A Fast, Efficient, Platform-Adaptive, and AIS-20/31 Compliant PLL-Based True Random Number Generator on an SoC FPGA

Oğuz Yayla[1] and Yunus Emre Yılmaz[2]

[1] Institute of Applied Mathematics, Middle East Technical University, 06800 Ankara, Turkey
oguz@metu.edu.tr
[2] Aselsan Inc., Ankara, Turkey
yeylmz@gmail.com

Abstract. Phase-locked loops (PLLs) embedded within field-programmable gate arrays (FPGAs) or system-on-chip FPGAs (SoCs) present a promising methodology for the generation of random numbers. Their widespread integration across these platforms, combined with their isolated operational characteristics and robust entropy generation, as substantiated by prior research, positions PLL-based true random number generators (PLL-TRNGs) as highly effective solutions for this purpose. The present study focuses on the implementation of PLL-TRNGs utilizing the ZC702 Rev1.1 Evaluation Board, which incorporates the Zynq-7020 SoC from Xilinx. For experimental validation, a configuration involving three such boards is employed. The parameters governing the PLL-TRNG are optimized through a backtracking algorithm. Additionally, a novel, platform-adaptive technique is proposed to enhance the rate of random data bit generation while preserving entropy characteristics. The system's performance is rigorously evaluated against the criteria established by the German Federal Office for Information Security (BSI) AIS-20/31 Tests, with a detailed account of the implementation process provided. Furthermore, the study demonstrates the minimal resource utilization of the PLL-TRNG design within a SoC, thereby underscoring its suitability for Internet-of-Things (IoT) applications, where logic resources are often highly constrained.

Keywords: random number generation, PLL-TRNG, AIS-20/31, SoC, FPGA

1 Introduction

Random numbers are essential for cryptographic systems, requiring high statistical quality, which is provided by True Random Number Generators (TRNGs) based on physical phenomena to ensure unpredictability. In this study, jitter, defined as the fluctuation of clock periods in the time domain, is used as the entropy source. Phase-Locked Loops (PLLs), commonly found in field-programmable

gate array (FPGA) and system-on-chip (SoC) platforms, are selected for entropy harvesting due to their inherent noise and jitter. The simplicity of PLL-based TRNGs (PLL-TRNGs) and their effective cryptographic properties have been demonstrated in previous works, with designs utilizing one or two PLLs, as explained in [1], [2], [3], and [4], along with a stochastic model introduced in [5]. Additionally, the implementation details of PLL-TRNGs are analyzed in [6] and [7], highlighting their straightforward design, coherent sampling, and isolated locations in FPGAs and SoCs, making them highly suitable for secure random number generation.

The primary challenge in PLL-TRNG design is the selection of optimal PLL settings from a vast configuration space. The chosen parameters must yield both a sufficient entropy rate and an adequate output bit rate. This study adopts the parameter determination process outlined in [10]. Notably, the backtracking algorithm employed in [10] offers significant advantages over previous search algorithms for PLL-TRNG design, as presented in [1], [8], and [9].

While PLL-TRNGs offer high entropy and benefit from the isolation of their PLLs, they suffer from low output speed. To address this, we propose a novel, platform-adaptive method that aggregates multiple PLL-TRNGs; our implementation on the Xilinx Zynq-7020 SoC utilizes all four available PLLs and can be extended to platforms with even more. We select an SoC rather than a standalone FPGA to leverage higher integration density, reduced power consumption, smaller board footprint, and better processor–FPGA communication bandwidth.

Once random numbers are generated, their quality must be assessed. For this evaluation, the AIS-20/31 standard [11], a methodology proposed by the German Federal Office for Information Security (BSI), is chosen.

This research culminates in the implementation of a four-PLL TRNG (4-PLL-TRNG) on the Xilinx Zynq-7020 SoC. To elucidate the design progression, a referenced configuration utilizing two PLLs and two intermediate configurations utilizing three PLLs are developed. PLL parameter optimization is achieved through a backtracking algorithm as outlined in [10]. Subsequently, four distinct PLL-TRNG configurations are implemented on the Xilinx (AMD) ZC702 Evaluation Kit Rev1.1, incorporating the Zynq-7020 SoC. To ensure design independence from the specific board, three evaluation kits are employed. Rigorous testing against AIS-20/31 standards is conducted on the generated random data. Hence, a comparative analysis with existing PLL-TRNG implementations is performed. Also, the utilization rates of 4-PLL-TRNG implementations in the Zynq-7020 SoC for three different configurations are calculated.

This paper constitutes a revision of [12]. Furthermore, the PLL-TRNG design methodology is described in detail in [13].

Our work presents three primary contributions:

1. We design a new structure of PLL-TRNG that is adaptive to new FPGAs or SoCs, which can be used to increase the bit rate without worsening cryptographic properties.

2. We conduct our new, fast, and adaptive design implemented on Xilinx Zynq-7020 SoC with respect to AIS-20/31 Tests and compare our results with previous works.
3. We demonstrate the low resource utilization of our proposed 4-PLL-TRNG and highlight its suitability for integration into IoT systems.

The paper is organized as follows. Section 2 provides the basic background information to explain how PLL-TRNG works. In Section 3, the implementation details of our proposed PLL-TRNG and the evaluation metrics for TRNGs are explained. In Section 4, the test results of one referenced and three proposed PLL-TRNGs are presented and compared with previous works. In the end, Section 5 concludes the paper.

2 Background Information About PLL-TRNG Implementation

2.1 Basics of Phase-Locked Loops (PLLs)

A PLL is a circuit (as depicted in Fig. 1) that uses an input signal to synchronize a signal from an embedded oscillator on it. The grey blocks represent the analog components, which cannot be parameterized, whereas the M, N, and C integer division coefficients, depicted in white blocks, need to be configured. These coefficients are essential for calculating the output frequency of the PLL (f_{out}) from the reference frequency (f_{ref}), as described in Equation (1).

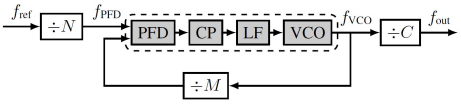

Fig. 1. Block diagram of a PLL (PFD: phase frequency detector, CP: charge pump, LF: loop filter, VCO: voltage-controlled oscillator) [10]

$$f_{out} = f_{ref} \times \frac{M}{N \times C} \tag{1}$$

2.2 Random Bit Generation Principle of the PLL-TRNG

Fig. 2 presents the working principle of the PLL-TRNG with one PLL. In this structure, the jittered clock signal clk_1 from the PLL is sampled by a data flip-flop (D-FF) using the reference clock signal clk_0. The 1-bit counter records the number of samples that equal one. Due to the frequency relationship established by the PLL, a pattern with a period $T_Q = K_D \times T_0 = K_M \times T_1$ emerges at the flip-flop output. As a result, certain samples consistently yield a value of one (indicated in blue in Fig. 2, specifically the 4th and 7th dots), while others consistently yield zero (shown in green for the 2nd and 5th dots). Additionally, some samples display random values (marked in red for the 1st, 3rd, 6th, and 8th

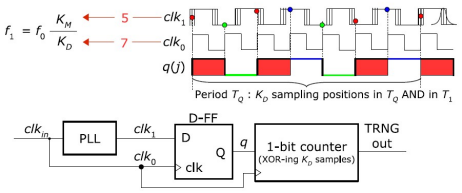

Fig. 2. Principle of the PLL-TRNG with one PLL [18]

dots). By applying the coherent sampling principle and rearranging the samples based on their positions, the waveform of one period of clk_1 can be reconstructed as explained in [1] and [18].

Fig. 3 illustrates a two-PLL variant of the same PLL-TRNG design, which we adopt as a reference model due to its better performance characteristics. The incorporation of two PLLs significantly enhances design flexibility by expanding the practical operating ranges for critical parameters, K_M and K_D, consequently increasing attainable bit and entropy rates. Moreover, this configuration substantially reduces autocorrelation between output bits. While incurring increased implementation costs, these can often be mitigated through resource sharing with other system components, as proposed in [8].

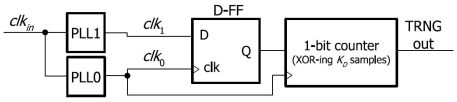

Fig. 3. PLL-TRNG with two PLLs configuration [18]

In this two-PLL case, firstly, as it is stated in Fig. 2:

$$\frac{f_1}{f_0} = \frac{K_M}{K_D} \tag{2}$$

where K_M and K_D are integer values representing frequency multiplication and division factors, depending on the configuration of PLLs. Each PLL has its multiplication and division factors. Moreover, they are related to K_M and K_D as:

$$K_M = K_{M_1} \cdot K_{D_0} \tag{3}$$
$$K_D = K_{M_0} \cdot K_{D_1} \tag{4}$$

The output (Q) of D-FF in the left part of both Fig. 2 and Fig. 3 has a pseudo-random pattern with a certain period. After XORing that pattern in the decimator or 1-bit counter, the bit rate of the PLL-TRNG is defined as follows:

$$R = \frac{f_0}{K_D} = \frac{f_1}{K_M} \tag{5}$$

The entropy rate per bit at the generator output depends on the parameters of the jitter and on the parameters of the generator, which are characterized by its sensitivity to the jitter:

$$S = \Delta^{-1} = f_0 \cdot K_M = f_1 \cdot K_D \tag{6}$$

The design of PLL-TRNG relies on choosing appropriate PLL multiplication and division factors. However, selecting these factors can be challenging due to the physical constraints of the PLL, such as the maximum and minimum values of N, M, C, and the input, output, PFD, and VCO frequency range. Consequently, determining these values is an optimization problem, and our solution to this problem is explained in Section 3.2 for Zynq-7020 SoC values listed in Table 1. The definitions of these parameters are provided in Section 2.1, and their illustration is presented in Fig. 1.

Table 1. Ranges of possible values for the PLL parameters and frequencies for Zynq-7000 SoC [16], [17]

Parameters	Min	Max
f_{ref} (MHz)	19	800
P_{VCOd}	1	1
M	2	64
N	1	56
C	1	128
f_{PFD} (MHz)	19	450
f_{VCO} (MHz)	800	1600
f_{out} (MHz)	6.25	464

2.3 Evaluation Metrics of TRNGs

The BSI test suite of AIS-20/31 [11] is highly regarded for evaluating the quality of RNGs, due to its widespread adoption as a reliable and effective tool. Therefore, AIS-20/31 was selected as the evaluation standard for the TRNG in this work. A summary of the tests outlined in Procedures A and B of AIS-20/31 [11], along with brief explanations based on the standard used for cryptographic RNG evaluation, is presented in this section.

Procedure A in AIS-20/31 Tests: Statistical Testing for Random Number Generators

This procedure focuses on the statistical randomness of the generated sequences and includes the following key tests.

Test T1 - Monobit Test:

Purpose: To evaluate the balance between 1s and 0s in the binary output of the RNG.

Explanation: This ensures that roughly half of the bits are 1s and half are 0s, which is expected from a random sequence.

Test T2 - Poker Test:

Purpose: To test the frequency of different bit patterns (e.g., a poker hand).

Explanation: The goal is to assess the uniform distribution of small groups of bits (for example, 4 bits) in the output. A non-uniform distribution would indicate potential bias or non-randomness.

Test T3 - Runs Test:

Purpose: To evaluate the length and frequency of consecutive sequences of identical bits (runs of 0s or 1s).

Explanation: This test checks if the runs of 0s and 1s appear with the expected frequency and length, as expected in a random sequence.

Test T4 - Long Runs Test:

Purpose: To detect overly long sequences of identical bits.

Explanation: If the RNG produces unusually long runs of 0s or 1s, this could indicate non-random behavior that the test aims to capture.

Test T5 - Autocorrelation Test:

Purpose: To measure the correlation between bits in a sequence at various spacings.

Explanation: Ensures that the sequence is not predictable and that the occurrence of one bit does not depend on the earlier bits.

Procedure B in AIS-20/31 Tests: Entropy and Stochastic Model Evaluation

This procedure emphasizes the evaluation of the RNG's entropy source and its model to ensure unpredictability. The focus is less on statistical randomness and more on the inherent unpredictability of generated bits.

Test T6 - Uniform Distribution Test:

Purpose: To evaluate whether the RNG output follows a uniform distribution.

Explanation: The RNG output should be uniformly distributed, meaning that possible output values should have an equal probability of occurring. If certain values are more or less frequent, this would indicate a bias that would compromise the randomness and unpredictability of the RNG. The Uniform Distribution Test checks for this by analyzing the distribution of the generated random numbers.

Test T7 - Test for Homogeneity:

Purpose: To evaluate whether the output from different sections or periods of the RNG behaves in a similar (homogeneous) manner.

Explanation: The homogeneity test checks whether the random numbers produced by RNG are consistent over time. This ensures that the randomness quality does not fluctuate between different runs or periods. If the output from various sections shows significant differences, it could indicate a problem with the RNG's stability or entropy source, potentially introducing weaknesses in cryptographic applications.

Test T8 - Entropy Estimation Test:

Purpose: To measure the amount of entropy in the RNG output while ensuring sufficient randomness.

Explanation: This test calculates the entropy (often min-entropy) of generated random numbers. Entropy estimation assesses the unpredictability of a sequence by examining the difficulty of predicting the most likely outcome. This ensures that the randomness generated by RNG has a high degree of unpredictability, which is crucial for cryptographic security. A lower-than-expected entropy value could indicate predictability, thus compromising the security of random numbers.

Entropy is a very important parameter for evaluating randomness. Hence, its evaluation must be carefully examined. The entropy values produced by Procedure B of the BSI suite are estimations of the min-entropy H_{min}, which is the most conservative measure of unpredictability and is calculated as the negative logarithm of the probability of the most likely outcome. Depending on the application of the implemented entropy source module, another related metric, Shannon entropy H_S, may be required. Shannon entropy can be derived from min-entropy using the following formula:

$$H_S = -2^{-H_{min}} \cdot \log_2\left(2^{-H_{min}}\right) - \left(1 - 2^{-H_{min}}\right) \cdot \log_2\left(1 - 2^{-H_{min}}\right) \quad (7)$$

In addition, BSI specifies that the confidence level of the results is 99.87% [11].

The Resource Utilization Rate on the Processor Although it is not included in any standard evaluation method, the resource utilization rate within the SoC or FPGA has also been a key evaluation metric in our study. This is because our goal is to minimize the resource usage of the hardware primitives that we utilize, ensuring that there is still space available for other designs that will be implemented for additional applications within the SoC or FPGA.

3 PLL-TRNG Implementation Details

3.1 Implemented PLL-TRNG Configurations

A significant drawback of PLL-based TRNGs is their relatively low data throughput. To overcome this, we introduce a new technique that exploits the multiple PLLs inherent in the SoC to boost output capacity. In our implementation on the Xilinx Zynq-7020 SoC, which provides four on-chip PLLs, we first explore intermediate designs using three PLLs to inform a step-wise development process. These preliminary configurations clarify the progression of our approach and support the final four-PLL architecture. Accordingly, Table 2 summarizes four distinct PLL-TRNG variants, and Fig. 4 illustrates each configuration graphically.

Table 2. Configurations of PLL-TRNG Implementations

Annotations for PLL-TRNG Designs Depicted in Fig. 4	Total Number of PLLs	Number of PLLs used as Reference Clock	Number of PLLs used as Jittered Clock	Type of PLL-TRNG Design
(a)	2	1	1	Referenced Design
(b)	3	1	2	Intermediate Design for the Proposed Approach
(c)	3	2	1	Intermediate Design for the Proposed Approach
(d)	4	2	2	Proposed Design

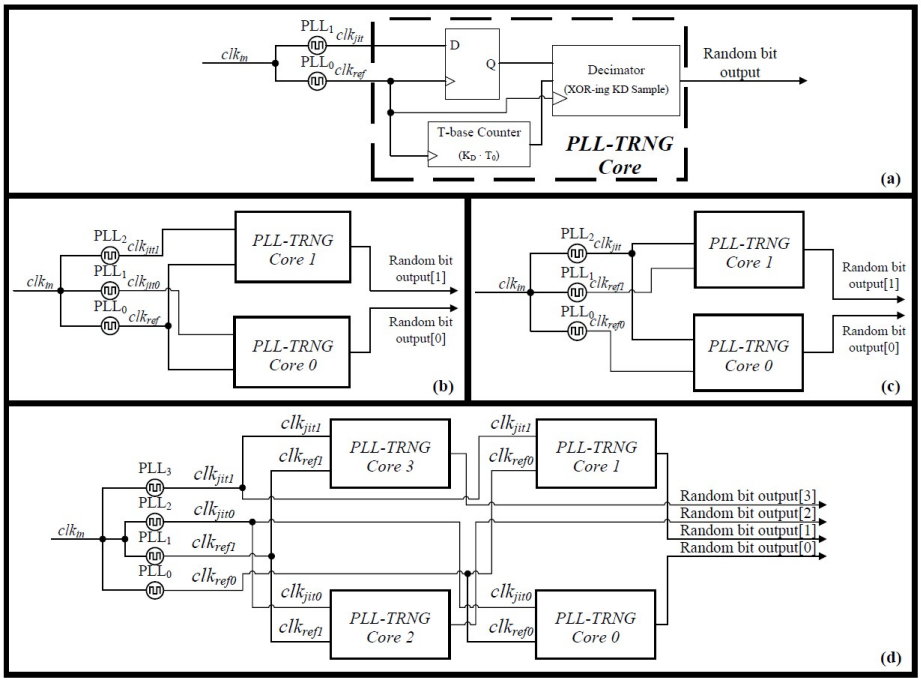

Fig. 4. Implemented PLL-TRNG Configurations: (a), (b), (c), and (d)

3.2 Determining PLL-TRNG Parameters

In this study, we adopt the backtracking algorithm proposed by Colombier et al. [10] for our parameter search. Based on the variables defined in Sections 2.1 and 2.2, and also the constraints enumerated in Table 1, the algorithm systematically explores candidate configurations. Rather than exhaustively testing every combination, it prunes any partial assignment that violates a constraint and then backtracks to consider alternative values, thereby efficiently converging on all feasible solutions. The implementation detailed in [10] thus ensures that the selected PLL-TRNG parameters satisfy both the underlying physical limitations and the specified application requirements.

An open-source Python implementation of the backtracking algorithm is provided by [10], with source code hosted on GitHub. Hence, we can modify it for Zynq-7020 SoC parameters provided in Fig. 1. Table 1, which lists the allowable PLL parameter ranges for the Zynq-7020 SoC, is derived from [17]. However, the upper limit of the output frequency, f_{out}, is governed not by the PLL settings but by the characteristics of the on-chip global buffer (BUFG). Since a BUFG must be employed in the SoC's clocking scheme, its maximum operating frequency constrains the search space for f_{out}. The corresponding BUFG frequency ceiling is also specified in [17].

Once the algorithm has produced its results, we rank them according to three criteria: the highest achievable bit rate (R), the greatest jitter sensitivity (S), and the optimal trade-off value given by the product $R \cdot S$ between these two extremes.

Following the selection of candidate configurations, an additional validation criterion is applied. As depicted in Fig. 2, the sampling of the jitter-affected clock against the reference clock must capture at least one jitter event to ensure true randomness in the output. Consequently, the temporal offset between any rising (or falling) edge of clk_0 and its corresponding edge of clk_0 must be less than Δ. This requirement is satisfied when the following inequality holds [4], [7]:

$$\sigma_{jit} > max(\Delta T_{min}) \tag{8}$$

where σ_{jit} is the standard deviation of the jitter at the output of the PLL, and $max(\Delta T_{min})$ is the largest distance between the two closest edges of clk_0 and clk_1. This can be computed as [4], [7]:

$$max(\Delta T_{min}) = \frac{T_{clk_0}}{4K_M} gcd(2K_M, K_D) = \frac{T_{clk_1}}{4K_D} gcd(2K_M, K_D) \tag{9}$$

where gcd is the greatest common divisor of two integers.

After running the backtracking algorithm for the chosen SoC, we determine the maximum value of $max(\Delta T_{min})$. However, estimating the clock jitter standard deviation (σ_{jit}) precisely is inherently difficult. To address this, we employ Vivado 2019.1's *Clocking Wizard*, which predicts output-clock jitter based on the configured PLL parameters. From the backtracking results, we first identify the configurations that maximize R, maximize S, and maximize the product $R \cdot S$. Each of these three candidate designs is then checked against the jitter constraint of Equation (8); those failing to meet it are discarded and replaced by the next viable options from the backtracking output. Consequently, the validated parameter sets are listed in Table 3, all of which satisfy Equation (8).

3.3 Implementation Setup

Fig. 5 depicts the hardware configuration used in this study. The design is implemented on the ZC702 Rev 1.1 Evaluation Board, which features the

Table 3. Determined Parameters for the PLL-TRNG Implementations

Config.	f_{ref} (MHz)	(M_0, N_0, C_0) (M_1, N_1, C_1)	f_0 (MHz) f_1 (MHz)	K_M	K_D	R (Mbit/s)	S (ps^{-1})	$R \cdot S$	σ_{jit}	$max(\Delta T_{min})$
Max. R	125	(51,4,4) (11,1,3)	398.438 458.333	176	153	2.60417	0.07013	0.18263	76.706	3.56506
Max. S	125	(51,4,4) (32,3,3)	398.438 444.444	512	459	0.86806	0.204	0.177084	100.882	1.22549
Max. $R \cdot S$	125	(37,5,2) (32,3,3)	462.5 444.444	320	333	1.38889	0.148	0.20556	100.882	1.68919

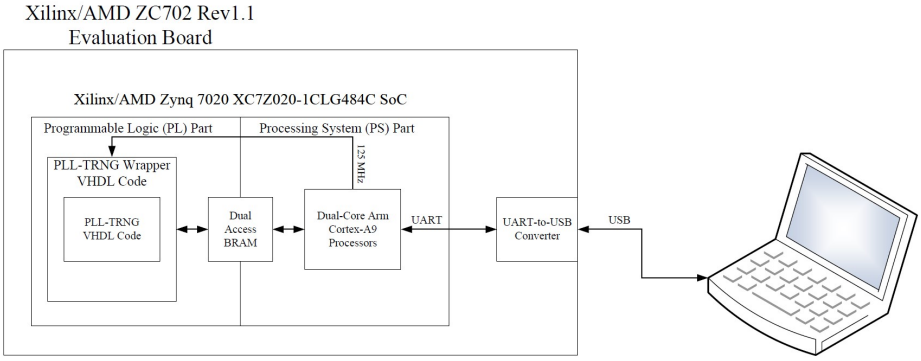

Fig. 5. Block Diagram of Implementation Setup

Zynq-7020 XC7Z020-1CLG484C SoC to host four PLL-TRNG variants detailed in Section 3.1). Within the programmable logic (PL), these four architectures, defined in Table 2, were coded in VHDL using Vivado 2019.1[15]. To stream random bits in real time, we employ a dual-port block random access memory (BRAM), with one port interfacing to the PL and the other to the processing system (PS). The PS runs C software that polls a BRAM flag set by the PL upon TRNG completion; it then retrieves the random data and transmits it via the asynchronous receiver/transmitter (UART) peripheral over USB to a host personal computer (PC). The PC logs the incoming bits in ASCII-encoded hexadecimal form, which are later converted offline to binary for AIS-20/31 statistical testing (Procedures A and B)[19], requiring ∼7 Mb per test. Hence, ∼7.2 Mb of random bits are collected. Finally, a 125 MHz reference clock (clk_{in}) is chosen to satisfy the timing constraints of the Zynq-7020's clocking network and global buffer (BUFG) limits.

To verify that our PLL-TRNG implementations are not bound to a single board, we deployed the four configurations across three separate ZC702 Rev 1.1 evaluation boards. Using the backtracking algorithm combined with the jitter constraint from Equation (8), we selected parameter sets optimized for maximal S, R, and $R \cdot S$. For each of the four configurations detailed in 3.1, we generated five separate random-bit output files on each board, captured the data on a host PC, and subjected it to AIS-20/31 testing. The detailed test results are presented in Section 4.

4 Results and Comparisons

The measured implementation outcomes are compiled in Table 4, where each entry corresponds to a specific PLL-TRNG setup defined by how many PLLs are used and the chosen parameter set. Four distinct PLL-TRNG architectures combined with three parameter choices yield twelve unique configurations, hence twelve table rows. For every configuration on each of the three ZC702 boards, we generate five random-bit output files of approximately 7.2 Mb each. The Shannon entropy value, each of them is calculated using (7), reported for each row is the arithmetic mean of those fifteen entropy measurements. Moreover, it should be noted that for all four PLL-TRNG designs, every generated file from each configuration successfully passed both Procedure A and Procedure B of the AIS-20/31 test suite [19] on all three boards.

Table 4. PLL-TRNG Implementation Results

PLL Configuration	Parameter Configuration	R (Mbit/s)	Output Bit Rate (Mbit/s)	S (ps^{-1})	$R \cdot S$	Entropy (Shannon)
2-PLL with one reference clock and one jittered clock	Max. R	2.6042	2.60417	0.0701	0.18263	**0.999999986833568**
	Max. S	0.8681	0.86806	0.204	0.17708	0.999999986516413
	Max. $R \cdot S$	1.3889	1.38889	0.148	0.20556	0.999999981069240
3-PLL with one reference clock and two jittered clocks	Max. R	2.6042	5.20834	0.0701	0.18263	0.999999976364641
	Max. S	0.8681	1.73612	0.204	0.17708	0.999999977771549
	Max. $R \cdot S$	1.3889	2.77778	0.148	0.20556	0.999999980834110
3-PLL with two reference clocks and one jittered clock	Max. R	2.6042	5.20834	0.0701	0.18263	0.999999985962200
	Max. S	0.8681	1.73612	0.204	0.17708	0.999999961780714
	Max. $R \cdot S$	1.3889	2.77778	0.148	0.20556	0.999999966076834
4-PLL with two reference clocks and two jittered clocks	Max. R	2.6042	**10.41668**	0.0701	0.18263	0.999999972332402
	Max. S	0.8681	3.47224	0.204	0.17708	0.999999971434251
	Max. $R \cdot S$	1.3889	5.55556	0.148	0.20556	0.999999956486246

Table 5 provides a comparative analysis of our work with previously implemented PLL-TRNGs. The results of our 4-PLL-TRNG implementation significantly enhance the output bit rate of the PLL-TRNG design while maintaining robust cryptographic properties. Specifically, the table shows that a speed of approximately 10.4 Mb/s can be achieved, notably higher than any other reported PLL-TRNG implementation. Additionally, our results exhibit superior Shannon Entropy compared to earlier PLL-TRNG designs.

Table 5 summarizes how our implementation compares with prior PLL-TRNG designs. The data demonstrate that our four-PLL configuration achieves a markedly higher output throughput, which is approximately 10.4 Mb/s, while preserving strong cryptographic characteristics. This performance surpasses all previously reported PLL-TRNG implementations. Moreover, our design attains superior Shannon entropy values relative to earlier PLL-based generators.

In Table 6, the utilization rates of different resource types for 4-PLL-TRNG are shown. Despite the inclusion of both the 4-PLL-TRNG and the necessary state machines in the PL section, the overall resource utilization remains relatively low, with the exception of the PLLs. These low utilization rates underscore the 4-PLL-TRNG's potential for TRNG applications in IoT systems.

Table 5. PLL-TRNG Implementation Results Comparison with [10], [7], and [20]

	Results of 4-PLL-TRNG			Results in [10] for Xilinx Spartan-6		
Parameter Configs	**Output Bit Rate (Mbit/s)**	**S (ps^{-1})**	**Entropy (Shannon)**	**Output Bit Rate (Mbit/s)**	**S (ps^{-1})**	**Entropy (Shannon)**
Max. R	10.41668	0.07013	0.999999972332402	1.042	0.094	1
Max. S	3.47224	0.204	0.999999971434251	0.521	0.167	0.99999
Max. R*S	5.55556	0.148	0.999999956486246	N/A	N/A	N/A
	Results in [7] for Xilinx Spartan-6			**Results in [20] for Xilinx Spartan-6**		
Parameter Configs	**Output Bit Rate (Mbit/s)**	**S (ps^{-1})**	**Entropy (Shannon)**	**Output Bit Rate (Mbit/s)**	**S (ps^{-1})**	**Entropy (Shannon)**
Max. R	0.555	0.0913	0.997	0.44	N/A	0.999931407560694
Max. S	0.555	0.0913	0.997	0.44	N/A	0.999931407560694
Max. R*S	0.555	0.0913	0.997	0.44	N/A	0.999931407560694

Table 6. Utilization Table Generated Using Vivado 2019.1 [15] for 4-PLL-TRNG Implementation

Resource Type	**Available Resource Quantity**	**Utilization Quantity (Utilization Rate as %) of Max. R Configuration**	**Utilization Quantity (Utilization Rate as %) of Max. S Configuration**	**Utilization Quantity (Utilization Rate as %) of Max. $R \cdot S$ Configuration**
LUT	53200	1539 (2.89%)	1542 (2.90%)	1536 (2.89%)
LUTRAM	17400	72 (0.41%)	72 (0.41%)	72 (0.41%)
FF	106400	1884 (1.77%)	1884 (1.77%)	1884 (1.77%)
BRAM	140	2 (1.43%)	2 (1.43%)	2 (1.43%)
IO	200	8 (4.00%)	8 (4.00%)	8 (4.00%)
PLL	4	4 (100.00%)	4 (100.00%)	4 (100.00%)

5 Conclusion

This study details the conception and realization of a novel, high-throughput PLL-based TRNG that leverages coherent sampling of jitter-induced phase deviations. A backtracking algorithm guides the selection of optimal PLL parameters, and a four-PLL arrangement, limited by the Zynq-7020 SoC's resources, yields a considerable improvement in bit-generation rate over traditional dual-PLL architectures. This approach is readily portable to any FPGA or SoC featuring three or more PLLs, broadening its applicability. Empirical evaluations confirm AIS-20/31 compliance, producing random sequences with both elevated output speeds and high Shannon entropy. The design's inherent scalability and flexibility position it as a strong candidate for integration into future FPGA, SoC, or ASIC platforms, particularly serving IoT applications where minimal resource consumption, rapid bit production, and robust randomness are paramount.

Acknowledgements. The authors acknowledge the support of Aselsan Inc. in the preparation of this work, as well as their provision of three Xilinx ZC702 Evaluation Boards used for the implementation of the PLL-TRNG algorithms presented in the paper.

References

1. 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, vol 2523. Springer, Berlin, Heidelberg (2002). https://doi.org/10.1007/3-540-36400-5_30
2. Drutarovský M., Simka M., Fischer V., Celle F.: A Simple PLL-Based True Random Number Generator for Embedded Digital Systems. In: Computing and Informatics, 23(5-6), 501–515 (2004). https://www.cai.sk/ojs/index.php/cai/article/view/442/351
3. Liu C., McNeill J.: A digital-PLL-based true random number generator. In: Research in Microelectronics and Electronics, 2005 PhD, Lausanne, Switzerland, pp. 113-116 vol.1 (2005). https://doi.org/10.1109/RME.2005.1543017
4. Fischer V., Drutarovský M., Simka M., Bochard N.: High Performance True Random Number Generator in Altera Stratix FPLDs. In: Field Programmable Logic and Application, J. Becker, M. Platzner, and S. Vernalde, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 555–564 (2004). https://doi.org/10.1007/978-3-540-30117-2_57
5. Bernard F., Fischer V., and Valtchanov B.: Mathematical model of physical RNGs based on coherent sampling. In: Tatra Mountains Mathematical Publications, vol. 45, no. 1, Slovak Academy of Sciences (2010). https://doi.org/10.2478/v10127-010-0001-1
6. Petura O., True random number generators for cryptography: Design, securing and evaluation. In: Micro and Nanotechnologies/Microelectronics. Université de Lyon. English. NNT: 2019LYSES053. tel-02895861 (2019). https://theses.hal.science/tel-02895861v1
7. Allini E. N., Characterization, evaluation and utilization of clock jitter as source of randomness in data security. In: Cryptography and Security [cs.CR]. Université de Lyon. English. NNT : 2020LYSES019. tel-03207261 (2020). https://ujm.hal.science/tel-02952931
8. Petura O., Mureddu U., Bochard N., Fischer V.: Optimization of the PLL-based TRNG design using the genetic algorithm. In: 2017 IEEE International Symposium on Circuits and Systems (ISCAS), 2017, pp. 1–4 (2017) https://doi.org/10.1109/ISCAS.2017.8050839
9. Allini E. N., Petura O., Fischer V., Bernard F.: Optimization of the PLL configuration in a PLL-based TRNG design. In: 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, France (2018). https://doi.org/10.23919/DATE.2018.8342209
10. Colombier B., Bochard N., Bernard F., Bossuet L.: Backtracking Search for Optimal Parameters of a PLL-based True Random Number Generator. In: Automation & Test in Europe Conference & Exhibition (DATE) (2020). https://hal.science/hal-02937862v1
11. Bundesamt für Sicherheit in der Informationstechnik (BSI), AIS 20/31 - Functionality Classes for Random Number Generators, version 2.0. Technical report (2011). https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_31_Functionality_classes_for_random_number_generators_e.html Accessed: 2023-11-01
12. Yayla O., Yılmaz Y.E.: Design and Implementation of a Fast, Platform-Adaptive, AIS-20/31 Compliant PLL-Based True Random Number Generator on a Zynq 7020 SoC FPGA. In: CISIS 2024. Lecture Notes in Networks and Systems, vol 957. Springer, Cham. (2024). https://doi.org/10.1007/978-3-031-75016-8_5
13. Yılmaz Y.E.: On an Efficient Implementation of Combined True Random Number Generator and Physically Unclonable Function on a SoC FPGA, Ph.D. Thesis, Middle East Technical University (2024). https://hdl.handle.net/11511/111066
14. AMD, Xilinx Zynq-7000 SoC ZC702 Evaluation Kit. https://www.xilinx.com/products/boards-and-kits/ek-z7-zc702-g.html Accessed: 2023-09-01
15. AMD, Xilinx Vivado 2019.1 Design Software for Xilinx (AMD) Adaptive SoCs and FPGAs. https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/archive.html Accessed: 2023-09-01
16. AMD, 7 Series FPGAs Clocking Resources User Guide (UG472) (v1.14) https://docs.amd.com/v/u/en-US/ug472_7Series_Clocking Accessed: 2023-10-01
17. AMD, Zynq-7000 SoC: DC and AC Switching Characteristics (DS187) (v1.21) https://docs.amd.com/v/u/en-US/ds187-XC7Z010-XC7Z020-Data-Sheet Accessed: 2023-10-01
18. Fischer, V., Bernard, F., Bochard, N.: Modern random number generator design – Case study on a secured PLL-based TRNG. In: it - Information Technology 61, no. 1 (2019): 3-13 (2019). https://doi.org/10.1515/itit-2018-0025
19. BSI. Implementation of Test Procedure A and Test Procedure B for Application Notes and Interpretation of the Scheme (AIS) 20/31 Standard. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_31_testsuit_zip.zip Accessed: 2023-12-01
20. Petura O., Mureddu U., Bochard N., Fischer V. and Bossuet L.: A survey of AIS-20/31 compliant TRNG cores suitable for FPGA devices. In: 2016 26th International Conference on Field Programmable Logic and Applications (FPL), Lausanne, Switzerland, 2016, pp. 1-10 (2016). https://doi.org/10.1109/FPL.2016.7577379