a number of integers, less than or equal to the number N of LSCs corresponding to a chaotic sequence $\left(U^k\right)_k$, is generated from the chaotic strategy used in the mixture stage. Thus, the U^k-th least significant coefficient of the carrier image is substituted by the k^{th} bit of the possibly mixed watermark. In the case of authentication, such a procedure leads to a choice of the LSCs that are highly dependent on the MSCs.

5.2.3 Extraction

The chaotic strategy can be regenerated, even in the case of an authenticated watermarking because the MSCs have not been changed during the stage of embedding the watermark. Thus, the few altered LSCs can be found, the mixed watermark can then be rebuilt, and the original watermark can be obtained. If the watermarked image is attacked, then the MSCs will change. Consequently, in the case of authentication and due to the high sensitivity of the embedding sequence, the LSCs designed to receive the watermark will be completely different. Hence, the result of the recovery will have no similarity with the original watermark: authentication is reached.

5.3 The FPGA setting

The 32-bit embedded-processor architecture designed specifically for the Altera family of FPGAs is applied in this information hiding specific application. Nios II incorporates many enhancements over the original Nios architecture, making it more suitable for a wider range of embedded computing applications, from DSP to system-control [26].

Figure 5(a) shows the structure of this application. The NIOS II system can read the image from the HOST computer side. Via the bus control, pseudorandom bits are produced into the FPGA and according to the CIPRNG. Then the results are transmitted back into the host.

In Figure 5(b), the NIOS II is using the most powerful version the CYCLONE II can support (namely, the NIOS II/f one). 4 KB on chip memory and 16 MB SDRAM are set, and the *PLL* device is used to enhance the clock frequency from 50 to 200 MHz. Finally, the data connection bus NIOS II system and generator works in 32 bits.

5.4 Results

For evaluating the efficiency and the robustness of the application, some attacks are performed on some chaotically watermarked images. For the attacks, the similarity percentages with the original watermark are computed. These percentages are the numbers of equal bits between the original and the extracted watermark, shown as a percentage. A result less than or equal to 50% implies that the image has probably not been watermarked.

5.4.1 Cropping attack

In this kind of attack, a watermarked image is cropped. In this case, the results in Tab.3 have been obtained. In Figure 6, the decrypted watermarks are shown after a crop of 50 pixels and after a crop of 10 pixels, in the authentication case.

By analyzing the similarity percentage between the original and the extracted watermark, we can conclude that in the case of unauthentication, the watermark still remains after a cropping attack. The desired robustness is reached. It can be noticed that cropping sizes and percentages are rather proportional. In the case of authentication, even a small change of the carrier image (a crop by 10×10 pixels) leads to a really different extracted watermark. In this case, any attempt to alter the carrier image will be signaled, thus the image is well authenticated.

5.4.2 Rotation attack

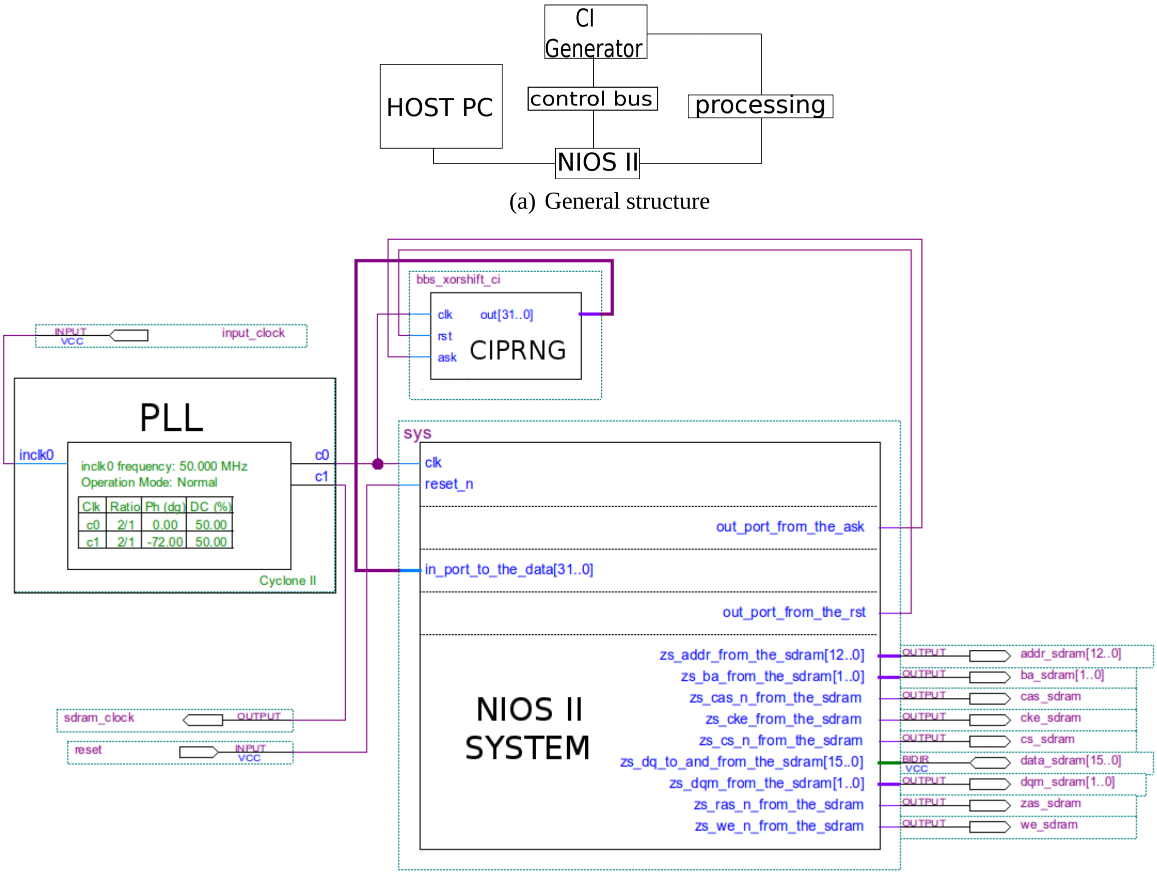
Let r_θ be the rotation of angle θ around the center $(128,128)$ of the carrier image. So, the transformation $r_{-\theta} \circ r_\theta$ is applied to the watermarked image. The results in Tab.3 have been obtained. The same conclusion as above can be declaimed.

5.4.3 JPEG compression

A JPEG compression is applied to the watermarked image, depending on a compression level. This attack leads to a change of the representation domain (from spatial to DCT domain). In this case, the results in Tab.3 have been obtained, illustrating a good authentication through JPEG attack. As for the unauthentication case, the watermark still remains after a compression level equal to 10. This is a good result if we take into account the fact that we use spatial embedding.

5.4.4 Gaussian noise

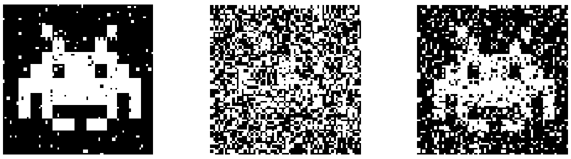
A watermarked image can be also attacked by the addition of a Gaussian noise, depending on a standard deviation. In this case, the results in Tab.3 are obtained, which are quite satisfactory another time.



(a) General structure

(b) Schematic view

Fig. 5: NIOS II setting in FPGA



(a) Unauthentication (10 × 10)

(b) Authentication (10 × 10)

(c) Unauthentication (50 × 50)

Fig. 6: Extracted watermark after a cropping attack (zoom ×2)

5.5 Discussion

Generally, the quality of a PRNG depends, to a large extent, on the following criteria: randomness, uniformity, independence, storage efficiency, and reproducibility. A chaotic sequence may satisfy these requirements and also other chaotic properties, as ergodicity, entropy, and expansivity. A chaotic sequence is extremely sensitive to the initial conditions. That is, even a minute difference in the initial state of the system can lead to enormous differences in the final state, even over fairly small timescales. Therefore, chaotic sequence fits the requirements of pseudorandom sequence well. Contrary to XORshift, our generator possesses these chaotic properties [17, 27]. However, despite a large number of papers published in the field of chaos-based pseudorandom generators, the impact of this research is rather marginal. This is due to the following reasons: almost all PRNG algorithms using chaos are based on dynamical systems defined on continuous sets (*e.g.*, the set of real numbers). So these generators are usually slow, requiring considerably more storage space and lose their chaotic properties during computations. These major problems restrict their use as generators [28].

In the CIPRNG method, we do not simply integrate chaotic maps hoping that the implemented algorithm remains chaotic. Indeed, the PRNG we conceive is just discrete chaotic iterations and we have proven in [27] that these iterations produce a topological chaos as defined by Devaney: they are regular, transitive, and sensitive to

Table 3: Robustness agains attacks

Attacks	UNAUTHENTICATION		AUTHENTICATION	
Cropping	Size (pixels)	Similarity	Size (pixels)	Similarity
	10	99.18%	10	50.06%
	50	96.13%	50	54.44%
	100	91.21%	100	52.04%
	200	66.16%	200	50.88%
Rotation	Angle (degree)	Similarity	Angle (degree)	Similarity
	2	96.11%	2	71.41%
	5	93.66%	5	60.03%
	10	92.55%	10	53.87%
	25	82.05%	25	50.09%
JPEG compression	Compression	Similarity	Compression	Similarity
	2	81.90%	2	53.79%
	5	66.43%	5	55.51%
	10	61.82%	10	51.24%
	20	54.17%	20	47.33%
Gaussian noise	Standard dev.	Similarity	Standard dev.	Similarity
	1	75.16%	1	51.05%
	2	62.33%	2	50.35%
	3	56.34%	3	49.95%

initial conditions. This famous definition of a chaotic behavior for a dynamical system implies unpredictability, mixture, sensitivity, and uniform repartition. Moreover, as only integers are manipulated in discrete chaotic iterations, the chaotic behavior of the system is preserved during computations, and these computations are fast.

These chaotic properties are behind the observed robustness of the proposed information hiding scheme: transitivity, for instance, implies that the watermark is spread over the whole host image, making it impossible to remove it by a simple crop. Regularity implies that the watermark is potentially inserted several times, reinforcing the robustness obtained by topological mixing and transitivity. Expansivity and sensitivity guarantee us that authentication is reached, as in an authenticated watermarking, MSBs are taken into account, and even a slight alteration of these bits leads to a completely different extracted watermark due to these metrical properties. Finally, unpredictability plays obviously an important role in the security of the whole process againts malicious attacks, even if this role is difficult to measure precisely in practice.

6 Conclusion and future work

In this paper, the pseudorandom generator proposed in our former research work has been developed in terms of efficiency. We also have proven that this generator based on hardware can be cryptographically secure. By using a BBS generator and due to a new approach in the way the Version 1 CI PRNG uses its strategies, the generator based on chaotic iterations works faster and is more secure. This new CIPRNG is able to pass NIST test suite when considering software implementation, and to reach 6000 Mbps (with the throughtput is about 132/16 each processing round) in FPGA hardware. These considerations enable us to claim that this CIPRNG(BBS, XORshift) offers a sufficient speed and level of security for a whole range of applications where secure generators are required as cryptography and information hiding.

In future work, we will continue to explore new strategies and iteration functions. The chaotic behavior of the proposed generator will be deepened by using the various tools provided by the mathematical theory of chaos. Additionally a probabilistic study of its security will be done. Lastly, new applications in computer science will be proposed, among other things in the Internet security field.

References

[1] Slobodan Bojanic, Gabriel Caffarena, Slobodan Petrovic, and Octavio Nieto-Taladriz. Fpga for pseudorandom generator cryptanalysis. *Microprocessors and Microsystems*, 30(2):63 – 71, 2006.

[2] J. L. Danger, S. Guilley, and P. Hoogvorst. High speed true random number generator based on open loop structures in fpgas. *Microelectron. J.*, 40(11):1650–1656, November 2009.

[3] K. H. Tsoi, K. H. Leung, and P. H. W. Leong. Compact fpga-based true and pseudo random number generators. In *Proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, FCCM '03, pages 51–61, Washington, DC, USA, 2003. IEEE Computer Society.

[4] Massimo Falcioni, Luigi Palatella, Simone Pigolotti, and Angelo Vulpiani. Properties making a chaotic system a good pseudo random number generator. *Phys. Rev. E*, 72:016220, Jul 2005.

[5] Songul Cecen, R. Murat Demirer, and Coskun Bayrak. A new hybrid nonlinear congruential number generator based on higher functional power of logistic maps. *Chaos, Solitons & amp; Fractals*, 42(2):847 – 853, 2009.

[6] Po-Han Lee, Yi Chen, Soo-Chang Pei, and Yih-Yuh Chen. Evidence of the correlation between positive lyapunov exponents and good chaotic random number sequences. *Computer Physics Communications*, 160(3):187 – 203, 2004.

[7] Laurent Larger and John M. Dudley. Nonlinear dynamics: Optoelectronic chaos. *Nature*, 465(7294):41–42, 05 2010.

[8] Robert L. Devaney. *An Introduction to Chaotic Dynamical Systems, 2nd Edition*. Westview Pr (Short Disc), March 2003.

[9] Jacques Bahi, Christophe Guyeux, and Qianxue Wang. A novel pseudo-random generator based on discrete chaotic iterations. In *INTERNET'09, 1-st Int. Conf. on Evolving Internet*, pages 71–76, Cannes, France, August 2009.

[10] Jacques Bahi, Xiaole Fang, and Christophe Guyeux. An optimization technique on pseudorandom generators based on chaotic iterations. In *INTERNET'2012, 4-th Int. Conf. on Evolving Internet*, pages 31–36, Venice, Italy, June 2012.

[11] Jacques Bahi, Xiaole Fang, Christophe Guyeux, and Qianxue Wang. Evaluating quality of chaotic pseudo-random generators. application to information hiding. *IJAS, International Journal On Advances in Security*, 4(1-2):118–130, 2011.

[12] Jacques Bahi and Christophe Guyeux. Topological chaos and chaotic iterations, application to hash functions. In *IJCNN'10, Int. Joint Conf. on Neural Networks, joint to WCCI'10, IEEE World Congress on Computational Intelligence*, pages 1–7, Barcelona, Spain, July 2010. Best paper award.

[13] Andrew Rukhin, Juan Soto, James Nechvatal, Elaine Barker, Stefan Leigh, Mark Levenson, David Banks, Alan Heckert, James Dray, San Vo, Andrew Rukhin, Juan Soto, Miles Smid, Stefan Leigh, Mark Vangel, Alan Heckert, James Dray, and Lawrence E Bassham Iii. A statistical test suite for random and pseudorandom number generators for cryptographic applications, Accessed: 30/09/2011. http://csrc.nist.gov/publications/nistpubs/800-22-rev1a/SP800-22rev1a.pdf.

[14] Lenore Blum, Manuel Blum, and Michael Shub. A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing*, 15:364–383, 1986.

[15] George Marsaglia. Xorshift rngs. *Journal of Statistical Software*, 8(14):1–6, 7 2003.

[16] J. Terno. Robert, f., discrete iterations. a metric study. berlin-heidelberg-new york-tokyo, springer-verlag 1986. xvi, 195 s., 126 abb., dm138,. isbn 3-540-13623-1 (springer series in computational mathematics 6) translation from the french. *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift fr Angewandte Mathematik und Mechanik*, 67(11):578–578, 1987.

[17] Qianxue Wang, Christophe Guyeux, and Jacques Bahi. A novel pseudo-random generator based on discrete chaotic iterations for cryptographic applications. *INTERNET '09*, pages 71–76, 2009.

[18] Jacques M. Bahi, Raphaël Couturier, Christophe Guyeux, and Pierre-Cyrille Héam. Efficient and cryptographically secure generation of chaotic pseudorandom numbers on gpu. *CoRR*, abs/1112.5239, submitted in Dec. 2011.

[19] F. Montoya Vitini, J. Monoz Masque, and A. Peinado Dominguez. Bound for linear complexity of bbs sequences. *Electronics Letters*, 34:450–451, 1998.

[20] Verilog hdl. http://www.verilog.com/IEEEVerilog.html, 2008. Accessed: 30/09/2012.

[21] X. Wu and Z. Guan. A novel digital watermark algorithm based on chaotic maps. *Physical Letters A*, 365:403—-406, 2007.

[22] Christophe Guyeux, Nicolas Friot, and Jacques M. Bahi. Chaotic iterations versus spread-spectrum: Chaos and stego security. In *IIH-MSP'10, 6-th Int. Conf. on Intelligent Information Hiding and Multimedia Signal Processing*, pages 208–211, Darmstadt, Germany, October 2010.

[23] Jacques M. Bahi and Christophe Guyeux. An improved watermarking algorithm for internet applications. In *INTERNET'2010. The 2nd Int. Conf. on Evolving Internet*, pages 119–124, Valencia, Spain, September 2010. IEEE seccion ESPANIA.

[24] Jacques Bahi, Jean-François Couchot, and Christophe Guyeux. Steganography: A class of secure and robust algorithms. *The Computer Journal*, 55(6):653–666, 2012.

[25] Jacques Bahi and Christophe Guyeux. A new chaos-based watermarking algorithm. In *SECRYPT'10, Int. conf. on security and cryptography*, pages 455–458, Athens, Greece, July 2010. SciTePress.

[26] Introduction to the altera nios ii soft processor. http://coen.boisestate.edu/smloo/files/2011/11/, 2011. Accessed: 30/09/2012.

[27] Jacques M. Bahi and Christophe Guyeux. Hash functions using chaotic iterations. *Journal of Algorithms & Computational Technology*, 4(2):167–181, 2010.

[28] L. Kocarev. Chaos-based cryptography: a brief overview. *IEEE Circ Syst Mag*, 7:6–21, 2001.

Jacques Mohcine Bahi received a Master of Science in applied mathematics from the University of Franche-Comte (France). He received his Ph.D. in Applied Mathematics from the same university in 1991. Until september 1999, he was a associate professor of applied mathematics at the Mathematical Laboratory of Besanon. Since then, he became a full professor of computer science at the University of Franche-Comte. He is a IEEE senior member and currently the Vice-President of the Scientific Council of the University of Franche-Comte.

Xiaole Fang is a third year doctoral student in the computer science department of complex system (DISC), FEMTO-ST Institute, University of Franche-Comt under Professor Jacques Bahi and Laurent Larger. The main objective of his thesis is to explore different possible approaches efficiently (in terms of speed and quality of randomness) to extract pseudo-random number sequence by adapting new mathematical topology properties. Before arriving in University of Franche-Comt, he completed two postgraduate studies: Master of Science in Theory and Engineering of Control (2006-2008) and Electronics and Electrical Engineering Systems (2009). Since 2010, he has published 2 articles in international journal, and 4 articles in peer reviewed international conferences.

Christophe Guyeux has taught mathematics and computer science in the Belfort-Montbliard University Institute of Technologies (IUT-BM) this last decade. He has defended a computer science thesis dealing with security, chaos, and dynamical systems in 2010 under Jacques Bahi's leadership, and is now an associated professor in the computer science department of complex system (DISC), FEMTO-ST Institute, University of Franche-Comt. Since 2010, he has published two books, 9 articles in international journals, and 25 articles in peer reviewed international conferences dealing with security or chaos.

Laurent Larger received the Degree in electronic engineering from the University of Paris XI, Orsay, France, in 1988, the Agrgation degree in applied physics in 1991, and the Ph.D. degree in optical engineering and the Habilitation degree from the University of Franche-Comt, Besanon, France, in1997 and 2002, respectively. He was in charge of the International Research Center GTL-CNRS Telecom, a joint laboratory between the French CNRS, Georgia Tech University, Atlanta, and the University of Franche-Comt, Besanon, from 2003 to 2006. He became a Full Professor with the University of FrancheComt in 2005. He is involved in research with the Franche Comt Electronique, Mcanique Thermique et Optique - Sciences et Technologies Institute, Besanon. His current research interests include the study of chaos in optical and electronic systems for secure communications, delayed nonlinear dynamics, optical telecommunication systems, high spectral purity optoelectronic oscillators, and neuromorphic photonic computing exploiting the complexity of nonlinear dynamical transients. Prof. Larger is a honorary member of the Institut Universitaire de France. He has been a Deputy Director of the FEMTO-ST Research Institute, Besanon, since 2012