Computer Science Review 27 (2018) 135–153

Contents lists available at ScienceDirect

Computer Science Review

journal homepage: www.elsevier.com/locate/cosrev

Survey on hardware implementation of random number generators on FPGA: Theory and experimental analyses

Mohammed Bakiri [a,b,*], Christophe Guyeux [a], Jean-François Couchot [a], Abdelkrim Kamel Oudjida [b]

[a] Femto-ST Institute, DISC Department, UMR 6174 CNRS, University of Bourgogne Franche-Comté, Belfort, 90010, France
[b] Centre de Développement des Technologies Avancées, ASM/DMN Department, Cité 20 août 1956 Baba Hassen, B. P 17, 16303, Alger, Algeria

ARTICLE INFO

Article history:
Received 13 December 2016
Received in revised form 7 October 2017
Accepted 18 January 2018

Keywords:
Random number generators
Field-Programmable Gate Array
Chaos
Physical security
Hardware Security
Applied cryptography

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 specifically mapped to FPGA are presented with deep 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.

Contents

* Corresponding author at: Femto-ST Institute, DISC Department, UMR 6174 CNRS, University of Bourgogne Franche-Comté, Belfort, 90010, France.

E-mail addresses: mbakiri@femto-st.fr (M. Bakiri), cguyeux@femto-st.fr (C. Guyeux), couchot@femto-st.fr (J.-F. Couchot), a_oudjida@cdta.dz (A.K. Oudjida).

https://doi.org/10.1016/j.cosrev.2018.01.002
1574-0137/

1. Background and motivation

Randomness is a common word used in many applications [1] such as simulations [2], numerical analysis [3], computer programming, 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]. Randomization 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], mathematical tables [11] (like logarithm or trigonometric tables, of integrals 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 importance to afford to reproduce exactly the same "random sequence" 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 computer 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 operations. 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. Therefore, 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*).

In most cases a random number generator algorithm can be defined by a tuple (S, f, g, U, x^0), in which S is the state space of the generator, U is the random output space, $f : S \rightarrow S$ is the transition mapping function, $g : S \rightarrow U$ is the output extractor function from a given state, and x^0 is the seed [18], see Fig. 1. The random output sequence is $y^1, y^2, \ldots$, where each $y^t \in U$ is generated by the two main steps described thereafter. The first step applies the transition function according to the recurrence $x^{t+1} = f(x^t)$, where f is an algorithm in the PRNG case and a physical phenomenon in the TRNG one. Then, the second step consists in applying the function generator to the new internal state leading to the output x^t, that is, $y^t = g(x^t)$. The period of a PRNG is the minimum number of iterations needed to obtain twice a given output (a PRNG being deterministic, it always finishes to enter into a cycle).

As stated previously, the old hardware manner to build such RNGs was to use a mechanical machine or a physical phenomenon as entropy source, which can thus be based on noise [19], metastability (frequency instability [20]), semiconductor commercial or industrial component circuit (PLL [21], amplifier, inverter, …), or a variation in the CMOS/MEMS process technologies (transistor). In spite of the quality of the generated randomness, most of these techniques are however, either slow processes (*i.e*, extracting noise from a component) or costly (*e.g.*, extracting or measuring noise may require specific equipment like an oscilloscope). All these previous drawbacks are the motivation behind the development of hardware generators based on a software design. The latter consist of developing deterministic algorithms by targeting a specific hardware system, like a Field Programmable Gate Array (FPGA), before automatically deploying it on the hardware architecture by using ad hoc tools.

FPGA 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. The general architecture of a FPGA presented in Fig. 2 is based on LCA (Logic Cell Array), which is composed of three parts, namely: *Configurable Logic Block* (CLB) [22], *Input Output Block* (IOB), and interconnect switches. FPGA could additionally include more complex components like a *Digital Signal Processing* (DSP), a *Random Access Memory* (RAM), a *Digital Clock Manager* (DCM), or an *Analog–Digital Converter* (ADC/DAC). The nomination of the internal blocks depends on the FPGA vendors (Xilinx, Altera, Actel …) even they have a similar functionality. The CLB structure is mainly based on Look-Up Tables (LUTs [23]), additionally with a Flip-Flop and some multiplexers. A K-input LUT is a $2^K \times 1$-bit memory array based on a truth table of K-bits inputs. These later can executes any logic functions as XOR/ADD/SHIFT…

Different implementations of RNG on FPGA have diverse characteristics. First of all, does it provide true random or pseudorandom numbers? In the second reproducible case, which algorithm is implemented? The next characteristic is the way each block is deployed on the FPGA, namely by computing or in a hardware manner. For instance, for a polynomial division, there is a choice between look-up table in software and a hardware shift register. Furthermore, the quality of the FPGA model that implements a random number generator can be evaluated according to many criteria. In a statistical perspective, the output has to be verified against some well-known test suite like the NIST [24], DieHARD [25], or TestU01 [26] ones. From the hardware perspective, one objective is to provide the highest frequency per randomly generated bit with less FPGA hardware resources (CLB, IOB, …).

This article surveys a large set of selected hardware implementations of random number generators on FPGA. Both pseudorandom and true random generators are investigated, while linear and non-linear generators are discussed in the PRNG case. Each approach is explained in details, and a discussion on the choices of both implementations and generations are systematically given. Performance with respect to frequency, area size, weaknesses,