Paweł DĄBAL, Ryszard PEŁKA
MILITARY UNIVERSITY OF TECHNOLOGY, FACULTY OF ELECTRONICS
2 Gen. Sylwestra Kaliskiego St., 00-908 Warszawa, Poland

Pipelined architecture of a chaotic pseudo-random number generator in a Cyclone V SoC device

Abstract

In this paper, we present a novel, optimized microarchitecture of a pseudo-random number generator (PRNG) based on the chaotic model with frequency dependent negative resistances (FDNR). The project was focused on optimization of the PRNG architecture to achieve the highest possible output throughput of the generated pseudo-random sequences. As a result we got a model of the pipelined PRNG that was implemented in Cyclone V SoC from Altera and verified experimentally. All versions of the PRNG were tested by standard statistical tests NIST SP800-22. In addition, we also provide a brief comparison with the PRNG implementation in SoC from Xilinx.

Keywords: chaotic system, random number generators, FPGA, SoC.

1. Introduction

Pseudo-random number generators (PRNGs) are widely used in many areas of modern science and technology, e.g. in communication systems for secure data transmission, simulation, diagnostics and testing. An ideal PRNG should generate a non-periodic, unpredictable sequence of numbers that meets rigorous statistical requirements formulated by a specific group of users. Limitations of the classic PRNG based on a shift register and feedback circuits cause an effort of many researchers to develop pseudo-random number generators based on chaotic deterministic relationships. Recently, an increasing number of papers have picked deterministic chaos as an alternative source of pseudo-random binary sequences. The example of mathematical models of deterministic chaos is an oscillator with a frequency dependent negative resistance (FDNR) element [1].

A common approach to the optimization of digital microsystems in programmable SoC devices is maximization of the system throughput (speed) versus the total amount of required logical resources and/or power consumption. In our case, we optimized the PRNG speed related to the number of required logical resources with a general requirement of fulfilling all rigors of the NIST SP-800-22 tests [2]. According to our earlier research on the chaotic PRNGs performance [3], the main limitation of the PRNG throughput is due to the relatively complicated arithmetic operations, and in particular due to multiplications of multi-bit words. An important advantage of the proposed architecture is the use of the FDNR element which allows us to substitute cumbersome multiplication by much faster and simpler bit shifting. In this case, the most important factor affecting the PRNG speed is the propagation time in carry-chain in adders. This limitation is of special importance for longer words, i.e. higher precision of arithmetic. Successive operating blocks were separated by registers. As a result we get a shorter propagation path and a higher operating frequency. Furthermore, we extended an initial word to the length that was sufficient for appropriate initial setting of all registers in the PRNG.

The project was focused on a careful and comprehensive optimization of the PRNG architecture to achieve the highest possible operating frequency and output throughput of the generated pseudo-random sequences. As a result we got a model of the pipelined PRNG that was implemented in Cyclone V SoC from Altera and verified experimentally. We considered a number of variants of the PRNG using 16-, 32-, 48-, and 64-bit precision of arithmetic calculations. All versions of the PRNG were tested by standard statistical tests NIST SP800-22.

The paper is organized as follows. In Section 2 we present an algorithm for pseudo-random bit generation with the use of FDNR element. Then, in Section 3 we discuss the most important implementation issues related to the Cyclone V platform. Experimental results, including throughput measures and results of the NIST statistical test are given in Section 4. Finally, Section 5 contains a brief summary and conclusions.

2. The PRNG based on a chaotic oscillator

The detailed analysis of the oscillator with FDNR element may be found in [1]. It is described by the following differential equation:

$$\dddot{X} = \ddot{X} + B\dot{X} + X, \tag{1}$$

where the nonlinear parameter B is defined as:

$$B = \begin{cases} \beta_1, & f(X,\dot{X}) \geq 1 \\ \beta_2, & f(X,\dot{X}) < 1 \end{cases}. \tag{2}$$

To implement a chaotic generator based on the differential equation in a digital system, we use well known numerical Euler's method. The Euler's method was chosen because of its simplicity, ease of use and good performance. By substituting: $Y = \dot{X}$ and $Z = \ddot{X}$ we get the solution of (1) in the form:

$$\begin{aligned} X_{t+h} &= X_t + hY_t \\ Y_{t+h} &= Y_t + hZ_t \\ Z_{t+h} &= Z_t - h(Z_t + BY_t + X_t) \end{aligned}, \tag{3}$$

where t denotes time and h is a step. An appropriate choice of the value of the step h and parameters β allows easy implementation of (3) with the use of adders, bit shifters and registers holding values of X, Y and Z.

3. Implementation of the PRNG

Direct implementation of (3) in programmable logic is troublesome, because we need four multipliers and some complex DSP blocks. By choosing an appropriate value of the step h and parameters β_1 and β_2 we can replace multiplications by much faster bit shifting. As a result we save a significant amount of FPGA resources and are able to apply a much higher clock frequency.

The significant improvement of the generator, comparing to the basic architecture presented in our earlier work [3], is the use of pipelined processing. It can be done by appropriate setting of the parameter *Number of pipeline stages* for each adder. In order to align the delays in channels calculating the values of X, Y and Z, four additional buffers *DelayX*, *DelayY* and *DelayZ* were included. The delays in the proposed PRNG model are defined by the following values: *pDelayH* – delay of the adding step h: 0 to 4 units; *pDelayS* – delay in the channel calculating the sum $(Z_t + BYt{+}Xt$ and in the additional buffers: 0 to 4 units. Figure 1 shows the block diagram of the PRNG based on the oscillator with FDNR.

Arithmetic calculations are performed using fixed point number representation and four different variants of precision (*pArith*): 16-, 32-, 48-, and 64-bit. The MSB holds the sign, the next 3 bits denote the integer part, and the remaining bits represent fractional part of the number. In our design we assumed $h = 2^{-4}$, $\beta_1 = 2^2$ and $\beta_2 = 0$. An actual value of B is determined by a comparator and a multiplexer. The multiplexer output is equal to Y if $Y \geq 1$, or is set to 0 when this condition is not fulfilled. Multiplications by constants $\beta_1 = 2^2$ and $h = 2^{-4}$ are executed as bit shifting by two positions to the left or four positions to the right, respectively. The generated output sequence depends on the initial values X_0, Y_0 and Z_0 from the input ports (*initX*, *initY*, *initZ*) and the multiplexers (*muxX*, *muxY*, *muxZ*). An acquisition of the PRNG output data is managed by a dedicated interface described in [5].

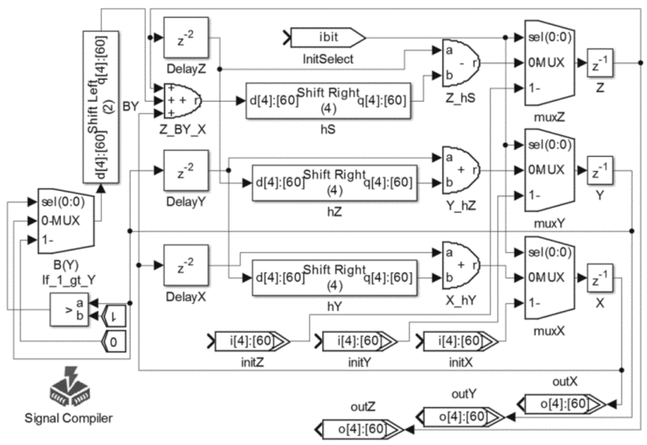

Fig. 1. Block diagram of the PRNG based on a chaotic oscillator

In order to find the best configuration (i.e. maximum operating frequency, number of used resources) of the PRNG, we checked 100 versions for different combinations of *pArith*, *pDelayH* and *pDelayS* parameters (4 × 5 × 5 values). For all variants of parameters (*pArith*, *pDelayH* and *pDelayS*) we performed the logic synthesis and time simulation to find the best solutions. Table 1 shows a part of the results of logic synthesis of the tested PRNG variants for selected arithmetic precisions and delays in each stage. The delay of adders is realized by a serial connection of flip-flops. Adders, multiplexers, and comparator were created using LUT logic blocks. By replacing multiplication with bit-shifting operation which does not require any additional logic resources, we get substantial savings in comparison to solutions, where the parameters of the generator are not a natural power of 2 [1].

Tab. 1. Required logic resources (Altera Cyclon 5CSXFC6D6F31)

pDelayH	pDelayS	FF				LUT			
		16b	*32b*	*48b*	*64b*	*16b*	*32b*	*48b*	*64b*
0	0	48	96	144	192	95	191	288	394
	1	172	348	524	700	95	191	288	394
	2	298	602	906	1210	127	255	383	517
	3	426	858	1290	1722	161	323	483	649
	4	556	1116	1676	2236	203	395	587	785
1	0	96	192	288	384	95	191	288	394
	1	220	444	668	892	95	191	288	394
	2	346	698	1050	1402	127	255	383	517
	3	474	954	1434	1914	161	323	483	649
	4	604	1212	1820	2428	203	395	587	785
2	0	148	292	436	580	151	295	440	588
	1	272	544	816	1088	151	295	440	588
	2	398	798	1198	1598	183	359	535	711
	3	526	1054	1582	2110	217	427	635	843
	4	656	1312	1968	2624	259	499	739	979
3	0	203	395	587	779	214	406	599	795
	1	327	647	967	1287	214	406	599	795
	2	453	901	1349	1797	246	470	694	918
	3	581	1157	1733	2309	280	538	794	1050
	4	711	1415	2119	2823	322	610	898	1186

The introduction of delays leads to 16.02 and 4.13-fold increase in the number of the used flip-flops and LUT blocks, respectively. However, this cost is marginally low in relation to the overall available logic resources (166 036 FFs, 83 018 ALUTs).

Table 2 shows a part of the results of time analysis of the tested PRNG variants for selected arithmetic precisions and delays in each stage. The PRNG without pipelining can operate at the maximum frequency f_{sim} within the range from 68 to 118 MHz, depending on the arithmetic precision. Setting the *pDelayH* = 1 at the output of the adder increases the maximum operating frequency. An increase in the operating frequency is equal to 9, 2, 5 and 6%, for precisions of 16-, 32-, 48- and 64-bit, respectively. The delays introduced during calculation of the sum $(Z_t + BY_t + X_t)$ by *pDelayS* = 1 connected to the outputs of adders have significantly greater contribution to the improvement of the operating frequency. The resulting improvement is up to 101, 88, 70, and 62% for 16-, 32-, 48- and 64-bit precision, respectively, at *pDelayH* = 0. In this case we obtain much greater speedup due to the fact that calculation of the Z value consists of 4 steps, while X and Y are calculated in a single step. The use of both *pDelayS* and *pDelayH* at the same time results in the speedup of around 132, 127, 135 and 217% for 16-, 32-, 48- and 64-bit precisions, respectively.

Tab. 2. Operating frequency (simulation results)

pDelayH	pDelayS	f_{sim}, MHz			
		pArith = 16b	*pArith = 32b*	*pArith = 48b*	*pArith = 64b*
0	0	118.16	108.84	97.59	68.24
	1	237.76	109.78	165.81	110.52
	2	246.73	210.04	178.13	130.62
	3	236.02	218.72	188.32	124.53
	4	269.32	216.08	193.69	216.45
1	0	129.08	110.91	101.99	72.23
	1	252.27	199.16	189.04	125.28
	2	265.18	229.52	178.13	143.27
	3	271.74	230.41	210.13	141.56
	4	258.26	242.07	211.46	216.45
2	0	122.96	110.66	101.99	78.81
	1	238.15	212.99	184.13	129.74
	2	267.24	229.46	206.02	190.01
	3	273.67	232.23	214.82	211.69
	4	258.26	233.48	222.92	200.32
3	0	129.37	106.93	102.03	77.93
	1	246.85	205.63	187.76	118.98
	2	265.18	229.57	215.98	189.18
	3	269.32	232.23	218.15	211.69
	4	249.69	246.85	229.25	210.79
4	0	123.29	109.78	103.48	80.95
	1	236.97	215.38	181.85	127.99
	2	267.38	231.11	211.37	204.54
	3	269.32	233.81	218.15	216.45
	4	253.04	245.52	227.01	214.22

4. Experimental results

All statistical tests of the PRNG used the NIST SP800-22 package. In our tests we assumed: number of sequences $m = 128$, sequence length $n = 2^{20}$ and confidence level $\alpha = 0.01$. Based on the knowledge obtained during our earlier research [3, 4] we know that for 16-bit precision all sequences failed the test. For other precisions we found that the 15th and lower bit positions successfully passed verification. Therefore, the final output stream of binary words having the length of *pWord* was created by rejection of a number of more significant bits.

According to the principle of pipeline, the latency, i.e. the time needed for calculation of a single result, is longer compared to non-pipelined architecture. However, the total throughput can be much higher, provided that the pipeline is started at each successive clock period with a new initial value. Instead of a single initial value, complex initial sequences were applied, composed of four values for each *X*, *Y*, and *Z* channel. It should be noted, that for 64-bit arithmetic all the recorded sequences passed the test at any output word length (*pWord*) in the range from 16 to 56 bits. The higher the precision of computing, the longer the usable part of the output word obtained by rejection of some more significant bits. By introducing a separate initial values for each successive pipeline cycle we get a significant speedup. In Table 3 we compare the obtained results with the data presented in [4]. Implementation performed at the Cyclone V is comparable to the performance level of the solution in a Zynq. However, the presented solution requires more flip-flops needed to implement a longer *pDelayS* path.

Tab. 3. Comparison with the previously reported generators

Altera Cyclon V						
***pArith*, b**	***pDelayH***	***pDelayS***	**f_{sim}, MHz**	**f_{real}, MHz**	***FF***	***LUT***
16	1	3	271.74	265.00	474	161
32	1	3	230.41	225.00	954	323
48	3	4	218.15	215.00	1733	794
64	1	4	216.65	205.00	2428	785
Xilinx Zynq 7020 [4]						
***pArith*, b**	***pDelayH***	***pDelayS***	**f_{sim}, MHz**	**f_{real}, MHz**	***FF***	***LUT***
16	1	1	327.90	320.00	188	152
32	1	1	281.10	275.00	380	308
48	1	1	225.30	220.00	572	464
64	1	1	216.50	205.00	764	608

Using 64-bit arithmetic, pipelining, multiple initial values and post-processing based on rejection of some more significant bits we get 3 output data streams with 11.48 Gbps throughput each (i.e. the total throughput is 34.4 Gbps) when *pDelayS* = 4 and *pDelayH* = 0 and 1. This result is competitive to approach based on a number of nonlinear equations described in [6], which in the fastest case provides 15.44 Gbps throughput.

5. Conclusions

The use of the proposed architecture significantly improves the speed of the PRNG implemented in a programmable SoC device. Chaotic mapping with the FDNR element can be effectively applied to generate pseudo-random number sequences that successfully pass the NIST 800-22 statistical test. The PRNG was implemented in the Cyclone V SoC from Altera and carefully tested in terms of its output throughput. We also present the numbers of the total required logical resources for different arithmetic precisions. In addition, we also provide a brief comparison with the PRNG implementation in SoC from Xilinx. The design based on SoC technology is particularly suitable for digital mobile systems.

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

6. References

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

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

[3] Dabal P., Pelka R.: FPGA Implementation of Chaotic Pseudo-Random Bit Generators. 19th International Conference Mixed Design of Integrated Circuits and Systems. Warsaw, 2012.

[4] Dabal P., Pelka R.: Fast pipelined pseudo-random number generator in programmable SoC device. International Conference on Signals and Electronic Systems, Poznań, 2014.

[5] Dabal P., Pelka R.: An integrated system for statistical testing of pseudo-random generators in FPGA devices. International Conference on Signals and Electronic Systems, Wroclaw, 2012.

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

Received: 21.04.2015 *Paper reviewed* *Accepted: 02.06.2015*

Paweł DĄBAL, MSc, eng.

He 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.

e-mail: pdabal@wat.edu.pl

Prof. Ryszard PEŁKA

He received the M.Sc. degree in electronic engineering from Warsaw Technical University, Warsaw, Poland, in 1979; and the Ph.D. and habilitation degrees in applied sciences from the Military University of Technology, Warsaw, Poland, in 1984 and 1997, respectively. Since 2004, he has been Professor of Electronics at the University. His current research interests are in methods of precise time-to-digital conversion, development and testing of digital systems with FPGA and SoC devices.

e-mail: rpelka@wat.edu.pl