A New TRNG Based on Coherent Sampling With Self-Timed Rings

Honorio Martin, Pedro Peris-Lopez, Juan E. Tapiador, and Enrique San Millan

***Abstract*—Random numbers play a key role in applications such as industrial simulations, laboratory experimentation, computer games, and engineering problem solving. The design of new true random generators (TRNGs) has attracted the attention of the research community for many years. Designs with little hardware requirements and high throughput are demanded by new and powerful applications. In this paper, we introduce the design of a novel TRNG based on the coherent sampling (CS) phenomenon. Contrary to most designs based on this phenomenon, ours uses self-timed rings (STRs) instead of the commonly employed ring oscillators (ROs). Our design has two key advantages over existing proposals based on CS. It does not depend on the FPGA vendor used and does not need manual placement and routing in the manufacturing process, resulting in a highly portable generator. Our experiments show that the TRNG offers a very high throughput with a moderate cost in hardware. The results obtained with ENT, DIEHARD, and National Institute of Standards and Technology (NIST) statistical test suites evidence that the output bitstream behaves as a truly random variable.**

***Index Terms*—Coherent sampling (CS), FPGAs, self-timed ring (STR), true random generator (TRNG).**

I. Introduction

SOURCES of random numbers are always in demand. They play a key role in computer games, problem solving techniques in engineering, industrial simulations, security primitives and protocols, and a variety of other applications. In many cases, the quality of the randomness must be as high as possible, e.g., when used in security applications to generate keys, nonces, session identifiers, etc., while in others it is also required a very high throughput in the generation process. In this regard, hardware-based pseudo and true random number generators [pseudo-random number generator (PRNG) and true random generator (TRNG), respectively] are very appealing because of their superior perfomance when compared with software implementations [1]–[3].

Motivated by this, many researchers have pointed out the convenience of using field programmable gate arrays (FPGAs) as TRNG platforms due to their low cost and versatility [4]–[6]. However, FPGAs offer a resource-constrained environment (fixed logic blocks) that does not include analog blocks, which are frequently employed to generate very entropic outputs. Thus, the typical phenomena used in the generation of randomness in FPGAs are metastability and jitter [7], [8]. Since FPGAs are initially designed and implemented to reduce their random behavior, it is considerably more challenging to implement a TRNG in an FPGA than in other digital devices.

From the implementation point of view, a TRNG should not be technology dependant. For instance, in [9] it is presented a TRNG that exploits the technique of coherent sampling (CS) and uses an analog phase-locked loop (PLL) to obtain a fine control over the clock signal. In particular, the proposed TRNG is implemented in an Altera FPGA, which includes a PLL. Unfortunately, this design is not portable to the FPGAs of other important manufacturers of the sector, e.g., such as the ones produced by Xilinx, whose FPGAs mainly use delay-locked loops (DLLs) instead of PLLs. Apart from being technological independent, proposed designs should not be device dependant. For example, Kohlbrenner and Gaj present in [10] a TRNG that uses ring oscillators (ROs) and takes advantage of the CS technique. The design consists of two independent and identically configured ROs with similar but not identical frequency. A sampling circuit uses one clock signal to sample the other clock signal. Although the design works well theoretically, Kohlbrenner and Gaj show how not only there is a great variation between the RO frequencies in the same FPGA (7%) but also between different FPGAs. In other words, the design depends so much on the used device that it has to be manually tuned (placement and routing) for each FPGA implementation.

To the best of our knowledge, one of the first simple and portable designs of a TRNG suitable for different FPGAs was presented by Sunar *et al.* in [11]. Nevertheless, this design was rapidly discarded since it suffers from implementation problems, mostly related to the number of signals handled by the XOR-tree. Moreover, the quality of the raw signal is rather poor and needs postprocessing. In [12], Wold and Tan present an enhanced design based on Sunar *et al.*'s, which attempt to solve its implementation problems while avoiding the need of a post-processing stage. This enhanced version proposes a reduction in the number of rings that, as reported in [13], cause the loss of entropy. However, such a lack of entropy is masked by

Manuscript received August 27, 2014; revised May 29, 2015 and August 28, 2015; accepted October 27, 2015. Date of publication November 20, 2015; date of current version February 02, 2016. This work was supported in part by the Ministerio de Economia y Competitividad (MINECO), Security and Privacy in the Internet of You (SPINY), under Grant TIN2013-46469-R, and in part by the Comunidad de Madrid (CAM), Cybersecurity, Data, and Risks (CIBERDINE), under Grant S2013/ICE-3095. Paper no. TII-15-0310.

H. Martin and E. S. Millan are with Department of Electronics Technology, Universidad Carlos III de Madrid, 28911 Madrid, Spain (e-mail: hmartin@ing.uc3m.es; quique@ing.uc3m.es).

P. Peris-Lopeze and J. E. Tapiador are with the Computer Security Laboratory, Department of Computer Science, Universidad Carlos III de Madrid, 28911 Madrid, Spain (e-mail: pperis@inf.uc3m.es; jestevez@inf.uc3m.es).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TII.2015.2502183

the pseudorandomness caused by XOR-ing clock signals having different frequencies. In addition, these two designs were successfully attacked in [14] and [15] exploiting the use of ROs and their vulnerabilities to frequency injection, in which the ROs are locked to an injected frequency and the jitter phenomena as a source of randomness is neutralized.

More recently, inspired by Sunar *et al.*'s design, Cherkaoui *et al.* [16] have presented a new design in which the ROs are replaced by a self-timed ring (STR). An STR is a multiphase generator that can maintain constant phase difference between the different outputs. Therefore, this construction is resistant to the common vulnerabilities used to attack TRNGs based on ROs. Cherkaoui *et al.*'s TRNG seems to be a secure design (no attack has been published yet), but the design consumes a substantial amount of resources in terms of power and circuit area. This renders it unsuitable for constrained devices. More specifically, the STR generates 63 signals that are sampled and finally passed through an XOR tree, generating a high activity and, correspondingly, having a high power consumption. Moreover, the hardware requirements are superior to the ones that can be afforded in most constrained devices.

In this paper, we present a new TRNG based on the CS technique. On the one hand, the design takes advantage of some key STR features, which help us to solve the implementation problems (mainly the device dependence) suffered by Kohlbrenner and Gaj's design [10], while simultaneously avoiding the vulnerabilities linked to the use of ROs. On the other hand, the proposed design is very efficient in hardware, which makes it suitable for devices with limited capabilities, and offers a relatively high throughput.

This paper is organized as follows. In Section II, we describe the CS technique. We explain the phenomenon and the randomness extraction technique. An overview of STRs and their operation principles is presented in Section III. In Section IV, our proposal is presented together with some implementation considerations needed. The experimental results, both about the randomness quality and hardware requirements, are presented in Section V, together with a comparison between our proposal and the most relevant designs. Finally, Section VI concludes the paper and summarizes our main contributions.

II. Coherent Sampling

In this section, we first introduce the principles of CS. After that, we present the main TRNG proposals that exploit this technique.

A. Background

CS is a well-known technique to sample periodic signals at finer time intervals. CS refers to an integer number of cycles that fits into a predefined sampling window. Mathematically, this can be expressed as

$$\frac{f_{\text{in}}}{f_{\text{sample}}} = \frac{N_{\text{cyc}}}{N_{\text{samples}}} \tag{1}$$

where f_{in} is the sampled signal (S_1) frequency, f_{sample} is the sampling signal (S_2) frequency, N_{cyc} is the number of cycles of the sampled signal, and N_{samples} is the number of samples.

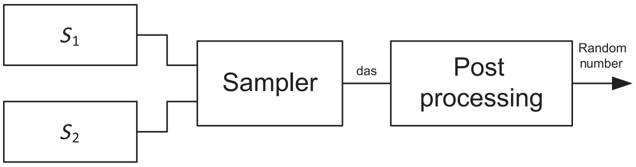


Fig. 1. General architecture of a TRNG based on CS.

If N_{cyc} and N_{samples} are high and coprimes, the repetition period of samples will be maximum, i.e., we will have the highest resolution of the sampled signal. This is an interesting feature because if the number of periods (frequencies) is constant for ideal sources of S_1 and S_2, in physical systems where these clock signals contain jitter, this number will be random because of the Gaussian random component contained in the jitter.

The general architecture of a TRNG using CS is depicted in Fig. 1. The signal S_1 would be sampled by the signal S_2, generating a digitized analog signal known as "das." If the quality of the raw output is not high enough, a postprocessing stage is added to guarantee a uniform output. A mathematical model of physical RNGs based on CS can be found in [17].

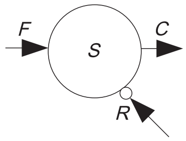
B. TRNGs Based on CS

The first time, to the best of our knowledge, that CS was used in an FPGA to generate random numbers was in [9]. In that work, Fischer and Drutarovsky used a PLL embedded in an Altera FPGA to guarantee the relation between N_{cyc} and N_{samples}. As explained in Section I, the main drawback of this proposal is that the TRNG is not portable to other FPGA vendors. Besides, PLLs are not supported in all FPGAs.

In [10], Kohlbrenner and Gaj replaced PLLs by ROs with the aim of obtaining a portable design for FPGAs from different vendors. The RO frequencies are selected to be close but not identical. The RO outputs are connected to a sampler circuit that generates a stream of 0's and 1's. The length of this stream is counted module 2 to generate a random bit. The weakest point of this design is that it requires a very complicated manual placement and routing process to finely set the ring frequencies. According to [10], this is a consequence of the high variation (up to 7%) between the RO frequencies in the same FPGA. To overcome such a sensitivity to placement, the authors suggest a design with four ROs that are sampled by a fifth one.

In [18], Cret *et al.* take up the basic idea of using only two ROs. In this design, the authors introduce a multiplexer to alternate the sampling signal. They claim that the placement sensitivity is overcome using a parametrizable postprocessing. The main weakness of this TRNG is that the quality of the raw output, without the postprocessing stage, is really poor. In addition, Cret *et al.* present the cycle lengths of the signal generated in the sampler and its distribution is not an evidence of the claimed randomness—which is actually far away from a uniform distribution.

Finally, in [19], the authors present three designs based on different clock generators for different FPGA models. More precisely, the generators are RO–RO, RO–PLL (for Altera FPGAs), and RO–DFS (for Xilinx FPGAs). Apart from the



F	R	C
0	0	C^{-1}
0	1	0
1	0	1
1	1	C^{-1}

Fig. 2. Structure and truth table of an STR stage.

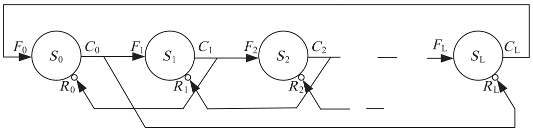


Fig. 3. STR structure.

technology dependency, the pair RO–RO cannot be fully automated since its design needs manual placement and routing. Finally, it is worth mentioning that the authors introduce the interesting idea of generating one random bit per half period by using mutual sampling.

III. Self-Timed Rings

STRs are a well-known structure to generate clock signals in digital devices. An STR implements an asynchronous handshake protocol that assures an even distribution of events through the different stages. For this, a multiphase oscillator based on a STR is tuneable by adjusting the frequency and the phase resolution between signals.

A. Architecture

The basic element in an STR is a stage. Each stage consists of a Muller gate and an inverter, and implements the truth table shown in Fig. 2. If the forward input F is different from the reverse input R, the output C takes the same value as F; otherwise, the previous output is maintained.

The STR architecture implements a micropipeline (ripple FIFO) introduced by Sutherland in [20] (see Fig. 3). The handshake protocol used guarantees the phase distribution between the micropipeline stages.

B. Behavior and Configuration

In order to understand the STR operation we need to define the following parameters.

1) L: The number of stages that compose the STR. Each stage can be initialized either to 0 or 1.
2) *Tokens* and *Bubbles*: A stage contains a *token* if its output C_i is not equal to the output C_{i+1}. Conversely, a stage contains a *bubble* if its output C_i is equal to the output C_{i+1}. The number of tokens (NT) and bubbles (NB) can be chosen during the initialization phase.
3) N: The number of events distributed throughout the ring, which equals the number of propagating tokens in the ring.

TTTTBBBB (01010000) → TTTBTBBB (01011000) →

TTBTBTBB (01011000) → TBTBTBTB (01100110) →

BTBTBTBT (11001100) → TBTBTBTB (10011001) →

BTBTBTBT (00110011) → TBTBTBTB (01100110) ···

Fig. 4. Example of tokens and bubbles propagation in an STR.

A token will propagate to the next stage (s_{i+1}) if this stage contains a bubble. The bubble will occupy the backward stage s_i. The STR will have an oscillatory behavior if there are at least three stages, one bubble, and an even number of tokens. For example, in an eight-stage STR with an initial distribution of four tokens and four bubbles, the events will propagate as shown in Fig. 4.

It is important to note that the propagation delay of the ring depends on two analog phenomena: 1) charlie effects; and 2) drafting effects. The Charlie effect is related to the separation time between events at the inputs: a shorter separation time of events results in a longer stage propagation delay. The drafting effect connects the gate propagation delay with the elapsed time for the output events. According to [21], the drafting effect is negligible in FPGAs. We refer the reader to [22], where a complete model that explain both phenomena and how they affect the ring propagation delay can be found.

As for the configuration possibilities, the frequency can be fine tuned in the initialization phase by changing the ratio between tokens and bubbles, i.e., NT/NB. The maximum frequency is reached when

$$\frac{\mathrm{NT}}{\mathrm{NB}} \simeq \frac{D_{\mathrm{ff}}}{D_{\mathrm{rr}}} \tag{2}$$

where D_{ff} and D_{rr} are the static forward and static reverse propagation delays, respectively.

The phase shift between two stages separated by n stages can be calculated as

$$\varphi_n = n \times \frac{N}{L} \times 90^\circ. \tag{3}$$

Note that if L is a multiple of N, some outputs will have the same phase, as it happens in the example shown in Fig. 4. In particular, there will be four different phases throughout the ring, with each phase appearing in two different stages (stage$_i$ and stage$_{i+4}$). Thus, for applications of this oscillator in which the goal is typically to generate the maximum number of different phases, $L \bmod N$ should not be equal to 0.

IV. Our Design

The design presented in this paper is inspired by the TRNG proposed by Kohlbrenner and Gaj in [10]. We use the same architecture but replace the ROs by two STRs. Cherkaoui *et al.* carried out in [21] an exhaustive comparison between ROs and STRs. According to this work, the main differences are as follows.

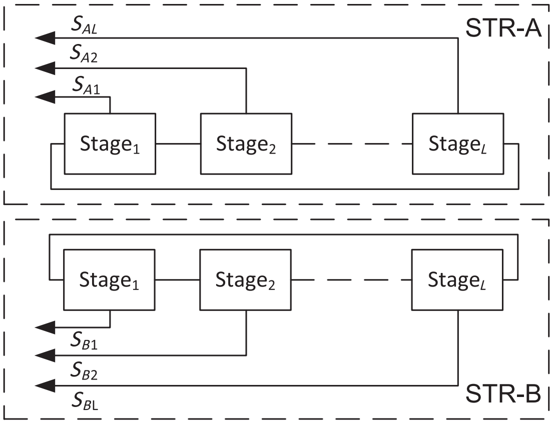


Fig. 5. STR structure of our TRNG.

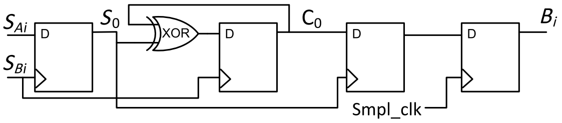


Fig. 6. Sampler structure of our TRNG.

1) STR robustness to voltage variations can be enhanced by adding more stages. ROs do not offer this feature.
2) STRs present a lower extra-device frequency variation when operating at high frequencies.
3) In STRs, the period jitter does not depend on the number of stages but it is mostly dependant on the jitter generated at each stage.

From the security point of view, these features are very interesting. In fact, in [21], the authors conclude that STRs are more robust to attacks than ROs and this property is inherited by our proposal. Furthermore, replacing ROs by STRs provides our design with the possibility of having at least L different signals in each STR. Each one of those L signals can be used as a sampling or sampled signal, since each stage can be considered as an independent source of entropy [16], the number of stages is equal to the number of independent entropy sources. Moreover, STRs are highly configurable. In particular, it is very easy to set the desired frequency for the STR output, which allows a fine-grained control over the resulting speed of the TRNG.

A. Architecture Overview

Figs. 5 and 6 show the different blocks that make up our TRNG. Fig. 5 depicts the two STRs used in our design. Both STRs are composed of L stages that generate L different outputs with a frequency f_{STR}. The number of tokens and bubbles are selected in the reset phase attending to the frequency and phase necessities.

The jitter contained in the STR outputs is extracted using the sampler circuit shown in Fig. 6. Each sampler circuit is composed of four-dimensional (4-D)-type flip-flops and one XOR gate. The first flip-flop uses the signal S_{Bi} to sample the signal S_{Ai}. The signal S_0 will be high while the rising edges of S_{Bi} occur during the high level of S_{Ai}. In Fig. 7, we show the behavior of S_0 taking into account that S_{Bi} contains jitter. As consequence of such a jitter, the cycle length of S_0 will not be constant.

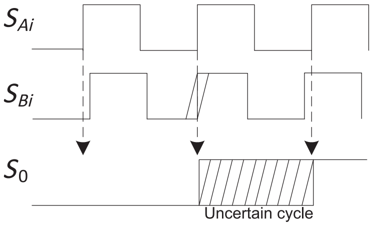


Fig. 7. Sampler behavior.

In our design, both signals S_{Bi} and S_{Ai} contain jitter. As a variation of the original sampler design that includes a 1-bit counter latched by S_0, in our design, we use the simplified version presented in [19]. In this scheme, instead of counting the cycles of S_{Bi}, we count the number of cycles that S_0 is at a high level. If such a number of cycles is even, the previous output is maintained; otherwise, the output changes. Two D flip-flops and one XOR gate are involved in this process. Finally, the last flip-flop samples the signal C_0 using an external clock. This external clock determines the TRNG throughput. As our design is composed of two STRs with L stages each, L sampler circuits are necessary (see Fig. 1).

Finally, our design includes a postprocessing unit that might be needed depending on the quality, in terms of randomness, of the raw data. The selected postprocessing is a parity filter, which has been widely used as postprocessing in previous proposals such as [16] and [18]. More precisely, an nth parity filter takes n consecutive bits and XOR all of them together to produce one bit. This postprocessing offers a simple bias reduction with the penalty of a throughput reduction—the filter reduces the bit generation by a factor of n.

V. Experimental Results

In order to evaluate the portability of our proposal, we have implemented our design on FPGAs from three different manufacturers: 1) a Spartan-3E XC3S500E FPGA from Xilinx; 2) an Igloo M1AGL1000 from Microsemi; and 3) a Cyclone II EP2C5F256C8 from Altera. As expected, the obtained results are similar in all of them. In addition, to show the independence of our design from the manufacturing technology, we have also implemented one final chosen design on another two different FPGAs that use different process technologies: 1) a Virtex 5 XC5VLX110T (65 nm); and 2) a Virtex 6 XC6VLX240T (40 nm) from Xilinx. In the following, we discuss our results in detail.

Two eight-stage STRs have been implemented and configured in the reset phase to obtain an STR output frequency of 300 MHz. Several frequencies have been used in the external clock that samples the signal C_0. Eight bits are generated with each rising edge of the sampling clock.

In order to obtain almost the same propagation delay in the different stages, a hard macro (or its equivalent for other FPGA vendors) has been designed. This hard macro implements a Muller gate and an inverter using a single look-up table (LUT).

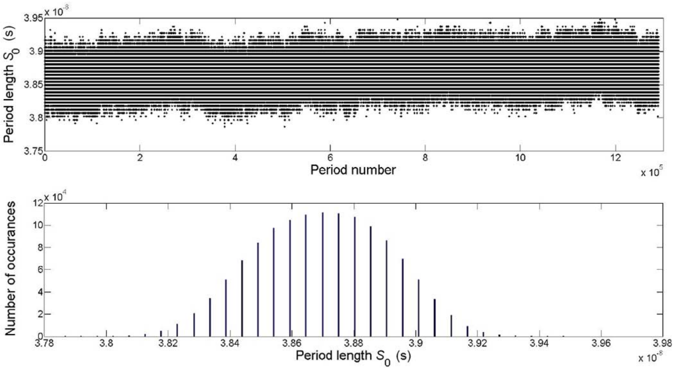


Fig. 8. Time evolution and histogram of S_0.

We have chosen two STRs with eight stages since this configuration is easily tunable and offers a good tradeoff between area and throughput. In addition, this configuration allows the generation in parallel of 8 bits (1 Byte), which is a typical bit length used in many applications. The throughput goal has been set to 1 Mb/s to be comparable to other TRNGs proposals based on CS. This throughput threshold will set the lowest sampling frequency that can be used in our design.

A. Testing Randomness

In this section, we discuss the quality of the TRNG output in terms of randomness. Following the standard practice in this field, we first show that the STR outputs have Gaussian jitter and then report the results obtained with three widely used suites of statistical tests for cryptographic applications. We have also carried out a restart experiment to provide evidence that the output is different after repeatedly restarting the system under the same conditions.

Due to space limitations, all results reported in this section correspond to traces obtained with the Spartan-3E XC3S500E FPGA. The conclusions for the other four FPGAs are identical to those shown here.

1) Evidence of the Gaussian Jitter: In order to show evidence of the presence of Gaussian jitter in the STR outputs, we have counted the number of cycles of the signal S_0, as done in [19] and [10]. Fig. 8 depicts the time evolution of the S_0 length (top) and a histogram of the cycles (bottom). The histogram population corresponds with $1.3{\times}10^6$ measurements. The average period of S_0 is 38.69 ns with an standard deviation of 0.215 ns. As the frequency of the STR has been set to 300 MHz, which means that the average cycle length is 11.61 cycles. In conclusion, the histrogram distribution clearly shows evidence of the underlying randomness in the sampling process, and by extension, in each stage of the STR.

2) Restart Experiment: Following the same procedure used in [12] and [23] to distinguish the amount of true randomness contained in a pseudorandom oscillating signal, we have carried out a restart experiment. In Fig. 9, nine oscillograms of repeated restarts from identical starting conditions are presented. The horizontal axis represents the time and shows the first 20 bits generated after each restart (the period of time shown for each restart is 20 μs using a sampling clock of 1 MHz). The vertical axis is the voltage of the output signal. Only nine curves out of the 1000 generated are shown. It is clear that the TRNG generates different traces after the same restarting point.

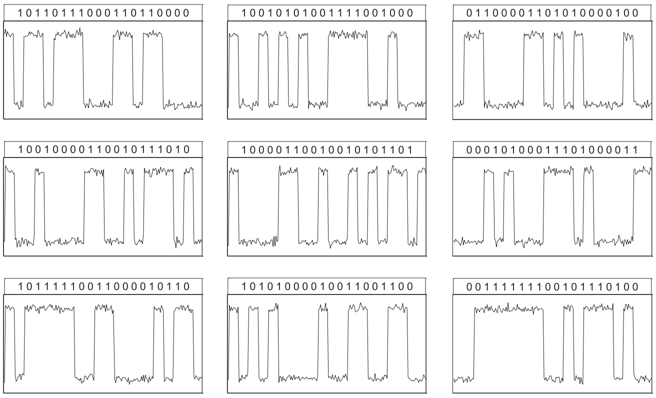


Fig. 9. Nine output sequences captured after restarting the TRNG. Note that all sequences are different.

3) Statistical Evaluation of the Output: The testing of our proposal has been carried out using the NIST statistical test suite [24], as commonly done to validate previous proposals (e.g., [9], [10], [18]). To transfer the bits generated by the TRNG in the FPGA to the host computer where the NIST tests are executed, a FIFO memory and an RS232 communication protocol have been used. In addition, the postprocessing has been conducted in the host computer in order to reduce the acquisition time of the traces.

We have evaluated the TRNG output for the following set of sampling frequencies: 50, 25, 10, 5, 1, and 0.5 MHz). A higher sampling frequency will imply a higher throughput, but also a lower quality of the random bits due to the fact that the jitter accumulation time is shorter. According to the study presented in [25], a longer accumulation time is desirable so that the contribution of the thermal noise (responsible of the nondeterministic jitter) is perceptible. On the other hand, the use of a longer accumulation time causes that the flicker noise (responsible of the deterministic jitter) dominates the jitter. This paradox forces designers to find a tradeoff to set the sampling frequency.

For the postprocessing, we have tested the minimum parity filter order (bit-wise XOR tree) necessary to pass the NIST tests for the different sampling frequencies studied. A third-order filter is needed for 50 MHz, while a second-order filter suffices for the rest. As expected, the postprocessing necessities are higher when higher sampling frequencies are used. Although many sampling frequencies need the same order parity filter, it is important to notice that the proportion of failed tests before the postprocessing rises when the sampling frequency is increased, as explained below. This is a crucial point if for some reason the TRNG will be used without the postprocessing block.

We have evaluated the quality of the raw data before the postprocessing for the six sampling frequencies studied. Fig. 10 shows boxplots of the p-value distribution for each sampling

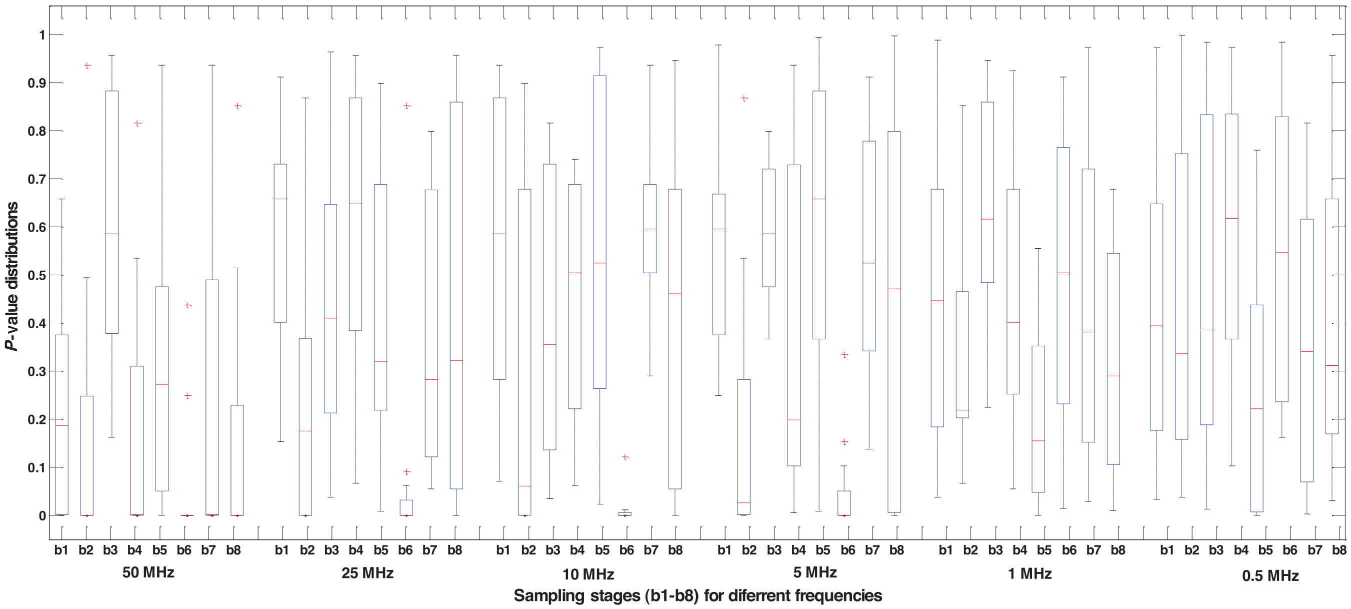


Fig. 10. Boxplots of *p*-value distributions for each sampling stage ($b1$ to $b8$) and different frequencies.

TABLE I
EXPERIMENTAL RESULTS: PASS RATE (PR) PROPORTION AND AVERAGE P-VALUE (PV) FOR GENERATED TRACES

	0.5 MHz		1 MHz		5 MHz		10 MHz		25 MHz		50 MHz	
	PR	PV	PR	PV	PR	PV	PR	PV	PR	PV	PR	PV
$b1$	97.91	0.41	98.58	0.44	98.41	0.57	99.08	0.55	99.5	0.58	93.83	0.22
$b2$	98.41	0.43	99.33	0.34	82.58	0.18	82.5	0.30	84.66	0.22	39.66	0.16
$b3$	98.83	0.46	99.08	0.62	98.58	0.59	98.91	0.41	98	0.44	99.33	0.60
$b4$	99.66	0.59	99.41	0.45	99.33	0.36	99.41	0.45	99	0.59	83.41	0.17
$b5$	92.66	0.25	87.16	0.20	98.5	0.59	98.25	0.55	99.16	0.39	97.91	0.30
$b6$	98.41	0.54	98.75	0.47	31.66	0.04	31.58	0.01	31.25	0.08	22.25	0.05
$b7$	98.75	0.36	99.16	0.45	98.83	0.53	98.91	0.61	98.58	0.36	66.58	0.24
$b8$	99.16	0.39	98.91	0.31	98	0.44	97.5	0.40	96.83	0.43	48.5	0.15
Total	97.97	0.43	97.55	0.41	88.23	0.41	88.27	0.41	88.37	0.39	68.93	0.24

stage ($b1$–$b8$) and different frequencies. According to the documentation provided by NIST, a random stream must present uniformity in the distribution of its p-values. It can be seen in Fig. 10 that higher sampling frequencies presents less uniformity for its p-values distribution than lower sampling frequencies, which are more uniform.

Further evidence of this phenomenon is presented in Table I, which shows the proportion of traces that pass the statistical tests (PR) and the average p-value (PV) for the different sampling stages and frequencies. Note that traces corresponding to $b5$ and $b6$ perform quite badly, specially $b6$. For sampling frequencies of 0.5 and 1 MHz, only a single trace ($b5$) fails the NIST tests before the postprocessing. It is noteworthy, however, that $b5$ fails the tests by a narrow margin. Three traces of $b5$ fail the tests for the sampling frequencies of 5, 10, and 25 MHz, and seven traces fail for 50 MHz. As for $b6$ traces, they fail badly for the sampling frequencies between 5 and 50 MHz. This consistent behavior in $b6$ is mainly due to the fact that the synthesizer has placed the sampling stage that generates the $b6$ stream in a way that causes a huge delay between the sampling (S_{A6}) and the sampled (S_{B6}) signals. This problem could be solved using a manual placement and routing process. In fact, we have tested this using manual placement and routing and setting the sampling frequency to 50 MHz results in a design such that the raw stream of bits without postprocessing passes the NIST tests. Nevertheless, one major design goal of our proposal is to avoid such a manual procedures.

We have evaluated the quality of our proposed TRNG after postprocessing. A sampling frequency of 1 MHz has been selected for this experimentation since this frequency offers a tradeoff between throughput and randomness quality before the postprocessing stage. We have opted for having a good quality signal without postprocessing to make stronger our TRNG proposal against some attacks. ENT [26], DIEHARD [27], NIST [24], and AIS31 [28] suites have been used for analyzing the randomness quality.

In Table II, we summarize the results obtained with ENT, which resemble those obtained with a genuine random variable, such as the chi-square test is passed, entropy is extremely high, the serial correlation is very low, etc. DIEHARD is a much more demanding battery of tests for checking randomness. As in the case of the NIST suite, DIEHARD is particularly

TABLE II
ENT RESULTS FOR A SAMPLING FREQUENCY SET TO 1 MHz

Entropy	7.999987
Compression	0%
Chi square distribution	261.92 (36.96%)
Arithmetic mean	127.5110
Monte Carlo value for Pi	3.141718075
Serial correlation coefficient	0.000368

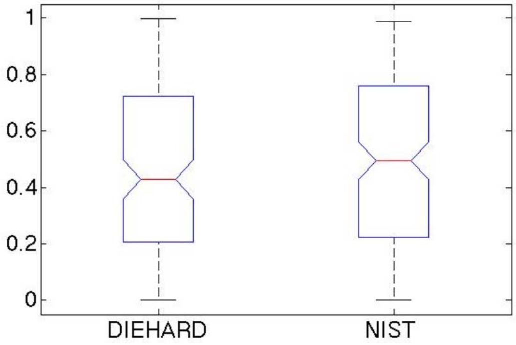


Fig. 11. Distribution of p-values for the DIEHARD and NIST test suites.

TABLE III
AIS31 RESULTS FOR THREE FPGAs

Device	Frequency (MHz)	Rawdata		Postprocessing
		T1-T4	T5-T8	n_{pmin}
Spartan-3E	1	4/4	3/4	2
	50	3/4	3/4	3
Igloo	1	4/4	4/4	-
	50	4/4	3/4	2
Cyclone II	1	4/4	3/4	2
	50	3/4	3/4	3

designed for cryptographic applications and includes a number of statistical tests (e.g., frequency, rank, fft, monkey, runs, and so on). A final p-value is obtained for each test. If we take a significance level of 0.05, $p < 0.025$ or $p > 0.975$ means that the TRNG fails the test. To show evidence that our proposed TRNG behaves as a random variable, in Fig. 11, we depict the distribution of p-values for all tests included in both suites. In particular, all the p-values in the NIST and DIEHARD suites are within the interval $[0.2, 0.8]$, so the TRNG passes all tests in both suites.

Finally, we have evaluated the data acquired from the three FPGAs using the AIS31 statistical test suite. Using two sampling frequencies of 50 and 1 MHz, we have gathered a 1-MB sequence of raw data. The results for the AIS31 statistical suite are depicted in Table III. Note that tests $T1$–$T4$ correspond to four FIPS 140-1 tests (poker, monobit, runs, and long runs). $T5$ is an autocorrelation test, $T6$ is a uniform distribution test, $T7$ is a comparative test for multinomial distributions, and $T8$ is an entropy test. According to the AIS31 recommendations, raw data from the TRNG output or at least data at the output of the arithmetic postprocessing, should pass $T5$ through $T8$. The column n_{pmin} represents the minimum filter order to comply with this requirement. For the AIS31 results—as shown in Table III and, equivalently, for NIST p-values in Fig. 10—we have obtained better results for lower frequencies.

TABLE IV
HARDWARE RESOURCES

FPGA	Hardware resources
Cyclone II EP2C5F256C8	43 LUTs and 48 Registers
Igloo M1AGL1000	93 Core Cells (48 DFFN)
Spartan-3E XC3S500E	32 LUTs and 48 Registers
Virtex 5 XC5VLX110T	32 LUTs and 48 Registers
Virtex 6 XC6VLX240T	32 LUTs and 48 Registers

From all of the above, we can conclude that our TRNG outputs a bit stream that looks like a true random variable.

B. Hardware Resources

The results presented in this section correspond to our chosen design with a sampling frequency of 1 MHz. The architecture consists of two eight-stage STRs, eight sampling stages, and a second-order parity filter as postprocessing block.

Since each STR stage uses a single LUT, the STRs occupies $2 \times L$ LUTs. As shown in Fig. 6, the sampler structure uses four registers and an XOR gate (one LUT). Therefore, the number of LUTs used by the sampler structure is L and the number of registers is $4L$. Finally, the postprocessing requirements depend on the parity filter of order n. The LUTs used by the postprocessing is also conditioned by the inputs of each LUT. Since a four-input XOR gate, as in the case of two-input XOR gates, can be implemented using one LUT, the number of LUTs, and registers will be L and nL, respectively—the filter order is 2 for 1 MHz sampling frequency, as explained in Section V-A3.

In summary, observing the results above we can conclude that each raw random bit (before postprocessing) has a cost of three LUTs and four registers. Therefore, for a given sampling frequency (f_{sampling}), a designer could improve the throughput by adding more stages to the STRs. This will result in f_{sampling} bps per additional stage. On the other hand, this improvement translates into a circuit area penalty of three LUTs and four registers per additional stage.

Table IV summarizes the amount of resources needed to implement our TRNG on five different FPGAs. The differences in terms of the combinational logic elements for the set of FPGAs analyzed are related to the optimizations carried out by the different synthesis tools. Note that the same amount of hardware resources are obtained for the three Xilinx FPGAs (i.e., Spartan and Virtex). This is due to the fact that we have tailored the hard macro created for the Spartan-3E to fit into the Virtex 5 and 6. It is important to emphasize the decision of implementing each STR stage in a single LUT to have almost the same delay between consecutive stages to avoid bottleneck effects.

Regarding throughput, our proposal is able to generate random bits in parallel. For the proposed architecture, eight random bits are generated each two clock cycles. This feature can be very interesting for some applications that require the generation of random bits in parallel.

TABLE V
TRNG COMPARISON

		Hardware resources	Throughput	Hardware complexity	Portability
Our proposal		32 LUTs 48 Registers	4 Mb/s	Medium	Yes
Fischer *et al.* [9]		121 LCs 4 ESBs and 1 PLL	69 Kb/s	Medium	No
Kohl *et al.* [10]		12 LUTs 24 Registers	300 Kb/s	High	Yes
Valtchanov *et al.* [19]	RO-RO	15 LUTs 4 Registers	2 Mb/s	High	Yes
	RO-PLL	12 LCs 4 Registers and 1 PLL	2 Mb/s	Medium	No
	RO-DFS	11 LUTs 6 Registers and 2 DFS	2 Mb/s	Medium	No
Cherkaoui *et al.* [16]		320 LUTs 320 Registers	200 Mb/s	Medium	Yes

C. Comparison With TRNGs Based on CS

Next, we present a comparison between our proposal and other TRNG designs that use CS. We also include the proposal of Cherkaoui *et al.* [16] since it is based on STRs. For each proposal, we have analyzed the hardware resources needed and the offered throughput. In addition, we have also considered the hardware complexity, including the degree of automation of the design, and its portability (device independence). Regarding hardware complexity, we distinguish three categories.

1) Low complexity is devoted for designs that can be easily implemented.
2) Medium complexity implies designs that need to use hard macros or specific components like PLL or DFS.
3) High complexity considers designs that require a manual place and route process.

Finally, the portability aspect represents whether the design needs special resources or efforts to be implemented in different FPGA vendors or devices.

We emphasize here that the hardware results presented in Table V constitute an estimation for the designs in which the authors do not provide specific results. For Cherkaoui *et al.*'s proposal, we selected the architecture that implements 255 stages.

Table V shows the comparison between our design and other TRNG proposals. It can be noted that our proposal offers a very good tradeoff between the set of parameters evaluated. TRNGs that need a complicated place and route process (e.g., [10] and RO–RO [19]) are superior in terms of hardware resources, but these designs have the drawback of requiring a specific design for each particular device. Among the TRNGs based on CS, our design offers the highest throughput. Note that this could be even better if a higher sampling frequency would have been selected, although this might degrade the quality of the random signal. On the other hand, Cherkaoui *et al.*'s TRNG presents the highest throughput, but uses around 10 times more resources than our proposal. As aforementioned, in terms of throughput our TRNG generates eight random bits in parallel. Finally, it is worth mentioning that our proposal is highly portable and complies with the two requirements set in Section I; i.e., our design is technology and device independent.

D. Comparison With Other FPGA-Based TRNGs

In Section V-C, we have carried out an exhaustive comparison between our proposal and other TRNGs based on CS. Now, we present a qualitative comparison with other state-of-the-art TRNGs implemented on FPGAs that present some interesting metrics.

Among the several proposals reported in this field, the TRNG proposed by Varchola and Drutarovsky [29] stands out because of its lightweightness. This design takes advantage of the metastability on a new bi-stable structure—transition effect ring oscillator (TERO). In terms of area (two CLBs), this TRNG is more lightweight than our proposal but present a very poor throughput (250 Kb/s). In addition, experiments show that proper placement and routing strategies are essential.

Exploiting some features of embedded RAMs has become a popular principle nowadays because of the high throughput that can be achieved. Among the proposals that take advantage of SRAMs, those that use write collisions to extract entropy are worth mentioning [30], [31]. The key idea here consists of generating a conflict in a particular address by trying to write opposite values at the same time. In comparison with our TRNG, these proposals present better results in terms of area and throughput, but their portability constitutes a handicap because an enrolment process is necessary in order to identify distinctive BRAMs in each FPGA.

Another interesting proposal based on high fanout nets was presented by Cret *et al.* in [32]. Its performance is remarkable regarding portability and its high throughput (60 Mb/s). However, in terms of area, our proposal outperforms this design. Very recently, Wieczorek presented a new FPGA-based TRNG [33] that offers metrics similar to those of our design. The portability of this TRNG is currently under study because only a Xilinx implementation has been reported. Another interesting work is the complementary design proposed in [34], which outperforms ours in terms of throughput but whose portability has not been deeply studied since the results are only validated on a Virtex-6 FPGA. Furthermore, the design in [34] includes a place and route procedure to guarantee the TRNG randomness, which implies some extra effort for the designer when implementing the TRNG in different FPGAs.

All in all, our TRNG presents an attractive tradeoff among hardware footprint, throughput, and portability when compared with existing proposals.

VI. CONCLUSION

There is a wide set of applications, ranging from security services to simulations, computer games, and problem solving tools, where pseudorandom number generators play a central role. In many cases, as a consequence of the high throughput required and the high-quality randomness demanded, the use of software-based solutions is simply infeasible. Motivated by this, many FPGA-based proposals have appeared recently.

TRNGs in FPGAs mainly exploit metastability and jitter phenomena as sources of randomness. In this paper, we have proposed a TRNG based on CS, which is a phenomenon that seems to provide good results in previous proposals. Most previous works based on CS rely on either a PLL or an RO. The use of these components has two major drawbacks.

1) It makes the design dependent of the FPGA vendor, for instance, not all FPGA vendors support PPLs.
2) It requires manual placement and routing for setting particular frequencies for each device.

To avoid these two drawbacks, we have proposed a novel design where ROs or PLLs are replaced by STRs. We argue that the use of STRs is very convenient, because it provides robustness against frequency and voltage variations while simultaneously offering one independent source of entropy for each ring stage. Thus, the resulting TRNG combines the power of CS and the robustness and portability linked to STRs. Furthermore, our design does not depend on the FPGA vendor, and the placement and routing is performed automatically by the synthesis tool.

Our proposal outperforms all previous TRNGs based on CS and its throughput could be further increased if we relax our requirements about the quality of the random signals before the postprocessing (e.g., for non-cryptographic applications). We have studied in detail the most restrictive design with a sampling frequency set to 1 MHz. In terms of randomness, our TRNG passes all batteries of tests for checking the randomness of a random number generator (ENT, DIEHARD, and AIS31), and also others like NIST that are devoted to evaluate generators designed for cryptographic applications.

References

[1] S. Saab, J. Hobeika, and I. Ouaiss, "A novel pseudorandom noise and band jammer generator using a composite sinusoidal function," *IEEE Trans. Signal Process.*, vol. 58, no. 2, pp. 535–543, Feb. 2010.

[2] J.-L. Danger, S. Guilley, and P. Hoogvorst, "High speed true random number generator based on open loop structures in FPGAS," *Microelectron. J.*, vol. 40, no. 11, pp. 1650–1656, 2009.

[3] R. Vaidyanathaswami and A. Thangaraj, "Robustness of physical layer security primitives against attacks on pseudorandom generators," *IEEE Trans. Commun.*, vol. 62, no. 3, pp. 1070–1079, Mar. 2014.

[4] X. Fang, Q. Wang, C. Guyeux, and J. M. Bahi, "Fpga acceleration of a pseudorandom number generator based on chaotic iterations," *J. Inf. Sec. Appl.*, vol. 19, no. 1, pp. 78–87, 2014.

[5] D. B. Thomas and W. Luk, "The lut-sr family of uniform random number generators for FPGA architectures," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 4, pp. 761–770, Apr. 2013.

[6] P. Wieczorek, "Dual-metastability FPGA-based true random number generator," *Electron. Lett.*, vol. 49, no. 12, pp. 744–745, Jun. 2013.

[7] D. Lubicz and N. Bochard, "Towards an oscillator based trng with a certified entropy rate," *IEEE Trans. Comput.*, vol. 64, no. 4, pp. 1191–1200, Apr. 2015.

[8] V. B. Suresh and W. Burleson, "Entropy extraction in metastability-based trng," in *Proc. IEEE Int. Symp. Hardware-Oriented Sec. Trust (HOST)*, Jun. 2010, pp. 135–140.

[9] M. Fischer and V. Drutarovsky, "True random number generator embedded in reconfigurable hardware," in *Proc. Int. Workshop Cryptogr. Hardware Embedded Syst. (CHES'02)*, 2002, vol. 2523, pp. 415–430.

[10] P. Kohlbrenner and K. Gaj, "An embedded true random number generator for FPGAS," in *Proc. 12th Int. Symp. Field Programm. Gate Arrays (ACM/SIGDA'04)*, 2004, pp. 71–78.

[11] S. Sunar, W. J. Martin, and D. R. Stinson, "A provably secure true random number generator with built-in tolerance to active attacks," *IEEE Trans. Comput.*, vol. 58, no. 1, pp. 109–119, Jan. 2007.

[12] K. Wold and C. H. Tan, "Analysis and enhancement of random number generator in FPGA based on oscillator rings," *Int. J. Reconfig. Comput.*, vol. 2009, pp. 4:1–4:8, 2009.

[13] N. Bochard, F. Bernard, V. Fischer, and B. Valtchanov, "True-randomness and pseudo-randomness in ring oscillator-based true random number generators," *Int. J. Reconfig. Comp.*, vol. 2010, 2010, Article ID 879281 [Online]. Available: http://dx.doi.org/10.1155/2010/879281

[14] P. Bayon *et al.*, "Contactless electromagnetic active attack on ring oscillator based true random number generator," in *Proc. 3rd Int. Workshop Construct. Side-Channel Anal. Secure Des. (COSADE)*, 2012, pp. 151–166.

[15] A. T. Markettos and S. W. Moore, "The frequency injection attack on ring-oscillator-based true random number generators," in *Proc. 11th Int. Workshop Cryptogr. Hardware Embedded Syst.*, 2009, pp. 317–331.

[16] A. Cherkaoui, V. Fischer, L. Fesquet, and A. Aubert, "A very high speed true random number generator with entropy assessment," in *Proc. Int. Workshop Cryptogr. Hardware Embedded Syst. (CHES'13)*, 2013, vol. 8086, pp. 179–196.

[17] F. Bernard, V. Fischer, and B. Valtchanov, "Mathematical model of physical RNGS based on coherent sampling," *Tatra Mt. Math. Publ.*, vol. 45, pp. 1–14, 2010.

[18] O. Cret, A. Suciu, and T. Gyorfi, "Practical issues in implementing TRNGS in FPGAS based on the ring oscillator sampling method," in *Proc. 10th Int. Symp. Symbol. Numer. Algorithms Sci. Comput. (SYNASC'08)*, 2008, pp. 433–438.

[19] B. Valtchanov, V. Fischer, and A. Aubert, "Enhanced TRNG based on the coherent sampling," in *Proc. 3rd Int. Conf. Signals Circuits Syst. (SCS'09)*, 2009, pp. 1–6.

[20] I. E. Sutherland, "Micropipelines," *ACM Commun.*, vol. 32, no. 6, pp. 720–738, 1989.

[21] A. Cherkaoui, V. Fischer, A. Aubert, and L. Fesquet, "Comparison of self-timed ring and inverter ring oscillators as entropy sources in FPGAS," in *Proc. Des. Autom. Test Eur. Conf. Exhib. (DATE'12)*, 2012, pp. 1325–1330.

[22] A. Winstanley and M. Greenstreet, "Temporal properties of self-timed rings," in *Correct Hardware Design and Verification Methods*, vol. 2144. New York, NY, USA: Springer, 2001, pp. 140–154.

[23] M. Dichtl and J. D. Golic, "High-speed true random number generation with logic gates only," in *Cryptographic Hardware and Embedded Systems (CHES)*, vol. 4727, P. Paillier and I. Verbauwhede, Eds. New York, NY, USA: Springer, 2007, pp. 45–62.

[24] A. Rukhin *et al.*, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," Natl. Inst. Stand. Technol., Gaithersburg, MD, USA, Tech. Rep., 2010 [Online]. Available: http://csrc.nist.gov/rng/

[25] P. Haddad, Y. Teglia, F. Bernard, and V. Fischer, "On the assumption of mutual independence of jitter realizations in P-TRNG stochastic models," in *Proc. Des. Autom. Test Eur. Conf. Exhib. (DATE'14)*, 2014, pp. 1–6.

[26] J. Walker. (1998). *Randomness Battery* [Online]. Available: http://www.fourmilab.ch/random/

[27] G. Marsaglia. (1996). *The Marsaglia Random Number CDROM Including the Diehard Battery of Tests of Randomness* [Online]. Available: http://stat.fsu.edu/pub/diehard

[28] W. Schindler and W. Killmann, "Evaluation criteria for true (physical) random number generators used in cryptographic applications," in *Proc. Revised Papers 4th Int. Workshop Cryptogr. Hardware Embedded Syst.*, 2003, pp. 431–449 [Online]. Available: http://dl.acm.org/citation.cfm?id=648255.752732

[29] M. Varchola and M. Drutarovsky, "New high entropy element for fpga based true random number generators," in *Cryptographic Hardware and Embedded Systems (CHES 2010)*, vol. 6225, S. Mangard and F.-X. Standaert, Eds. New York, NY, USA: Springer, 2010, pp. 351–365.

[30] T. Guneysu and C. Paar, "Transforming write collisions in block RAMs into security applications," in *Proc. Int. Conf. Field-Programm. Technol. (FPT'09)*, Dec. 2009, pp. 128–134.

[31] T. Gyorfi, O. Cret, and A. Suciu, "High performance true random number generator based on FPGA block RAMs," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. (IPDPS'09)*, May 2009, pp. 1–8.

[32] O. Cret, T. Gyorfi, and A. Suciu, "Implementing true random number generators based on high fanout nets," *Rom. J. Inf. Sci. Technol.*, vol. 15, no. 3, pp. 277–298, 2012 [Online]. Available: www.scopus.com

[33] P. Wieczorek, "An FPGA implementation of the resolve time-based true random number generator with quality control," *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 61, no. 12, pp. 3450–3459, Dec. 2014.

[34] X. Yang and R. C. C. Cheung, "A complementary architecture for high-speed true random number generator," in *Proc. Int. Conf. Field-Programm. Technol. (FPT)*, Dec. 2014, pp. 248–251.

Honorio Martin received the Ph.D. degree in advanced electronics systems, from Universidad Carlos III de Madrid, Spain, in 2015.

He is a Postdoctoral Researcher with the Department of Electronics Technology, Universidad Carlos III de Madrid, Madrid, Spain. His research interests include the study of lightweight cryptography hardware implementations, radio-frequency identification systems, and low-power designs.

Pedro Peris-Lopez received the M.Sc. degree in telecommunications engineering and the Ph.D. degree in computer science from Universidad Carlos III de Madrid, Spain, in 2007.

He is a Visiting Lecturer with the Department of Computer Science, Universidad Carlos III de Madrid, Madrid, Spain. He has authored a great number of papers in specialized journals and conference proceedings on radio-frequency identification systems (RFID), and implantable medical devices (IMD). His research interests include protocols design, primitives design, lightweight cryptography, cryptanalysis, RFID, and IMD.

Juan E. Tapiador received the M.Sc. and Ph.D. degrees in computer science from the University of Granada, Granada, Spain, in 2000 and 2004, respectively.

He is an Associate Professor with the Department of Computer Science, Universidad Carlos III de Madrid (UC3M), Madrid, Spain. Between 2009 and 2011, he was Research Associate with the University of York, York, U.K., before joining UC3M. His research interests include applied cryptography, and computer and network security.

Enrique San Millan received the M.Sc. degree in mathematics from La Rioja University, Logroño, Spain, in 1996 and the Ph.D. degree in mathematics engineering from the Universidad Carlos III de Madrid, Madrid, Spain, in 2003.

He is an Associate Professor with the Department of Electronics Technology, Universidad Carlos III de Madrid. His research interests include hardware design of digital circuits and systems for several fields (cryptography, biometry, fault tolerant systems, and communications), and computer assisted design (CAD) tools for design automation and optimization of digital integrated circuits.