Design and Implementation of a Fast, Platform-Adaptive, AIS-20/31 Compliant PLL-Based True Random Number Generator on a Zynq 7020 SoC FPGA

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

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

Abstract. Phase-locked loops (PLLs) integrated within field-programmable gate arrays (FPGAs) or System-on-Chip FPGAs (SoCs) represent a promising approach for generating random numbers. Their widespread deployment, isolated functionality within these devices, and robust entropy, as demonstrated in prior studies, position PLL-based true random number generators (PLL-TRNGs) as highly viable solutions for this purpose. This study explicitly examines PLL-TRNG implementations using the ZC702 Rev1.1 evaluation board featuring the Zynq 7020 SoC from Xilinx, utilizing a configuration involving three such boards for experimental validation. Parameters governing the PLL-TRNG are optimized using a backtracking algorithm. Additionally, a novel methodology is proposed to enhance the rate of random data bit generation while preserving entropy characteristics. Performance metrics are rigorously evaluated against the criteria set by the German Federal Office for Information Security (BSI) AIS-20/31 Tests, accompanied by detailed descriptions of the implementation process.

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

1 Introduction

Random numbers are critical and also the most required elements in the majority of cryptographic systems. For these numbers, excellent statistical quality is required, and that is provided by true random number generators (TRNGs) that are based on some physical random phenomena to guarantee unpredictability. Different types of entropies, concerning the application and the chosen technology, can be harvested. In this research, the jitter is selected as the entropy source, and the jitter can be defined as the fluctuation of the period in the clock in the time domain.

In this study, Phase-Locked Loops (PLLs) are selected as the source of the entropy harvester. These PLLs, which vary in number and properties, can be

found in any field-programmable gate array (FPGA) or System-on-Chip FPGA (SoC). PLLs are feedback control systems that automatically adjust the phase of a locally generated signal to align with the phase of an input signal. Due to their internal structures and inherent noise, the clocks within PLLs exhibit jitter. This jitter serves as a source of randomness, forming the basis for PLL-based True Random Number Generators (PLL-TRNGs). The PLL-TRNG boasts a straightforward and comprehensive design, utilizing coherent sampling. Furthermore, PLLs have isolated locations in FPGAs and SoCs, and previous works about PLL-TRNG conclude that PLL-TRNG has good cryptographic properties. The PLL-TRNG design using one PLL is explained in [1], [2], and [3], while the design with two PLLs is explained in [4]. Moreover, in [5], the stochastic model of the PLL-TRNG is presented. In addition to these works, details of PLL-TRNG implementation are analyzed in [6] and [7].

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 it is advantageous to implement PLL-TRNG considering its high entropy and isolated locations of PLL, one of the main drawbacks of the PLL-TRNG is its relatively low random data output speed. A new PLL-TRNG method using four PLLs is proposed to overcome this disadvantage. For this purpose, we chose Xilinx Zynq 7020 SoC instead of an FPGA. Since they offer higher integration, lower power, smaller board sizes, and higher bandwidth communication between the processor and FPGA.

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 True Random Number Generator (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. Finally, a comparative analysis with existing PLL-TRNG implementations is performed.

Our work presents two primary contributions:

– 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.

– 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.

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 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 PLL

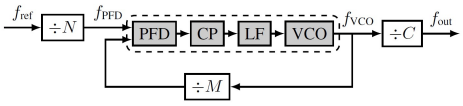

Fig. 1. Block diagram of a PLL (PFD: phase frequency detector, CP: charge pump, LF: loop filter, VCO: voltage-controlled oscillator) [10]

Table 1. Table of ranges of possible values for the PLL parameters and frequencies for Zynq-7000 SoC [15], [16]

Parameters	Xilinx Zynq-7000	
	Min	Max
f_{ref} (MHz)	19	800
P_{VCO}	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

A phase-locked loop (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).

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

2.2 Random Bit Generation Principle of the PLL-TRNG

The working principle of the PLL-TRNG with one PLL, and also two PLL versions of PLL-TRNG, is presented in Fig. 2.

The jittered clock signal clk_1 from the PLL is sampled by a D 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, some samples are consistently one (shown as blue in Fig. 2 and these are 4^{th} and 7^{th} dots), some are always zero (shown as green and these are 2^{nd} and 5^{th} dots), and others are random (shown as red and these are 1^{st}, 3^{rd}, 6^{th}, and 8^{th} 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 [1], [17]. This work adopts a two PLL-TRNG architecture as a

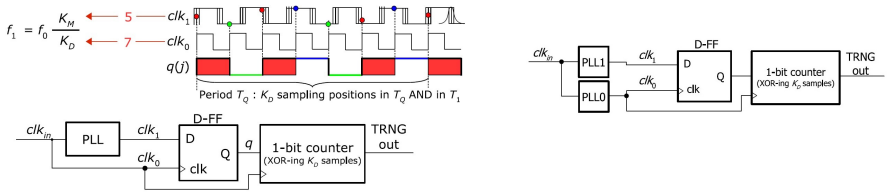

Fig. 2. [Left Figure]: Principle of the PLL-TRNG with one PLL [17]
[Right Figure]: PLL-TRNG with two PLLs Configuration [17]

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].

In this two PLLs 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 DFF in the left part of Fig. 2 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 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.

3 PLL-TRNG Implementation Details

3.1 Implemented PLL-TRNG Configurations

A primary limitation of PLL-TRNGs is their comparatively low output data rate. To address this constraint, this work proposes a methodology to enhance output capacity by leveraging additional PLLs available within the SoC. The Zynq 7020 SoC, featuring four PLLs, represents the upper bound for this implementation. However, prior to full-scale implementation, intermediate configurations employing three PLLs are investigated to facilitate a systematic design process. This study elucidates the design rationale for the four-PLL system by providing detailed explanations of these intermediate steps. Consequently, four distinct PLL-TRNG configurations are presented in Table 2 and visually depicted in Fig. 3:

Table 2. Configurations of PLL-TRNG Implementations

Codes of PLL-TRNG Designs Depicted in Fig. 3	Number of PLLs	Purpose of Use of PLL		PLL-TRNG Design Type
		As Reference Clock	As Jittered Clock	
(a)	2	1	1	Referenced Design
(b)	3	1	2	Intermediate Step for Proposed Approach
(c)	3	2	1	Intermediate Step for Proposed Approach
(d)	4	2	2	Proposed Design

3.2 Determining PLL-TRNG Parameters

In this work, as the parameter search algorithm, the backtracking algorithm in [10] is selected. Given a set of variables explained in Section 2.1 and Section 2.2 and constraints listed in Table 1, this backtracking method iteratively investigates potential solutions. Unlike a brute-force approach, it promptly eliminates any variable values that fail to meet a constraint, then backtracks to explore other possible values until all valid solutions are identified. The algorithm detailed in [10] involves determining the PLL-TRNG parameters that comply with both physical constraints and application requirements.

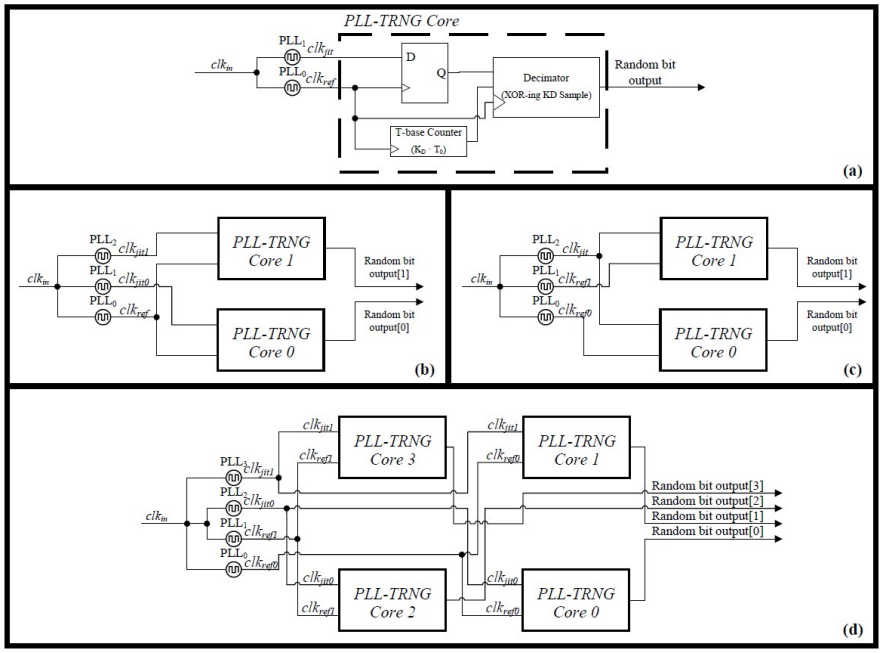

Fig. 3. Implemented PLL-TRNG Configurations: (a), (b), (c), and (d)

The code in the backtracking is open-source shared in [20]. Hence, we can modify it for Zynq 7020 SoC parameters provided in Fig. 1. Table 1 is generated with PLL properties of the SoC except for f_{out} which is determined concerning the maximum frequency value of BUFG clock buffer in Zynq 7000 Series [16]. BUFG must be used in the SoC design, and hence, it restricts f_{out} value for the search algorithm.

After generating results for our case, the results of the algorithm are ordered with respect to three different configurations. Those are the maximum bit rate (max. R), the maximum sensitivity to jitter (max. S), and the maximum $R \cdot S$ value as the optimization between max. S and max. R.

After obtaining the candidate results for three different configurations, those results must be tested with one more criterion. The sampling process of the jittered clock with the reference clock is illustrated in Fig. 2. In order to obtain random numbers at the output of this PLL-TRNG, at least one sample is required to be affected by the jitter. This necessitates that the distance between any edge of clk_0 and its corresponding edge on clk_1 must be less than Δ. This condition is met if the following condition holds [4], [7]:

$$\sigma_{jit} > max(\Delta T_{min}) \tag{7}$$

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]:

$$max(\Delta T_{min}) = \frac{T_{clk_0}}{4K_M} gcd(2K_M, K_D) = \frac{T_{clk_1}}{4K_D} gcd(2K_M, K_D) \tag{8}$$

where *gcd* is the greatest common divisor of two integers.

Upon executing the backtracking algorithm and obtaining results for the selected SoC, the maximum value of $max(\Delta T_{min})$ can be determined. However, accurately measuring or estimating σ_{jit} presents significant challenges. At this juncture, the estimation tool named *Clocking Wizard* in Vivado 2019.1 can be utilized. This tool provides an estimation of the jitter at the PLL's output clock, given the PLL parameters. Consequently, the results from the backtracking algorithm are first examined, and max. R, max. S and the max. $R \cdot S$ are identified. These three candidates are then evaluated against Equation (7). Candidates failing to satisfy the equation are discarded, and alternative candidates from the backtracking results are considered.

The results of the search algorithms are listed in Table 3. As it can be seen, all the selected configurations satisfy Equation (7).

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

3.3 Implementation Setup

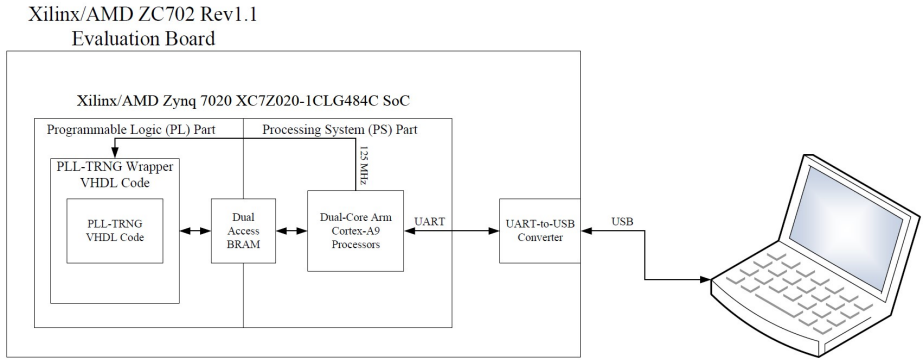

Fig. 4. Block Diagram of Implementation Setup

The implementation setup employed in this study is illustrated in Fig. 4. It utilizes the ZC702 Rev1.1 Evaluation Board [12], which incorporates the Zynq 7020 XC7Z020-1CLG484C SoC to facilitate the implementation of four distinct PLL-TRNG configurations as detailed in Section 3.1. In the Programmable Logic (PL) section, four distinct designs, specified in Table 2, are developed using Vivado 2019.1 [13] in VHDL [14]. To enable real-time transmission of generated random numbers to a personal computer (PC), a dual-access Block RAM (BRAM) is employed. One port of this BRAM is connected to the PL, while the other is connected to the processing system (PS) section. The requisite code for the PS section is written in the C programming language. The PL section

generates random numbers and writes a predefined value to a specific BRAM address to indicate that the random bits are ready. Once this indication is given, the software in the PS section outputs the random bits to the UART serial port, which are then converted to USB and transmitted to the PC. The received bits on the PC are saved in their ASCII-coded hexadecimal form and later converted to binary form offline to serve as input for AIS-20/31 Tests [18]. Both Procedure A and Procedure B Tests of AIS-20/31 are conducted for each result. Given that these tests require ∼7Mb of random bits, each output file is generated to have a size of ∼7.2Mb. Additionally, a 125 MHz clock frequency is selected for the system's main clock (clk_{in}) due to timing constraints inherent in the SoC.

In conclusion, three ZC702 Rev1.1 Evaluation Boards are employed to demonstrate that the implemented PLL-TRNG configurations are not specific to a particular device. The backtracking algorithm and the elimination criteria outlined in Equation (7) are used to determine the PLL-TRNG configuration parameters for maximizing S, R, and the product $R \cdot S$. Subsequently, five random output bit files are generated for each of the four configurations described in Section 3.1 and tested on the three evaluation boards. The results are stored on a PC, and AIS-20/31 Tests are conducted. The outcomes of these tests are detailed in Section 4.

4 Results and Comparisons

The implementation results are presented in Table 4. In this table, each row corresponds to a unique configuration defined by the number of PLLs in the PLL-TRNG and the parameter configuration. By considering our four different PLL-TRNG configurations and three different PLL parameter selections, we have twelve distinct rows, in other words, twelve different results. For each row, five different ∼7.2Mb random number files are generated for each of the ZC702 Boards. Hence, the arithmetic mean of these fifteen values is used for the Shannon Entropy calculation in the table.

Although the Shannon Entropies are provided by the test suite, it is not indicated in Table 4, but it must be emphasized that all four PLL-TRNG designs passed all the AIS-20/31 Tests for all three different configurations on three distinct boards for all generated files.

Table 5 provides a comparative analysis of our work with previously implemented PLL-TRNGs. The results indicate that our 4-PLL implementation significantly enhances 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, which is notably higher than any other reported PLL-TRNG implementation. Additionally, our results exhibit superior Shannon Entropy compared to earlier PLL-TRNG designs.

5 Conclusion

In this paper, we delineate the design and implementation procedures of an innovative and fast PLL-TRNG utilizing the coherent sampling method of jit-

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. PLL-TRNG Implementation Results Comparison with [10], [7], and [19]

	Results of 4-PLL-TRNG			Results in [10] for Xilinx Spartan-6			Results in [7] 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)	Output Bit Rate (Mbit/s)	S (ps^{-1})	Entropy (Shannon)
Max. R	10.41668	0.07013	0.999999972332402	1.042	0.094	1	0.555	0.0913	0.997
Max. S	3.47224	0.204	0.999999971434251	0.521	0.167	0.99999	0.555	0.0913	0.997
Max. $R \cdot S$	5.55556	0.148	0.999999956486246	N/A	N/A	N/A	0.555	0.0913	0.997
	Results of 4-PLL-TRNG			Results in [19] 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	0.44	N/A	0.999931407560694			
Max. S	3.47224	0.204	0.999999971434251	0.44	N/A	0.999931407560694			
Max. $R \cdot S$	5.55556	0.148	0.999999956486246	0.44	N/A	0.999931407560694			

tered PLL clocks. For parameter selection, we employ the backtracking algorithm [10]. Unlike conventional designs, which typically incorporate two PLLs, our approach leverages four PLLs to enhance the output bit rate. The choice of four PLLs is constrained by the Xilinx Zynq 7020 SoC, which accommodates exactly four PLLs. Nevertheless, the methodology illustrated in Fig. 3 is adaptable to any FPGA or SoC platform, provided the target device includes at least three PLLs. This flexibility ensures the broad applicability of our approach across various hardware configurations.

We show that our proposed methods can generate random numbers with AIS-20/31 compliance. With their excellent results compared to the previous works, it can be concluded that our proposed method is promising.

The proposed PLL-TRNG approach demonstrates scalability and adaptability, rendering it suitable for integration into future FPGA and SoC designs featuring enhanced performance characteristics. Moreover, the proposed design constitutes a viable candidate for implementation as a TRNG core within an application-specific integrated circuit (ASIC) architecture.

Acknowledgements. We thank the anonymous reviewers whose comments/ suggestions helped improve and clarify this paper. Also, the authors acknowledge Aselsan Inc. for its support during the preparation of this paper and for providing three Xilinx ZC702 Evaluation Boards that were utilized during the implementation of the PLL-TRNG algorithms described in the paper.

References

1. Fischer V., Drutarovský M.: True Random Number Generator Embedded in Reconfigurable Hardware. CHES 2002, (2002)
2. Drutarovský M., Simka M., Fischer V., Celle F.: A Simple PLL-based True Random Number Generator for Embedded Digital Systems. (2004)
3. Liu C., McNeill J.: A Digital-PLL-based True Random Number Generator. PhD Research in Microelectronics and Electronics, vol. 1, 2005, pp. 113–116, (2005)
4. Fischer V., Drutarovský M., Simka M., Bochard N.: High Performance True Random Number Generator in Altera Stratix FPLDs, (2004)
5. Bernard F., Fischer V., and Valtchanov B.: Mathematical Model of Physical RNGs Based on Coherent Sampling, (2010)
6. Petura O., True Random Number Generators for Cryptography: Design, Securing and Evaluation. Micro and Nanotechnologies/Microelectronics. Université de Lyon. English. NNT: 2019LYSES053. tel-02895861, (2019)
7. Allini E. N., Characterization, Evaluation and Utilization of Clock Jitter as Source of Randomness in Data Security. Cryptography and Security [cs.CR]. Université de Lyon. English. NNT : 2020LYSES019. tel-03207261, (2020)
8. Petura O., Mureddu U., Bochard N., Fischer V.: Optimization of the PLL-based TRNG Design Using the Genetic Algorithm, (2017)
9. Allini E. N., Petura O., Fischer V., Bernard F.: Optimization of the PLL Configuration in a PLL-based TRNG Design, (2018)
10. Colombier B., Bochard N., Bernard F., Bossuet L.: Backtracking Search for Optimal Parameters of a PLL-based True Random Number Generator, (2020)
11. B. für Sicherheit in der Informationstechnik (BSI), AIS 20/31 - Functionality Classes for Random Number Generators (2011) https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_31_Functionality_classes_for_random_number_generators_e.html
12. Xilinx Zynq-7000 SoC ZC702 Evaluation Kit. https://www.xilinx.com/products/boards-and-kits/ek-z7-zc702-g.html
13. Xilinx (AMD) 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
14. IEEE Standard VHDL Language Reference Manual, IEEE Computer Society, IEEE Std 1076-2008 (2008)
15. 7 Series FPGAs Clocking Resources User Guide (UG472) (v1.14). https://docs.amd.com/v/u/en-US/ug472_7Series_Clocking
16. Zynq-7000 SoC: DC and AC Switching Characteristics (DS187) (v1.21). https://docs.amd.com/v/u/en-US/ds187-XC7Z010-XC7Z020-Data-Sheet
17. Fischer, V., Bernard, F., Bochard, N.: Modern random number generator design – Case study on a secured PLL-based TRNG, (2019)
18. 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
19. Petura O., Mureddu U., Bochard N., Fischer V. and Bossuet L.: A survey of AIS-20/31 compliant TRNG cores suitable for FPGA devices, (2016)
20. The source code of the backtracking algorithm in [10] https://gitlab.univ-st-etienne.fr/sesam/pll-trng-constraint-programming/tree/master