


# Design and Test of an Integrated Random Number Generator with All-Digital Entropy Source

Luca Crocetti <sup>†</sup>, Stefano Di Matteo <sup>†</sup>, Pietro Nannipieri <sup>\*,†</sup>, Luca Fanucci <sup>†</sup> and Sergio Saponara <sup>†</sup>

Department of Information Engineering, University of Pisa, Via G. Caruso, 16, 56122 Pisa, Italy; luca.crocetti@phd.unipi.it (L.C.); stefano.dimatteo@phd.unipi.it (S.D.M.); luca.fanucci@unipi.it (L.F.); sergio.saponara@unipi.it (S.S.)

\* Correspondence: [pietro.nannipieri@ing.unipi.it](mailto:pietro.nannipieri@ing.unipi.it); Tel.: +39-050-27660

† These authors contributed equally to this work.

**Abstract:** In the cybersecurity field, the generation of random numbers is extremely important because they are employed in different applications such as the generation/derivation of cryptographic keys, nonces, and initialization vectors. The more unpredictable the random sequence, the higher its quality and the lower the probability of recovering the value of those random numbers for an adversary. Cryptographically Secure Pseudo-Random Number Generators (CSPRNGs) are random number generators (RNGs) with specific properties and whose output sequence has such a degree of randomness that it cannot be distinguished from an ideal random sequence. In this work, we designed an all-digital RNG, which includes a Deterministic Random Bit Generator (DRBG) that meets the security requirements for cryptographic applications as CSPRNG, plus an entropy source that showed high portability and a high level of entropy. The proposed design has been intensively tested against both NIST and BSI suites to assess its entropy and randomness, and it is ready to be integrated into the European Processor Initiative (EPI) chip.

**Keywords:** random number generator; FPGA; entropy; ASIC; EPI



**Citation:** Crocetti, L.; Di Matteo, S.; Nannipieri, P.; Fanucci, L.;

Saponarara, S. Design and Test of an Integrated Random Number Generator with All-Digital Entropy Source. *Entropy* **2022**, *24*, 139.

<https://doi.org/10.3390/e24020139>

Academic Editor: Gregg Jaeger

Received: 29 November 2021

Accepted: 12 January 2022

Published: 18 January 2022

**Publisher's Note:** MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Copyright:** © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

The generation of random bits (or numbers) represents one of the fundamental and most significant aspects concerning cybersecurity, because they are employed to generate and/or derive cryptographic keys, one-time passwords, initialization vectors (IVs) for some cryptographic algorithms, and more in general non-repeating values (as the nonces and others). From here, the requirement of having good-quality random numbers becomes a mandatory feature for those security modules aimed to provide and strengthen the security level of a system or an application. With good-quality random numbers, they indicated those sequences of bits generated unpredictably, for which the higher the unpredictability, the higher the quality (or the security strength). Indeed, if the method used to produce such streams of bits was predictable, also only in part, this would expose an entire system to severe security threats. For example, it can be assumed that a cybersecurity device offers confidentiality protection of data by using a symmetric-key encryption/decryption scheme. The key employed within the encryption and decryption functions must be protected as well, because if an attacker was able to discover or to guess such a key, then it would have access to the content of the communication, compromising the privacy between the entities. Random numbers could be used to generate or refresh such keys, or also as the initial value of a packet number that is integrated into the communication to protect against replay attacks. Anyway, the higher the probability to retrieve the value of those random numbers, the higher the probability for an adversary to obtain such data.

In addition to next-generation technologies, such as quantum-based random number generation [1], there are essentially two methods to generate random bits exploitable in digital circuits. One strategy consists of producing bits non-deterministically, for which the

value of every output bit is based on a physical process that is intrinsically unpredictable: this class of Random Bit Generators (RBGs) is commonly known as Non-deterministic Random Bit Generator (NRBG). The other approach consists of computing random bits using a deterministic algorithm, and accordingly, this class of RBGs is known as Deterministic Random Bit Generator (DRBG). Such a class of RBGs produces sequences of bits from an initial value that is denoted as seed, and it is required to be the output of a source of randomness. As a result of the deterministic nature of the process, a DRBG is said to produce pseudo-random bits rather than random bits. Moreover, the seed used to initialize the DRBG can be compared to a cipher key that must be kept secret; otherwise, its knowledge can allow an attacker to disclose the future values of pseudo-random sequence, as it is generated in a deterministic way. If on one hand, the terms DRBG and NRBG refer to the construction techniques and properties of RBGs, on the other one, their implementations are usually denoted as Pseudo-Random Number Generators (PRNGs) and True Random Number Generators (TRNGs), respectively, because they are typically used to generate random values with a width greater than 1 bit (notably a number). In this field, the adoption of modules named Cryptographically Secure Pseudo-Random Number Generators (CSPRNGs) also acquired relevance, which are PRNG with specific properties regulated by standards, and whose output sequence has a degree of randomness such that it cannot be distinguished from an ideal random sequence within certain limits; for this reason, they can be employed in cryptographic applications. A CSPRNG can be implemented both integrating a source of randomness or a TRNG by itself, thus beginning an only output module that does not require any input (i.e., the seed), and without integrating such source of randomness: in such case, the CSPRNG module is provided also with an input interface for seeds that must respect specific and stringent rules specified by the corresponding standards.

This article focuses on the design and test of an all-digital hardware accelerator for random number generation, NIST-compliant at entropy level, with sustained throughput and technology independence. The complete hardware accelerator is to be exploited within the 7 nm European Processor Initiative [2] ASIC, together with other hardware accelerators that will rely on it (AES [3], ECC [4], SHA [5]). The paper is organized as follows. In Section 2, the architecture of the proposed RNG is explained. In particular, this section covers also the aspects related to the design of both the entropy source and the DRBG modules as well as the synthesis process on the target technology; Section 3 focuses on the achieved results in terms of complexity, entropy, and randomness of the proposed design; Section 4 compares the obtained results with the State of the Art, and finally, Section 5 points out the conclusions for this work.

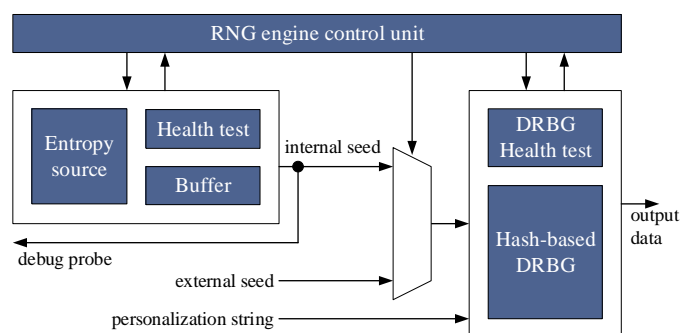
## 2. Design of the Random Number Generator

### 2.1. RNG Engine

The RNG engine implements a CSPRNG to provide random sequences of bits (or numbers) with an entropy level that can be considered sufficient for cryptographic applications requiring a high level of security (or security strength).

The internal architecture of the RNG engine is illustrated in Figure 1, and it mainly consists of the integration of an entropy source module and a hash-based DRBG module, which are respectively described in [6–8]. The former is responsible for the generation of (internal) seeds that are used to initialize the internal state of the DRBG and that are built byte by byte, exploiting the Buffer unit to collect them, while the latter produces sequences of random words that constitute the output data of the whole RNG engine. Both modules are equipped with their own dedicated health test mechanism that continuously evaluates the randomness of the generated bitstreams, and both are managed by a control unit that regulates their usage and interaction, which is also based on the response of the health test. Moreover, the DRBG module accepts as input also personalization string to further increase the degree of randomness of output sequences, according to the specifications of [9,10], and it also seeds from the external: this choice was guided by the fact that the

entropy level of seeds generated by the entropy source module is strongly dependent on the technology used for its implementation, and it was not possible to evaluate its characteristics in advance, using some statistical models. Thus, it was foreseen to permit the usage of seeds from external to improve the robustness of the RNG engine and prevent the impossibility of using at least the DRBG part of the CSPRNG, in case the integrated entropy source module is not able to generate seeds featuring the required amount of entropy. This solution allows also to continue using the DRBG mechanism of the RNG engine in case the entropy module shows some flaws during its operation, and that is signaled by the corresponding Health test unit. Finally, an additional and dedicated output line has been included in the RNG engine to access and acquire the raw data from the entropy source module and conduct tests to assess what the entropy level can provide: such an output line corresponds to the signal debug probe in Figure 1. With respect to the previous works published in [6–8], this paper presents the whole design of the RNG engine and introduces some architectural features for improving its robustness such as the input ports for external seed and personalization string, the output port for the entropy source assessment, and the health test modules for both the entropy source and the DRBG module. In addition, with respect to the work in [6], this work shows the complete evaluation procedure for the whole RNG engine and demonstrates the portability of the entropy source in different FPGA technologies.



**Figure 1.** Internal architecture of RNG engine.

## 2.2. Entropy and Design of the All-Digital Entropy Source

Concerning the entropy source module, its development was guided by the purpose of implementing a digital design featuring the highest level of entropy, and that was portable on a variety of technologies, exploiting the properties of HDL digital designs. As shown in [6], such design activity starts with the analysis (and the comparison) of the available solutions, which all rely on the usage of ROs for the generation of entropy. A RO essentially consists of a chain of combinational logic gates and cells that closes on itself, forming a circle, or a loop, in which the output of the last element corresponds also to the input of the first element of the chain. The output of the last element of the chain is used also as the output of the RO, and it oscillates between two voltage levels corresponding to the logic states *low* (or 0) and *high* (or 1). The characteristics of the oscillation depend on the topology of the combinational loop and the logic cells used to build it. Based on this, different approaches can be used to exploit some intrinsic physical phenomena such as temperature, voltage, or noise fluctuations to randomize the oscillation and thus generate a random sequence of 0 s and 1 s by sampling the output of the RO. The most diffused and main solutions are Transition Effect Ring Oscillator (TERO), originally proposed in [11], Metastable Ring Oscillator (Meta-RO), proposed for the first time in [12], Fibonacci Ring Oscillator (FiRO), and Galois Ring Oscillator (GaRO), illustrated in [13] and Fibonacci-Galois Ring Oscillator (FiGaRO), which is a combination of the last two previous candidates, and it is described, for example, in [14].

FiRO derives from its namesake LFSR architecture, replacing all D flip-flops with inverters. Randomness is introduced by the dependence of the inverter delay on temperature and supply voltage: as these parameters vary, due to both noise and environmental changes,

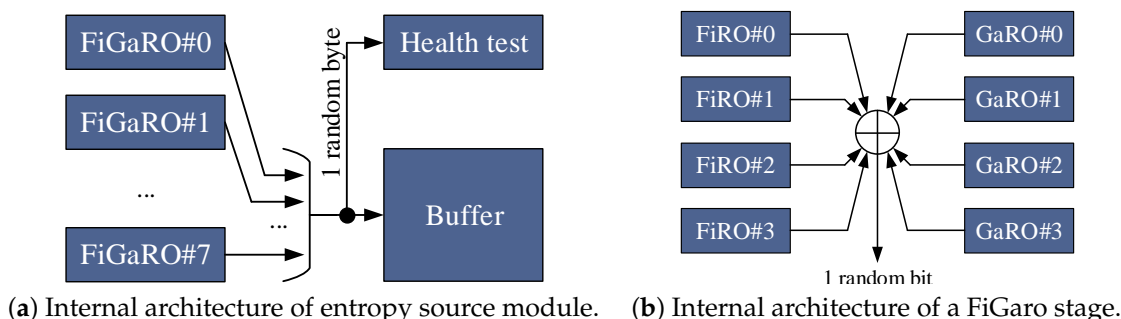
the ring evolves with different output patterns, resulting in a chaotic and unpredictable signal. Further randomness can be acquired during the sampling phase with the violation of setup and hold times. Similarly, GaRO is derived from the Galois LFSR structure, again replacing all D flip-flops with inverters. The same considerations made with FiRO on the stochastic evolution of this circuit apply. The structure formed by XORing FiRO and GaRO is called FiGaRO structure. For more details, please refer to [6], where architectures of the mentioned oscillators are explained in detail. The selection of the FiGaRO oscillator has been made after the experimental campaign performed in [6], where it was demonstrated to be the best among all the other presented oscillators in terms of entropy for an FPGA device.

Table 1 shows the main outcome of the comparative analysis in [6], for which the FiGaRO approach was selected, because of the advantages offered by its features, including its independence to the placement of design elements and its enhanced robustness to the implementation of single FiRO or GaRO.

**Table 1.** Comparison of design strategies for digital entropy source modules. The approach based on TERO is illustrated in [11,15–17], the one based on Meta-RO is described in [12], while some examples of FiROs and GaROs can be found in [13], and the one employing FiGaRO is presented in [13].

Design Strategy	Physical Phenomena Generating Entropy	Main Characteristics
TERO	Latches oscillatory metastability	Low throughputs, large dependence on placement of logic cells
Meta-RO	Analogue metastability of inverter gates	PLL required, dependence on placement of logic cells
FiRO	Jitter and metastability	Good independence from placing
GaRO	Jitter and metastability	Good independence from placing
FiGaRO	Jitter and metastability	Independence from placing, higher entropy and robustness respect to single FiRO and GaRO

According to these considerations, our proposed entropy source module has been implemented by instantiating eight parallel FiGaRO stages, to generate a random byte made of eight independent bits, each one from a different FiGaRO stage. The architecture of this module is illustrated in Figure 2a, while Figure 2b shows the internal structure of a FiGaRO stage, which counts four FiROs and four GaROs units mixed through the XOR gate, as this solution showed in [6] to provide an entropy per bit of 0.995, independently from the placement and the value of the sampling frequency.



**Figure 2.** Internal architecture of entropy source module of RNG engine (a), and of FiGaRO stages composing it (b).

### 2.3. Design of the Deterministic Random Bit Generator Module

Figure 3 shows the internal architecture of the DRBG module, which relies on a SHA2-256 core and integrates also a buffer containing the current state and a reseed counter. The output of the SHA2-256 [5] core is used to generate the output stream of the DRBG;

therefore, it is constituted by random words of 256 bits, corresponding to the digest of the hashing unit. The reseed counter is in charge to signal when a new seed is required in input to the module, to refresh the entropy level of the output random stream, and to avoid it decreasing too much.

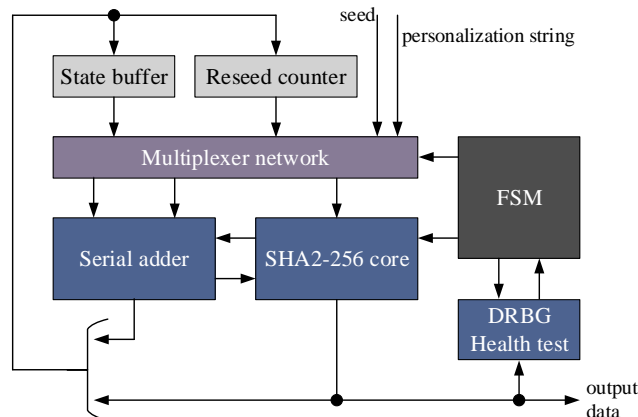


Figure 3. Internal architecture of DRBG module of RNG engine.

The choice of a hash-based mechanism for the implementation of the DRBG module can be found in the work presented in the publications [7,8], in which they have investigated the structures and the characteristics in terms of security and performance of the implementation approach specified by the NIST standard for the construction of cryptographically secure deterministic RBGs, i.e., [9]. We propose and specify the following three main architectures:

- CTR-DRBG, which relies on the CTR mode of operation of block ciphers;
- Hash-DRBG (ref. [9] specifies also the usage SHA1 function for the Hash-DRBG class; anyway, it has not been included in the analysis presented in [8] because it offers a lower security strength to the SHA2 counterpart, and it is considered outdated), which relies on the SHA2 functions;
- HMAC-DRBG, which relies on the HMAC scheme of hash algorithms.

All these approaches can provide a security strength up to 256 bits, under specific design rules, and show different characteristics at an architectural level, which reflects also on performance. Although the CTR-DRBG solution permits implementing more efficient PRNGs both for the relative metric of throughput per area and for the absolute metrics of throughput (the highest one) and consumption of logic resources (the lowest one), it has been discarded, as [18] expresses about the effective capability of this mechanism to reach maximum security strength: its authors claim that the usage of DRBGs based on block ciphers should be avoided since the pseudo-random permutation inside each AES [3] round outputs a sequence that is distinguishable from a random source, while the DRBGs based on hash functions satisfy the security requirements. On the other hand, the DRBGs based on HMAC can be interpreted as a more complex version of the ones relying on hash functions, because they exploit the same underlying hash algorithm to which the HMAC scheme adds resources, data (the key), and operations (hence latency), therefore leading to a less efficient solution and introducing the issues related to the establishment of cryptographic keys. After these considerations, the most convenient choice lies in the Hash-DRBG approach. In this regard, it is useful to recall that SHA2-224 and SHA2-256 are in essence the same algorithm that only provides outputs of different sizes, hence requiring the same amount of resources and showing the same critical path (i.e., the same maximum frequency) but outputting a different amount of data. For this reason, the function with the highest data size (SHA2-256) must be preferred, because it offers a higher throughput at the same area cost. Similarly, such evaluation applies also to the SHA2-384 and SHA2-512 functions, for which the SHA2-512 algorithm is shown to be more efficient than the

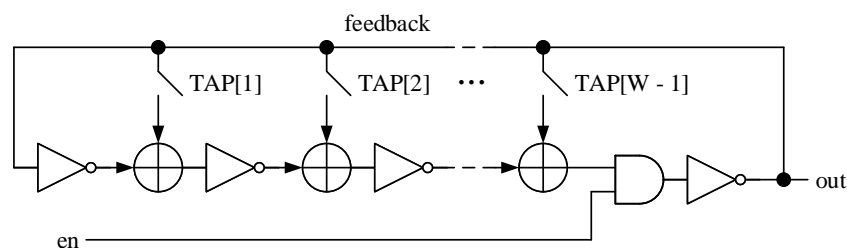
SHA2-384 one. Proceeding with a comparison of solutions based on SHA2-256 and SHA2-512 routines, no matter the implementation strategy, the function for the generation of 512-bit digests requires almost double the resources requested by the 256-bit counterpart because it operates on vectors of double size: input blocks of 1024 bits, rather than 512 bits, and internal data path of 64 bits, rather than 32 bits. On the timing performance side, the two solutions can support approximately the same clock frequency ( $f_{clk}$ ) but with different latencies: 64 clock cycles for the SHA2-256 case and 80 clock cycles for the SHA2-512 case. Including also the length of the output digests, their throughput (At confirmation of the considerations on the HMAC-DRBGs approach, the corresponding throughput of HMAC version of the same 256-bit and 512-bit PRNGs can be expressed, respectively, such as  $1 \cdot f_{clk}$  bit/s and  $1.6 \cdot f_{clk}$  bit/s) can be expressed, respectively, like  $4 \cdot f_{clk}$  bit/s and  $6.4 \cdot f_{clk}$  bit/s. If including also the area cost, a heuristic efficiency factor can be derived by computing from the ratio between the throughput and the resources consumption as:

- $4 \cdot f_{clk} \frac{\text{bit/s}}{\text{area}}$ , for the SHA2-256 case;
- $3.2 \cdot f_{clk} \frac{\text{bit/s}}{\text{area}}$ , for the SHA2-512 case.

Therefore, in conclusion, the implementation of a Hash-DRBG module based on the SHA2-256 function should be preferred to ensure a compact and efficient solution, and that was the choice for the DRBG block in Figure 1. According to NIST specifications [9], the length of seed required by this deterministic RBG is 440 bits and the minimum level of entropy is 256 bits (while the maximum is of  $2^{35}$  bits). The input seed has been extended to 512 bits both for simplicity of usage, because the SHA2 function takes as input data blocks of 512 bits, and for relaxing the constraints on the entropy degree (This minimum level of entropy per bit is calculated as the ratio between the minimum required entropy, i.e., 256 bits, and the length in bits of input seed after extension, i.e., 512 bits; if using the length of the bit of input seed specified by the NIST in [9], i.e., 440 bits, the corresponding minimum level of entropy per bit is  $256/440 = 0.582$  bits, which constitutes a more stringent requirement for the seed properties to the previous case) of input seed that must be at least of  $256/512 = 0.5$  bits of entropy per bit.

#### 2.4. Synthesis Design

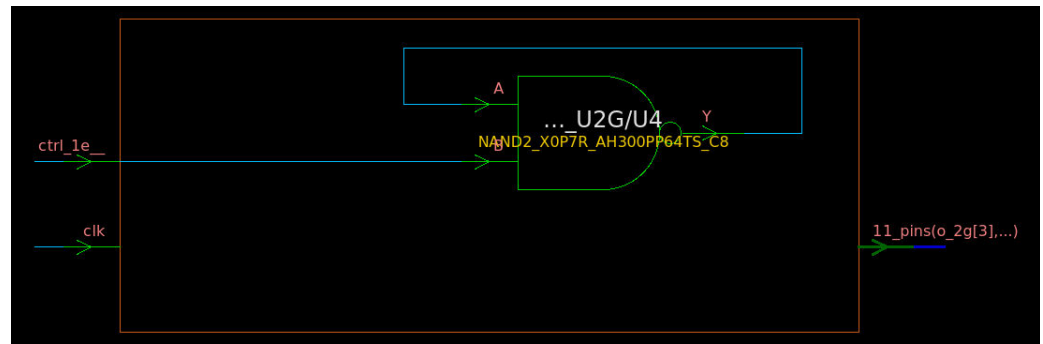
The synthesis process of the RNG engine was performed using the Electronic Design Automation (EDA) tools Design Compiler by Synopsys [19] and Vivado by Xilinx [20], targeting, respectively, standard-cell and FPGA technologies. Concerning this step, some issues are required to be addressed for guiding the logic synthesis to generate a netlist corresponding to the expected outcome. The main issue faced during the synthesis concerned the ring oscillators (FiROs and GaROs) that are essentially a looped chain of *NOT* and *XOR* gates, as shown by Figure 4 for a GaRO of  $W$  inverting elements. The inverting elements are the *NOT* gate and the global enable signal of the circuit is *en*, while the TAP[i] elements are the coefficient of the polynomial associated to the feedback net: if TAP[i] = 1, the corresponding connection is a short-circuit; hence, the corresponding XOR gate is instantiated within the circuit; otherwise, (TAP[i] = 0) the corresponding connection is an open circuit, and the associate *XOR* gate is not instantiated.



**Figure 4.** Schematic of a GaRO with enable (*en* signal) and  $W$  inverting elements.

When generating the corresponding netlist with the library cells of the target technology, the synthesis engine performs several optimizations that can include also the

modification of some parts of the circuit described in HDL but without modifying its logical functionality. This can happen also to the ring oscillator in Figure 4, and indeed, this phenomenon was observed during the synthesis of the RNG engine. Figure 5 shows the typical outcome of the logic synthesis when trying to synthesize a circuit similar to the one reported in Figure 4.



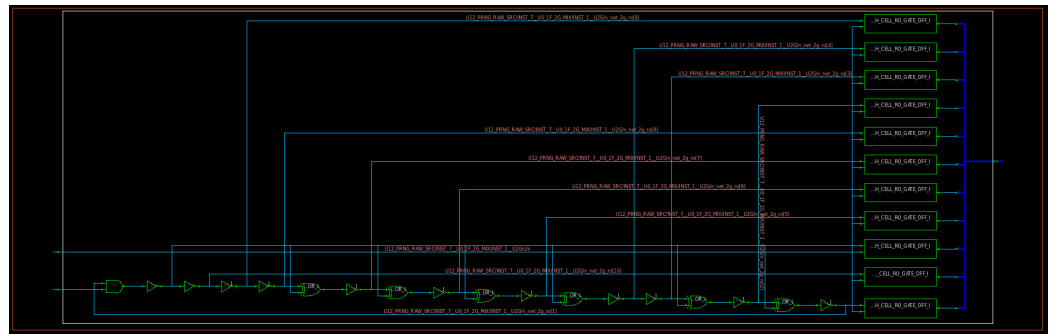
**Figure 5.** Typical outcome of the synthesis of a GaRO. This schematic has been extracted by using the Synopsys Design Compiler tool.

As illustrated in Figure 5, the chain of *NOT* and *XOR* gates forming the ring oscillator loop can be replaced by a single *NAND* gate by the synthesis process, because they are logically equivalent. Indeed, it can be easily proved by building the truth table associated with the *NAND* gate in Figure 5, by referring to its input ports, i.e., *A* and *B*, and its output port, i.e., *Y*.

Combining the information in Table 2 with the schematic in Figure 5, it derives that when input *B* is 0, the signal on port *Y* is forced to the high logic level (1), whatever the initial value of port *A*, and it remains on that logic level, because due to the feedback wire, the value of *Y* is applied also to port *A*, and when  $B = 0$  and  $A = 1$ ,  $Y = 1$  (Table 2). In other words, when the module in Figure 5 is disabled (i.e., *B* which is connected and corresponds to the module enable signal, *ctrl\_1e\_\_*, is zero), the loop does not oscillate and the output of the module (*Y*) is stable to the logic value 1. On the other hand, when the module in Figure 5 is enabled (i.e.,  $\text{ctrl\_1e\_} = B = 1$ ), the oscillation is triggered and the output *Y* starts to bounce between the logic states 0 and 1. Assuming  $A = 1$  for instance, *Y* becomes 0 (case  $B = 1$  and  $A = 0$  in Table 2), and this value is transmitted to port *A*, which moves to the low logic value, too. Now that  $A = 0$ , *Y* moves to the logic value 1 (case  $B = 1$  and  $A = 0$  in Table 2); thus, also, *A* assumes again the high logic state, and the cycle starts again from the beginning, being continuously repeated until  $B = 1$ . To resolve this issue, specific constraints have been specified relying on the *set\_dont\_touch* attribute that was integrated within the Synopsys Design Constraint (SDC) file used in the synthesis flow, to force the synthesis engine to generate a netlist as close as possible to the one in Figure 4 and described accordingly in the RTL code of the RNG. An example of the results of this strategy is illustrated in Figure 6.

**Table 2.** Truth table of *NAND* gate.

<i>B</i>	<i>A</i>	$C = B \cdot A$	$Y = \overline{C} = \overline{B \cdot A}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0



**Figure 6.** Outcome of GaRO synthesis with proper constraints. This schematic has been extracted by using the Synopsys Design Compiler tool and refers to the synthesis of the same RTL code used also for the synthesis of the circuit represented in Figure 5 but including dedicated synthesis constraints.

### 3. Results

#### 3.1. Results on FPGA and 7 nm Standard Cell Technologies

In the case of FPGA, the full implementation flow was performed, i.e., logic synthesis, placement, and routing steps. The target device for this flow was a Xilinx Virtex UltraScale+ HBM FPGA VU37P [21] (device XCVU37P-L2FSVH2892EES9837) that is manufactured using a 16 nm low-power FinFET+ process technology from TSMC. The main programmable logic element of this device is named Configurable Logic Block (CLB), and it integrates several logic resources, including Look-Up Tables (LUTs) (referred to as CLB LUTs), flip-flops (referred to as CLB Registers), and other logic blocks; specifically, that FPGA counts 162,960 CLB, and each one is provided with eight LUTs (for a total of 1,303,680 CLB LUTs) and 16 flip-flops (for a total of 2,607,360 CLB Registers or bits). Moreover, such an FPGA is equipped also with other embedded hardware resources as RAM blocks, DSPs, PLLs, and others. The tool Vivado EDA has been used for the synthesis and implementation process, adopting strategies oriented to the optimization of the performance (of timing and power, notably the synthesis strategy denoted as Flow\_PerfOptimized\_high and the implementation strategy denoted as Performance\_ExtraTimingOpt, respectively). Table 3 reports the final results of this activity.

**Table 3.** Implementation results on FPGA VU37P. The percentage data between round brackets refer to the relative utilization of the corresponding entity with respect to the total of resources offered by the FPGA device.

Entity	Frequency [MHz]	CLB (162,960)	CLB LUTs (1,303,680)	CLB Registers (2,607,360)
RNG engine	260	2151	9842	7121
Entropy Source	260	384	1567	2137
DRBG	260	1528	7327	3685

Regarding the synthesis results on standard-cell technology, in this case, the Design Compiler EDA by Synopsys was used, and the target technology was the one proposed by the EPI project, i.e., the H300 BASE SVT C8 of the 7 nm TSMC process CLN07FF41001 SVT, and released by ARM as part of the package of logic products named Artisan 7 nm TSMC CLN07FF41001. The operating conditions and the technology corner case used in the synthesis were 0.90 V for the voltage supply, 125 °C for temperature, and slow process. Table 4 summarizes the final post-synthesis results on the 7 nm standard-cell technology by ARM, in which the area data are expressed in Gate Equivalent (GE), being 1 GE corresponding to the area of the two-inputs NAND gate of the ARM Artisan 7 nm technology with the smallest area, i.e., 0.0768  $\mu^2$ .



**Table 4.** Post-synthesis results for the RNG engine on 7 nm technology. Frequency data are expressed in GHz, while area data are expressed in kGE.

Entity	Frequency [GHz]	Area [kGE]
RNG engine	4.325	127.16
Entropy Source	4.325	69.29
DRBG	4.325	46.51

### 3.2. RNG Assessment

Several suites exist to evaluate the randomness of an RNG; over the years, researchers have proposed different statistical methods and tests to assess the output sequences of both hardware and software implementations of RNG. The first and most notorious was the Diehard, which has been updated and improved to become the suite Dieharder [22]. Other suites of the test have been proposed in recent years, including the following:

- The ENT suite [23], for example used in [24,25];
- The batteries of tests PractRand [26], employed in [27,28], and TestU01 [29], whose usage is reported in [27,28,30–33]: both essentially represent an enhancement of Diehard(er) suite, because they include some improvements such as the possibility of setting the parameters of some of the offered statical tests (feature not offered by Diehard(er));
- The NIST Statistical Test Suite (STS) [34], used in [24,25,30];
- The battery of tests described in [35], in order to meet the requirements of the BSI standards AIS 20 [36], for DRBGs and AIS 31 [37], for TRNGs, whose usage is documented by [38,39];

A review on the available statistical tests available can be found in [40]. In case of cryptographic applications, the most indicated statistical suites, as confirmed by [38,41], are the ones offered by the standardization organizations, i.e., NIST and BSI. Hence, their usage is indicated to evaluate two fundamental aspects for implementations of RNGs: the entropy and the randomness of bitstreams. The goal of the former metric concerns the quantification of how many bits of entropy an RNG module can generate; the latter is to evaluate how much the sequences of random bits generated by a real RNG differ from the sequences produced by an ideal RNG. In this work, we adopted both the NIST and BSI suites for the assessment of our RNG.

The BSI suite is made available as a Java-based tool that can be downloaded at [42], and it includes a battery of statistical tests divided into two distinct procedures: procedure A, comprising the tests  $T_0$  and  $T_1$  through  $T_5$ , which is aimed to evaluate the randomness of sequences of bits, and procedure B, comprising the tests  $T_6$  to  $T_8$ , whose purpose is to calculate the entropy level of such sequences [35], which is an update of both [43,44], gives mathematical references of the tests, and describes the procedures to be followed for both TRNGs (AIS 31) and DRBGs (AIS 20).

The NIST offers two distinct suites for the two distinct characteristics to be analyzed: the Entropy Assessment (EA) suite, illustrated in the standard NIST SP 800-90B [45], and the already mentioned STS described in the standard NIST SP 800-22 [34]. The former is required for the evaluation of the entropy and is available as a collection of software applications written in C++ language that can be downloaded through a git repository at [46]. The latter can be used for testing the randomness of sequences and is made available as a C software application, whose most recent version is 2.1.2, and it can be downloaded from the NIST website at [47].

#### 3.2.1. Entropy Evaluation

The entropy evaluation of an RNG becomes relevant when it is used to generate seeds for PRNGs or CSPRNGs, as in the case of the entropy source we used in this work to supply the DRBG. By evaluating the level of entropy, it is then possible to determine if the RNG

module can satisfy the requirements of the minimal amount of bits of entropy required by the DRBG mechanism intended to use such seeds as the ones specified by the NIST in [9]. The EA suite for the evaluation of the entropy essentially counts two steps: a preliminary entropy assessment routine for the calculation of the initial entropy  $H_I$  and a successive entropy assessment routine that estimates the entropy per row  $H_R$ , and the entropy per column  $H_C$ , of a  $1000 \times 1000$  matrix of random samples. The routine outputs the final value of entropy  $H$ , which is calculated as the minimum between  $H_I$ ,  $H_R$ , and  $H_C$ :

$$H = \min(H_I, H_R, H_C).$$

The required samples have been acquired from the entropy source module exploiting the dedicated debug probe signal in Figure 1, and the estimated entropy is as follows:

- 7.888 bits of entropy per byte, corresponding to 0.986 bit of entropy per bit, estimated with the NIST EA suite;
- 7.999 bits of entropy per byte, corresponding to 0.999 bit of entropy per bit, estimated with the BSI suite.

The difference among values generated by the two statistical suites may find an explanation in the fact that, as reported and highlighted by [48,49], the estimators of NIST SP 800-90B (i.e., NIST EA) are subject to systematic underestimates of entropy. Anyway, also in the worst case (the NIST suite one), the obtained level of entropy per bit allow achieving an overall entropy of  $0.986 \times 512 = 504.832$  bits for the seeds generated by the entropy source module of the RNG engine (Section 2.1, Figure 1) and used as inputs of DRBG mechanism of the same engine, and such a value largely satisfies the requirement of entropy of at least 256 bits, according to NIST specifications in [9].

### 3.2.2. Randomness Tests

For the randomness evaluation, both the NIST STS and the BSI suite are formulated to test a specific null hypothesis  $H_0$  (i.e., the sequence being tested is random), and an alternative hypothesis  $H_a$  (i.e., the sequence being tested is not random). Such hypotheses are defined to address all the possible cases that can occur in the generation–conclusion procedure when testing the randomness of a bitstream, as reported by Table 5. The probability of Type I error, i.e., the event that a random generator is declared being non-random, is defined as  $\alpha$ , called significance level, while the probability of Type II error, i.e., the event that a non-random generator is declared being random, is denoted by  $\beta$  [34]. The author of [34] defines a range of admitted values for  $\alpha$  and two metrics to evaluate the randomness of a sequence under test. Such metrics rely on the calculation of the so-called  $p$ -value, which is defined as the probability that a perfect random number generator would have produced a sequence less random than the tested sequence. Then, the conclusion on the randomness of a sequence can be determined basing on the relation between the  $p$ -value and  $\alpha$ : if  $p\text{-value} \geq \alpha$ , then the sequence can be considered random with a confidence of  $(1 - \alpha) \cdot 100\%$ ; otherwise, if  $p\text{-value} < \alpha$ , the sequence can be considered as non-random with a confidence of  $(1 - \alpha) \cdot 100\%$ . For cryptographic applications,  $\alpha$  is typically chosen in the range (0.001–0.01). In addition, also, the number of sequences to be tested is regulated by the indication that if we defined  $k$  as the number of  $n$ -bit sequences to be tested, then  $k \geq 1/\alpha$ : i.e., for  $\alpha = 0.01$ , at least 100 sequences have to be tested, while for  $\alpha = 0.001$ , at least 1000 have to be tested. Based on this, each of the 15 statistical tests composing the NIST STS operates as follows:

- The  $p$ -value of each sequence is calculated, discarding the sequences for which  $p\text{-value} < \alpha$ ;
- The ratio between the number of sequences that passed the test (i.e., the one for  $p\text{-value} \geq \alpha$ ) and the total number of tested sequences (i.e.,  $k$ ) is computed, and it is labeled as PProportion (PR);
- The  $p$ -value of sequences that passed the test are distributed in the range  $[0, 1)$  by splitting it into 10 equal sub-intervals, and the uniformity of the distribution of  $p$ -value

is calculated: basing on the chi-square (chi-squared or  $\chi^2$ ) function, the uniformity of distribution is determined by computing a figure that can be considered as a  $p$ -value of  $p$ -value (PoP);

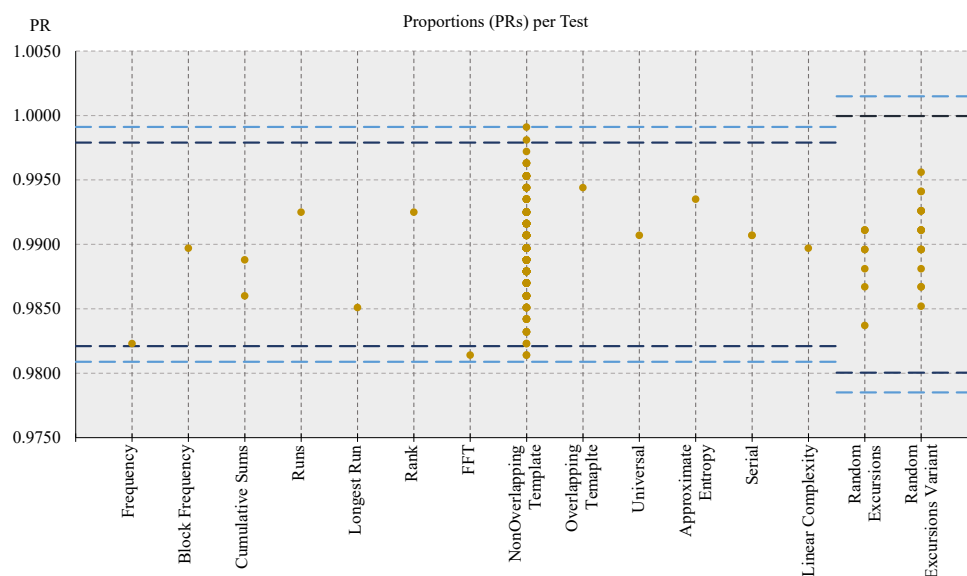
**Table 5.** Possible outcomes of randomness evaluation procedure.

True Situation	Conclusion	
	Data Are Random (Accept H0)	Data Are Not Random (Accept Ha)
Data are random (H0 is true)	No error	Type I error
Data are not random (Ha is true)	Type II error	No error

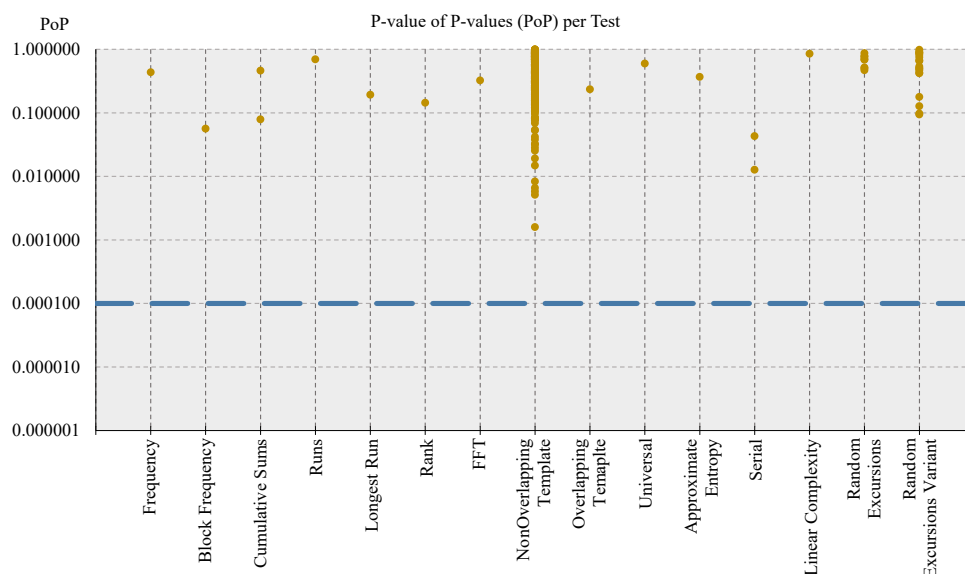
Then, all of the results above are reported for each of the tests, and a final and global conclusion on the randomness of the generator that produced the tested sequences can be derived, by declaring it as random, with a confidence of  $(1 - \alpha) \cdot 100\%$ , if:

- For each test, PR lies in the confidence interval defined as  $(1 - \alpha) \pm 3\sqrt{\frac{\alpha(1-\alpha)}{k}}$ ;
- For each test,  $PoP \geq 0.0001$  (i.e., the  $p$ -values of sequences that passed the test are uniformly distributed).

Regarding the evaluation of the randomness of the DRBG module (and hence of the overall CSPRNG unit corresponding to the RNG engine), its output sequences have been tested with both procedure A of the BSI suite (notably the battery of tests T0 and T1 through T5) and the Fast NIST STS, using the configuration parameters for NIST STS indicated by the BSI in [35] ( $\alpha = 0.01, k = 1073, n = 1,000,000$ , block length of Block Frequency test  $M = 20,000$ , template and block length for Non-Overlapping Template and Overlapping Template tests  $m = 10$ , block lengths of Approximate Entropy, Serial, and Linear Complexity tests, respectively,  $m = 8, m = 16$ , and  $M = 1000$ ). In both cases, all tests passed, confirming the indistinguishability of the bistreams generated by the RNG engine from the ones produced by an ideal RNG, in particular, with a confidence of 99%, according to the specifications of SP 800-22 document. Figure 7 reports a representation of the results for the proportion metric, while Figure 8 reports the one for the uniformity metric.



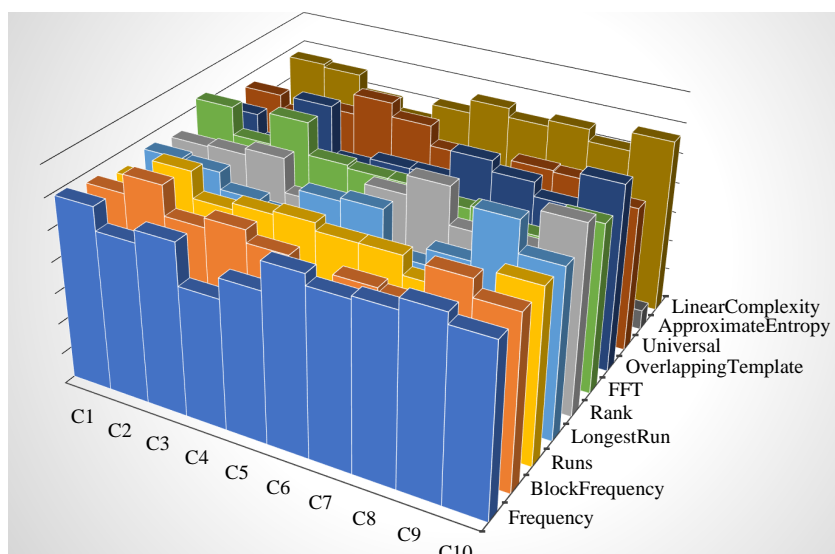
**Figure 7.** Graph of NIST STS PR metric results for tested DRBG sequences. The dark golden points represent the PR values for each test, or sub-test, and the blue dashed lines represent the boundaries of the confidence interval(s): if the PR value lies outside the confidence interval(s), the test is failed: three (six) failing tests are tolerated for the widest (narrowest) confidence interval.



**Figure 8.** Graph NIST STS PoP metric results for tested DRBG sequences. The dark golden points represent the PoP values for each test, and the light blue dashed line traces the threshold for the pass–fail criterion: if the PoP value is greater than (or equal to) threshold, then the test is passed; otherwise, it is failed. The vertical axis uses the logarithmic scale.

Referring to Figure 7, the dark golden dots represent the PR values for each test, while the blue dashed lines represent the upper and lower limits of the confidence interval used to determine if a sequence passed a test: the light blue ones delimit the confidence interval deriving from the original formula of NIST SP 800-22, i.e., the one with the value 3 for the constant, and the dark blue ones delimit the corresponding confidence interval using a value of 2.6 for the constant in the formula, according to [50]. Interpreting the graphical representation of results, on one hand, multiple dark golden dots (i.e., PR values) on the same vertical line means that the corresponding test is composed of multiple sub-tests, and the outcome of each of them is reported; on the other one, it can be noted also that some of the tested sequences failed some of the test (i.e., the associated dark golden dot lies outside the confidence interval). This must not be interpreted as a failure of the NIST STS testing procedure, because, according to [50], a certain number of failing tests can be tolerated without affecting the outcome of the experiment. The number of tolerated failing tests is three (or six) when referring to a confidence interval computed using a constant value of 3 (or 2.6). In the former case, no test (0) exceeded the limit of the confidence interval related to the constant value of 3, while in the latter case, only five tests lie outside the confidence interval related to the constant value of 2.6; hence, the final result is that the tested sequences successfully passed the NIST STS tests, according to the proportion metric. The same graphical representation approach applies also to Figure 8, showing that the PoP values (dark golden dots) of all tests are greater than the threshold of 0.0001 (dashed light blue line) used for the pass–fail criterium, indicating the distributions of *p*-values were shown to be uniform, as illustrated by Figure 9.

As an additional step, also, the bitstreams of the entropy source module were tested with the NIST STS software, and also, in this case, the sequences passed all tests, confirming that such a unit can be used also as a stand-alone TRNG, being already equipped with hardware resources to the health test of the raw binary stream.



**Figure 9.** Histograms of  $p$ -values distributions from NIST STS tests. The three-dimensional histograms of the distribution of  $p$ -values are reported only for single-experiment tests of NIST STS and, according to the results of PoP metric (Figure 8), their uniformity can be noted.

#### 4. Comparison to the State of the Art

To make a comparison with the state of the art, ref. [39] offers a good reference point, because it offers a review of several implementations of TRNG on FPGA technologies, by using the AIS 31 suite for the estimation of entropy. The FPGA analyzed in [39] are the Intel Cyclone V [51], the Xilinx Spartan 6 [52], and the Microsemi SmartFusion 2 [53], respectively manufactured on technology nodes at 28, 45, and 65 nm, and the characteristics reported for the TRNG involve both the entropy per bit and the entropy rate, which were calculated as the entropy per bit multiplied by the bit rate, according to the main target application of these modules to generate seeds with a certain amount of bits of entropy (as expressed above). Even if a TRNG, or an entropy source module, can produce an elevated level of entropy per bit, it may anyway not be suitable for some applications in case the generation rate is too low, or, in other words, the latency it features is too high, when compared to the interval time that can be considered acceptable for the accumulation of a certain amount of entropy bits. Table 6 includes data extracted from [39] that refer to the corresponding publications in the literature.

**Table 6.** Comparison between FPGA implementations of TRNG. Data reported in this table have been extracted from the results of this works and from [39]. In case multiple citations are present, it indicates that the analyzed TRNG implementation was built merging the contributions from each of the works documented in the corresponding citation.

Implementation	FPGA	Bit Rate [Mbit/s]	Entropy per Bit (from BSI Suite)	Entropy Rate [Mbit/s]
This work	Virtex Ultrascale+ VU37P	2080	0.999	2077.92
[54]	Spartan 6	0.0042	0.999	0.004
	Cyclone V	0.0027	0.990	0.003
	SmartFusion 2	0.014	0.980	0.013
[55]	Spartan 6	0.54	0.999	0.539
	Cyclone V	1.44	0.999	1.438
	SmartFusion 2	0.328	0.999	0.327

Table 6. Cont.

Implementation	FPGA	Bit Rate [Mbit/s]	Entropy per Bit (from BSI Suite)	Entropy Rate [Mbit/s]
[56–58]	Spartan 6	2.57	0.999	2.567
	Cyclone V	2.2	0.999	2.197
	SmartFusion 2	3.62	0.999	3.616
[59,60]	Spartan 6	0.44	0.981	0.431
	Cyclone V	0.6	0.986	0.592
	SmartFusion 2	0.37	0.921	0.340
[11,61]	Spartan 6	0.625	0.999	0.624
	Cyclone V	1	0.987	0.985
	SmartFusion 2	1	0.999	0.999
[62,63]	Spartan 6	154	0.998	154.121
	Cyclone V	245	0.999	244.755
	SmartFusion 2	188	0.999	188.522

The results in Table 6 show that the entropy source module integrated into the RNG engine and implemented on the VCU128 demo board offers a level of entropy equal to the maximum one provided also by other solutions (and very close to the maximum theoretical one of 1.000), and that it features also the best entropy rate, which is about 8.5 times greater than the best one of other FPGA implementation. In addition, if assuming the entropy estimated by the NIST EA, i.e., 0.986, the corresponding bit rate was  $2080 \times 0.986 = 2050.88$  Mbit/s, which is much greater than the best one of the counterparts that are 244.755 Mbit/s. This was confirmed also by the work presented in [6] in which it presented the implementation of a TRNG on an Intel Stratix IV FPGA [64], and on which basis the entropy source module was integrated into the RNG engine: in that work, the measured entropy was of 0.995, and the bit rate was 400 Mbit/s, thus leading to an entropy rate of 398 Mbit/s.

Please note that the largely better performance in terms of throughput is due to both the FPGA technology being significantly newer and the amount of resources used (e.g., the more resources you use in parallel, the larger throughput you can achieve). We reported the throughput as a figure of merit, but we did not center our analysis on that, knowing that it would not be fair. As a further and general metric to evaluate the entropy estimation, as suggested by the authors of [65], the minimum value of entropy per bit should satisfy the requirement of 0.910, as specified by [35] in the form of the Shannon entropy of 0.997. The Shannon entropy ( $H_S$ ) can be calculated from the value of minimum entropy (or min-entropy,  $H_{min}$ , i.e., the one computed by the NIST EA and BSI suite), by the following formula:

$$H_S = -2^{-H_{min}} \cdot \log_2(2^{-H_{min}}) - (1 - 2^{-H_{min}}) \cdot \log_2(1 - 2^{-H_{min}}).$$

To derive the equation above, we relied on the following assumptions: approximation of Shannon entropy as the sum of probabilities [36], and the fact that the random bit can assume only two values (0 and 1). These are also specified and exploited in [65].

Using only the expression above, the minimum value of  $H_S = 0.997$  corresponds to the value of  $H_{min}$  equal to 0.910. The author of [35] specifies such a value as one of the requirements that an RNG must respect to be compliant with the class-defined *PTG.2*, the one collecting PTRNG whose target is the generation of cryptographic keys, seeds, and random padding bits. Hence, the corresponding value (the effective values of  $H_S$  are 0.99993141 and 0.9999965 for  $H_{min}$  values of 0.986 (NIST EA) and 0.999 (procedure B of BSI suite), respectively, that can be rounded both to the value 1.000 when using 4 significant digits for representing those numbers) of  $H_S$  is 1.000 for both the min-entropy

values of 0.986 and 0.999, enabling the entropy source module to be an RNG of class PTG.2. Successively, procedure A of the BSI suite was applied to the random sequences of the entropy source module, according to [35], and all tests passed, confirming the compliance to the European requirements for TRNG expressed by the AIS 31 standard.

## 5. Conclusions

In this work, we implemented a high-entropy and high-throughput RNG, which demonstrated to be portable in different technologies. We validated the correct implementation of the RNG engine as a CSPRNG, and we verified that it satisfies the security requirements for cryptographic applications, because as confirmed by the results, once seeds with appropriate entropy levels are used, it can generate sequences whose randomness cannot be distinguished from the one of an ideal random generator, with the confidence of 99%. As such characteristics are depending only on the deterministic part of the RNG engine, it will be maintained also for the EPI chip on the 7 nm standard-cell technology. In addition, the measured level of entropy contributed to proving the portability of the digital entropy source module, showing also characteristics in terms of entropy rate, which outperforms the other main solutions in the field of digital TRNGs. The extracted value of 1.000 for the Shannon entropy per bit makes it a candidate for the PTG.2 class of AIS 31 standard, i.e., a (physical) TRNG for the generation also of certified and highly-qualified cryptographic keys.

**Author Contributions:** Conceptualization and methodology, L.C., S.D.M. and P.N.; validation, L.C.; project administration and funding acquisition, S.S. and L.F. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work has been partially funded by MIUR, in the Crosslab project under the Dipartimento di excellence programme and by the European Processor Initiative (EPI) project, under grant agreement No. 826647.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Not applicable.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Herrero-Collantes, M.; Garcia-Escartin, J.C. Quantum random number generators. *Rev. Mod. Phys.* **2017**, *89*, 015004. [CrossRef]
2. Consortium, E. EPI Website. Available online: <https://www.european-processor-initiative.eu/> (accessed on 10 December 2021).
3. Nannipieri, P.; Di Matteo, S.; Baldanzi, L.; Crocetti, L.; Zulferti, L.; Saponara, S.; Fanucci, L. VLSI design of Advanced-Features AES CryptoProcessor in the framework of the European Processor Initiative. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2021**, in press. [CrossRef]
4. Di Matteo, S.; Baldanzi, L.; Crocetti, L.; Nannipieri, P.; Fanucci, L.; Saponara, S. Secure Elliptic Curve Crypto-Processor for Real-Time IoT Applications. *Energies* **2021**, *14*, 4676–2. [CrossRef]
5. Nannipieri, P.; Bertolucci, M.; Baldanzi, L.; Crocetti, L.; Di Matteo, S.; Falaschi, F.; Fanucci, L.; Saponara, S. SHA2 and SHA-3 Accelerator Design in a 7nm Technology within the European Processor Initiative. *Microprocess. Microsyst.* **2020**, *87*, 103444. in press. [CrossRef]
6. Nannipieri, P.; Di Matteo, S.; Baldanzi, L.; Crocetti, L.; Belli, J.; Fanucci, L.; Saponara, S. True Random Number Generator Based on Fibonacci-Galois Ring Oscillators for FPGA. *Appl. Sci.* **2021**, *11*, 3330. [CrossRef]
7. Baldanzi, L.; Crocetti, L.; Falaschi, F.; Belli, J.; Fanucci, L.; Saponara, S. Digital Random Number Generator Hardware Accelerator IP-Core for Security Applications. In *Applications in Electronics Pervading Industry, Environment and Society. ApplePies 2019; Lecture Notes in Electrical Engineering (LNEE)*; Springer: Berlin/Heidelberg, Germany, 2020; Volume 627, pp. 117–123. [CrossRef]
8. Baldanzi, L.; Crocetti, L.; Falaschi, F.; Bertolucci, M.; Belli, J.; Fanucci, L.; Saponara, S. Cryptographically Secure Pseudo-Random Number Generator IP-Core Based on SHA2 Algorithm. *Sensors* **2020**, *20*, 1869. [CrossRef]
9. NIST. *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*; SP 800-90A Rev. 1; NIST: Gaithersburg, MD, USA, 2015.
10. NIST. *Recommendation for Random Bit Generator (RBG) Constructions*; SP 800-90C (Draft); NIST: Gaithersburg, MD, USA, 2016.

11. Varchola, M.; Drutarovsky, M. New High Entropy Element for FPGA Based True Random Number Generators. In *Cryptographic Hardware and Embedded Systems (CHES) 2010*; Lecture Notes in Computer Science (LNCS); Springer: Berlin/Heidelberg, Germany, 2010; Volume 6225, pp. 351–365.
12. Vasylytsov, I.; Hambardzumyan, E.; Kim, Y.S.; Karpinsky, B. Fast Digital TRNG Based on Metastable Ring Oscillator. In *Cryptographic Hardware and Embedded Systems (CHES) 2008*; Lecture Notes in Computer Science (LNCS); Springer: Berlin/Heidelberg, Germany, 2008; Volume 5154, pp. 164–180.
13. Schramm, M.; Dojen, R.; Heigl, M. Experimental assessment of FIRO- and GARO-based noise sources for digital TRNG designs on FPGAs. In Proceedings of the 2017 International Conference on Applied Electronics (AE), Pilsen, Czech Republic, 5–6 September 2017; pp. 1–6.
14. Demir, K.; Ergün, S. A Comparative Study on Fibonacci-Galois Ring Oscillators for Random Number Generation. In Proceedings of the 2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS), Springfield, MA, USA, 9–12 August 2020; pp. 631–634.
15. Cao, Y.; Rožić, V.; Yang, B.; Balasch, J.; Verbauwhede, I. Exploring active manipulation attacks on the TERO random number generator. In Proceedings of the 2016 IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS), Abu Dhabi, United Arab Emirates, 16–19 October 2016; pp. 1–4.
16. Li, T.; Wu, L.; Zhang, X.; Wu, X.; Zhou, J.; Wang, X. A Novel Transition Effect Ring Oscillator Based True Random Number Generator for a Security SoC. In Proceedings of the 2017 International Conference on Electron Devices and Solid-State Circuits (EDSSC), Hsinchu, Taiwan, 18–20 October 2017; pp. 1–2.
17. Fujieda, N. On the Feasibility of TERO-Based True Random Number Generator on Xilinx FPGAs. In Proceedings of the 2020 30th International Conference on Field-Programmable Logic and Applications (FPL), Gothenburg, Sweden, 31 August–4 September 2020; pp. 103–108.
18. Kan, W. Analysis of Underlying Assumptions in NIST DRBGs. In *IACR Cryptology ePrint Archive*; IACR: Lyon, France, 2007; p. 345.
19. Synopsys. Design Compiler. Available online: <https://www.synopsys.com/support/training/rtl-synthesis.html> (accessed on 10 December 2021).
20. Xilinx. Vivado. Available online: <https://www.xilinx.com/products/design-tools/vivado/implementation.html> (accessed on 10 December 2021).
21. Xilinx. UltraScale+ FPGAs—Product Tables and Product Selection Guide. 2015–2021. Available online: <https://www.xilinx.com/support/documentation/selection-guides/ultrascale-plus-fpga-product-selection-guide.pdf> (accessed on 10 December 2021).
22. Brown, R. Dieharder: A Random Number Test Suite. Version 3.31.1. 2004. Available online: <https://webhome.phy.duke.edu/~rgb/General/dieharder.php> (accessed on 10 December 2021).
23. Walker, J. ENT—A Pseudorandom Number Sequence Test Program. Available online: <https://www.fourmilab.ch/random/> (accessed on 10 December 2021).
24. Aggarwal, D.; Ghatikar, R.; Chennuri, S.; Banerjee, A. Generation of 1 Gb full entropy random numbers with the enhanced-NRNG method. *arXiv* **2021**, arXiv:2108.04331.
25. Sun, Y.; Lo, B. Random number generation using inertial measurement unit signals for on-body IoT devices. In Proceedings of the Living in the Internet of Things: Cybersecurity of the IoT-2018, London, UK, 28–29 March 2018; pp. 1–9. [CrossRef]
26. PractRand (Practically Random) C++ Library. Available online: <http://prcrand.sourceforge.net/> (accessed on 10 December 2021).
27. Gevorkyan, M.N.; Demidova, A.; Korol'kova, A.; Kulyabov, D. A Practical Approach to Testing Random Number Generators in Computer Algebra Systems. *Comput. Math. Math. Phys.* **2020**, *60*, 65–73. [CrossRef]
28. Sleem, L.; Couturier, R. TestU01 and Pracrnd: Tools for a randomness evaluation for famous multimedia ciphers. *Multimed. Tools Appl.* **2020**, *79*, 24075–24088. [CrossRef]
29. L'Ecuyer, P.; Simard, R. TestU01: AC library for empirical testing of random number generators. *ACM Trans. Math. Softw. (TOMS)* **2007**, *33*, 1–40. [CrossRef]
30. Bisadi, Z.; Fontana, G.; Moser, E.; Pucker, G.; Pavesi, L. Robust Quantum Random Number Generation With Silicon Nanocrystals Light Source. *J. Light. Technol.* **2017**, *35*, 1588–1594. [CrossRef]
31. Bakiri, M.; Couchot, J.F.; Guyeux, C. CIPRNG: A VLSI Family of Chaotic Iterations Post-Processings for  $\mathbb{F}_2$ -Linear Pseudorandom Number Generation Based on Zynq MPSoC. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2018**, *65*, 1628–1641. [CrossRef]
32. Chen, S.; Yu, S.; Lü, J.; Chen, G.; He, J. Design and FPGA-Based Realization of a Chaotic Secure Video Communication System. *IEEE Trans. Circuits Syst. Video Technol.* **2018**, *28*, 2359–2371. [CrossRef]
33. Sharipov, B.R.; Perukhin, M.Y.; Mullayanov, B.I. Statistical Analysis of Pseudorandom Sequences and Stegocontainers. In Proceedings of the 2021 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM), Sochi, Russia, 17–21 May 2021; pp. 434–439. [CrossRef]
34. NIST. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*; SP 800-22; NIST: Gaithersburg, MA, USA, 2010.
35. Killmann, W.; Schindler, W. *A Proposal for: Functionality Classes for Random Number Generators, Version 2.0*; Mathematical-Technical Reference of AIS 20/31; NIST: Gaithersburg, MA, USA, 2011.
36. BSI. *Functionality Classes and Evaluation Methodology for Deterministic Random Number Generators, Version 1*; AIS 20; BSI: Bonn, Germany, 1999.



37. BSI. *Functionality Classes and Evaluation Methodology for Physical Random Number Generators*, Version 1; AIS 31; BSI: Bonn, Germany, 2001.
38. Zhang, M.; Zhang, X.; Zhu, Y.; Miao, S. Overview of Randomness Test on Cryptographic Algorithms. *J. Phys. Conf. Ser. IOP Publ.* **2021**, *1861*, 012009.
39. Petura, O.; Mureddu, U.; Bochar, N.; Fischer, V.; Bossuet, L. A Survey of AIS-20/31 Compliant TRNG Cores Suitable for FPGA Devices. In Proceedings of the 2016 26th International Conference on Field Programmable Logic and Applications (FPL), Lausanne, Switzerland, 29 August–2 September 2016.
40. Demirhan, H.; Bitirim, N. Statistical testing of cryptographic randomness. *İstat. Derg. İstat. Aktüerya* **2016**, *9*, 1–11.
41. Balasch, J.; Bernard, F.; Fischer, V.; Grujić, M.; Laban, M.; Petura, O.; Rožić, V.; Van Battum, G.; Verbauwhede, I.; Wakker, M.; others. Design and Testing Methodologies for True Random Number Generators Towards Industry Certification. In Proceedings of the 2018 IEEE 23rd European Test Symposium (ETS), Bremen, Germany, 28 May–1 June 2018; pp. 1–10.
42. BSI. Implementation of Test Procedure A and Test Procedure B for AIS 20/31 Standard. Available online: [https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS\\_31\\_testsuit\\_zip.zip](https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_31_testsuit_zip.zip) (accessed on 10 December 2021).
43. Schindler, W. *AIS 20: Functionality Classes and Evaluation Methodology for Deterministic Random Number Generators*; Version 2.0; Mathematical-Technical Reference of AIS 20; BSI: Bonn, Germany, 1999.
44. Killmann, W.; Schindler, W. *A Proposal for: Functionality Classes and Evaluation Methodology for True (Physical) Random Number Generators*, version 3.1; Mathematical-Technical Reference of AIS 31; BSI: Bonn, Germany, 2001.
45. NIST. *Recommendation for the Entropy Sources Used for Random Bit Generation*; SP 800-90B; NIST: Gaithersburg, MD, USA, 2018.
46. NIST. SP 800-90B Entropy Assessment Software, Version 1.0. 2019. Available online: [https://github.com/usnistgov/SP800-90B\\_EntropyAssessment](https://github.com/usnistgov/SP800-90B_EntropyAssessment) (accessed on 10 December 2021).
47. NIST. SP 800-22 STS Software, Version 2.1.1. 2010. Available online: [https://csrc.nist.gov/CSRC/media/Projects/Random-Bit-Generation/documents/sts-2\\_1\\_2.zip](https://csrc.nist.gov/CSRC/media/Projects/Random-Bit-Generation/documents/sts-2_1_2.zip) (accessed on 10 December 2021).
48. Kim, Y.; Guyot, C.; Kim, Y.S. On the Efficient Estimation of Min-Entropy. *IEEE Trans. Inf. Forensics Secur.* **2021**, *16*, 3013–3025. [[CrossRef](#)]
49. Kelsey, J.; McKay, K.A.; Turan, M.S. Predictive Models for Min-Entropy Estimation. In *International Workshop on Cryptographic Hardware and Embedded Systems*; Springer: Berlin/Heidelberg, Germany, 2015; Volume 9293, pp. 373–392.
50. Sýs, M.; Riha, Z.; Matyas, V.; Marton, K.; Suci, A. On the Interpretation of Results from the NIST Statistical Test Suite. *Rom. J. Inf. Sci. Technol. (ROMJIST)* **2015**, *18*, 18–32.
51. Intel. Cyclone V Device Overview. 2018. Available online: [https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyclone-v/cv\\_51001.pdf](https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyclone-v/cv_51001.pdf) (accessed on 10 December 2021).
52. Xilinx. *Spartan-6 FPGAs: Performance, Power, and I/O Optimized for Cost-Sensitive Applications*; Technical Report; Xilinx: San Jose, CA, USA, 2017. Available online: [https://www.xilinx.com/support/documentation/white\\_papers/wp396\\_S6\\_HV\\_Perf\\_Power.pdf](https://www.xilinx.com/support/documentation/white_papers/wp396_S6_HV_Perf_Power.pdf) (accessed on 10 December 2021).
53. Microsemi. User Guide SmartFusion2 and IGLOO2 FPGA Security and Best Practices (UG0443), Revision 10.0. 2019. Available online: [https://www.microsemi.com/document-portal/doc\\_download/132037-ug0443-smartfusion2-and-igloo2-fpga-security-best-practices-user-guide](https://www.microsemi.com/document-portal/doc_download/132037-ug0443-smartfusion2-and-igloo2-fpga-security-best-practices-user-guide) (accessed on 10 December 2021).
54. Baudet, M.; Lubicz, D.; Micolod, J.; Tassiaux, A. On the Security of Oscillator-Based Random Number Generators. *J. Cryptol.* **2011**, *24*, 398–425. [[CrossRef](#)]
55. Kohlbrenner, P.; Gaj, K. An Embedded True Random Number Generator for FPGAs. In Proceedings of the 12th International Symposium on Field Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2004; pp. 71–78.
56. 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.* **2007**, *56*, 109–119. [[CrossRef](#)]
57. Wold, K.; Tan, C.H. Analysis and Enhancement of Random Number Generator in FPGA Based on Oscillator Rings. In Proceedings of the 2008 International Conference on Reconfigurable Computing and FPGAs, Cancun, Mexico, 3–5 December 2008; pp. 385–390.
58. Bochar, N.; Bernard, F.; Fischer, V.; Valtchanov, B. True-Randomness and Pseudo-Randomness in Ring Oscillator-Based True Random Number Generators. *Int. J. Reconfig. Comput.* **2010**, *2010*, 879281. [[CrossRef](#)]
59. Fischer, V.; Drutarovský, M. True Random Number Generator Embedded in Reconfigurable Hardware. In *Cryptographic Hardware and Embedded Systems (CHES) 2002*; Lecture Notes in Computer Science (LNCS); Springer: Berlin/Heidelberg, Germany, 2002; Volume 2523, pp. 415–430.
60. Bernard, F.; Fischer, V.; Valtchanov, B. Mathematical Model of Physical RNGs Based on Coherent Sampling. *Tatra Mt.-Math. Publ.* **2010**, *45*, 1–14. [[CrossRef](#)]
61. Haddad, P.; Fischer, V.; Bernard, F.; Nicolai, J. A Physical Approach for Stochastic Modeling of TERO-Based TRNG. In *Cryptographic Hardware and Embedded Systems (CHES) 2015*; Lecture Notes in Computer Science (LNCS); Springer: Berlin/Heidelberg, Germany, 2015; Volume 9293, pp. 357–372.
62. Cherkaoui, A.; Fischer, V.; Aubert, A.; Fesquet, L. A Self-Timed Ring Based True Random Number Generator. In Proceedings of the 2013 IEEE 19th International Symposium on Asynchronous Circuits and Systems, Santa Monica, CA, USA, 19–22 May 2013; pp. 99–106.

63. Cherkaoui, A.; Fischer, V.; Fesquet, L.; Aubert, A. A Very High Speed True Random Number Generator with Entropy Assessment. In *Cryptographic Hardware and Embedded Systems (CHES) 2013*; Lecture Notes in Computer Science (LNCS); Springer: Berlin/Heidelberg, Germany, 2013; Volume 8086, pp. 179–196.
64. Intel. Overview for the Stratix IV Device Family. In *Stratix IV Device Handbook*; 2020; Volume 1. Available online: [https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-iv/stx4\\_siv51001.pdf](https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-iv/stx4_siv51001.pdf) (accessed on 10 December 2021).
65. Peetermans, A.; Rozic, V.; Verbauwhede, I. A Highly-Portable True Random Number Generator Based on Coherent Sampling. In *Proceedings of the 2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, Barcelona, Spain, 8–9 September 2019; pp. 218–224.