Survey on Hardware Implementation of Random Number Generators on FPGA: Theories and Experiments Analysis

Mohammed Bakiri[1], Christophe Guyeux[2], Jean-Francois Chouchot[2], Abdelkrim Oudjida[1]

[1] Center for Development of Advanced Technologies, Algiers, Algeria
[2] FEMTO-ST Institute,UMR CNRS 6174, University of Franche-Comte, France
mbakiri@cdta.dz,Christophe.Guyeux@femto-st.fr

Abstract. this paper introduce a survey on PRNG on FPGA....

Keywords: Random Number Generator, PRNG, TRNG, Chaotic PRNG, Cryptography, Security, FPGA

1 Introduction

1

2 Random Number Generator: Theories and Classification

We can find many implementation of RNG in Software and Hardware but all can be classifies generally as PRNG, TRNG and Hybrid, but new concept has been introduce this last year's defined by parallel and chaotic generator.

2.1 Pseudorandom Galois Generators

F2-linear generators are a well-known PRNG class of and a special case of matrix linear recurrence modulo 2, where they are appropriate for low power and high speed requirement. However with the limitation of the shift register state, this generators enter to a finite and repeated state cycle and have a short period cycle. Because of that many hardware optimization are proposed to perform a good random and increase the period. A linear Random Number generator are defined following thous equations, where the first equation (1) $x_t = (x_{t,0}, ..., x_{t,k1})^s \in {F^k}_2$ is the k-bit state vector at step n. the second equation (2) $y_t = (y_{t,0}, ..., y_{t,w1})^s \in {F^k}_2$ is the w-bit output vector at step t, k and w are positive integers. Where, A is a $k * k$ transition matrix with elements in F_2, B is a $w * k$ output transformation matrix with elements in F_2, and $u_i \in [0, 1]$ is the output at step t:
$x_t = A * x_{t1}$ (1)

$y_t = B * xt$ (2)
$u_t = \sum_{\ell=l}^{w} y_{t,\ell-1}\ 2^{-\ell} = y_{t,0}\ y_{t,1}\ y_{t,2}$ (3)

The characteristic polynomial of the matrix A is $x_t = a_i xt - 1 + \quad + a_k x_{t-k}$

2.2 Discrete Dynamics

3 Pseaudo Random Number Generator

We illustrate in this part of the survey all theory and concept implementation in FPGA of the PRNG generators. An important property of all these generators is that they are special cases of a general class of generators whose state evolves according to a (matrix) linear recurrence modulo 2 and the bits that form the output are also determined by a linear transformation modulo 2 applied to the state. Except for the chaotic system where are dynamic and non linear PRNG.

Linear Feedback Shift Register Generators or LFSR use a feedback polynomial driven by an exclusive-OR (XOR) as a coefficients to shift bits of the register based on Flip-Flop. It uses an initial input called "seed" and a n-bits generation period of $2^n - 1$. However, even there are many FPGA implementation of LFSR can be founded, few of them has a optimize version on FPGA. As in this work [15], where the authors present two types of PRNG based LFSR. The first is called *Shrinking Generator* (SG), that uses two LFSRs of $67 - bit$. It consist for every clock cycle, the bit output generated will be the same of the second LFSR-2 if the 1-bits LSB of the first LFSR-1 is "1" or put "0". The second PRNG is called *alternating step generator* (ASG). It uses a third LFSR-3 of $141 - bits$ to control which bit output will be token from the others two LFSR of $131 and 137 bit$. However, for a small comparison purpose, the SG has a Total period of $L_T = \ (2^{L_2-1}) * 2^{L_1-1}$, where the ASG has a period of $L_T = \ 2^{L_1}(2^{L_2-1})(2^{L_3-1})$.

Linear Congruential Generators or LCG are another linear recurrence described by $x_{i+1} = (ax_i + b)\ mod\ 2^n$, and "a" is called the multiplier, "b" the increment, and "m" the modulus. A FPGA implementation of LCG can be demonstrated in [22], where the authors propose two version of PRNG based LCG. The first version is a *coupled LCG* (CLCG), coupling two linear congruential generator. It consists the use of the first LCG-1 to generate a control bit, and to select the output bit from the second LCG-2 following this equation $x_{t+1} = (a_1 x_t + b_1)\ mod\ 2^n$. However, the second version is a 4-stages pipeline of two CLCG that can generate a 4-bit simultaneous. As a consequences and for every pipeline level, a n-bit/n-pipeline will be computed with a two n-bit additions generated by the flowing equation $x_{t+1} = (2^r * x_t\ mod\ 2^n) + (x_i + b_1)\ mod\ 2^n)$. The results of every stage will be feedback and processed in the earlier and following stage for more complexity.

LUT and Accumulator Generators are a digital component used or implemented on FPGA to accelerate hardware optimization. The *Lookup-Tables* are a fully dependent FPGA technologies vendors. Every configurable logic block

(CLB) of FPGA include a n-Look-up Table (LUT 2/4/6/8-inputs and one outputs), some carry, a control logic and storage elements. In the other side, the *accumulators* are an user combinatory logic circuit that uses the FPGA resources as a components. Some of the works based on LUT are cited in [40], [38] and [39]. The authors of thous papers presents a series of a LUT optimized PRNG based on F-2 linear matrix recursive algorithms. The basic idea is to provide a maximum area efficiency using thous optimization. As consequences and for each row of the recurrence matrix "A" mapped as a XOR gate, can also be implemented in a single LUT. Also, it's allow in the same time, the test of the characteristic polynomial of each matrix for primitivity. Thous PRNGs proposed as LFSR or *masterine tewister MT* will use the LUT as a Flip-Flips (FF), Shift Registers (SR) or block of RAMs and then compare between them in FPGA. However, the big drawback of this method (LUT) is when creating a long period sequence. To do that, a large number of LUT (FF, SR, RAM) pairs must be used. Even if an application only needs 64 bits per cycle, it must use 512 LUT-FFs to get a period of 2^{512-1} for example. Even that, the authors claim a 87 percent the performance will be wasted using this optimization.

Another implementation in [17], where the authors propose a 9-stage PRNG of 8 bit using digital accumulator. Here, it based on $\sum \Delta$ modulators M for fractional frequency synthesizers and implemented as self-recursive structure. They claims by taking a constant X as an inputs, the accumulator sum P0 is the state variable of the system. Then and for every cycle, two outputs are generated, the sum modulus of M or the *quantization error* e0 $P0[i] = X + e0[i-1]$ and the carry output y0 for overloads if the $sum >= M$. However, the proposal solution implement a multistage of $\sum \Delta$, where the previews stage of the accumulator and the output M will be feedback to the early stage. Also, the authors scale the error e0 by a coefficient inputs accumulator and multiplied by a binary variable $d \in 0, 1$ generated by a linear feedback shift register (LFSR).

Blum Blum Shub Generators or BBS are a known efficient cryptography secure PRNG proposed [1986]. It has been proposed to solve the quadratic residue problem [1999] represented by the Rabin function of $x_t = {x_{t-1}}^2 \ mod \ n$. It's based on the product "n" of two big primes number called "Blum prime" and are congruent to (3 modulo 4). The output bit "j" from x_t of this generator is in first extracted by the least significant bits implemented in this equation $j = c * log\ (log(n))$, where c is a constant. Then it will be added to the final generated binary sequence $y_t = {x_t}^2 \ mod \ j$. However, just a few work using BBS are implemented in FPGA. As for this paper in [34], the authors present an area/speed comparison without any optimization between a 4-bits LFSR and a 16-bits BBS PRNG. Another work is founded in [32] for RFID tag applications. Here, the authors present some FPGA implementation of BBS of 160 to 512 bits using different multiplication algorithms to satisfy the main BBS equation ($A * BmodM$). The algorithms used for comparison are *classical combinational multiplier*, *classical shift adder and multiplication*, *Karatsuba's multiplication* and *Montgomery's direct modular multiplication*. However, the authors

compare just the area results and they claim the uses of the "Montgomery" iterative approach is the low cost area in FPGA.

Mersenne Twister Generators or MT are a word-wise recurrence matrix instead of bit-wise as for LFSR. The first implementation of this generator use a generalized version of Feedback Shift Register called "GFSR". It generates an output as a state vector or an array flowing this equation $x[t] = (x_{M1} \bigotimes x_{M2} \bigotimes ...)$, where M is the middle word $1 < M < t$. However, the GFSR are known to not reach the maximum period. To resolve that, a matrix recurrence twisted for GFSR "TGFSR" has been proposed to increase the period. It uses a feedback polynomial of $x_{t+k} = [x_{M+k} \bigotimes (x_k^u \mid x_{k+1}^l) .A]$, where k=1,2,3 and $x[0]^u$, $x[1]^l$ are the less and most significant bit of x_t (u,l are tempering bit shifts). To uniform the outputs of TGFSR, they are tempered by a bitwise multiplication using a binary matrix $y = x_t * T$. Also, it performs the statistical properties by reducing the dimensionality of equidistribution. In the same purpose for a long period PRNG, the Mersenne twister (MT) is proposed as a special case of TGFSR. We can find two main implementation of MT, the MT11213 with a period of $2^{19937-1}$ and the MT19937 with a period of $2^{19937-1}$.

We can start by some implementation of TGFSR that are cited in [2]. Here, the authors compare two FPGA implementation version of TGFSR. The first version implement two Mersenne Twister PRNG "MT19937" and "MT11213" but using *Generalised Feedback Shift Registers* (GFSRs) and without any hardware optimization. Then, two optimized TGFSR version "Ran" and "Ranq1" has been proposed based on just multiplication of fixed precision integers (with overflow). The comparison presented, shows the two TGFSR of the second version has a good area optimization unlike of the first version that use block memories RAMs for multiplications.

As for Mersenne Twister, we can find in [45]. Here, the authors propose a 3-stages pipeline version of MT19937. It uses a dual-port BRAM memories of the FPGA to generate a long period and for multiplication operation. However, it takes 3 cycle for every sample bit generated. In the other hand and in [27], the authors present an MT Implementation with just 3 read for ever 1 write into the RAM structure. This technique is resulting a single sample per cycle. Another pipeline MT19937 implementation is in [4] using a single block RAMs to store the state vector. The same approach has been done in [41] for "MT2203" but using multi ported block of RAMs to reduce HW resource instead of using multi/single RAMs with limited block ports. Hence, compared to the two early version of parallel MT. In [11], the authors propose two parallel version of "MT19937" by reducing the IO ports and bank memories RAMs. The first version is the *Interleaved Parallelization* (IP) that generate a N-bit for every "P" memories bank separately and during a period of $T = (P/N_{bits}) * (N_{IO}/clock)$. The second version is *Chunked Parallelization* (CP) that claim to use a fewer block RAMs compared to the IP version. It's based on the idea to use the output bit of each RAM bank as the far recurrence input for the following RAM bank.

Finally, a recent paper presented in [14]. Here, the authors compare another simple implementation of MT19937 that use RAM banks memories as storage

element with a new alternative solution of RAMs called *circular buffer*. The new solution is based on the fixed relationship between the indexes of the words. It considers the replacement of each step of the recurrence x_t with x_{t+k} in the work area. This way, the linear recurrence and the buffer of registers can be considered as a circular buffer. Here, the linear recurrence is carried out by some combinational logic between the input and the output of the buffer. Hence, the architecture is simplified as no logic for the table indexes is needed.

Cellular Automata or CA are proposed as a third class for exhibit chaotic or pseudo-random behavior presented by Wolfram "A New Kind of Science". It's can be represented as an array of cells that can hold and update their internal state dependent on local rules and the state of their neighborhood. The basic CA can combine 3 cells, each of them has two possibilities $0, 1$. It's resuming a state machine with a total of 8 (2^3 binary configuration possibilities for each CA and resulting a 256 (2^8) possible rules generally referred to by their Wolfram code. If the rule of a CA involves only XOR logic it's called a linear rule. It will be referred to as complement rules when using XNOR logic. Otherwise, a linear CA represent all cells that has linear rules and an additive CA if they combine the linear and complement rules. Another type is the uniform CA, which has the same rule for all cells else it's a hybrid CA. Finally, a null boundary CA is when both the left and right neighborhood of the leftmost and rightmost terminal cell is connected to logic 0-state.

In 2003 [**?**], the author implemented a custom IC of HCA that combines the outputs of a hybrid 90/150 rules of a 37-bit CA with a 43-bit of LFSR to produce a maximum length PRNG. However and to pass pass all the statistic tests, the LFSR and HCA must be clocked at different frequencies to create a random output sequence. To resolve Clocking issue, a new solution presented in [3]. Here, the authors propose to XOR the last bit of HCA with the last bit of LFSR to generate 1-bit per clock cycle. Following that, they found the best combination for a high quality of PRNG is 16-bit CA with a 37-bit LFSR. In [7], they compare the previews work LFSR/HCA [3] with a new implementation of CA called *Self-Programmable* CA (SPCA) and presented in [18]. It uses the super-rule 90/156 to determine when to make a dynamic rule change in each CA cell. Here, the the inputs rules of his neighboring cells is it self a second CA working in parallel with the main CA. However, even it gives a better throughput than the LFSR/HCA combination [7], it fails in the statistical test. Another CA combination solution is cited in [33]. Here, the authors propose another combination circuit of CA and Non-LFSR block based on A2U2 design. The main objective is to resist more of the various forms of cryptanalysis, such as correlation attacks and algebraic attacks. As for the outputs generated, they will pass to a mixer mechanism to increase the complexity for decryption.

In [1], the authors propose two Hybrid CA for an encryption system application. The first HCA-1 implemention work as a PRNG and use two rules 90/150 to generate a real-time key stream. The second HCA-2 is implemented as a block cipher and use a combination of 51/153/195 rules. However, to select witch rules will be used by the block cipher HCA-2, the PRNG or HCA-1 gen-

erate an encryption rules to switch the rules and provide each cell of HCA-2 is own rules.

A new implementation CA is presented in [13], where the authors create an automatic software tool to generator CA. It's based on the *algebraic normal form* (ANF) representation to generate a RTL code of any hybrid CA depending on ID rules provided as inputs. Using ANF representation and with ID=101 for 3 neighbor, the outputs generated will be identical to Matlab results following this formula ($y = [K_0 \bigotimes K_1(U_1) \bigotimes K_2(U_2) \bigotimes K_3(U_3) \bigotimes K_4(U_1 * U_2) \bigotimes ... K_7(U_1 * U_2 * U_3)] ... * mask$). Then in recent papers [20], they claim by using a chain of HCA instead of single HCA will increase the ratio of frequency/ares and cryptography.

Another and different concept of CA is proposed in [25]. It uses a new way to implement the rules for 1-d CA using the real time computer clock sequence. Here, the initial state configuration and his length is the product of day, month, year, hour, min and seconds. However, over a period of $t = x(60 - x)$, it uses two functional rules. The first rule is the product of minutes by seconds $m * s$, and the second rule is the number of minutes divided by the number of seconds multiplied by a constant $\frac{m}{s} * c$.

Chaotic Systems are interests theory to generate random number. Here, the use of this theory has been increase due to the sensitivity to initial conditions, unpredictability and ability to reciprocal synchronization. However, the finite precision of arithmetic and quantization generate a non-ideal and periodic random number. We can found chaotic PRNG proposed by using a differential equations in a continuous time domain. In other side, the most chaotic PRNG are implemented based on the recurrent sequences using chaotic maps in a discrete time domain $x^0 \in R$ where $x^{n+1} = f(x^n)$. Hence, from the cryptographers point of view, the most essential drawbacks of the chaotic systems are: weak resistance to recovery of parameters, relatively low digital precision comparing to the analog generators, periodicity problems, non ideal probability distributions, higher level of correlation and suffer from serious dynamical degradation due to quantization error and finite representation of system states, including loss of periodicity and shorter pseudo-orbits.

Chaotic Map Generators describe here are a generators based on a polynomial mapping that is equivalent to recurrence matrix degree 2. Here, the chaotic map generators uses a non-linear dynamic transformation. For this survey we found a logistic map based on this equation of $x_{t+1} = rx_t \ (1x_t)$, where r is what called the *biotic potential* $(3,57 < r < 4,0)$. Also, there is Hénon map which is also a discrete-time dynamical system according to the equations of $x_{t+1} = y_t + (1 - ax_t^2)$ and $y_{t+1} = bx_{t+1}$, where "a" and "b" are called *canonical parameters*. Following that, a chaotic map PRNG is implemented on FPGA and presented in in [10]. Here, the authors start from earlier study done of some FPGA implementation of thous chaotic map PRNG and presented in [8] and [9]. The authors implement two new optimization version of PRNG based on chaotic logistic map using the XSG Xilinx tool for more HW optimization. The first is a log-LUT based on LUT blocs and a high-speed carry line of the FPGA. Then,

the second is Log-DSP that use directly DSP component of FPGA. However, they add delays to ensure parallel sequence generation and a complex initial sequence used for a better NIST test results.

Another work presented in [31], where the authors demonstrate a mixed version using tow chaotic map to increase the security level against plaintex attacks. They show by coupling a chaotic encryption system (ENS) based on 2-D Hénon map and control system (CRS) based on 1-D logistic map. The ENS is used to generate the chaotic sequence, and the CSR to control the multiplexer and choose the outputs bits of ENS according to the value generated by the logistic map. Here, the final outputs bits are the XOR of the MSB of 32-bit CSR with his neighbor LSB following some rules. In the same approach and in [19], the authors uses a chaotic logistic map to generate output. They increase the period by a reseeding module, where the output sequence will be XORed with a vector mixing module based on auxiliary linear generator. Another solution proposed in [46], where the authors presents a chaos system based on three dynamic nonlinear transform arithmetic DNT process. It's a parallel structure that transform an input 3 times and improve a high cycle-length and distribution output sequence. However, for each DNT module it initiates his 2x256 code book and obtain the binary input sequence. Then, transformed and look up it using the inputs as parameters $x_{t+1} = x_t\ (C(w)\ R(q))$.

Spationtemporal Chaos Geneators are a temporally chaotic system as for a chaotic map. It's also a spatially and there is many mathematical models can be use to represent this type of generator. In [30] and to achieve a high operation speed, a bi-directional coupled chaotic map lattices (CML) has been used as a models represented by $x_{t+1}(i) = (1-\epsilon)f(x_t(i) + \frac{\epsilon}{2}(f(x_t(i-1) + f(x_n(i+1)$, where n and i are respectively temporal and spatial indexes of discrete lattices. ϵ is the couple coefficient, L is the number of the total spatial lattices and f(x) is a logistic map. They first deal with continuous domain with digitized all operand to be suited for HW implementation by using and modifying the CML to a finite integer set. Then second and to avoid finite precision chaotic map problem, they compute only the insignificant bit is subject to be an output.

Fibonacci post-processing Generators presented in [29]. It's uses a non-autonomous 4D hyperchaotic-based PRNG post processing based on Fibonacci series and driven by a 256-bit LFSR. The post processing is based on two loop feedback, where in first loop, they use a fixed 1-bit static rotation to suppress the short-term predictability. Then, the second loop is based on a variable rotation controlled using Fibonacci series of K-bit to enhances differential sensitivity if there is a change at any bit when the other bit propagate during n-cycle.

Chaotic Iteration or CI has been proposed by the authors in [**?**] [**?**] to implement a new post processing with the same chaotic theory characteristic defined by Devaney, Li-Yorke, to give better statistic test and increase cryptography. However, they claim their chaotic iterations generate a set of vectors (Boolean) and they are defined by an initial state x^0, an iteration function f, and a strategy S said to be a *chaotic strategy*. Where, at the n-th iteration, only the $S^n - th$ cell is *iterated*. Note that in a more general formulation, S^n can be a subset

of components and $f((x^{n-1})_{S^n})$ can be replaced by $f((x^n)_{S^n})$, where $k < n$, describing for example delays transmission. The main requirement is to prevent the machine from working in silos, by taking at each iterate a new input from the outside world. By doing so, the finite state machine does not necessarily enter into a loop: a same state can be visited twice, but with two completely different future evolution, depending on the inputs the machine receive. Over four version proposed, one of them has been implemented on FPGA using Chaotic iteration as post processing which use a BBS and XORshift PRNG as generator. The internal state x is a vector of 16-bits, whereas two 64-bit XORshift generators are provided as entropy sources and then the outputs are spread into four 32-bit integers. Then for each integer, there are 16 (2-bits) components that can be found and every 12 of these components are used to update the states. Lastly, the 4 least significant bits (LSBs) of the output BBS generator decide if the state must be updated with the considered 13-bits block or not.

4 True Random Number Generator

Passing now to the second section of the RNG theory and concept part. It contains analyses that been found for the fpga implementations of True RNG. As been defined earlier, TRNG are completely physical generator that use many hardware component and polynomial been produced, where FPGA is one HW support that this survey illustrate all works around it as an accelerator and for security purpose. However, man techniques and HW optimization has been demonstrated are show in this survey, where we can found the use the FPGA component optimized or mixing external component with FPGA or post processing FPGA to generate RNG.

Phase-Locked Loop or PLL circuit in general is derived by an external clock generator source like quartz or RC circuit, and which can set static or dynamic configuration. A PLL is completely depended of the physical environment like power, temperature or others that cause a very high secure RNG and reduce attacks. the TRNG generator based PLL will use a jitter extractor to generate randomness, wich is a short-term variation of the clock propagation. The most common jitter measurements used by FPGA vendors are period jitter and cycle-to-cycle jitter. The period jitter is defined as the difference between the n-th clock period and the mean clock period. However and in FPGA, a PLL is is based on the size of the clock jitter, the frequency divided by the VCO and the his loop filter bandwidth. Generally, we can find an analog PLL TRNG that extract the "intrinsic jitter" causes by the nois of his VCO for example or a digital PLL generator that use a synchronous/asynchronous Flip-Flop as an extractor. we can find an old implementation in [16], where the authors propose an analysis about extracting randomness from the jitter of an PLL implemented on Altera FPLD. Their studies is based on detecting the jitter by the sampling the reference clock signal (F_{CLK}) using a correlated signal synthesized in the PLL (F_{CLG}) where $F_{CLG} = F_{CLK}(K_M/K_D)$, and the maximum distance between the two clock (CLK,CLG) must be minimum

$MAX(\Delta T_{min}) < \sigma_{jit}$. However, they confirm in ideal environment condition and without a jitter the sampled output or random is deterministic under a period of $TQ = K_D T_{CLK} = K_m T_{CLG}$. Then, they conclude in a real condition $\sigma_{jit} \neq 0$ the randomness is not deterministic and depending on jitter distribution where the $MAX(\Delta T_{min}) = T_{CLK} * GCD(2K_M, K_D)/4K_M$.

Where, in [35] the authors demonstrate by taking [16] as model and combined more than one PLL in parallel or series to increase the significantly sensitivity on the jitter $S = F_{CLK} MAX(\Delta T_{min})$ and the output-bit of the generator compared to the use of one PLL. The configuration of multiple PLL are based on input/output length, VCO frequency and MUL/DIV factors (K_M/K_D). In [36] the authors test the impact of the the change on operation condition environment as temperature of an PLL and illustrate that with low bandwidth of PFF cause a higher number of the critical samples, decreases the output jitter and thus increase the tracking jitter. As an PLL application system, the work proposed in [43] they explore an embedded system with TRNG and based on PLL to extract randomness from the jitter and propose two version where the slower of 40kbps can pass the statistic tests.

ring oxialltor In [24] and in [23], the authors propose a TRNG based on two ring oscillators clocked by different clock generated by an internal PLL on FPGA. The authors also extract the jitter of the 2 RO implement in only one CLB slice using a simpler. Where, in [12], propose a new approach that can replace RO based on inverters using XOR combination between Fibonacci (FIRO) and Galois ring oscillators (GARO). the main key consists of a number of inverters and connected in a cascade together with XOR logic gates forming a feedback in an analogous way where the feedback polynomial form is $f(x) = (1+x)h(x)$ where $h(1) = 1$ and the result show with the new method can achieve a stable state less than classical RO.

In [42], the authors propose a Hybrid implementation on FPGA of TRNG based on RO and PRNG based on BBS generators and with high operation frequency of 400Mhz. However, they generate a low off-chip frequency based on resistor and capacitors RC and it was notice by implementing BBS using ALU structure for squaring and modulo operation the period will be $[(4.5 * n^2 + n)]$.

Self-timed ring STR In [6] and [5], the authors propose another alternative more based on Self-Timed Ring (STR) robust to environment (power, temperature) than RO based inverter. The SRT approach consist of a ripple of L stage of FIFO as a ring $(C_i)_{1 \leq i \leq L}$ with a phase of $\Delta\varphi = T/2L$, and extract jitter of each oscillator stage using two asynchronous handshaking protocol as even that can be "taken" or "bubble". However, the outputs randomness bits event $(S_i)_{1 \leq i \leq L}$ will be samples using a flip-flop by the main clock and the result will be combined with a XOR operation $\psi = s_1 \oplus s_2 \oplus ... \oplus s_L$. Secondly, the authors suggest that to avoid the limitation frequency of the STR by the long period delay, the maximum frequency is achieve when the propagation delay (forward and reverse static delay) is near to ring accuracy (N of token and bubble) and $[N_T/N_B \cong D_{ff}/_D rr \simeq 1]$.

Metastability In [44], the authors present a study of using *Metastability* phenomena as a entropy source generated by 5 IRO stage. They claim that by implementing the inverter as loop ring and using a Control Clock Generator to switch the connectivity between the IRO stages flowing two mode (MS, Generation), the output voltage converges to metastability level and stays longer than using bi-stable circuit (Flip-Flop) causing a high entropy. However, the authors wan to estimate the robustness of the system after applying the sampling process in a different process and environment variation modes using CMOS process and FPGA, and they find that it must added another stage for a higher quality output as decreasing the operation rate, applying a Von-Neumann post-processing and influences the loads (RC parasitic) of the last inverter, when it was noted that just post-processing is used in FPGA.

Another metastability uses as a RNG founded in [28], where the authors propose a TRNG using the metastability of the flip-flop when there is a violation in setup/hold time. The system is based on closed-loop feedback mechanism for auto-adjustment on delay Δ controlled by the Programmable delay lines (PDLs) stage based on LUT to avoid violation and maintain the metastability. However, the system use at-speed monitor to keep tracking the output bit probability and proportional-integral (PI) controller to decides to add/subtract the delay difference ($\Delta \rightarrow 0$). The probability of the output is $ProbOut = 1 = Q(\Delta/sigma)$ where $Q(x) = 1/\sqrt{4\Pi}\int_x{}^{\infty} expr(-u^2/2)du$. Where, the updated/corrected delay difference is the difference between the bias/skew caused by the routing asymmetric with the delay induce by the environment condition and the correct delay injected by the PDL ($\Delta = \Delta_p + \Delta_b - \Delta_f$). A revision version proposed by [26], to analyze the probability and maintain metastability state for a long period to avoid the deterministic state. however they use an extract hardware resource as memory for storing the outputs and use Hamming weight to calculate the probability bits histories.

5 Rundom Number Generator:Experiments Analysis

5.1 Statistic Test

National Institute of Standrs and technologies or **NIST** is based on hypothesis testing and includes a 15 batteries of mathematical and physical properties tests for RNG. It was developed to test the randomness of (arbitrary long) binary sequences produced by either HW/SW based cryptographic random or pseudo random bit generators with a fixed input parameters as sequence length of ($10^3 < N < 10^7$) and ($M < 55$) of binary sequences (sample size). For each N sequence produced, the NIST determine by prbability computed by $P - value$ (level of test) of all different type of non-randomness exist, where the $P - value$ must be more than a significance level $\alpha[0.0001, 0.01]$ to classify it as a good RNG and equal to 1 to have perfect randomness. Practically, the NIST test evaluate in first the range of acceptable proportions of binary sequences passing the statistical test, then in second it evaluate the uniformity of the test sequence and computing on the basis of x^2 test to the $P - value$ obtained.

TestU01 is now the most complete and difficult batterie of test of RNG. it was developed by Pierre L'Ecuyer and was implemented in the ANSI C languages with more than 516 tests resumed in 7 big batteries ($P - value[0.001, 0.999]$). This new Battery of tests covers divers classical tests of the other batteries with new algorithm performance and cryptographic tests. Six predefined batteries of tests are already included, where the last batterie **FIPS** is the recall for NIST tests. The first three batterie (319 tests) are for sequences of random numbers **SmallCrush**, **Crush** and **BigCrush**, and the three others (181 tests) are for bit sequences **Rabbit**, **Alphabit** and **Pseudo-DieHARD** designed to test a finite sequence contained in a random bits.

The other test batteries are the **Diehard**, it's include 18 test of Randomness and was developed by George Marsaglia. It was supposed to give a better way of analysis in comparison to original **FIPS** (16 tests) statistical tests. However and unlike NIST, the $P - value$ have to belong to some fixed chosen interval $[0 + \alpha, 1 - \alpha]$ with a level of signification α of 5% for example. Where the **ENT** battery tests was developed to sequences of bytes stored in files and reports the results of those tests as Entropy and x^2 tests.

The results of this survey presented by the authors, show some different level of what it concerned testing statistic of the RNG implementation from theories to practical results. Where the tests standard evaluate and updated to covert more the characteristic of the RNG starting by the simple to a complex and hard batteries of test. We can analysis the result by two category that it seen more stable and trust tests as the NIST and TESTU01, where the other batteries are even for old tests or they are included in those two batteries.

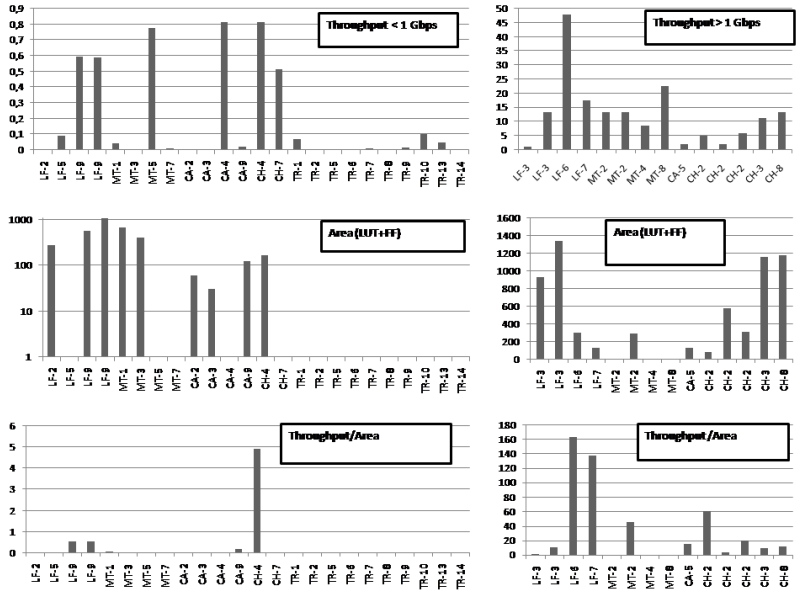
The PRNG analysis table show that not all papers has pass the NIST test, where Chaotic PRNG are th most success implementation of FPGA using different meothods. However, all work that use FPGA opetimised resourcec as LUT or acuumulator can pass the test where others not. give a details about this section of tests, where there is some difficult to say it pass it or now specially the coverege of tests.

RNG	DieHard	FIPS	NIST	TestU01	AIS
PRNG	LF[4,6] MT[1,3,8,7] CA[2,3,5,8]	CHO[7]	LF[1*,3,5] CA[5-9] CHO[1,3,4,6,7,8]	LF[4*,6*,7*,8*] MT[1*,3*,8*,6**]	
TRNG	IRO[7,8] STR[10]	PLL[4] STR[10,12] MTS[13]	PLL[1,2,5] IRO[6,8] STR[10,12] MTS[14,15]	IRO[7**]	STR[10] MTS[13]

Cryptography Secure

5.2 Hardware Implementation

Hardware Implementation



6 Conclusion

Conclusion

7 References

References

1. Petre Anghelescu, Emil Sofron, and Silviu Ionita. Vlsi implementation of high-speed cellular automata encryption algorithm. In *Semiconductor Conference, 2007. CAS 2007. International*, volume 2, pages 509–512. IEEE, 2007.
2. Simon Banks, Philip Beadling, and Andras Ferencz. Fpga implementation of pseudo random number generators for monte carlo methods in quantitative finance. In *Reconfigurable Computing and FPGAs, 2008. ReConFig'08. International Conference on*, pages 271–276. IEEE, 2008.
3. Juan C Cerda, Chris D Martinez, Jonathan M Comer, and David HK Hoe. An efficient fpga random number generator using lfsrs and cellular automata. In *Circuits and Systems (MWSCAS), 2012 IEEE 55th International Midwest Symposium on*, pages 912–915. IEEE, 2012.
4. Shrutisagar Chandrasekaran and Abbes Amira. High performance fpga implementation of the mersenne twister. In *Electronic Design, Test and Applications, 2008. DELTA 2008. 4th IEEE International Symposium on*, pages 482–485. IEEE, 2008.

5. Abdelkarim Cherkaoui, Viktor Fischer, Alain Aubert, and Laurent Fesquet. Comparison of self-timed ring and inverter ring oscillators as entropy sources in fpgas. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012*, pages 1325–1330. IEEE, 2012.
6. Abdelkarim Cherkaoui, Viktor Fischer, Alain Aubert, and Laurent Fesquet. A self-timed ring based true random number generator. In *Asynchronous Circuits and Systems (ASYNC), 2013 IEEE 19th International Symposium on*, pages 99–106. IEEE, 2013.
7. Jonathan M Comer, Juan C Cerda, Chris D Martinez, and David HK Hoe. Random number generators using cellular automata implemented on fpgas. In *System Theory (SSST), 2012 44th Southeastern Symposium on*, pages 67–72. IEEE, 2012.
8. Pawel Dabal and Ryszard Pelka. A chaos-based pseudo-random bit generator implemented in fpga device. In *Design and Diagnostics of Electronic Circuits & Systems (DDECS), 2011 IEEE 14th International Symposium on*, pages 151–154. IEEE, 2011.
9. Pawel Dabal and Ryszard Pelka. Fpga implementation of chaotic pseudo-random bit generators. In *Mixed Design of Integrated Circuits and Systems (MIXDES), 2012 Proceedings of the 19th International Conference*, pages 260–264. IEEE, 2012.
10. Pawel Dabal and Ryszard Pelka. A study on fast pipelined pseudo-random number generator based on chaotic logistic map. In *Design and Diagnostics of Electronic Circuits Systems, 17th International Symposium on*, pages 195–200, April 2014.
11. Ishaan L Dalal, Jared Harwayne-Gidansky, and Deian Stefan. On the fast generation of long-period pseudorandom number sequences. In *Systems, Applications and Technology Conference, 2008 IEEE Long Island*, pages 1–9. IEEE, 2008.
12. Markus Dichtl and Jovan Dj Golić. *High-speed true random number generation with logic gates only*. Springer, 2007.
13. Ioana Dogaru and Radu Dogaru. Algebraic normal form for rapid prototyping of elementary hybrid cellular automata in fpga. In *Electrical and Electronics Engineering (ISEEE), 2010 3rd International Symposium on*, pages 277–280. IEEE, 2010.
14. Pedro Echeverría and Marisa López-Vallejo. High performance fpga-oriented mersenne twister uniform random number generator. *Journal of Signal Processing Systems*, 71(2):105–109, 2013.
15. E. Erkek and T. Tuncer. The implementation of asg and sg random number generators. In *System Science and Engineering (ICSSE), 2013 International Conference on*, pages 363–367, July 2013.
16. Viktor Fischer and Miloš Drutarovskỳ. True random number generator embedded in reconfigurable hardware. In *Cryptographic Hardware and Embedded Systems-CHES 2002*, pages 415–430. Springer, 2003.
17. Victor R Gonzalez-Diaz, Fabio Pareschi, Gianluca Setti, and Franco Maloberti. A pseudorandom number generator based on time-variant recursion of accumulators. *Circuits and Systems II: Express Briefs, IEEE Transactions on*, 58(9):580–584, 2011.
18. Sheng-Uei Guan and Syn Kiat Tan. Pseudorandom number generator–the self programmable cellular automata. In *Knowledge-Based Intelligent Information and Engineering Systems*, pages 1230–1235. Springer, 2003.
19. NagaDeepa Hariprasad et al. Fpga implementation of a cryptography technology using pseudo random number generator. In *International Journal of Engineering Research and Technology*, volume 2. ESRSA Publications, 2013.

20. Dogaru Ioana and Dogaru Radu. Fpga implementation and evaluation of two cryptographically secure hybrid cellular automata. In *Communications (COMM), 2014 10th International Conference on*, pages 1–4. IEEE, 2014.
21. Minsu Kang. Fpga implementation of gaussian-distributed pseudo-random number generator. In *6th International Conference on Digital Content, Multimedia Technology and its Applications*, pages 11–13, 2010.
22. Raj S Katti and Sudarshan K Srinivasan. Efficient hardware implementation of a new pseudo-random bit sequence generator. In *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*, pages 1393–1396. IEEE, 2009.
23. Cristian Klein, Octavian Cret, and Alin Suciu. Design and implementation of a high quality and high throughput trng in fpga. *arXiv preprint arXiv:0906.4762*, 2009.
24. Paul Kohlbrenner and Kris Gaj. An embedded true random number generator for fpgas. In *Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, pages 71–78. ACM, 2004.
25. Leonidas Kotoulas, Demetrios Tsarouchis, Georgios Ch Sirakoulis, and Ioannis Andreadis. 1-d cellular automaton for pseudorandom number generation and its reconfigurable hardware implementation. In *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, pages 4–pp. IEEE, 2006.
26. Donggeon Lee, Hwajeong Seo, and Howon Kim. Metastability-based feedback method for enhancing fpga-based trng. *International Journal of Multimedia & Ubiquitous Engineering*, 9(3), 2014.
27. Yuan Li, Jiang Jiang, Hanqiang Cheng, Minxuan Zhang, and Shaojun Wei. An efficient hardware random number generator based on the mt method. In *Computer and Information Technology (CIT), 2012 IEEE 12th International Conference on*, pages 1011–1015. IEEE, 2012.
28. Mehrdad Majzoobi, Farinaz Koushanfar, and Srinivas Devadas. Fpga-based true random number generation using circuit metastability with adaptive feedback control. In *Cryptographic Hardware and Embedded Systems–CHES 2011*, pages 17–32. Springer, 2011.
29. Abhinav S Mansingka, Mohamed L Barakat, M Affan Zidan, Ahmed G Radwan, and Khaled N Salama. Fibonacci-based hardware post-processing for non-autonomous signum hyperchaotic system. In *IT Convergence and Security (ICITCS), 2013 International Conference on*, pages 1–4. IEEE, 2013.
30. Yaobin Mao, Liu Cao, and Wenbo Liu. Design and fpga implementation of a pseudo-random bit sequence generator using spatiotemporal chaos. In *Communications, Circuits and Systems Proceedings, 2006 International Conference on*, volume 3, pages 2114–2118. IEEE, 2006.
31. Lahcene Merah, Adda ALI-PACHA, and Naima HADJ SAID. Coupling two chaotic systems in order to increasing the security of a communication system-study and real time fpga implementation.
32. Pedro Peris-Lopez, Enrique San Millan, Jan CA van der Lubbe, and Luis A Entrena. Cryptographically secure pseudo-random bit generator for rfid tags. In *Internet Technology and Secured Transactions (ICITST), 2010 International Conference for*, pages 1–6. IEEE, 2010.
33. Lakshman Raut and David HK Hoe. Stream cipher design using cellular automata implemented on fpgas. In *System Theory (SSST), 2013 45th Southeastern Symposium on*, pages 146–149. IEEE, 2013.

34. Khushboo Sewak, Praveena Rajput, and Amit Kumar Panda. Fpga implementation of 16 bit bbs and lfsr pn sequence generator: A comparative study. In *Electrical, Electronics and Computer Science (SCEECS), 2012 IEEE Students' Conference on*, pages 1–3. IEEE, 2012.
35. Martin Šimka, Miloš Drutarovskỳ, and Viktor Fischer. Embedded true random number generator in actel fpgas. In *Workshop on Cryptographic Advances in Secure Hardware–CRASH*, pages 6–7, 2005.
36. Martin Simka, Milos Drutarovskỳ, Viktor Fischer, et al. Testing of pll-based true random number generator in changingworking conditions. *RADIOENGINEERING,*, 20:94–101, 2011.
37. David B Thomas. Fpga gaussian random number generators with guaranteed statistical accuracy. In *Field-Programmable Custom Computing Machines (FCCM), 2014 IEEE 22nd Annual International Symposium on*, pages 149–156. IEEE, 2014.
38. David B Thomas and Wayne Luk. Fpga-optimised uniform random number generators using luts and shift registers. In *Field Programmable Logic and Applications (FPL), 2010 International Conference on*, pages 77–82. IEEE, 2010.
39. David B Thomas and Wayne Luk. The lut-sr family of uniform random number generators for fpga architectures. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 21(4):761–770, 2013.
40. David Barrie Thomas and Wayne Luk. Fpga-optimised high-quality uniform random number generators. In *Proceedings of the 16th international ACM/SIGDA symposium on Field programmable gate arrays*, pages 235–244. ACM, 2008.
41. Xiang Tian and Khaled Benkrid. Mersenne twister random number generation on fpga, cpu and gpu. In *Adaptive Hardware and Systems, 2009. AHS 2009. NASA/ESA Conference on*, pages 460–464. IEEE, 2009.
42. Kuen Hung Tsoi, KH Leung, and Philip Heng Wai Leong. Compact fpga-based true and pseudo random number generators. In *Field-Programmable Custom Computing Machines, 2003. FCCM 2003. 11th Annual IEEE Symposium on*, pages 51–61. IEEE, 2003.
43. Michal Varchola, Milos Drutarovsky, Robert Fouquet, and Viktor Fischer. Hardware platform for testing performance of trngs embedded in actel fusion fpga. In *Radioelektronika, 2008 18th International Conference*, pages 1–4. IEEE, 2008.
44. Ihor Vasyltsov, Eduard Hambardzumyan, Young-Sik Kim, and Bohdan Karpinskyy. Fast digital trng based on metastable ring oscillator. In *Cryptographic Hardware and Embedded Systems–CHES 2008*, pages 164–180. Springer, 2008.
45. Shengfei Wu, Jiang Jiang, and Yuzhuo Fu. Hardware architecture for the parallel generation of long-period random numbers using mt method. In *Computer Engineering and Technology*, pages 8–15. Springer, 2013.
46. Ziqi Zhu and Hanping Hu. A dynamic nonlinear transform arithmetic for improving the properties chaos-based prng. In *Intelligent Control and Automation (WCICA), 2010 8th World Congress on*, pages 7055–7060, July 2010.