# Generation of Large Random Sequences on FPGA using TRNG and 2-D chaotic post-processing

Ciro Fabian Bermudez-Marquez, Estaban Tlelo-Cuautle and Cuauhtemoc Mancillas-López

***Abstract*—This work presents two co-design implementations of a true random number generation mechanism. Physical components provided by the Programmable Logic of FPGA are used for true random seeds generation. The seed conditioning and generation of the large sequences were implemented using block cipher AES implemented on the Cortex-A9 processor (embedded in the Zynq FPGA) or specific AESNI instructions in modern processors. Our implementations use less than 10% of the available resources on the target FPGAs and pass all the NIST tests for random generators.**

***Index Terms*—Cryptography, pseudorandomness, True Random Number Generator, TERO**

## I. Introduction

One of the crucial aspects of cryptographic systems' security is the unpredictable and fast generation of random bits. Bit randomness is obtaining a result 0 or 1 with the same probability of occurrence, taken as independent events along the output bit sequence of the system that generates it. This randomness can be generated through two mechanisms: Deterministic Random (DRBG) and Non-Deterministic Random (NRBG), whose outcome should ideally be unpredictable or almost impossible to determine [2]. Some of its specific applications are the generation of keys, initialization vectors in the different modes of operation of block ciphers, or the generation of random scalar in elliptic curves cryptosystems.

The random generation of these bits is performed by a True Random Number Generator (TRNG) [1], whose randomness can be obtained from a non-deterministic source. For this purpose, a physical resource from the environment (known as an entropy source) is used, employing a natural or constructed mechanism. Entropy reflects the uncertainty associated with predicting the outcome of an experiment [11], and in the case of generators of this type, an ideal entropy measure is required.

The generation of truly random numbers is an inefficient mechanism because of the time and resources it takes. Since the requests to these mechanisms are of high demand in time and quantity, generating values of this type implies having enough entropy for each of these requests to generate a truly random value, which is difficult to achieve in a reduced time. For this reason, pseudo-random -also called deterministic- mechanisms have been developed. These mechanisms work based on the generation of a seed obtained by a random mechanism [8] and are known as Pseudo-random Number Generators (PRNG). From that seed, a process is derived in which any of the system's outputs are reproducible; therefore, such seed is secret. Since the pseudo-random mechanism is derived from a random one, both have a source of entropy. However, this deterministic process is not bound to have an ideal entropy measure.

The quality of the random sequences obtained utilizing a type of generator acquires a high degree of importance depending on their purpose. In the case of cryptography, these numbers or bit strings are required to be sufficiently random (validated through the various NIST tests [2]) to maintain the integrity of the cryptographic primitives. Cryptographic RNGs focus on producing values that can withstand adversarial attacks against the cryptographic system since their security is based on a randomly generated secret key. So, it must maximize its entropy value and have a uniform distribution to be considered ideal. In addition, to assess these generators and measure the quality of the output bitstreams, it is necessary to consider the following characteristics:

1) **Uniformity**: Let the occurrence of both "0"s and "1"s in the output bit sequence be as similar as possible, i.e., since both have a probability of $\frac{1}{2}$ and an output of $n$-bits, the number of zeros and ones should be as close to $\frac{n}{2}$.
2) **Independence**: That despite knowing k bits of the output string, the probability of obtaining the next result bit is -ideally- of $\frac{1}{2}$ when trying to predict it by using a computationally efficient algorithm in time (polynomial time complexity or better). This property makes it possible to determine no correlation between the previous and future results of a random experiment, despite knowledge of some of its outcomes
3) **Non-periodicity**: That even though an attacker knows a part of the output bit string $R_t, R_{t+1}, R_{t+2}, \ldots$, it is not possible for him to determine other bits part of the same string, either with a $i < t$ or$i > t$.
4) **Scalability**: Any sub-sequence extracted from the total output sequence is also random.
5) **Consistency**: Despite changing the generator seed, the output sequences remain random in behavior and do not differ in quality.

Some mechanisms are implemented in high-end Intel processors [1]; in this work, we explore the implementation of a generator of true large random sequences using an SoC FPGA and a combination of FPGA and a personal computer. Our system's main advantage is that it combines a widely known TRNG mechanism and a PRNG based on a secure block cipher. Also, all the system can work stand alone on a Zynq FPGA, or in the case of high-speed applications, Zynq FPGA is used only as a seed generator, and a modern PC generates the large sequences.

The paper is organized as follows: In Section II we introduce the basic blocks; the implementation aspects are detailed in Section III; in Section IV we present the results and discussions about them; finally, in Section V, the conclusions are given.

## II. Basic Blocks

A standard ring oscillator (RO) is a resource to obtain hardware randomness (entropy source). It obtains its properties from the noise and meta-stability that depend on the implementation platform and allows oscillating outputs through cyclic feedback for an indeterminate time[10, p. 42].

On the other hand, the Transition Effect Ring Oscillator (TERO) is a type of ring oscillator that produces meta-stability, returning oscillations between low and high states for an indeterminate period. It operates through delays, obtained by cascading an odd number of buffers with two NAND gates, similar to an RS-latch circuit. TERO must be set to an illegal state (setting and resetting for the RS-latch) to generate such meta-stability [10]. As is shown in Figure 1, the NAND gates obtain the clock signal from a standard ring oscillator. Its purpose is to sample the device clock signal to obtain the jitter accumulation, which works as the random source coming directly from the implementation platform.

The Advanced Encryption Standard (AES) is a block cipher standardized by the NIST. It iterates a round function composed of four transformations: *Bytes Substitution*, *Shift Rows*, and *Mix Columns* functions. The last round does not execute the Mix Columns transformation. Initially, it adds the key and the plaintext, then executes 10, 12, or 14 rounds, depending on the key size. Many embedded platforms include an AES co-processor, and modern Intel processors have special instruction set for AES called AES-NI [6]; so, the use of AES guarantees efficiency on different platforms. According to the NIST standard in SP 800-90A [9, p. 88], using Counter Mode (CTR) with the AES block cipher ensures that in case of a possible attack whose objective is to distinguish bits to predict future outputs, the probability of success is practically zero because duplicate outputs cannot be produced, being able to make up to $2^{48}$ calls for the generation of random sequences of $2^{19}$ length at most.

## III. Implementation

We implemented two generator versions on the PYNQ-Z2 development board and Vivado tool version 2020.2. The difference between versions is in the post-processing phase; one uses the ARM processor embedded in the FPGA, while the other uses a personal computer equipped with an Intel processor using AES-NI instructions. The implementation of the first version followed the following steps:

1) Obtaining the true randomness using the selected entropy source TERO (Transition Effect Ring Oscillator) implemented in VHDL.
2) Capture the entropy values as raw random values obtained from the 128 or 32 implemented copies of the TERO using the AXI communication protocol and the registers in the embedded ARM processor.
3) Post-processing of the entropy bits in C language using the conditioning algorithms defined by NIST in SP 800-90C.
4) Pseudo-random bit generation using the process described for AES block cipher in counter mode, with inputs and outputs 128 bits long as defined in the SP 800-90A standard.
5) The generated sequence is sent to a personal computer through the serial port using the embedded processor.
6) The quality of the generated sequence of random bits is checked by executing the NIST test suit over them.

For the second version, steps 1 and 2 are the same. In this case, the embedded processor sends the values from the TERO copies to the personal computer through the serial port (similar to step 5). Steps 3 and 4 are executed on the personal computer using the AES-NI instructions.

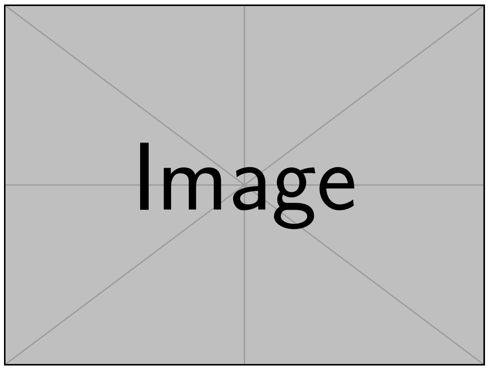

Fig. 1: Complete structure of the *Transition Effect Ring Oscillator* obtained from [10, p. 19].

For the generation of large sequences (PRNG), we considred the incorporation of the AES block cipher in CTR mode of operation with 128 bits input and output sizes, endorsed in the NIST SP 800-90A standard [9, p. 88]. It ensures that the implementation of a cryptographic PRNG is reliable in terms of performance and minimizing its vulnerabilities to potential attackers.

### *A. Implementation of the TERO on the FPGA*

Implementing a ring oscillator requires using resources capable of generating delays to the signals; in the case of FPGAs, such resources are the LUTs (Lookup Tables). Each logic gate in Figure 1 should be implemented in one LUT, and all utilized LUTs should be located near them. To achieve that, we used placement constraints. We have used the primitives LUT1 and LUT2 included in the unisim library. It is important to enable the option `keep_equivalent_registers` and disable `resource_sharing` and `Opt_design` in the Vivado Tool. This allow us to get the correct implementation and functionality of the TERO design.

### *B. Software implementation of AES*

For the post-processing in the embedded processor, we implemented AES block cipher using T-boxes [3], which is an implementation strategy optimized for 32-bit processors. T-boxes precomputes the Mix Columns transformation and combines it with Bytes Substitution. The extra cost of this strategy is in the amount of memory, as it needs large tables of 256 values of 32 bits compared to traditional S-Box has 256 values of 8 bits. When the post-processing is performed on a personal computer, we implemented AES using the AES-NI instructions.

### C. Communication Interface

One hundred twenty-eight in- stances of the TERO are communicated with the embedded processor through AXI4 protocol using four 32-bit registers. Each register has a memory address accessible by the embed- ded processor. The communication with the personal computer is done through the serial port.

### D. Design Integration

The first implemented version (see figure 2a), the complete design was integrated into the Zynq FPGA by separating it into two parts:

1) *Programmable Logic* (PL). Multiple instantiations of TERO are used as an entropy source to generate random seeds.
2) *Processing System* (PS). The conditioning of the seeds and the PRNG process are executed in software by the embedded processor. The large sequence is sent to a personal computer using the serial port.

In the case of the second implementation (see figure 2b), the same design was used for the PL part. While PS only sends the outputs of the TERO without any post-processing to the personal computer using the serial port. The post-processing of the seed and the large sequence generation (PRNG) are executed on the personal computer. So, the FPGA is used in this case only as a random seed generator.

The utilization statistics for the proposed seed generators, two versions, one using 32 TERO instantiations and the other with 128, are shown in Table I. The results show the utilized resources of the device to implement the described design and the comparison with two similar designs based on the combination of Galois and Fibonacci ring oscillators.

| TRNG type | LUTs | FF | Device | PostProcessing |
|---|---|---|---|---|
| 32-TEROs | 1416 | 416 | Zynq | Block Cipher |
| 128-TEROs | 5451 | 1664 | Zynq | Block Cipher |
| FiGaRO [4] | 1557 | 2137 | V UltraScale+ | Hash Funtion |
| FiGaRO [7] | 1140 | 698 | Stratix 4 | - |

TABLE I: Implementation statistics of the proposed seeds generator design for the ZYNQ FPGA and comparison with two existing proposals based on Galois and Fibonacci RO

## IV. Results and discussion

The quality of the randomness generated by a specific system is validated using the tests suite established by the National Institute of Standards and Technology (NIST) [2]. Such a suite includes fifteen tests to determine if a mechanism designed to provide randomness for a cryptographic purpose is secure enough. If the generated sequences are good quality, they can be used in cryptographic protocols.

NIST STS recommends introducing a minimum of fifty-five sequences of 100,000 bits in length, mainly because more information assessed ensures more accurate results. We compare our two implemented mechanisms and the Intel random generator. The results corresponding to each test applied to one hundred sequences of 100,000 bits each. The table II is a concise compendium that brings together general results of the NIST tests applied to the implemented mechanisms.

Each test checks one of the properties mentioned above of random sequences (from a sequence of bits taken as a sample of the generators). We can obtain probabilistic values that give a comparison parameter to know if it is possible to predict or distinguish values in the analyzed sequence. However, obtaining perfect values as a result of these tests does not assure that the values returned by the generator are random, so they are inconclusive [5]; on the other hand, the failure in any of them shows us evidence of critical defects within the randomness generators.

We have computed the confidence interval as is defined in the standard in [2, p. 4-2]. For hundred sequences and significance value $\alpha = 0.01$ the interval is $[0.96016, 1.01984]$. According to the results returned in each test output file, the minimum pass ratio for each statistical test (except the Random Excursions Test and its variant) is approximately equal to 96 for a sample size of 100 binary sequences of length $100,000$ and 8 for ten sequences of $1,000,000$ bits. The minimum pass ratio of the Random Excursions Test and its variant is approximately $n-1$ for a sample size equal to $n$ binary sequences. The number of rounds for each test is arbitrarily determined for each run.

Some of the tests have more than one round such as the Cumulative Sums Test (2 rounds), the Non-Overlapping Template Matching Test (148 rounds), the Random Excursions Test (8 rounds), the Random Excursions Variant Test (18 rounds), and the Serial Test (2 rounds). Only the lowest obtained result in terms of proportion for each case and each mechanism was included.

Our design implements cryptographic post-processing based on the block cipher AES; in contrast, the design in [4] performs the post-processing using the hash function SHA256. The design in [7] has no post-processing.We have taken only the resources consumed for the TRNG, so the post-processing phase is not considered to have a fair comparison as we perform it using the embedded processor. From Table I we can observe that our proposal to use 32-TEROs as a seed generator for the embedded processor is a cheap option and uses fewer FFs, and the number of LUTs is comparable to FiGaROs-based TRNGs. The version with 128-TEROs can be used to provide more speed as it provides 128 bits in parallel.

## V. Conclusions

We presented a co-design capable of generating large random sequences in this work. Our proposal combines a TRNG to produce random seeds implemented in the logic part of the Zynq FPGA and a PRNG and the conditioning of the seeds implemented in software. As TRNG, we used the widely known TERO; we used 128 instantiations to improve the variability of the generated seeds. As PRNG, we used AES in counter mode. Also, the conditioning of the seeds was performed with AES. Two versions of our proposal were

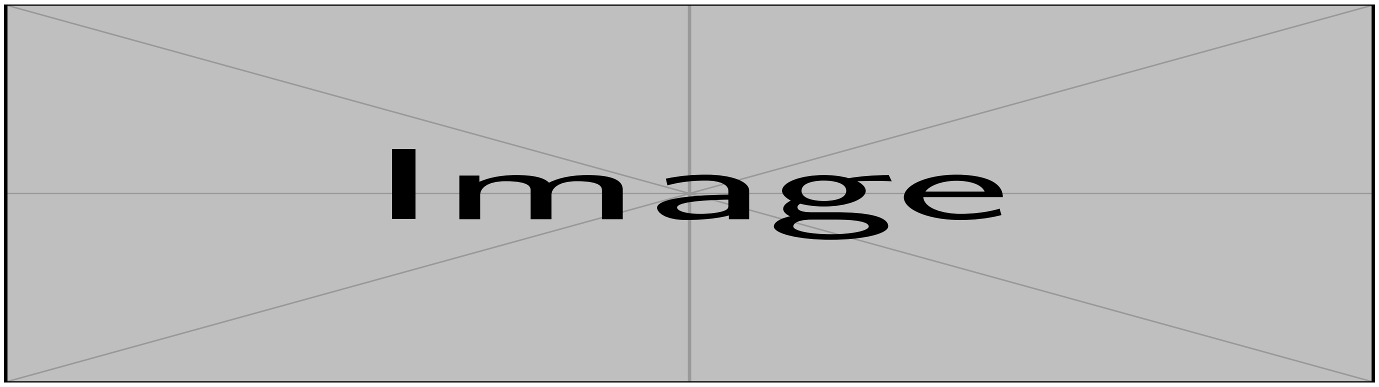

Fig. 2: Figure (a) shows the stages of PL and PS for the pseudorandom number generator with entropy source, post-processing and Deterministic Random Bit Generation. Figure (b) shows the generator stages in hardware (FPGA) and post-processing with Pseudo-random Generation on the personal computer using the Intel Processor.

| **Test** | **RNG Mechanism** | | | | | | **Test** | **RNG Mechanism** | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | (1) | | (2) | | (2) | | | (1) | | (2) | | (2) | |
| | % | p-value | % | p-value | % | p-value | | % | p-value | % | p-value | % | p-value |
| Frequency | 1 | 0.383827 | 1 | 0.137282 | 0.98 | 0.834308 | OverTemp | 0.95 | 0.99425 | 0.99 | 0.759756 | 1 | 0.883171 |
| BlockFreq | 0.97 | 0.883171 | 0.99 | 0.883171 | 1 | 0.616305 | Universal | 1 | 0.350485 | 1 | 0.739918 | 1 | 0.534146 |
| Cumsums | 0.99 | 0.719747 | 1 | 0.816537 | 0.98 | 0.249284 | Entropy | 0.99 | 0.595549 | 0.99 | 0.851383 | 1 | 0.12962 |
| Runs | 1 | 0.759756 | 0.99 | 0.514124 | 0.98 | 0.883171 | RandExc | 0.92 | 0.437274 | 0.92 | 0.437274 | 1 | - |
| LongestRun | 1 | 0.834308 | 0.98 | 0.759756 | 0.99 | 0.045675 | RandExcVar | 0.92 | 0.162606 | 0.92 | 0.275709 | 0.88 | - |
| Rank | 0.99 | 0.987896 | 0.99 | 0.55442 | 0.99 | 0.657933 | Serial | 0.99 | 0.678686 | 0.99 | 0.911413 | 0.98 | 0.798139 |
| FFT | 0.99 | 0.383827 | 1 | 0.595549 | 0.99 | 0.275709 | LinearComp | 0.99 | 0.437274 | 0.98 | 0.115387 | 0.97 | 0.249284 |
| NonOverTemp | 0.95 | 0.494392 | 0.95 | 0.401199 | 0.97 | 0.01265 | | | | | | | |

TABLE II: Results of the ratios corresponding to the number of NIST tests passed, applied to 100 pseudorandom sequences of 100,000 bits each from the output of the corresponding PRNG mechanism: (1) ZYNQ full hardware (PL and PS) RNG, (2) ZYNQ (PL) and Intel processor for the PRNG, and (3) Intel® Pseudorandom Number Generator (Intrinsics Library).

presented; both use the same TRNG and differ only in the way they execute the PRNG and the conditioning of the seed. One implements AES in the Zynq embedded processor, and the other sends the seeds directly to a personal computer equipped with AES-NI instructions. Finally, we compared the quality of the sequences generated for our mechanism (both versions) with the Intel RNG available in moderns processors. The comparison was performed in terms of the output of the NIST tests for random sequences. The obtained results show that our implementation can be used for cryptographic applications. The version implemented totally on the FPGAs can be used when we need and stand-alone generator, while the version that only produces random seeds can be used when a personal computer is available.

## References

[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] K. Elissa, "Title of paper if known," unpublished.

[5] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[8] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[9] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[10] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[11] K. Elissa, "Title of paper if known," unpublished.