Pipelined Pseudo-Random Number Generator with the Efficient Post-Processing Method

Paweł Dąbal

Abstract—**This brief proposes a novel architecture of the chaotic pseudo-random bit generators (PRBGs) based on the chaotic nonlinear model and pipelined data processing. We investigated PRBG built on the chaotic logistic map and frequency dependent negative resistances (FDNR). A significant enhancement in terms of output throughput has been achieved by combining the advantages of pipelining with post-processing based on fast logical operations like bit shifting and XOR. The proposed method has been implemented using programmable SoC Zynq device from Xilinx. We verified output pseudo-random bit stream by standard statistical tests NIST SP800-22. We also present detailed comparison of the proposed post-processing method with the methods reported previously by the other authors. In particular, we compared the maximum output throughput and amount of total logical resources required by PRBG implementation in the programmable SoC device. For PRBGs based on the logistic chaotic map and frequency dependent negative resistance (FDNR) we obtained speed-up factors equal to 33% and 14%, respectively. By composing the output stream of 3 data channels in PRBG with FDNR element, we get the maximum throughput equal to 38.43 Gbps. That is significantly greater comparing to the chaotic PRBGs described so far.**

Index Terms—**chaotic system, post-processing, pseudo-random number generation, system on chip**

I. Introduction

RANDOM number generator (RNG) and random bit generator (RBG) are widely used in many applications like computer simulation, cryptography, gaming and randomized design. Hardware random number generators based on physical microscopic phenomena such as thermal noise [1, 2, 3], time jitter in ring oscillators [4, 5, 6], flip-flop metastability [7, 8, 9] or quantum effects are called true random number generators (TRNGs) since these processes are, in theory, completely unpredictable. Due to the well-known disadvantages of TRNGs (relatively low output bit rate, complex design and vulnerability to external synchronization) very often true random sequences are replaced by pseudo-random numbers produced by pseudo-random number generators (PRNGs) and pseudo-random bit generators (PRBGs). An ideal PRBG should generate a non-periodic, unpredictable sequence of numbers that meets rigorous statistical requirements formulated by a specific group of users. Most common principles for generation of pseudo-random sequences use linear feedback shift registers (LFSR) [10] or modulo m operations (linear congruential generators – LCG) [11]. Recently, an increasing number of researchers explore chaotic mapping as an alternative source of pseudo-random binary sequences [12, 13, 14, 15, 16, 17].

The main limitation of PRBGs is finite length of generated sequences observed as a periodicity of output stream of numbers. Moreover, the period of output sequences produced by generators based on the LFSR or LCG principle can be calculated analytically. In a case of chaotic PRBGs such a calculation is usually impossible, because the solutions of chaotic equations are irrational numbers while PRBGs are implemented using a finite precision of arithmetic. Therefore, in practice, an analysis of the periodicity of chaotic PRBGs requires full search over all possible initial values [18].

The bit stream obtained from both TRBGs and PRBGs is usually biased, which means that some output symbols are predominating. A common approach to coping with bias, and in general to improving the other statistical properties of the output sequences, is an appropriate post-processing of data produced by the RNG [19, 20].

In this paper we present a novel post-processing technique developed for the use with pipelined chaotic pseudo-random bit generators presented previously in our earlier works [21, 22]. The project was focused on a careful and comprehensive optimization of the PRBG’s architecture to achieve the highest possible operating frequency and output throughput of the generated pseudo-random sequences. We get models of the pipelined PRBG that has been implemented in Zynq 7020 from Xilinx. We considered a number of variants of the PRBG using 16-, 32-, 48-, and 64-bit precision of arithmetic calculations. The models have been verified experimentally. The proposed post-processing method is focused on the maximization of the number of effective bits used for assembling the final PRBG’s output sequence. As a result, we get a significant enhancement of the PRBG performance in terms of its output throughput. All statistical test has been performed using the NIST SP800-22 battery of tests [23] and some dedicated software described in [24]. Experimental results have been confronted with some other post-processing methods applied earlier to the chaotic PRBGs by the other authors [25, 26].

In Sec. II we describe an operating principle and design of two chaotic PRBGs based on the logistic mapping and frequency dependent negative resistance. We also explain the proposed post-processing technique. Then, in Sec. III we present main practical issues related to the implementation of PRBGs in programmable device. Experimental results and

This work has been supported by the Military University of Technology, Warsaw, Poland, as a part of the projects RMN 751 and PBS 918.

The author is with the Faculty of Electronics, Military University of Technology, Warsaw, Poland (e-mail: pawel.dabal@wat.edu.pl).

discussion of the impact of the proposed PRBGs architecture on their statistical quality and output throughput are given in Sec. IV. Sec. V contains some final remarks and brief summary of the paper.

II. Chaos-Based PRBGs

Different chaotic systems have been tested by many authors for use for the generation of pseudo-random sequences: logistic map [27, 28, 29], Henon [30], Lorenz [31], and quasi-chaotic non-linear filters [32]. They differ from each other in terms of maximum output throughput, total amount of required logical resources, and computational complexity. Most of them use fixed-point arithmetic to simplify design, to reduce the demands on the required logical resources and due to the lack of dedicated FPGA blocks supporting floating-point arithmetic. Meanwhile, an actual precision of the arithmetic used for calculations of the chaotic attractors according to a given equation or a system of equations is a key factor for the statistical properties of pseudo-random sequence, including its maximum length (period).

In practice, even after the proper selection of this precision, some bit positions in generated binary words should be rejected to keep the resulting bit stream within the assumed limits of statistical quality. Such a rejection of some selected bit positions may be considered as a simple yet effective post-processing method. However, as a result the final output bit stream has significantly lower throughput. In our previous work we proposed a method to overcome this drawback by introducing a new pipelined architecture of the chaotic PRBG designed as a combinatorial logic block [33, 34]. However, to exploit the full benefit from this architecture the operating frequency should be much greater than the delay of a single pipeline stage. We solved this problem by initializing the *M*-stage pipeline by *M* different initial values. In this way, after rejection of some more significant bits from the output words, we get the overall speedup of the PRBG equal to *M*.

In the following text the symbol *pArith* denote precision of the arithmetic used in calculations. By *pWord* we will refer to the word length after rejection of *pDrop* most significant bits. The symbol Q_i will be used to denote an output word after post-processing and X_i will describe the result obtained in *i*-th iteration of the chaotic equation.

A. Chaotic PRBG with the logistic mapping

Logistic map [12] has been used in many pseudo-random number generators [21, 22, 25, 33]. It was proposed for the first time by P.F. Verhulst in 1845, and popularized by R. May, who proposed the use of its properties to generate a chaotic sequence of numbers [12].

$$x_{i+1} = r x_i \left(1 - x_i\right), \tag{1}$$

where $0 \le x_i \le 1$, $0 < r \le 4$, $n = 1, 2, 3 \ldots$.

The logistic map is perhaps the most frequently used model of the chaotic system due to its relatively low computational complexity (two multiplications and single subtraction per iteration). Depending on the value of the parameter *r*, the dynamics of the related sequence may change dramatically. When *r* is in the range $3.569945672 < r \le 4$, the numbers generated in successive iterations of the mapping become chaotic, and there is no constant pattern in the derived series. However, there are so called windows of stability within this range, when one can observe stable cycles. The last of these windows is located at the value approximately equal to 3.828427. Of course, the generated sequence is also affected by the choice of the initial value x_0. In a case when $r = 4$, the calculations are even simpler and require a single multiplication per iteration. However, to achieve reasonably good statistical properties of the output sequence, a sufficient precision of arithmetic (commonly 48- or 64-bit) should be used. This results in more complex multiplier and significant decrease of the operating frequency. The behavior of the logistic mapping with respect to the value of the parameter *r* is shown in Fig. 1 as a bifurcation diagram.

B. Chaotic PRBG with the FDNR oscillator

Chaotic oscillators described by 3-rd order differential equations, e.g. Chen [34], Lorentz [35], Sprott [36], and FDNR [37] are also used in chaotic PRBGs. Detailed analysis of oscillator with FDNR element may be found in [38]. It is described by the following differential equation:

$$-\dddot{X} = \ddot{X} + B\dot{X} + X \,, \tag{2}$$

where the nonlinear parameter B is defined as

$$B = \begin{cases} \beta_1, & f(X, \dot{X}) \ge 1 \\ \beta_2, & f(X, \dot{X}) < 1 \end{cases}. \tag{3}$$

To implement a chaotic generator based on differential equation in a digital system we can use well known numerical methods, e.g. Euler's method, fourth-order Runge-Kutta method or midpoint method. A comprehensive discussion of their application to chaotic generators is presented in [39]. According to the author, the recommended method is the Euler's method, because of its simplicity, easy of use and good performance.

Let $Y = \dot{X}$ and $Z = \ddot{X}$, then the numerical solution of equation (2) is evaluated as:

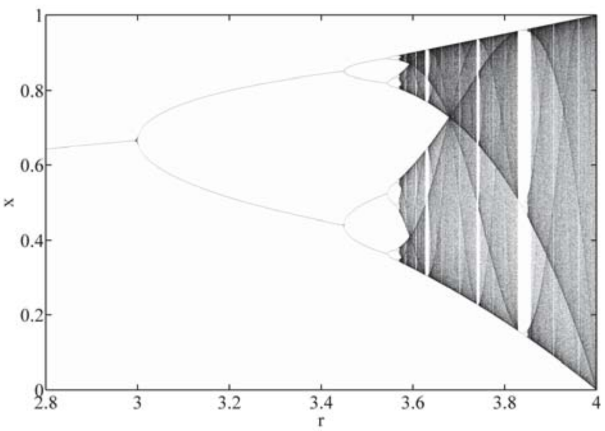

Fig. 1. Bifurcation diagram of the logistic mapping.

$$\begin{aligned} X_{t+h} &= X_t + hY_t \\ Y_{t+h} &= Y_t + hZ_t \\ Z_{t+h} &= Z_t - h\left(Z_t + BY_t + X_t\right) \end{aligned} , \quad (4)$$

where t denotes time and h is a step. An appropriate choice of the value of step h and parameters β allows easy implementation of eq. (4) with the use of adders, bit shifters and registers holding values of X, Y and Z. Fig. 2 shows the chaotic behavior of nonlinear system described by (4).

C. *Study on the known post-processing methods*

An interesting concept of the PRBG based on the logistic map and implemented in a programmable logic was presented in [25]. In order to enhance the output throughput, the authors proposed 2-stage pipeline and manually optimized multiplication which saves the number of DSP blocks. Each new output word is composed of three successive sub-words X_i, X_{i-1} and X_{i-2} obtained as 16 less significant bits and coupled together by mod 2 operation. The authors reported the maximum output throughput equal to 1.45 Gbps @ 93 MHz but did not provide any results of statistical tests.

In [26] the authors presented simple yet effective post-processing method for chaotic oscillator based PRBG implemented with the use of 32-bit arithmetic. The post-processing is based on the mod 2 operation for γ most significant bits shifted left by β positions. In a case of the PRBG with FDNR element and $\gamma = 32$ it leads to the simple relationship:

$$Q_{i+1} = X_i \oplus \left(Q_i << \beta\right). \quad (5)$$

The sequence uses all available bits without rejection of any positions and allows for very good efficiency of the PRBG reported as 14.07 Gbps @ 146.56 MHz. The output data stream passed statistical test NIST SP800-22.

D. *A new method of post-processing for pipelined PRBGs*

As one can see from results presented in [33] and [34], to fulfill the requirements of standard statistical NIST test, some more significant bit positions (including the sign bit) in output words generated by the chaotic PRBG should be rejected. Apparently, these bits contain much less entropy and change less frequently than LSB bits. According to our observations, the number of such a kind of statistically 'weak' bits depends on many factors. Of course, one of them is the assumed precision of arithmetic, in particular the number of bits representing the integer part and the sign. Furthermore, we observed that the number of 'weak' bits depends also on the particular chaotic system and even on the order of calculations and differentiation step. As a result, 12,5% to 50% of MSB bit positions should be rejected for successful validation of the output sequence by the NIST SP800-22 tests.

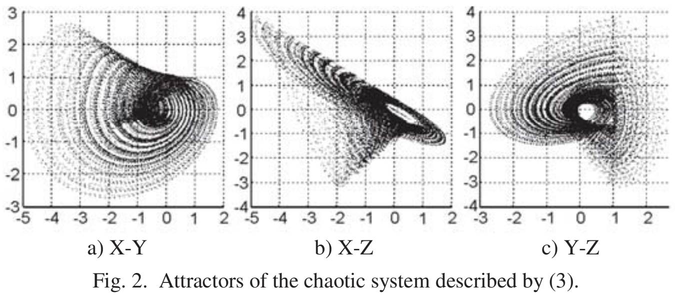

Fig. 2. Attractors of the chaotic system described by (3).

Taking into account all remarks mentioned above, we propose a new method of post-processing composed of XOR operation, bit shifting (rotation) and feedback loop. We apply this method to the pipelined pseudo-random generators and combine with some ideas described in [26]. Increasing the number of bits independently added with the use of mod 2 operation we get significantly better dispersion of high entropy contained in LSB bits over the MSB bit positions. As a result, the entropy in output words is distributed in a more uniform way and rejection of the selected bit positions is no longer necessary.

An original architecture of the pipelined PRBG presented in [33] and [34] features the output data stream composed of alternate values obtained for different initial values (seeds). Thanks to the principle of pipelined operation, each two successive words X_{i-1} and X_{i-2} are available in a single iteration at the same time and can be analyzed simultaneously. In a case of non-pipelined PRBG such an analysis would require additional registers for storing the values of X_{i-1} and X_{i-2}. An overlapping of 'weak' bits in these words can be easily obtained by rotating bits to the left by $RL = pAritch/4$ bit positions. Hence, the new output word Q_{i+1} is given by the relationship:

$$Q_{i+1} = X_i \oplus \left(X_{i-1} << RL\right) \oplus \left(X_{i-2} << 2RL\right) \oplus \left(Q_i << 3RL\right). \quad (6)$$

Thanks to the use of two successive internal values of X_{i-1} and X_{i-2} combined with the previous output value Q_i this method of post-processing allows for more uniform distribution of the entropy over the entire output word than the basic method reported in [26]. It should be noted however, that in a case when the number of 'weak' bits exceeds half of the word this effect of entropy dispersion may be not sufficiently effective. Such a situation can be detected by careful monitoring of statistical properties for particular binary subsequences isolated from the original output data stream.

III. Implementation

For each of two chaotic pseudo-random number generators described in [33] and [34] (based on the logistic map and FDNR element) we designed, implemented and experimentally verified three different configurations. The PRBG based on the logistic map will be denoted as *PrngLog*: *PrngLogClassic* is the basic (non-pipelined) architecture, *PrngLogDelayed* denotes pipelined architecture with a single initial value having the length of *pArith* bits, and *PrngLogPiplined* is pipelined architecture with M initial values, each of *pArith* length.

Similarly, the PRBG operating with the use of FDNR element will be denoted as *PrngOsc* and tested in three versions: *PrngOscClassic*, *PrngOscDelayed*, *PrngOscPiplined*. A new method of post-processing described in Sec. II D has been applied to the pipelined PRBGs with multiple initial values (*PrngLogPiplined* and *PrngOscPiplined*).

Operating models of all PRBG versions listed above have been developed using the MATLAB 2013a Simulink environment with System Generator tool from Xilinx. In this way we can easily get all necessary files with the relevant VHDL description of the analyzed PRBG architectures.

Figures 3 and 4 show block diagrams of two PRBGs based on the logistic map and FDNR element, respectively. Both models include appropriate blocks for data post-processing. To perform this post-processing we need *pArith* LUTs for XOR operations and the same number of flip-flops (FFs) for storing the output value Q_n. Since a single LUT can perform XOR operation on 6 bits simultaneously, we can optimize the number of used LUTs and use *pArith* LUTs to calculate XOR for many words. Successive values of X are easily available thanks to the use of pipelined architecture.

In a case of the *PrngOsc* each of three channels (X, Y, Z) has a separate post-processing block and additional block is used for composing the output Q having the length of 3×*pArith* bits. It should be noted that introduction of post-processing does not affect (decrease) the maximum operating frequency of the PRBG in any way. The maximum speed of the PRBG is still determined by the delay in the main feedback loop.

As the test-bed we used the evaluation board ZedBoard (Avnet) with SoC programmable device Zynq XC7Z020 from *Xilinx*. This device contains 28-nm programmable logic, an efficient dual core ARM Cortex-A9 processor, and versatile hardware controllers.

IV. Experimental Results

All statistical tests of the PRBG have been performed using the NIST SP800-22 package [23]. This suite is the most popular battery of statistical tests for evaluation of the quality of RBGs. It is composed of 15 tests. These tests focus on a variety of different types of non-randomness that could exist in a sequence of numbers. Each of tests is applied to the same sequence of n bits and gives $P_{\text{-}value}$, i.e. the probabilistic measure of the randomness of the sequence under test. If the significance level α is chosen to be 0.01 (common values of α in cryptography are about 0.01), then about 1% of the sequences are expected to be nonrandom. A sequence passes

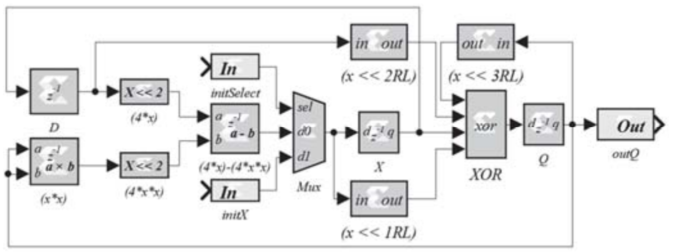

Fig. 3. Block diagram of the chaotic PRBG based on the logistic map with built-in post-processing.

a statistical test whenever the $P_{\text{-}values} \geq \alpha$, and fails otherwise. A more intensive test, involves a number m of different sequences generated by the PRBG under test. NIST suggests the following strategy: check if the $P_{\text{-}values}$ are uniformly distributed within the interval [0, 1] with a goodness of fit test ($P_{\text{-}valueT}$), then calculate proportion (*Prop*) of sequences passing the test and compare it to the expected value.

A random generator should produce all kinds of sequences, even bad ones (i.e., sequences not passing a statistical test). This approach was already proposed by NIST in its document ([23], Ch. 4). The distribution of $P_{\text{-}values}$ for m of binary sequences has been examined to check the uniform distribution of $P_{\text{-}values}$ for each test. The uniformity of the $P_{\text{-}values}$ has been examined via χ^2 test. After that, the determination of $P_{\text{-}values}$ corresponding to the goodness-of-fit distributional test on the $P_{\text{-}values}$ obtained for each statistical test was made. Then, for each individual test we can calculate $P_{\text{-}valueT}$:

$$P_{-valueT} = igamc\left({}^{9}\!/_{2}, {}^{\chi^2}\!/_{2} \right) \tag{7}$$

According to the NIST recommendation, if $P_{\text{-}valueT} \geq 0.0001$, then the sequences can be considered to be uniformly distributed. If all results are positive, we can say that at the assumed confidence level α the sequence is random. In our tests we assumed: $m = 128$, $n = 2^{20}$ and $\alpha = 0.01$.

Experimental tests have confirmed that the proposed post-processing method is effective. According to our previous analysis of the chaotic PRBG with FDNR element (*PrngOsc*) the sequence composed of X, Y and Z outputs did not pass statistical tests, although the separate data streams obtained at these outputs have passed the test [34]. In the current design of the PRBG with post-processing module the composed output stream successfully passes NIST tests. All tests have been repeated with positive results for over a dozen different initial conditions.

We also estimated the efficiency of PRBGs with implemented post-processing method in terms of maximum throughput. The fastest version of the PRBG (*PrngOscPiplined* – pipelined architecture based on the FDNR with post-processing) generates output data at the speed of 25.75, 30.94 and 38.43 Gbps for 32-, 48-, and 64-bit precision of arithmetic, respectively. As far as we know, this is the best result achieved so far for the considered class of PRBGs.

Tables I shows detailed comparison of tested PRBGs. It contains the total amount of required logic resources (LUTs, FFs and DSP blocks), precision of arithmetic (*pArith*), an effective length of output data word (*pWord*), and estimated throughput with and without post-processing. It may be seen that by applying the pipelined architecture we get speed-up factor of the PRBG from 3 to 5 times, depending on the PRBG version. Of course, it is achieved at a cost of proportional increase of required logic resources.

Experimental results from Tab. I are very competitive compared to known results obtained by the other authors. For the PRBG based on the logistic map we get the throughput equal to 14.56 Gbps @ 233 MHz, while in [25] the authors

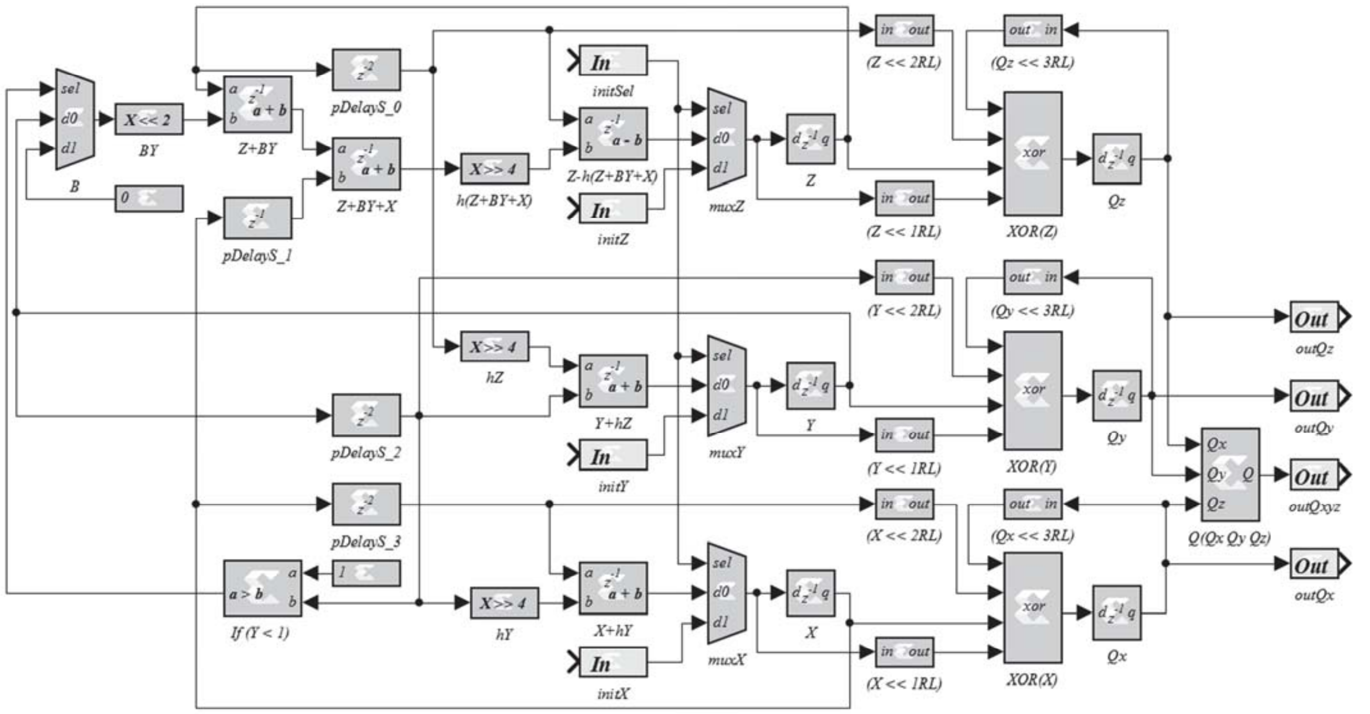

Fig. 4. Block diagram of the chaotic PRBG with FDNR element and built-in post-processing.

achieved 1.45 Gbps @ 93 MHz. Similarly, for the PRBG with FDNR element we get the speed of 25.75 Gbps @ 275 MHz, while the authors in [26] have reported 14.07 Gbps @ 146.56 MHz.

V. Conclusion

We proposed and experimentally verified a dedicated post-processing method for chaotic PRBGs with pipelined architecture. This method significantly improves the efficiency of the PRBGs because we do not need to reject bit positions having lower entropy. The method has been applied to two different PRBG models based on the logistic map and FDNR element. As a result we get significant improvement of the PRBG's throughput: 14.56 Gbps for the PRBG with logistic map (33% better than reported in [33]) and 12.81 Gbps for the PRBG with FDNR element (14% better than reported in [34]). Furthermore, by composing the output stream of 3 data channels in PRBG with FDNR element, we get the maximum throughput equal to 38.43 Gbps @ 205 MHz, which is the best result obtained so far for this class of chaotic PRBGs. Since the complete PRBG with post-processing module can be easily implemented in a single, low-cost programmable SoC device, it can be used in many applications, including complex microsystems for scientific purposes as well as commercial mobile digital systems with embedded gaming and/or communication features.

TABLE I.
Comparison of the required logic resources and throughputs of tested PRBGs

		PrngLogClassic		*PrngLogDelayed*		*PrngLogPiplined*		*PrngOscClassic*			*PrngOscDelayed*			*PrngOscPiplined*		
pArith [b]		**48**	**64**	**48**	**64**	**48**	**64**	**32**	**48**	**64**	**32**	**48**	**64**	**32**	**48**	**64**
pWord [b]		16	32	16	32	24/48[a]	48/64[a]	16	32	56	16	32	56	8/32[a]	40/48[a]	56/64[a]
Delay		1	1	8	13	1[b]	1[b]	1	1	1	4	4	4	1[b]	1[b]	1[b]
Resources	**LUT**	90	107	134	313	182	377	208	312	416	308	464	608	404	608	800
	FF	48	64	272	842	320	906	96	144	192	380	572	764	476	716	956
	DSP	9	16	9	16	9	16	0	0	0	0	0	0	0	0	0
f_{real} **[MHz]**		45	29	240	233	240	233	110	100	95	275	220	205	275	220	205
Speed before post-processing [Gbps]		0.703	0.906	0.469	0.561	5.625	10.922	1.719	3.125	5.195	1.074	1.719	2.803	2.148	8.594	11.211
Speed after post-processing [Gbps]		-	-	-	-	11.250	14.563	-	-	-	-	-	-	8.594 25.781[c]	10.313 30.938[c]	12.813 38.438[c]

[a] without post-processing / with post-processing
[b] effective delay between new values

REFERENCES

[1] C. S. Petrie, and A. J. Connelly, “A noise-based IC random number generator for applications in cryptography,” *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 47, no. 5, pp. 615-621, May 2000.

[2] T. W. Holman, A. J. Connelly, and A. B. Dowlatabadi, „An integrated analog/digital random noise source,” *Trans. Circuits Syst. I, Fundam. Theory Appl.*, pp. 521-528, 1997.

[3] M. Bucci, L. Germani, R. Luzzi, P. Tommasino, A. Trifiletti, and M. Varanonuovo, “A high-speed IC random-number source for SmartCard microcontrollers,” *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 50, no. 11, pp. 1373-1380, Nov. 2003.

[4] B. Sunar, W. J. Martin, and D. R. Stinson, “A Provably Secure True Random Number Generator with Built-In Tolerance to Active Attacks,” *IEEE Trans. Comput.*, vol. 56, no. 1, pp. 109-119, Jan. 2007.

[5] V. Fischer, F. Bernard, N. Bochard, and M. Varchola, „Enhancing security of ring oscillator-based TRNG implemented in FPGA,” in *Proc. Int. Conf. on Field Programmable Logic and Applications*, 2008.

[6] M. Jessa, and Ł. Matuszewski, “Producing Random Bits with Delay-Line-Based Ring Oscillators,” *Int. Journal of Electronics and Telecommunications*, vol. 59, no. 1, pp. 41-50, 2013.

[7] C. Tokunaga, D. Blaauw, and T. Mudge, “True Random Number Generator With a Metastability-Based Quality Control,” *IEEE J. Solid-State Circuits*, vol. 43, no. 1, pp. 78-85, Jan. 2008.

[8] S. Robson, B. Leung, and G. Gong, „Truly Random Number Generator Based on a Ring Oscillator Utilizing Last Passage Time,” *IEEE Trans. Circuits Syst II: Express Briefs*, vol. 61, no. 12, pp. 937-941, 2014.

[9] Z. Wieczorek, and K. Gołofit, “Dual-Metastability Time-Competitive True Random Number Generator,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 1, pp. 134-145, Jan. 2014.

[10] H. Rahimov, M. Babaei and M. Farhadi, “Cryptographic PRNG Based on Combination of LFSR and Chaotic Logistic Map,” *Applied Mathematics*, vol. 2, no. 12, pp. 1531-1534, 2011.

[11] K. Entacher, A. Uhl, S. Wegenkittl, “Linear congruential generators for parallel Monte Carlo: the Leap-Frog case,” *Monte Carlo Methods and Applications*, vol. 4, no. 1, pp. 1-16, 1998.

[12] R. M. May, “Simple mathematical models with very complicated dynamics,” *Nature*, vol. 261, pp. 459-467, Jun. 1976.

[13] M. Hénon, “A Two-Dimensional Mapping with a Strange Attractor”, *Commun. Math. Phys.*, vol. 50, no. 1, pp. 69-77, 1976.

[14] O. E. Rössler, “An Equation for Continuous Chaos”, *Phys. Lett. A.*, vol. 57, no. 5, pp. 397-398, Jul. 1976.

[15] A. S. Elwakil and M. P. Kennedy, “Chaotic oscillator configuration using a frequency dependent negative resistor,” in *Proc. Int. Symp. on Circuits and Systems ISCAS '99* , vol.5, 1999, pp. 399-402.

[16] C. Tanougast, “Hardware Implementation of Chaos Based Cipher: Design of Embedded Systems for Security Applications,” *Chaos-Based Cryptography - Studies in Computational Intelligence*, pp. 297-330, 2011.

[17] A. G. Radwan, A. S. Mansingka, M. A. Zidan, and K. N. Salama, "On the short-term predictability of fully digital chaotic oscillators for pseudo-random number generation," on *20th IEEE Int. Conf. on Electronics, Circuits, and Systems*, pp. 373-376, 2013.

[18] K. J. Persohn, and R. J. Povinelli, “Analyzing logistic map pseudorandom number generators for periodicity induced by finite precision floating-point representation,” *Chaos, Solitons & Fractals*, vol. 45, no. 3, pp. 238-245, 2012.

[19] J. von Neumann, “Various techniques used in connection with random digits,” *National Bureau of Standards Applied Math Series*, no 12, pp. 36-38, 1951.

[20] S.-H. Kwok, at al., “A Comparison of Post-Processing Techniques for Biased Random Number Generators,” in *Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication*, LNCS vol. 6633, pp. 175-190, 2011.

[21] P. Dabal, and R. Pelka, “A Chaos-Based Pseudo-Random Bit Generator Implemented in FPGA Device,” in *Proc. 14th IEEE Symp. Design and Diagnostics of Electronic Circuits and Systems*, Cottbus, pp. 151-154, 2011.

[22] P. Dabal, and R. Pelka, “FPGA Implementation of Chaotic Pseudo-Random Bit Generators,” in *Proc. 19th Int. Conf. Mixed Design of Integrated Circuits and Systems*, Warsaw, pp. 260-264, 2012.

[23] A. Rukhin, et al., “A statistical test suite for random and pseudorandom number generators for cryptographic applications,” *NIST Special publication 800-22, Revision 1a*, Aug. 2010.

[24] P. Dabal, and R. Pelka, “An integrated system for statistical testing of pseudo-random generators in FPGA devices,” in *Proc. Int. Conf. on Signals and Electronic Systems*, Wrocław, 2012.

[25] A. Pande, and J. Zambreno, “A chaotic encryption scheme for real-time embedded systems: design and implementation,” *Telecommunication Systems*, vol. 52, no. 2, pp. 551-561, 2013.

[26] M. L. Barakat, A. S. Mansingka, A. G. Radwan, and K. N. Salama, „Generalized Hardware Post-processing Technique for Chaos-Based Pseudorandom Number Generators,” *ETRI Journal*, vol. 35, no. 3, pp. 448-458, 2013.

[27] A. P. Kurian, and S. Puthusserypady, “Self-synchronizing chaotic stream ciphers,” Signal Processing, vol. 88, issue 10, pp. 2442-2452, 2008.

[28] S. Liu, J. Sun, Z. Xu and Z. Cai, “An improved chaos-based stream cipher algorithm and its VLSI implementation”, in *Proc. Int. Conf. on Networked Computing and Advanced Information Management*, vol. 2, pp. 191-197, 2008.

[29] N.P. Sajeeth and K.J. Babu, “Chaos for stream cipher,” in Proc. Recent Adv. Computing Communications, ADCOM2000 New York: Tata McGraw-Hill, pp. 35-42, 2000.

[30] R. Forre, “The Henon attractor as a keystream generator,”. in *Advances in cryptology EUROCRYPT 91*. LNCS, Berlin: Springer, pp. 76-81, 1990.

[31] D. Frey, “Chaotic digital encoding: an approach to secure communication,” *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 40, no. 10, pp. 660-666, 1993.

[32] T. Habutsu, Y. Nishio, I. Sasase and Y. Nishio, “A secret key cryptosystem by iterating a chaotic map,” in *Advances in cryptology EUROCRYPT 91. LNCS 547*, pp. 127-140, 1991.

[33] P. Dabal, and R. Pelka, “A study on fast pipelined pseudo-random number generator based on chaotic logistic map,” on *17th Int. Symp. on Design and Diagnostics of Electronic Circuits and Systems*, Warsaw, pp. 195-205, 2014.

[34] P. Dabal, and R. Pelka, “Fast pipelined pseudo-random number generator in programmable SoC device,” on *Int. Conf. on Signals and Electronic Systems*, Poznań, 2014.

[35] G. R. Chen and J.H. Lu, “Dynamics of the Lorenz System Family: Analysis, Control and Synchronization,” Beijing: Sci. Press, 2003.

[36] E. Lorenz, “Deterministic Nonperiodic Flow,” *J. Atmospheric Sci.*, vol. 20, no. 2, pp. 130-141, 1963.

[37] J. C. Sprott, “Chaos and Time-Series Analysis,” Oxford, UK: Oxford University Press, 2003.

[38] A. S. Elwakil, M. P. Kennedy, “Chaotic oscillator configuration using a frequency dependent negative resistor,” on *Int. Symp. on Circuits and Systems*, vol.5, pp. 399-402, 1999

[39] M.A. Zidan, A.G. Radwan, and K.N. Salama, “The effect of numerical techniques on differential equation based chaotic generators,” in *Proc. Int. Conf. on Microelectronics* (ICM), pp. 1-4, 2011.

Paweł Dąbal received the M.Sc. degree in electronic engineering, in 2009, from the Military University of Technology, Warsaw, Poland, where he is currently working toward the Ph.D. degree in electronic engineering. His research interests concern the design of digital circuits and systems using FPGA programmable structures for random number generation use in cryptography.