Pushing the Optimization Limits of Ring Oscillator-Based True Random Number Generators

Andrei Marghescu[1,2(✉)] and Paul Svasta[1]

[1] "Politehnica" University of Bucharest, CETTI, Splaiul Independentei, nr. 313, Sector 6, 060042 Bucharest, Romania
{andrei.marghescu,paul.svasta}@cetti.ro

[2] Advanced Technology Institute, Str. Dinu Vintila nr. 10, Sector 2, 021102 Bucharest, Romania
ati@dcti.ro

Abstract. True Random Numbers are widely used in different security areas, like Public Key Cryptography, Symmetric Encryption Algorithms, security protocols (key exchange, nonce generator), etc., because of their defining unpredictability. True Random Number Generators (TRNG) are formally composed of three main components: a Noise Generator, which is based on a physical nondeterministic phenomenon (like cosmic radiations or the jitter of an oscillator), a Randomness Extractor and a Randomness Tester. Ring Oscillators (RO) are commonly chosen for this generators because of their simplicity in FPGA implementation. A RO consists of an odd number of inverters representing basically a clock signal of whose frequency depends mainly on the number of inverters. This paper describes a novel optimization technique (aiming the speed and resource consumption) for the implementation of TRNG based on Ring Oscillators and some good conclusive results.

Keywords: FPGA · CPLD · TRNG · Security · Randomness

1 Introduction

Random Number Generators are widely spread in engineering, being used in various applications like cryptography, artificial intelligence, simulations, gaming, etc. These generators split into two categories: Pseudo Random Number Generators (PRNG), which are reproducible, being based on a mathematical function and True Random Number Generators (TRNG), which are non-predictable, being based on physical nondeterministic phenomenon. The PRNGs are mostly used in stream cipher algorithms (to generate the same cryptographic key by 2 parties, at the same time).

True Random Numbers represent a very sensible part of a cryptographic system. They are mainly used in generating either symmetric (for algorithms like One Time Pad) or asymmetric (when generating a public/private key, a good

© Springer International Publishing AG 2016
I. Bica and R. Reyhanitabar (Eds.): SECITC 2016, LNCS 10006, pp. 209–224, 2016.
DOI: 10.1007/978-3-319-47238-6_15

generator is needed to output a random sequence that is further tested according to some requirements) encryption keys. These generators are also used in key-exchange protocols like Diffie-Hellmann [9] or "challenge-response" schemes.

This paper is structured as follows: the second chapter describes the True Random Number Generator concept and its components while the third chapter will present the Ring Oscillator along with two noise acquisition techniques. The forth chapter presents the steps needed for a personalized solution of a True Random Number Generator that is based on Ring Oscillators to be optimal. The fifth chapter will present the statistical testing results of the proposed TRNG running in different setups, demonstrating that the generator is stable. Finally, the last chapter will present some conclusions.

Since True Random Numbers requires a dedicated hardware resource, the challenge is to develop a small-sized and cost-efficient generator. This paper describes how to create a good TRNG while using FPGA (or CPLD) resources at minimum.

2 True Random Number Generators

Generating True Random Numbers implies the interconnection of three main components, as described in Fig. 1.

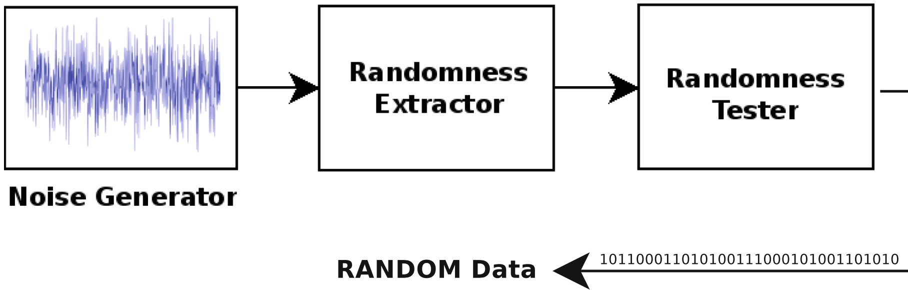

Fig. 1. True random number generator scheme.

The first component (the Noise Generator) is basically the one responsible for the generation process, outputting Random Data based on some unpredictable and non-deterministic phenomenon. The Randomness Extractor is responsible for uniform distribution of the data acquired from the Noise Generator and the last component, the Randomness Tester, is responsible for testing the random sequences according to a battery of statistical tests.

2.1 Noise Generators

Noise Generators are basically the pillars of a True Random Number Generator. They consists mainly of a hardware structure with unpredictable properties. The

unpredictability comes from a physical process like cosmic radiations, hardware imperfections, the reaction of a specific component while exposed to certain external factors, etc.

2.2 Randomness Extractors

Usually, the output (0 or 1) of a Noise Generator tends towards 50 %, defining a Gaussian Distribution. The Randomness Extractor is used to prevent the eventual deviations, by trying to uniformly distribute the output bits as much as possible.

The most usual and simple Randomness Extractor is Von Neumann which works with pairs of two bits, dropping the pairs where there are two identical bits and outputting the first bit of the others. Von Neumann's Randomness Extractor's output table can be seen in Table 1.

Table 1. Von Neumann randomness extractor output

Input1	Input2	Output
0	0	DROP
0	1	0
1	0	1
1	1	DROP

The main idea behind the Randomness Extractor is that if we play a heads or tails game with a biased coin and if we toss the coin twice, the probability that the first result is head and the second is tail and the probability that the first result is tail and the second is head tends to equality.

2.3 Randomness Testers

Randomness Tests are used to find correlations of some sort over a bunch of random data. Their aim is to apply a battery of statistical algorithms and problems and trying to find out in which way the next outputted bit (or sequence) can be predicted [6,7].

The most known Statistical Tests were developed by the United States of America's National Institute of Standards and Technology [10] treating the following:

Frequency - This test is based on the counting of "1" and "0" bits.
Block Frequency - This test analyzes the frequencies of blocks of data, having the same algorithm as the first one.
Cumulative sums - This test calculates the sums of partial sequences within the tested ones;
Runs - This test tries to identify sequences of bits that occur multiple times among the tested ones, calculating the number of occurred runs;

Longest Run - This test uses the data from the previous one and calculates the length of the longest run;
Rank - This test calculates the rank of disjoint matrices that could be computed with the input sequence;
FFT - This test calculates and interprets the Fast Fourier Transform peak heights;
NonOverlappingTemplates - This test comes with a set of predefined patterns, calculating their occurrences.
OverlappingTemplate - This test works the same as the previous one but it uses different search engines.
Universal - This test is trying to apply compression algorithms over the sequence, knowing that a True Random Sequence cannot be efficiently compressed.
Aproximate Entropy - This test compares the frequencies of n bit blocks and the n + 1 bit blocks.
Serials - This test searches for fixed length patterns and counting their apparitions among the data;
LinearComplexity - Any random data can be regenerated using a custom LFSR (Linear Feedback Shift Register). This test calculates the length of such LFSR that could generate the tested sequence.

3 Rig Oscillators as Noise Generators

Using an odd number of inverters ("*NOT*" gates) that are interconnected like in Fig. 2 provides a digital clock signal (alternating the logical states 0 and 1). The signal frequency is directly dependent on the number of inverters as well as their position inside the FPGA logic (the distance between them influences the timing).

Due to fabric and/or technology imperfections, a phenomenon called *jitter* occur, resulting in a slightly different clock period (Fig. 3).

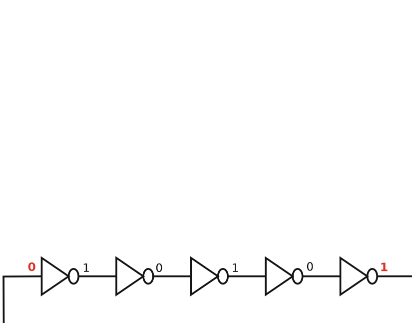

Fig. 2. Ring oscillator.

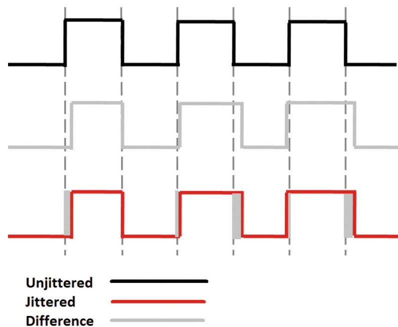

Fig. 3. Jittery oscillator.

The jitter, which is very small in terms of period, has a Gaussian Distribution and could be very hard to enhance and emphasize through measurements.

A lot of different setups are used in order to exploit the imperfections of the ring oscillators [4] some by using schemes consisting of a large number of them (emphasizing a randomness acquisition technique called De-synchronization Technique) and others by using a jittery oscillators in which the jitter is measured (using the randomness acquisition technique called Jitter Counting Technique).

Some other setups are using the scheme in a slightly different way in a generator named TERO [2,3], which is mainly based on both de-synchronization and counting as measurement, also providing a reliable TRNG.

3.1 Jitter Counting Technique

The first approach in the Randomness Acquisition Techniques is based on the jitter measurement. This technique implies a very fast counter, that is usually implemented in FPGA logic. Since the jitter is not a reproducible phenomenon and its behavior is fully random, it can be used as a good and reliable Noise Source. Figure 4 emphasizes the jitter of an analog Trigger Schmitt Inverter-based Oscillator.

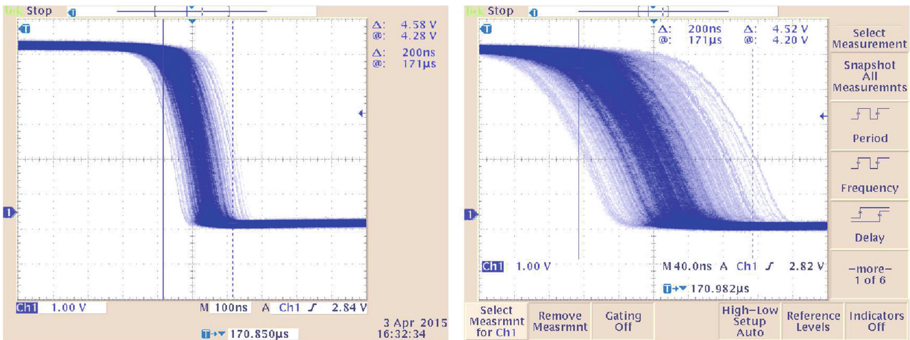

Fig. 4. Highlighted jitter from a trigger-schmitt oscillator [4]

It can be clearly observed from Fig. 4 that the jitter has a Gaussian Distribution. The Jitter Counting Technique highlights the jitter presence and works as follow (Fig. 5):

1. The Ring Oscillator is running freely (having no input source), outputting a clock signal;
2. A very fast counter starts counting while the RO signal is 1, emphasizing the period differences between clock periods. The counter resets itself when the RO outputs 0;
3. This technique usually uses the Least Significant Bit (LSB) of the counter's output.

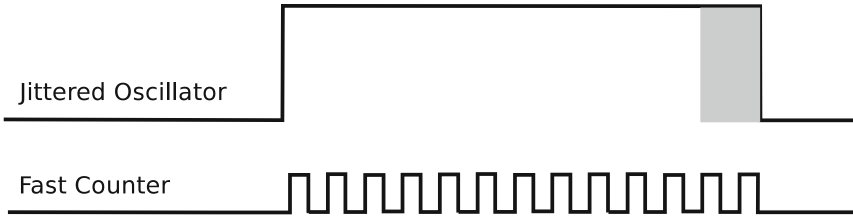

Fig. 5. Jitter counting technique

3.2 De-synchronization Technique

This technique uses a large number of different frequencies free running Ring Oscillators, which are connected to a XOR gate (Fig. 6). The scheme works using the following properties:

1. Each Ring Oscillator freely oscillates (does not require an input clock signal) at the frequency of F_i;
2. The XOR logical gate is powered by a clock signal running at the frequency of F_{sample};
3. $F_i \neq F_{sample}, \forall\, i \in (1, n)$, where n = the total number of RO's used;
4. The output of the Generator is the output of the XOR gate.

Even if the number of inverter gates per each Ring Oscillators is the same, their frequencies usually differ, depending on the physical distances between the corresponding logic gates that were used.

It is a good practice, when it comes to select the Ring Oscillators Frequencies to choose them as relative prime numbers. In this way, the probability for some oscillators to synchronize tends to nearly 0. The synchronization frequency can be approximate to *Least Common Multiplier* (Fi).

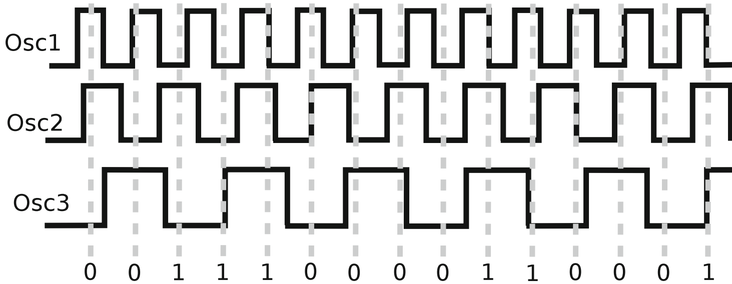

Fig. 6. De-synchronization technique

4 Proposed Solution

4.1 Related Work

Sunar et al. [1] proposed and demonstrated that a generator consisting of a large number of Ring Oscillators (114 for that paper) is provably secure. Their scheme works as presented in Fig. 7.

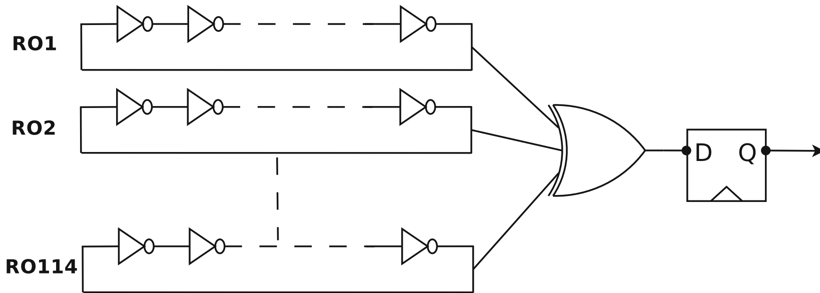

Fig. 7. Sunar's et al. TRNG scheme [1] and adapted for the ZYBO Zynq development board by Marghescu et al. in [5].

Sunar's scheme has the advantage of being secure and mathematically provable while having the disadvantage of using a lot of hardware (FPGA or CPLD) resources.

Marghescu et al. adapted Sunar's solution in [5] for a custom hardware that was used for this paper as well, using a slight different setup, obtaining positive results. This adaptation is one of the pillars of the proposed TRNG presented in this paper, being the speed "booster" of the scheme.

4.2 Chosen Hardware

The chosen hardware for this research is the Zybo Zynq-7000 System on Chip Development Board [11] that is based on an ARM Cortex A9 which powers a FPGA. The FPGA is essential for our project because it will store the TRNG, while the ARM side will manage the Randomness Testing and the communication protocol with the user (Fig. 8).

In other words, this hardware provides the capabilities of both generating and statistical testing of True Random Numbers.

4.3 Description of the Solution

Firstly, the first scheme, that is presented in Fig. 9, uses free running Ring Oscillators and works as follows:

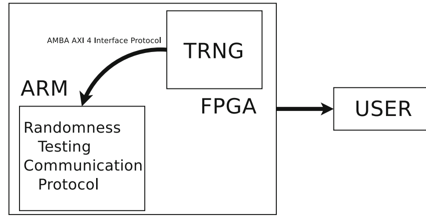

Fig. 8. Zynq TRNG schematic.

– Each RO consists of a prime number of inverter gates and a latch. The latch is present within the circuit to bypass the optimization of the compiler which doesn't recognize such schemes as valid ones;
– Each RO is connected to a Von Neumann Randomness (VN) Extractor Block;
– The VN block is connected to a clock signal as input (in our case the clock of the FPGA = 150 MHz) which tells it when to sample the free running Ring Oscillators;
– Each VN block has a *data_valid* signal, telling a controller when it has a valid bit to offer;
– The controller passes to each individual RO + VN joint, acquiring and storing the corresponding valid bit only when the *data_valid* signal of the VN block is 1;
– After passing to all RO + VN joints, the controller calculates the modulo 2 sum of the bits, outputting the resulting one, that is to be considered the True Random one.

By introducing the VN blocks within the scheme, we can assure that sequences collected from each RO are uniformly distributed and by combining them altogether we can compute a complex generator.

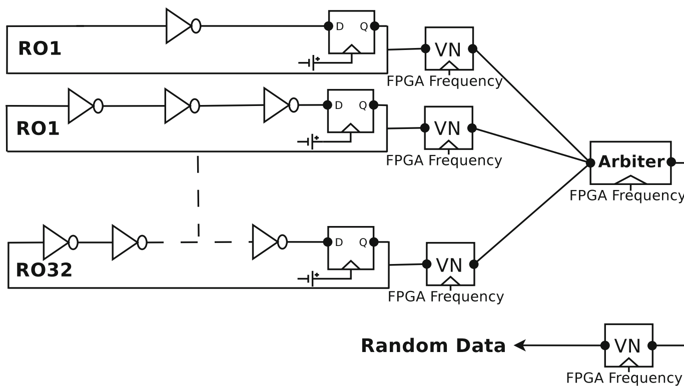

Fig. 9. Proposed TRNG scheme.

Sunar's scheme has two big advantages (the stability and its security) but it uses a lot of FPGA resources. Therefore, the authors tried to use its principle and combine it with the first generator. As a result, the second scheme, presented in Fig. 10, works mainly the same as the first one, but it splits the output of each Ring Oscillator in two. The first "half" of the signal is connected to the VN block (just as in the first one) powering the mechanism presented above.

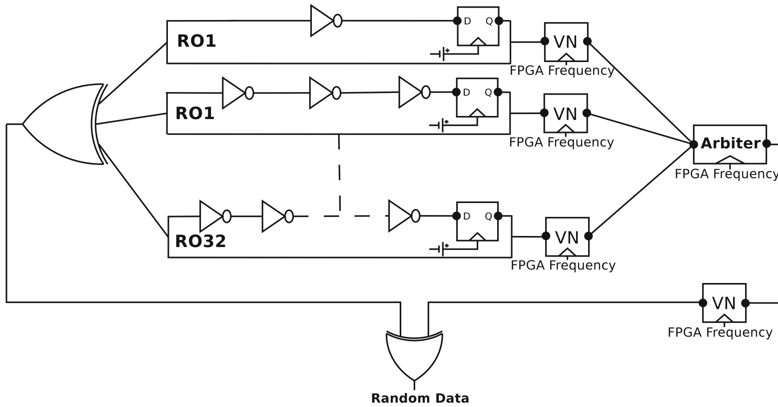

Fig. 10. Optimized TRNG scheme.

The second "half" is connected to a parallel scheme which uses the principle used by Sunar [1] and Marghescu et al. in [5]. Therefore, the second part of the generator is based on a free running XOR gate which combines the signals from all Ring Oscillators, outputting one bit. The output of the generator consists of the modulo 2 sum of the outputs of the two component parts.

This second "half" provides a very fast generation rate approximated at $CLK_{FPGA}/32$ (while working with 32 bit buffers), although, if it is taken alone, it doesn't offer a good generator itself (because the number of ROs is quite small).

Since it is well known that if a random sequence is XORed with any other, the result will still be random, the merging of the first half (greater speed, reduced complexity and statistical properties) with the second one (low speed and good statistical properties) provides a high speed True Random Number Generator with good statistical properties.

The Statistical Test Results of this two schemes are presented in Table 2.

For the proposed generator, the first 32 prime numbers were chosen for the number of inverter gates for each Ring Oscillator. The number 32 was presumed to be high enough for the first test and it led to positive results and since one of the goals of this paper is to optimize the generator by reducing the number of ROs, this number was chosen to be the starting point of the analysis.

The further optimization implies reducing the number of RO (by 2 for each step, dropping the biggest ROs each time) and testing the solution if it still provides good data.

5 Results

After the implementation of the presented schemes, the testing platform consists of the following:

- The TRNG IP from the FPGA, which runs at 150 MHz frequency, is communicating with the ARM side via AXI4 protocol;
- The ARM standalone application receives the random data from the FPGA and tests it using the NIST Statistical Test Suite;
- After the data is statistically validated, it is transmitted to the user using the UART at 115200 baud rate (this baud rate was chosen just for demonstrating the concept, the generator's output being much higher);

The proposed TRNG was the subject of the NIST Statistical Test Suit, and the results are presented in the next Tables, including the following:

1. A table that presents the results of the two generators described in the previous subsection (with and without speed acceleration);
2. The results of other 15 different TRNG setups (containing 32 RO, 30 RO, ..., 4 RO), that aim to optimize the resource consumption of the FPGA;

Each table presents on each row the statistical test that was applied to the random data, the P-value (described by NIST STS documentation [10]), the number of passed tests within the total amount of them (for instance 98/100) and the result (also described by the NIST STS documentation [10]).

As we can see from the presented tables, we can conclude that, from a statistical testing point of view, the generator which consumes the less hardware resources (4 Ring Oscillators), is suitable for using in TRNG applications (Tables 3, 4, 5, 6, 7, 8, 9, 10).

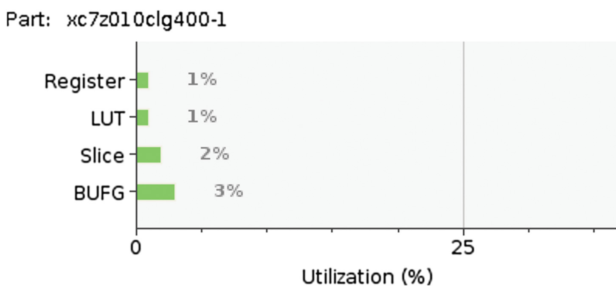

Fig. 11. Resource consumption of the 4osc implementation.

Table 2. NIST STS results for the TRNG with and without speed acceleration at a 150 MHz FPGA frequency

No	Statistical test	Without acceleration			With acceleration		
		P-value	Proportion	Result	P-value	Proportion	Result
1	Frequency	0.935716	97/100	Pass	0.946308	100/100	Pass
2	Block frequency	0.719747	100/100	Pass	0.534146	99/100	Pass
3	Cumulative sums	0.304126	98/100	Pass	0.514124	100/100	Pass
4	Runs	0.017912	100/100	Pass	0.262249	96/100	Pass
5	Longest run	0.275709	99/100	Pass	0.249284	97/100	Pass
6	Rank	0.494392	100/100	Pass	0.719747	98/100	Pass
7	FFT	0.419021	100/100	Pass	0.657933	100/100	Pass
8	NonOverlappingTemplates	0.035174	99/100	Pass	0.595549	98/100	Pass
9	OverlappingTemplate	0.987896	99/100	Pass	0.055361	96/100	Pass
10	Universal	0.213309	98/100	Pass	0.181557	98/100	Pass
11	Aproximate entropy	0.851383	98/100	Pass	0.994250	99/100	Pass
12	RandomExcursions	0.706149	60/60	Pass	0.327854	70/70	Pass
13	RandomExcursionsVariants	0.772760	60/60	Pass	0.169178	70/70	Pass
14	Serials	0.401199	99/100	Pass	0.867692	98/100	Pass
15	LinearComplexity	0.249284	100/100	Pass	0.474986	98/100	Pass

Table 3. 32osc and 30osc NIST STS results

No	Statistical test	32osc			30osc		
		P-value	Proportion	Result	P-value	Proportion	Result
1	Frequency	0.514124	100/100	Pass	0.514124	100/100	Pass
2	Block frequency	0.030806	97/100	Pass	0.419021	100/100	Pass
3	Cumulative sums	0.816537	100/100	Pass	0.996335	100/100	Pass
4	Runs	0.006196	100/100	Pass	0.350485	99/100	Pass
5	Longest run	0.350485	100/100	Pass	0.964295	100/100	Pass
6	Rank	0.494392	98/100	Pass	0.595549	97/100	Pass
7	FFT	0.494392	100/100	Pass	0.066882	96/100	Pass
8	NonOverlappingTemplates	0.455937	100/100	Pass	0.071177	99/100	Pass
9	OverlappingTemplate	0.401199	99/100	Pass	0.637119	100/100	Pass
10	Universal	0.678686	99/100	Pass	0.637119	100/100	Pass
11	Aproximate entropy	0.236810	99/100	Pass	0.616305	100/100	Pass
12	RandomExcursions	0.08217	60/60	Pass	0.976060	65/66	Pass
13	RandomExcursionsVariants	0.350485	60/60	Pass	0.739918	63/66	Pass
14	Serials	0.935716	99/100	Pass	0.946308	100/100	Pass
15	LinearComplexity	0.534146	100/100	Pass	0.419021	100/100	Pass

Figure 11 presents the FPGA resource consumption of the generator which uses 4 Ring Oscillators (with 1, 3, 5 and 7 inverters). Taking this in account and correlated with the statistical testing results from the Tables above, we can state that this generator is optimum and that 4 oscillators are sufficient for the proposed construction.

Table 4. 28osc and 26osc NIST STS results

No	Statistical test	28osc			26osc		
		P-value	Proportion	Result	P-value	Proportion	Result
1	Frequency	0.494392	98/100	Pass	0.779188	98/100	Pass
2	Block frequency	0.574903	99/100	Pass	0.145326	100/100	Pass
3	Cumulative sums	0.699313	98/100	Pass	0.401199	99/100	Pass
4	Runs	0.202268	97/100	Pass	0.739918	99/100	Pass
5	Longest run	0.759756	97/100	Pass	0.719747	100/100	Pass
6	Rank	0.145326	100/100	Pass	0.595549	97/100	Pass
7	FFT	0.304126	98/100	Pass	0.935716	100/100	Pass
8	NonOverlappingTemplates	0.048716	97/100	Pass	0.678686	99/100	Pass
9	OverlappingTemplate	0.834308	99/100	Pass	0.455937	97/100	Pass
10	Universal	0.574903	99/100	Pass	0.637119	100/100	Pass
11	Aproximate entropy	0.080519	99/100	Pass	0.678686	99/100	Pass
12	RandomExcursions	0.723129	60/61	Pass	0.407091	61/62	Pass
13	RandomExcursionsVariants	0.186566	60/61	Pass	0.534146	61/62	Pass
14	Serials	0.366918	100/100	Pass	0.236810	99/100	Pass
15	LinearComplexity	0.739918	99/100	Pass	0.554420	99/100	Pass

Table 5. 24osc and 22osc NIST STS results

No	Statistical test	24osc			22osc		
		P-value	Proportion	Result	P-value	Proportion	Result
1	Frequency	0.534146	98/100	Pass	0.739918	99/100	Pass
2	Block frequency	0.955835	99/100	Pass	0.455937	98/100	Pass
3	Cumulative sums	0.955835	99/100	Pass	0.455937	98/100	Pass
4	Runs	0.978072	98/100	Pass	0.275709	100/100	Pass
5	Longest run	0.275709	98/100	Pass	0.616305	100/100	Pass
6	Rank	0.616305	100/100	Pass	0.319084	99/100	Pass
7	FFT	0.897763	100/100	Pass	0.867692	98/100	Pass
8	NonOverlappingTemplates	0.319084	100/100	Pass	0.455937	98/100	Pass
9	OverlappingTemplate	0.779188	100/100	Pass	0.851383	100/100	Pass
10	Universal	0.474986	100/100	Pass	0.236810	98/100	Pass
11	Aproximate entropy	0.946308	99/100	Pass	0.657933	100/100	Pass
12	RandomExcursions	0.619772	63/63	Pass	0.551026	63/63	Pass
13	RandomExcursionsVariants	0.551026	62/63	Pass	0.070445	63/63	Pass
14	Serials	0.637119	98/100	Pass	0.366918	98/100	Pass
15	LinearComplexity	0.171867	97/100	Pass	0.383827	98/100	Pass

Table 6. 20osc and 18osc NIST STS results

No	Statistical test	20osc			18osc		
		P-value	Proportion	Result	P-value	Proportion	Result
1	Frequency	0.897763	99/100	Pass	0.637119	99/100	Pass
2	Block frequency	0.816537	97/100	Pass	0.017912	100/100	Pass
3	Cumulative sums	0.678686	100/100	Pass	0.437274	99/100	Pass
4	Runs	0.739918	99/100	Pass	0.030806	97/100	Pass
5	Longest run	0.055361	99100	Pass	0.678686	100/100	Pass
6	Rank	0.437274	99/100	Pass	0.224821	99/100	Pass
7	FFT	0.419021	99/100	Pass	0.334538	99/100	Pass
8	NonOverlappingTemplates	0.419021	99/100	Pass	0.350485	99/100	Pass
9	OverlappingTemplate	0.514124	99/100	Pass	0.437274	98/100	Pass
10	Universal	0.678686	98/100	Pass	0.657933	99/100	Pass
11	Aproximate entropy	0.554420	100/100	Pass	0.739918	99/100	Pass
12	RandomExcursions	0.534146	67/68	Pass	0.162606	66/66	Pass
13	RandomExcursionsVariants	0.637119	67/68	Pass	0.350485	66/66	Pass
14	Serials	0.304126	100100	Pass	0.798139	98/100	Pass
15	LinearComplexity	0.554420	97/100	Pass	0.834308	99/100	Pass

Table 7. 16osc and 14osc NIST STS results

No	Statistical test	16osc			14osc		
		P-value	Proportion	Result	P-value	Proportion	Result
1	Frequency	0.419021	100/100	Pass	0.834308	99/100	Pass
2	Block frequency	0.637119	98/100	Pass	0.129620	100/100	Pass
3	Cumulative sums	0.699313	100/100	Pass	0.153763	99/100	Pass
4	Runs	0.289667	100/100	Pass	0.102526	100/100	Pass
5	Longest run	0.334538	97100	Pass	0.202268	100/100	Pass
6	Rank	0.699313	100/100	Pass	0.978072	100/100	Pass
7	FFT	0.851383	98/100	Pass	0.955835	99/100	Pass
8	NonOverlappingTemplates	0.366918	98/100	Pass	0.514124	99/100	Pass
9	OverlappingTemplate	0.213309	98/100	Pass	0.474986	99/100	Pass
10	Universal	0.851383	99/100	Pass	0.455937	99/100	Pass
11	Aproximate entropy	0.236810	100/100	Pass	0.455937	99/100	Pass
12	RandomExcursions	0.033552	74/75	Pass	0.671779	60/60	Pass
13	RandomExcursionsVariants	0.411329	75/75	Pass	0.213309	60/60	Pass
14	Serials	0.657933	100/100	Pass	0.699313	100/100	Pass
15	LinearComplexity	0.719747	98/100	Pass	0.122325	100/100	Pass

Table 8. 12osc and 10osc NIST STS results

No	Statistical test	12osc			10osc		
		P-value	Proportion	Result	P-value	Proportion	Result
1	Frequency	0.096578	99/100	Pass	0.383827	99/100	Pass
2	Block frequency	0.971699	98/100	Pass	0.275709	99/100	Pass
3	Cumulative sums	0.071177	99/100	Pass	0.262249	99/100	Pass
4	Runs	0.090936	98/100	Pass	0.678686	99/100	Pass
5	Longest run	0.366918	98/100	Pass	0.851383	99/100	Pass
6	Rank	0.595549	100/100	Pass	0.719747	98/100	Pass
7	FFT	0.191687	100/100	Pass	0.851383	97/100	Pass
8	NonOverlappingTemplates	0.115387	100/100	Pass	0.129620	99/100	Pass
9	OverlappingTemplate	0.202268	99/100	Pass	0.000757	97/100	Pass
10	Universal	0.401199	99/100	Pass	0.262249	100/100	Pass
11	Aproximate entropy	0.401199	99/100	Pass	0.678686	99/100	Pass
12	RandomExcursions	0.474986	56/57	Pass	0.819544	64/65	Pass
13	RandomExcursionsVariants	0.554420	57/57	Pass	0.287306	65/65	Pass
14	Serials	0.816537	100/100	Pass	0.366918	99/100	Pass
15	LinearComplexity	0.003712	99/100	Pass	0.401199	100/100	Pass

Table 9. 8osc and 6osc NIST STS results

No	Statistical test	8osc			6osc		
		P-value	Proportion	Result	P-value	Proportion	Result
1	Frequency	0.419021	99/100	Pass	0.401199	100/100	Pass
2	Block frequency	0.181557	99/100	Pass	0.275709	100/100	Pass
3	Cumulative sums	0.883171	99/100	Pass	0.191687	100/100	Pass
4	Runs	0.816537	98/100	Pass	0.798139	99/100	Pass
5	Longest run	0.035174	98/100	Pass	0.071177	98/100	Pass
6	Rank	0.637119	98/100	Pass	0.334538	100/100	Pass
7	FFT	0.637119	99/100	Pass	0.964295	99/100	Pass
8	NonOverlappingTemplates	0.719747	100/100	Pass	0.816537	97/100	Pass
9	OverlappingTemplate	0.883171	100/100	Pass	0.096578	97/100	Pass
10	Universal	0.779188	99/100	Pass	0.051942	99/100	Pass
11	Aproximate entropy	0.595549	98/100	Pass	0.834308	97/100	Pass
12	RandomExcursions	0.759756	59/59	Pass	0.452799	61/61	Pass
13	RandomExcursionsVariants	0.595549	58/59	Pass	0.078086	61/61	Pass
14	Serials	0.013569	99/100	Pass	0.437274	100/100	Pass
15	LinearComplexity	0.759756	100/100	Pass	0.055361	99/100	Pass

Table 10. 4osc NIST STS results

No	Statistical test	4osc		
		P-value	Proportion	Result
1	Frequency	0.834308	99/100	Pass
2	Block frequency	0.616305	99/100	Pass
3	Cumulative sums	0.739918	100/100	Pass
4	Runs	0.037566	100/100	Pass
5	Longest run	0.996335	99/100	Pass
6	Rank	0.897763	100/100	Pass
7	FFT	0.066882	98/100	Pass
8	NonOverlappingTemplates	0.129620	100/100	Pass
9	OverlappingTemplate	0.137282	100/100	Pass
10	Universal	0.236810	99/100	Pass
11	Aproximate entropy	0.798139	100/100	Pass
12	RandomExcursions	0.468595	60/60	Pass
13	RandomExcursionsVariants	0.378138	59/60	Pass
14	Serials	0.851383	99/100	Pass
15	LinearComplexity	0.071177	99/100	Pass

6 Conclusions

This paper described the concept of True Random Number Generators and the steps needed to be made in order to create one. Moreover it presented a personalized TRNG, based on Ring Oscillators, and the optimization techniques used for reducing the number of ROs and therefore the FPGA resources that were allocated for the generator.

The optimizations aimed not only the resource consumption but the speed of the generator as well, obtaining a high speed True Random Number Generator with good statistical properties.

In the final part, this paper presented some conclusive results which demonstrate that the proposed TRNG is suitable for using in sensible applications and/or environments (cryptographic usage).

Acknowledgments. This work was supported by the Romanian National Authority for Scientific Research (CNCSUEFISCDI) under the project PN-II-PT-PCCA-2013-4-1651.

References

1. Sunar, B., Martin, W.J., Stinson, D.R.: A provably secure true random number generator with built-in tolerance to active attacks. IEEE Trans. Comput. **56**(1), 109–119 (2007)
2. Varchola, M., Drutarovsky, M.: New high entropy element for FPGA based true random number generators. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 351–365. Springer, Heidelberg (2010). doi:10.1007/978-3-642-15031-9_24

3. Haddad, P., Fischer, V., Bernard, F., Nicolai, J.: A physical approach for stochastic modeling of TERO-based TRNG. In: Güneysu, T., Handschuh, H. (eds.) CHES 2015. LNCS, vol. 9293, pp. 357–372. Springer, Heidelberg (2015). doi:10.1007/978-3-662-48324-4_18
4. Marghescu, A., Svasta, P., Simion, E.: Randomness extraction techniques for jittery oscillators. In: 38th International Spring Seminar on Electronics Technology (ISSE), pp. 161–166 (2015)
5. Marghescu, A., Teeleanu, G., Maimut, D., Neaca, T., Svasta, P.: Adapting a ring oscillator-based true random number generator for Zynq system on chip embedded platform. In: 20th International Symposium for Design and Technology in Electronic Packaging (SIITME), pp. 197–202 (2014)
6. Simion, E.: The relevance of statistical tests in cryptography. IEEE Secur. Priv. **13**(1), 66–70 (2015)
7. Oprina, A., Popescu, A.S.E., Simion, G., Simion, G.: Walsh-Hadamard randomness test and new methods of test results integration. Bull. Transilv. Univ. Braov **2**, 51 (2009)
8. Drumea, A., Dobre, R.: Clicks counting methods for a scope knob. Hidraulica **4**, 79 (2013)
9. Diffie-Hellmann Key Exchange Protocol. https://tools.ietf.org/html/rfc2631
10. National Institute of Standards and Technology. http://csrc.nist.gov/groups/ST/toolkit/rng/documentation_software.html
11. http://www.xilinx.com/products/boards-and-kits/1-4azfte.html