Jyoti al. International Journal of Recent Research Aspects ISSN: 2349-7688, Vol. 4, Issue 2, June 2017, pp. 213-216

A Study of Hardware Implementation of Random Number Generators on FPGA

Jyoti

M.C. A. dept. M.D. University Rohtak

Abstract: **Random number generation refers to many applications such as simulation, numerical analysis, cryptography etc. Field Programmable Gate Array (FPGA) are reconfigurable hardware systems, which allow rapid prototyping. This research work is the first comprehensive survey on how random number generators are implemented on Field Programmable Gate Arrays (FPGAs). A rich and up-to-date list of generators specifi- cally mapped to FPGA are presented with deepned technical details on their definitions and implementations. A classification of these generators is presented, which encompasses linear and nonlinear (chaotic) pseudo and truly random number generators. A statistical comparison through standard batteries of tests, as well as implementation comparison based on speed and area performances, are finally presented.s.**

Keywords: Hardware Implementation of Random Number Generators on FPGA

I. INTRODUCTION

Randomness is a common word used in many applications [1] such as simulations [2], numerical analysis [3], computer pro- gramming, cryptography [4], decision making, sampling, etc. The general idea lying behind this generic word most of the times refers to sequences, distribution, or uniform outputs generated by a specific source of entropy. In other words, the probabilities to generate the same output are equal (50% to have ‘‘0’’ or ‘‘1’’). If we take the security aspect, many cryptosystem algorithms rely on the generation of random numbers. These random numbers can serve for instance to produce large prime numbers which are at the origin of cipher key construction [5] (for example, in RSA algorithm [6], in Memory Encryption [7] or Rabin signatures [8]). Furthermore, when the generators satisfy some very stringent properties of security, the generated numbers can act as stream cyphers in symmetric cryptosystems like the one-time pad, proven cryptographically secure under some assumptions [9]. Randomiza- tion techniques are especially critical since these keys are usually updated for each exchanged message. Even if an adversary has partial knowledge about the random generator, the behavior of this latter should remain unpredictable to preserve the overall security. From a historical point of view, numerical tables and physical devices have provided the first sources of randomness designed for scientific applications. On the one hand, random numbers were extracted from numerical tables like census reports [10], mathe- matical tables [11] (like logarithm or trigonometric tables, of inte- grals and of transcendental functions, etc.), telephone directories, and so on. On the other hand, random numbers were extracted also from some kind of mechanical or physical computation like the first machine of Kendall and Babington-Smith [12], Ferranti Mark 1 computer system [13] that uses the resistance noise as a physical entropy to implement the random number instruction in the accumulator, the RAND Corporation [14] machine based on an electronic roulette wheel, or ERNIE (Electronic Random Number Indicator Equipment [15]), which was a famous random number machine based on the noise of neon tubes and used in Monte Carlo simulations [16,17].

These techniques cannot satisfy today’s needs of randomness due to their mechanical structure, size limitation when tables are used [11], and memory space. Furthermore, it may be of im- portance to afford to reproduce exactly the same ‘‘random se- quence’’ given an initial condition (called a ‘‘seed’’), for instance in numerical simulations that must be reproducible — but physical generation of randomness presented above does not allow such a reproducibility. With the evolution of technologies leading to com- puter machines, researchers start searching for low cost, efficient, and possibly reproducible Random Number Generators (RNGs). This search historically began with John von Neumann, who presented a generation way based on some computer arithmetic op- erations. Neumann generated numbers by extracting the middle digits from the square of the previously generated number and by repeating this operation again and again. This method called mid-square is periodic and terminates in a very short cycle. There- fore, periodicity and deterministic outputs that use an operator or arithmetic functions are the main difference with the earlier generators. They are known in literature as ‘‘pseudorandom’’ or ‘‘quasirandom’’ number generators (PRNGs), while circuits that use a physical source to produce randomness are called ‘‘true’’ random number generators (TRNGs).

TheFPGA devices are reconfigurable hardware systems. They allow a rapid prototyping, i.e., explore a number of hardware solutions and select the best one in a shorter time. The design methodology on FPGA relies on the use of a High Description Language (i.e, Verilog, VHDL, or SystemC) and a synthesis tool. Because of this, FPGA has become popular platforms for implementing random generators or complete cryptographic schemes, due to the possibility to achieve high-speed and high-quality generation of random.

II. STATISTICAL TEST ANALYSIS

Statistical tests are used to evaluate whether the output of a given RNG can be separated from a real random sequence

obtained, for instance, by rolling a dice. Such tests are usually grouped in ‘‘Batteries’’, like the FIPS [132], DieHARD [25], NIST SP800 22 [133], TestU01 [26], or AIS [134] ones. In what follows, the content of these tests is recalled, for completeness purpose so as to make our article self-contained.

The National Institute of Standard and Technologies introduced their first test battery namely Federal Information Processing Stan- dard (FIPS) 140-1 [132] in 1994. These quick result tests have been further updated to the FIPS 140-2 [135] version, which covers more complex test batteries (focused for instance on security level).

Meanwhile, the DieHARD battery has been proposed by George Marsaglia [25]. It contains 18 tests of randomness. It was designed to provide a better way of analysis in comparison to the pre- viously released NIST tests. Unlike this latter, the p-values have now to belong to some fixed chosen interval α, 1 α , with a signification level of α for 5% for instance. An example of these batteries are: ‘‘Birthday spacings’’, ‘‘Overlapping permutations’’, ‘‘Ranks of matrices’’, ‘‘Monkey tests’’, ‘‘Count the 1′s’’, ‘‘Parking lot’’, ‘‘Minimum distance’’, ‘‘Random spheres’’, ‘‘The sqeeze test’’, ‘‘Overlapping sums’’, ‘‘Runs’’, and ‘‘The craps’’.

The AIS-31 battery [134] is a German standard to test and eval- uate the security properties of truly random number generators. It uses 9 statistical tests for the evaluation of a TRNG. AIS can be divided in two categories: the first one consists of T0-T4, which are the same function of FIPS 140-1 [132]. These later are mostly used to test the outputs of a post-processing. T0 is the ‘‘disjointedness test’’, which collects 65 536 of 48-bit and verifies that two adjacent values must not be equal. T1 is the monobit test, T2 is the poker test, T3 is the run test, and T4 is the longest run test. As for T5, it is part, is the auto-correlation test, and T6 is a ‘‘uniform distribution test’’ including of 2 sub-tests. T7 is a ‘‘comparative test for multinomial distributions’’, and finally T8 is an entropy test (Coron’s test).

In the other side, National Institute of Standard and Tech- nologies introduces a new test battery known as ‘‘NIST SP800 22’’ [133]. This one aims at testing the random profile of a given sequence using 15 tests. More precisely, it evaluates a long binary sequences generated by the RNG for the randomness and a higher security testing level than the FIPS 140-2. The tested sequences must have a fixed length N , where the parameter N is such that 103 < N < 107. Then, for each statistical test, a set of s sequences is produced by the RNG under test, and p-values are obtained. They all need to be larger than 0.0001 to reasonably consider the as- sociated sequences as uniformly distributed and cryptographically secure according to NIST standards.

The TestU01 battery is now the most complete and stringent battery of tests for RNG [26]. It was initially developed by ‘‘Pierre L’Ecuyer’’ and was implemented in the ANSI C language with more than 516 tests grouped inside 7 big sub-batteries. This new battery of tests covers various classical tests already present in other batteries with new algorithms for performance and cryptographic tests.

III. STATISTICAL RESULTS OF FPGA BASED RNG

In Tables 1 and 2, a number of generators are classified accord- ing to the battery test they have undergone. As it can be observed, the most stringent battery (Big crush) has only been applied twice in the literature, namely [107,122]. Let us notice that most (P)RNGs pass the Diehard and NIST batteries, while only a few PRNGs have

next have deeply investigated the non-linear ones, based on Blum– Blum–Shub or on chaotic maps. Then a large review of the true random number generators for FPGA has been proposed, encom- passing respectively the phase-locked loop, the ring oscillator, the self-timed ring, and the stability TRNG. For each type of RNG, a hardware analysis regarding area and throughput has been pro- vided. A section about statistical tests has finally been proposed, containing the detail of state-of-the-art batteries of tests, and the test results of some generators reviewed in this article against these batteries. [–]

It has been tested using the FIPS that has been integrated latter inside the NIST. Considering the TestU01 one, only crush batteries are usually considered. All generators fail at least one test, with the exception of chaotic iterations generators that can pass the whole battery.

Authors in [111,112] investigate the related problem for linear PRNGs. They show too that usual chaotic PRNGs are not passing the BigCrush when they consider its non linearity. However, being linear does not lead to a high linear complexity, which is defined by the degree of their polynomial characteristic function. How- ever, most random number generators are linear recursive, and so they fail in the so-called statistical Linear Complexity Test of TestU01 [26]. This test characterizes the (P)RNGs by their longest LFSR model: non randomness is claimed when the model is too short. This model is estimated by using the well-known Berlekamp– Massey algorithm [136]. It determines the shortest polynomial of a linearly recurrent finite output sequence in GF2. Note that all the other generators fail too the linear complexity test, except for PCG32 and MRG32K 3a: indeed, only PRNGs based on chaotic iterations are passing TestU01. Under this category, the authors propose too an extended internal space of 64 bits (CIPRNG-XOR) for 32 bits generators, when they increase the number of internal iterations to be uniformly distributed and to pass statistical tests.

Finally, TRNGs are hard to test with TestU01 (specially the BigCrush battery), as it needs 1038 random bits for a full test. Fig. 18 shows a general throughput of the order of kbps, which makes it difficult to collect the minimum amount of data needed in such tests. Under these conditions, only the TRNG of [122] based on ring oscillators has been proven to pass with success the BigCrush battery. Note finally that other batteries offer more flexibility and need a lower amount of bits for their embedded tests (namely, Diehard, NIST, and AIS), but they are less stringent and trustworthy than TestU01.

IV. EXPERIMENTAL RESULTS AND HARDWARE ANALYSIS

Methodology

Formally speaking, the space represents the allocation cost of most objects used in the algorithm (tables, indexes, loops, etc.). It can also be combination of many PRNG algorithms. In terms of FPGAs, the latter can be translated in memories, registers, and LUT resources, etc. These resources can be a single basic operation (like addition or subtraction, multiplication of variables or constants), algebraic functions (division, modulo, etc.), or any other elemen- tary function. The question raised in this section is thus: how much hardware resources are needed to provide pseudorandom numbers with a good statistical profile? And which algorithms outperform the other ones in terms of internal resources, while providing higher throughput?

Almost aforementioned (P)RNGs have been evaluated regarding their hardware performance according to three parameters: (1) the area, which is the result of (LUT FF) 8, (2) the throughput being the frequency (clock-to-setup) multiplied by the RNG output length for one clock cycle, and (3) the ratio between throughput over area in Mega bits per area unit.

Hardware implementation resources required by linear (P)RNGs, their throughput, and the rate area over throughput are presented in Fig. 16, when nonlinear ones are in Fig. 17. Finally, the TRNGs are represented in Fig. 18.

Let us start to discuss the results obtained with linear PRNGs, as illustrated in Fig. 16. It appears clearly that the cellular automata has the lowest area, when compared to the other approaches. Such results can be explained by the need of a low amount of resources to store both the states and the rules in the cellular automata. Conversely, the TGFSR family deploys BRAM block memories to read 3 word and write the output in one cycle, whereas LFSR family uses more LUTs in order to parallelize the shifting process based on the polynomial equation. Another parameter is the use of black box as DSP and block memories. The latter optimize the logic operation as multiplication, support the floating point, store internal process in a multidimensional bloc, and finally read and write multiple states in parallel from the BRAM. These advantages, leading to the difficulty to compare such designs to other ones that do not have that, lead naturally to further area bloc consumption in the case of an ASIC implementation. As a consequence, we will consider that (P)RNGs without black boxes are better and more recommended for cryptographic applications.

V. STATISTICAL TEST ANALYSIS

Statistical tests are used to evaluate whether the output of a given RNG can be separated from a real random sequence obtained, for instance, by rolling a dice. Such tests are usually grouped in ‘‘Batteries’’, like the FIPS [132], DieHARD [25], NIST SP800 22 [133], TestU01 [26], or AIS [134] ones. In what follows, the content of these tests is recalled, for completeness purpose so as to make our article self-contained.

The National Institute of Standard and Technologies introduced their first test battery namely Federal Information Processing Stan- dard (FIPS) 140-1 [132] in 1994. These quick result tests have been further updated to the FIPS 140-2 [135] version, which covers more complex test batteries (focused for instance on security level). Meanwhile, the DieHARD battery has been proposed by George Marsaglia [25]. It contains 18 tests of randomness. It was designed to provide a better way of analysis in comparison to the previously released NIST tests. Unlike this latter, the p-values have now to belong to some fixed chosen interval $\alpha, 1\ \alpha$, with a signification level of α for 5% for instance. An example of these batteries are: ‘‘Birthday spacings’’, ‘‘Overlapping permutations’’, ‘‘Ranks of matrices’’, ‘‘Monkey tests’’, ‘‘Count the 1′s’’, ‘‘Parking lot’’, ‘‘Minimum distance’’, ‘‘Random spheres’’, ‘‘The sqeeze test’’, ‘‘Overlapping sums’’, ‘‘Runs’’, and ‘‘The craps’’.

The AIS-31 battery [134] is a German standard to test and eval- uate the security properties of truly random number generators. It uses 9 statistical tests for the evaluation of a TRNG. AIS can be divided in two categories: the first one consists of T0-T4, which are the same function of FIPS 140-1 [132]. These later are mostly used to test the outputs of a post-processing. T0 is the ‘‘disjointedness test’’, which collects 65 536 of 48-bit and verifies that two adjacent values must not be equal. T1 is the monobit test, T2 is the poker test, T3 is the run test, and T4 is the longest run test. As for T5, it is part, is the auto-correlation test, and T6 is a ‘‘uniform distribution test’’ including of 2 sub-tests. T7 is a ‘‘comparative test for multinomial distributions’’, and finally T8 is an entropy test (Coron’s test).

In the other side, National Institute of Standard and Technologies introduces a new test battery known as ‘‘NIST SP800 22’’ [133]. This one aims at testing the random profile of a given sequence using 15 tests. More precisely, it evaluates a long binary sequences generated by the RNG for the randomness and a higher security testing level than the FIPS 140-2. The tested sequences must have a fixed length N , where the parameter N is such that $103 < N < 107$. Then, for each statistical test, a set of s sequences is produced by the RNG under test, and p-values are obtained. They all need to be larger than 0.0001 to reasonably consider the associated sequences as uniformly distributed and cryptographically secure according to NIST standards.

VI. CONCLUSION

We have provided a widespread coverage of the current research in hardware implementation of random number generators

[1]. M. Henson, S. Taylor, Memory encryption: a survey of existing techniques, ACM Comput. Surv. 46 (4) (2014) 53.

[2]. L. Lamport, Constructing digital signatures from a one-way function, Techni- cal Report CSL-98, SRI International Palo Alto, 1979.

[3]. C. Shannon, Communication theory of secrecy systems, Bell Syst. Tech. J. 28 (4) (1949) 656–715.

[4]. http://dx.doi.org/10.1002/j.1538-7305.1949.tb00928.x.

[5]. L. Tippett, Random sampling numbers. Arranged

by L.H.C. Tippett, Etc, [Tracts for Computers. no. 15.], 1927. http://books.google.dz/books?id= CZJTMwEACAAJ.
[6]. M. Campbell-Kelly, M. Croarken, R. Flood, E. Robson, The history of mathe- matical tables, AMC 10 (2005) 12.
[7]. M.G. Kendall, B.B. Smith, Randomness and random sampling numbers, J. Roy. Stat. Soc. (1938) 147–166.
[8]. S.H. Lavington, A History of Manchester Computers, NCC Publications, 1975.
[9]. G.W. Brown, History of Rand's Random Digits, Summary, DTIC Document, 1949.
[10]. W. Thomson, Ernie–A mathematical and statistical analysis, J. Roy. Statist. Soc. Ser. A (1959) 301–333.
[11]. N. Metropolis, The beginning of the Monte Carlo method, Los Alamos Science 15 (584) (1987) 125–130.
[12]. H. Niederreiter, N.-C. R. C. on Random Number Generation, in: Random Number Generation and Quasi-Monte Carlo Methods, vol. 63, SIAM, 1992.
[13]. P. L'Ecuyer, Uniform random number generation, Ann. Oper. Res. 53 (1) (1994) 77–120.
[14]. C.D. Motchenbacher, J.A. Connelly, Low-noise Electronic System Design, Wiley New York, 1993.
[15]. L. Kleeman, A. Cantoni, Metastable behavior in digital systems, IEEE Des. Test Comput. 4 (6) (1987) 4–19.
[16]. G.-C. Hsieh, J.C. Hung, Phase-locked loop techniques. A survey, Ind. Electron., IEEE Trans. 43 (6) (1996) 609–615.
[17]. R.H. Freeman, H.-C. Hsieh, Distributed memory architecture for a config- urable logic array and method for using distributed memory, Google Patents, US Patent 5,343,406, 1994.
[18]. P.M. Freidin, Logic block with look-up table for configuration and memory, Google Patents, US Patent 5,414,377, 1995.
[19]. E. Barker, A. Roginsky, DRAFT nIST Special Publication 800-131 Recommen- dation for the Transitioning of Cryptographic Algorithms and Key Sizes, 2010.
[20]. G. Marsaglia, (1995) The diehard test suite, 1995..
[21]. P. L'Ecuyer, R. Simard, TestU01: AC library for empirical testing of random number generators, ACM Trans. Math. Softw. 33 (4) (2007) 22.
[22]. P. L'Ecuyer, F. Panneton, Fast random number generators based on linear recurrences modulo 2: overview and comparison, in: Proceedings of the Winter Simulation Conference, 2005, 2005, p. 10. http://dx.doi.org/10.1109/ WSC.2005.1574244.
[23]. M. Matsumoto, T. Nishimura, Mersenne twister: a 623-dimensionally equidis- tributed uniform pseudo-random number generator, ACM Trans. Model. Comput. Simul. 8 (1) (1998) 3–30.
[24]. R.C. Tausworthe, Random numbers generated by linear recurrence modulo two, Math. Comp. 19 (90) (1965) 201–209.
[25]. D.E. Knuth, Deciphering a linear congruential encryption, IEEE Trans. Inform. Theory 31 (1) (1985) 49–52.

