A Comparative Study of Cryptographically Secure Random Number Generators

Felicia Guo, Jingyi Xu
{felicia_guo, jingyixu}@berkeley.edu

***Abstract*—Good random numbers are essential for a variety of cryptographic applications. Compared with their software counterparts, the hardware RNGs often have higher efficiency and better security. In this project, we implement and compare two state-of-the-art hardware RNGs that are cryptographically secure. One is Blum Blum Shub PRNG, which has strong cryptographic properties and compact hardware implementation suitable for ASICs. The other is an optimized TRNG with Markov Chain decorrelation and IVN debiasing, which has improved energy efficiency and throughput. A comprehensive analysis of the BBS and optimized TRNG is presented from a hardware perspective. We find that the optimized TRNG has better energy and area efficiency, whereas the BBS PRNG has slightly higher throughput and produces bits of better statistical quality.**

***Index Terms*—Cryptographic-quality, True random number generators (TRNGs), iterative von Neumann (IVN), Markov chain (MC), Pseudorandom number generators (PRNGs), Blum Blum Shub(BBS),**

I. Introduction

Increasing amounts of private data transfer warrant the development of cryptography quality key security. Random number generators (RNGs) have thus become essential components in hardware systems to produce such keys. These generators come in the form of true random number generators (TRNGs) and cryptographically secure pseudorandom number generators (CSPRNGs).

For cryptographic applications TRNG are frequently used for their unpredictability. They use physical entropy sources, such as oscillator jitter, meta-stability and thermal noise to produce random sequences. However, TRNG design is often complicated in order to eliminate sources of bias that can degrade their statistical randomness, such as offset voltage or colored noise. Furthermore, their energy per bit efficiency and throughput is often much lower than those of PRNGs. In contrast, PRNGs, despite better efficiency and throughput, are deterministic. PRNGs operate by taking a numerical seed, and expanding the seed algorithmically to produce a sequence to appear random. Unfortunately, this random number generation method can PRNGs easier to predict and causes the period of the resulting sequence to be limited by the size of the PRNG's internal states. Certain PRNGs are designed to be resist attacks even if part of the sequence has been reverse engineered, and are classified as CSPRNGs. These PRNGs, despite higher random number quality and robustness, are still intrinsically deterministic, and can have the same pitfalls if their seed is determined. This is especially so as many CSPRNG algorithms are published.

In this report, we will draw comparisons between current implementations of CSPRNGs and TRNGs. Both types of RNGs have similar cryptography use cases. Thus, we aim to evaluate if TRNGs can achieve efficiency and throughput comparable to those CSPRNG implementations while maintaining unpredictability. We will begin with background on PRNGs to provide context for our choice of design comparisons (Section 2). Then, the Blum Blum Shub PRNG and a TRNG design optimized for energy efficiency and throughput will be discussed in detail (Section 3 and 4). We will detail our testing methodology to validate that both RNGs are cryptographic quality (Section 4) and pose our prediction for the throughput and energy efficiency results of the two implementations (Section 5).

II. Background

There are many algorithms to generate pseudo-random sequences, in this section, we present a short review of some PRNGs that have been implemented in hardware(mostly on FPGAs), to facilitate our discussion of the Blum Blum Shub PRNG.

One of the most common and simple method to implement a PRNG is to use a linear feedback shift register(LFSR), which can be implemented in hardware very efficiently. While the LFSR's outputs are highly uniform and thus has high entropy, they are notably correlated. In addition, the Berlekamp-Massey algorithm can efficiently find the recurrence relation given the output bit sequence, so LFSRs are not cryptographically secure.

Cellular Automata (CA) has also been used to generate random sequences. The CA consists of a grid of connected cells, which communicate with neighbouring cells according to the CA rules. Even simple rules can produce complex behaviors and hence good randomness in its output sequence. Depending on the implementations, a CA may or may not be secure for cryptographic applications.

Mersenne twisters are another family of PRNGs that has been widely used in software and implemented in hardware. In particular, the MT19937 has a considerably long period ($2^19937-1$), 623-dimensional equidistribution and high throughput(one output number per cycle). However, MT19937 needs to store a large internal state(624 32-bit numbers) to obtain such long period. It is also not cryptographically secure, after observing 624 iterations, one can predict all future iterations.

Chaos-based random number generators produce random sequences by using chaotic maps, usually defined as piecewise functions. The chaotic PRNGs can be relatively compactly implemented in hardware. While many of the chaotic PRNGs has been reported to fail cryptographic security, recently research proposed cryptographically secure chaos-based PRNGs.

III. Implementation

A. Blum Blum Shub PRNG

Since pseudo-random numbers are generated deterministically, they need to resist reverse engineering in order to be cryptographically secure. In particular, cryptographically secure pseudorandom number generators (CSPRNG) should pass the next-bit test: there is no polynomical time algorithm which, given the first l bits of the output sequences, can predict the $l+1$th bit with a probability significantly greater than 50%. Common CSPRNGs include the the Yarrow algorithm, Fortuna and Blum Blum Shub(BBS).

The Blum Blum Shub generates random sequences with the simple equation:

$$x_{n+1} = x_n^2 mod M \tag{1}$$

where M is the product of two large distinct primes p ad q which are both congruent to 3 modulo 4. The seed x_0 is an integer other than 0 or 1 that is co-prime to M.

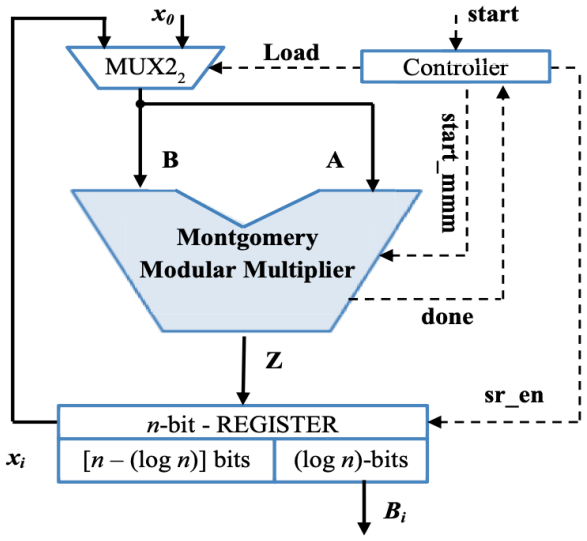

Fig. 1. BBS PRNG architecture using Montgomery Modular Multiplier

The BBS has strong cryptographic properties because predicting its output requires solving the quadratic residue problem: find the unique $x_{-1}\ mod\ M$ such that $x_{-1}^2\ mod\ M = x_0$. It has been shown that factoring M is needed to solve this, and there is known no polynomial time algorithm for integer factorization. The original BBS algorithm only produce the least significant bit of x_i at every iteration, but it has been proved that the least $log_2(log_2(M))$ bits of x_i can be used while maintaining equivalent security. Larger M value does not only make the generator more cryptographically secure, but also increase the number of available output bits. Typically, the size of M is between 256 and 1024 bits.

The BBS algorithm has 3 major steps:

1) choose distinct primes p and q such that
   $p = 4p_1 + 3$
   $q = 4q_1 + 3$
   $M = p * q$
2) repeatedly test different random seeds x_0 until gcd(x0, M) = 1
3) for each iteration i:
   $x_i = x_{i-1}^2\ mod\ M$
   output = least significant $log_2(log_2(M))$ bits of x_i

Squaring/multiplication and modulus operation are the most frequently used computation kernels. Instead of performing these two operations separately, we can reduce the latency by using modular multipliers. The Montgomery modular multiplier is a good candidate for the BBS PRNG, as it performs modular operation in one pass, thus has lower latency and requires less hardware.

Our BBS PRNG has 16-bit seeds and 16-bit parameter M, so that the 2 least significant bits of x_i can be used securely. The seed validation module is essentially a greatest common divisor unit that outputs if gcd(seed, M) = 1. The GCD algorithm takes a variable number of cycles to complete. Our experiments show that the gcd takes between 40 and 110. We have to consider how long seed validation takes because BBS, like most PRNGs, has a limited period of output sequence. There is no straight-forward formula to compute the period of BBS, however, the safe primes p and q, i.e. $gcd((\frac{p-3}{2}, \frac{q-3}{2}))$ is small, generally has longer periods. We find from empirical data that the period of 16-bit M generally ranges from 70 to 250, regardless of the seeds, yet a small subset of seeds result in a cycle length less than 10. Therefore, reseeding is necessary to ensure the cryptographic quality of the random output stream. In practice, the size of M is generally much larger than 16-bits, and consequently has a longer period, so reseeding can be performed much less frequently. Our design pipelines the reseeding and the modular multiplication, so that the BBS can steadily output 2 bit per clock cycle.

B. TRNG with Integrated Post Processing

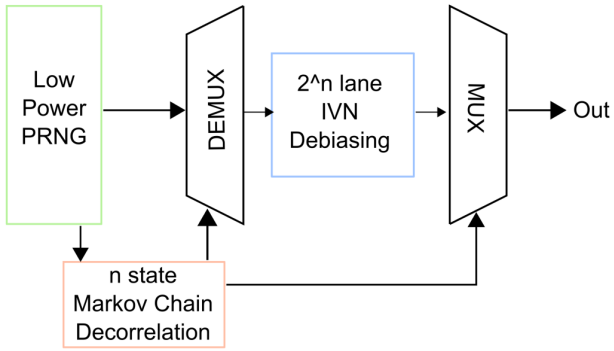

Fig. 2. Post processing with a lower quality RNG

Recent work on TRNGs has aimed to improve both the energy efficiency (in terms of energy per bit) and throughput (in terms of bits per second). The former may be improved by using a lower power RNG that does not meet cryptography standards, and processing the bit output for decorrelation and debiasing, i.e. integrating post-processing with hardware design [1]. Such design methods have the additional benefit of being highly digital. However, many such processing implementations rely on extracting bits from the provided sequence, resulting in a much lower throughput. We are therefore interested in processing methods that will not significantly decrease throughput while maintaining the energy and design benefits of using such an architecture. The proposed TRNG architecture is shown in Fig 3.

A low power option for the RNG is a strongARM latch utilizing noise from the input transistors to generate random bits(Figure 4). The strongARM latch is a good option because of 1) near zero power consumption and 2) rail to rail output [3].

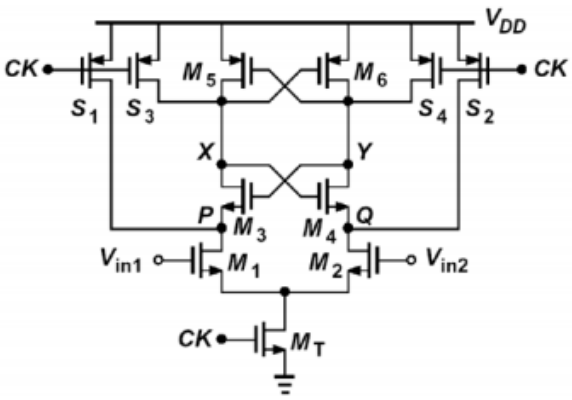

Fig. 3. StrongARM latch

The bits generated by the strongARM latch will likely need to be decorrelated, as correlation can be produced from sources such as colored noise. Correlation can be suppressed through a Markov Chain routing, where the source of randomness is the strongARM latch. The Markov chain state can be stored in an n-bit shift register. In previous works, the state is then used to direct the RNG bits to debiasing lanes, such that every set of bits debiased by a lane has the same RNG history[3]. This is to ensure that the bits are independently and identically distributed for debiasing.

The second part of post processing is debiasing, for which the iterating von Neumann (IVN) method is frequently used [2]. Given a sequence of independently and identically distributed (IID) bits, the IVN method extracts Shannon entropy, defined as:

$$H = -\Sigma p log(p) \tag{2}$$

As we target bias removal, bias for a binary bit sequence is defined as:

$$b = \frac{|p_1 - p_0|}{2} \tag{3}$$

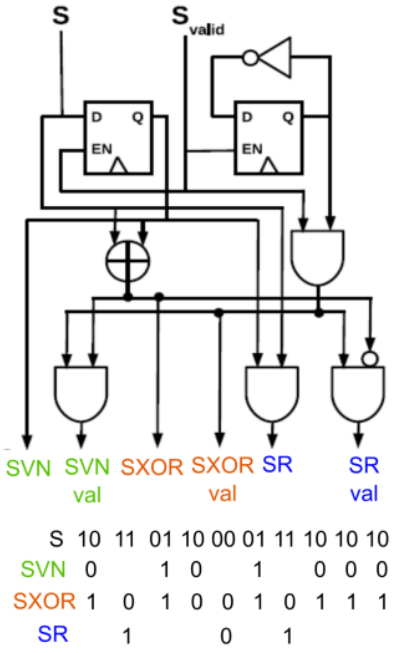

Fig. 4. IVN computation elements and example bit output

To remove bias and extract entropy from a sequence, the classical von Neumann method takes input bits in pairs, discards the pair if the bits are the same (00 and 11), replaces 01 with 1, and 10 with 0. Bias is removed as differing pairs of bits have the same probability of occurrence; the probability of getting a 01 or a 10 is the same if there is a non-zero probability of getting either 0 or 1, and the bit sequence is independently distributed. To improve throughput, the IVN method performs the same procedure as the classical von Neumann method on the discarded bits. If the amount of iterations is infinite, then the maximum bitrate will be the entropy of the incoming sequence times the sequence bit rate. A tree structure may be used to extract the entropy and throughput from the discarded bits.

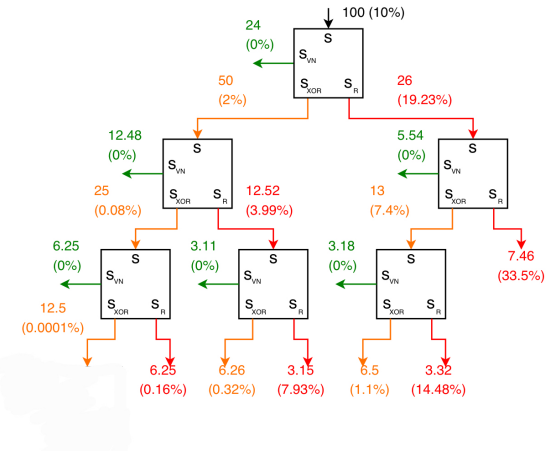

Fig. 5. IVN tree, with annotated throughput and bias(in parenthesis)

Our True Random Number Generator with integrated decorrelation and bias correction uses a 16 state Markov Chain that tracks the history of the previous 4 bits. Accordingly, it has 16 IVN lanes(or IVN trees), each with a 12-bit internal state. However, as only one lane is active at any point in time, we only need one shared IVN bias-removal lane and store the internal states in registers. This incurs loads and stores between the IVN lane and IVN registers, in exchange for smaller area and lower power.

In addition, as illustrated in the IVN tree, not every processing element outputs a valid bit every cycle. Thus we use a buffer to store the valid output bits so that on average the TRNG produces one valid bit per cycle. In the original work[3], the buffered output bits are fed into a 16-bit LFSR to remove any residual correlation. However, our experiments show that the LFSR can increase the output bias fairly significantly. We suspect that this is because LFSRs can get trapped in all-zeros if there are long sequences of the same bit, which causes the output to be biased towards 0. This is confirmed by the comparatively poor performance of the "Runs" test (discussed more in Section IV and V). Therefore, we excluded the LFSR in our implementation.

IV. Evaluation

Our evaluation methodology consists of RTL implementation, simulation, synthesis, and place-and-route of the Blum Blum Shub PRNG and the TRNG with Markov Chain decorrelation and IVN debiasing. Both RNGs generate one random bit per iteration. The two RNGs are compared based on throughput, energy efficiency, area, power consumption and quality of random sequence.

While it is not possible to prove a sequence is random, there are some properties that good cryptographically secure RNGs need - uniform distribution, polynomial-time unpredictability and for PRNGs, large period. We have evaluated the two random number generators with tests from the NIST 800-22 statistical test suite for random and pseudorandom number generators for cryptographic applications using 10 bitstreams of 100 bits each. For our purposes, we were interested in the Runs, Block Frequency, Approximate Entropy, FFT, and Serial tests. "Runs" provides a metric of how frequently changes between 0s and 1s happen, and check whether the probability of the $(i+1)th$ bit being larger or smaller than the ith bit forms a binomial distribution consistent with randomness. "Block Frequency" tests the proportion of 1s in a defined block length, which is directly related to our measure of bias. "Approximate Entropy" measures the overlapping patterns in a sequence, and is thus a measure of regularity. "FFT" detects repetitive patterns that happen in the number sequence, and help determine if we are producing a predictable period. "Serial" tests allow us to determine if the appearance of m-bit sequences is uniformly distributed, and is thus another measure of bias.

A. BBS

To test the statistical randomness of our BBS PRNG, we set up a verilog testbench which regularly feeds random seeds to the BBS and collects output sequence. We choose the parameter M to have a relatively large period of 250, in order to reduce the number of reseedings. We reseed every 150 cycles to guarantee that no cycles are repeated as well as to tolerate invalid seeds and extra reseeding.

B. TRNG

The TRNG was evaluated in two parts. The strongARM latch was implemented on the transistor level. Its outputs were generated through a custom testbench and simulated with Spectre. Inputs were set to be the same voltage ($\frac{VDD}{2}$), and transient simulations with noise were run for a nominal VDD of 700mV $\pm$ 10 %, with device noise being the contributor of randomness. To utilize noise with minimal output bit bias in physical implementation, the DC offset between terminals V_in1 and V_in2 should be compensated. In this experiment, however, we assumed constant DC offset and no mismatch to evaluate the structure itself. The resulting waveform was sampled and became the input bitstream for the post-processing blocks, implemented in RTL. The power consumption was evaluated through the current draw of the latch, from which we could find the peak and average current draw, and thus power from $P = IV$.

The post-processing blocks were evaluated for area, clock speed, and power constraints post place and route. The bias of the bitstreams from both the strongARM latch and the post processing blocks for each supply voltage were calculated as in (3) to determine the effectiveness of post processing debiasing, as well as determine the effect of supply voltage on the TRNG. Both bitstreams were also run through the NIST testbench.

V. Results

A. Area

The area estimates produced by place-and-route results are shown in table I.

BBS		**TRNG**	
modular multiplier	3112.18	IVN	1757.06
seed validation	646.18	buffer	715.46
control	30.55	MC router	32.19
total	4153.55	total	2535.28

TABLE I
AREA BREAKDOWN OF BBS AND TRNG IN um^2

The total area of the BBS PRNG is around 1.6x that of the Markov Chain based TRNG. The BBS area is dominated by the modular multiplier, because it needs to perform 32-bit modulo operation, rather than 16-bit, in order obtain correct results. The area of the latch was not included, as the device itself has a negligible size estimate (approximately 0.019 um^2).

B. Power

Table II shows the post place-and-route power consumption of the BBS PRNG and the TRNG.

	BBS	TRNG
Internal	0.551	0.273
Switching	0.956	0.225
Leakage	0.167	0.079
Total	1.675	0.577

TABLE II
POWER CONSUMPTION OF BBS AND TRNG IN mV

The BBS PRNG consumes 2.9x power compared to the TRNG. Granted the BBS PRNG has more cells and thus consume more power, it is notable that the switching power of the BBS PRNG is 10x that of the TRNG. This is likely because the BBS PRNG has more logic cells that actually switches during execution, inside the modular multiplier. The IVN module in the TRNG only has one logic lane that executes von Neumann operations and contribute to switching power, and the rest of the states are simply stored in registers.

Under nominal VDD the strongARM latch drew a peak of 34uA when running at 2GHz. The average current draw was 300nA.

C. Throughput

The BBS PRNG and TRNG are repeatedly synthesized while decreasing the clock period to measure the highest achievable clock frequency. Table III shows the max clock frequency and bitrate for each random number generator.

	BBS	TRNG
max achievable clock frequency(GHz)	0.66	1.25
bitrate(Gbit/s)	1.332	1.25

TABLE III
HIGHEST ACHIEVABLE CLOCK FREQUENCY FOR BBS AND TRNG

The strongARM could be pushed to run at a 2GHz clock. Thus the limiting frequency was still that of the processing unit. The TRNG produces roughly 1 bit per cycle, and thus has 1.25 Gbit/s throughput. This is higher than the original work[3], which has 86 Mbit/s bitrate in 65nm node and 1V Vdd. We presume this is partially due to the presence of a DC compensation scheme involving capacitive precharging, non-ideal components, and an older process (65nm).

Although the BBS PRNG's critical path is limited by the modular multiplier, and thus has lower frequency, it is able to generate 2 output bits per cycle, doubling the bitrate. If only one least significant bit is used for the output, the bitrate would be halved.

D. Statistical Quality

Both runsets were run through a mutliple NIST tests to evaluate them statistically. As shown in Table 2, the Blum Blum Shub PRNG evaluates much better than the TRNG. This shows that overall PRNGs can produce bits that are equally good, or in this case better, statistically when compared to TRNGs. We see that the BBS PRNG does not do as well on the FFT test, which confirms some expectations as it depends on seeding and has a set period for every seed. The TRNG does not do well with Runs and Approx. Entropy, which suggests correlations in the bits generated, though biasing appears to be within an acceptable range. The performance of Runs in particular, in conjunction with poor performance after addition of an LFSR, suggests that long sequences of the same bit are being produced.

	BBS	TRNG
Runs	10/10	8/10
Block Frequency	10/10	10/10
Approx. Entropy	10/10	8/10
FFT	9/10	10/10
Serial	10/10	10/10

TABLE IV
SELECTED NIST TEST RESULTS

Results show that the strongARM latch produces some bias as a result of supply voltage, as higher voltages produce biases resulting from more 1's, while lower voltages produce more 0s. The TRNG appears to be able to do some correction for higher TRNG biases (e.g. at 630mV supply), however, our results are inconclusive. It is likely that the quality of bits produced from the strongARM latch was actually too high, as post processing is tolerant of a bias up to 0.1 [1]. The amount of colored noise was thus lower than expected.

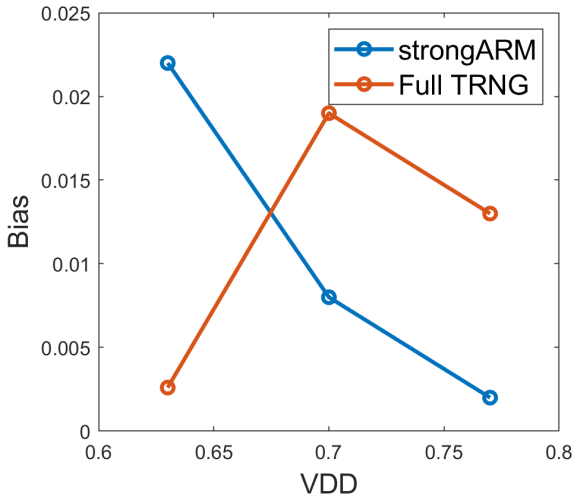

Fig. 6. Bias of strongARM vs full TRNG

VI. CONCLUSION

From this study, we have evaluated the efficiency and throughput, as well as quality of random numbers, of the Blum Blum Shub PRNG and an optimized TRNG. In general, CSPRNGs are used more frequently used for cases where large quantities of random numbers are needed, whereas TRNGs

are used for cases where high quality random numbers are needed and throughput matters less - such as in one time key generations. However, our results show that a BBS block can have comparable throughput and better bit statistical quality than a proposed TRNG. Our TRNG, however, has much better energy and area efficiency, as shown earlier. It is possible that this efficiency has contributed to lower quality numbers, and further investigation into the used algorithms and devices would be a next step.

Given more time and resources, more extensive testbenches should be run to confirm the findings of this study. More statistically significant results should involve at least 100 tests with longer bitstreams. More extensive testing on debiasing capabilities of the IVN processing would also be done with better sources of colored noise.

REFERENCES

[1] V. R. Pamula, X. Sun, S. M. Kim, F. u. Rahman, B. Zhang and V. S. Sathe, "A 65-nm CMOS 3.2-to-86 Mb/s 2.58 pJ/bit Highly Digital True-Random-Number Generator With Integrated De-Correlation and Bias Correction," in IEEE Solid-State Circuits Letters, vol. 1, no. 12, pp. 237-240, Dec. 2018.

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: ClareV. Rožić, B. Yang, W. Dehaene and I. Verbauwhede, "Iterating Von Neumann's post-processing under hardware constraints," 2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), McLean, VA, USA, 2016, pp. 37-42.

[3] B. Razavi, "The StrongARM Latch [A Circuit for All Seasons]," in IEEE Solid-State Circuits Magazine, vol. 7, no. 2, pp. 12-17, Spring 2015.

[4] K. H. Tsoi, K. H. Leung and P. H. W. Leong, "Compact FPGA-based true and pseudo random number generators," 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2003. FCCM 2003.

[5] A. K. Panda and K. C. Ray, "FPGA Prototype of Low Latency BBS PRNG," 2015 IEEE International Symposium on Nanoelectronic and Information Systems, 2015.

[6] K. Bhattacharjee, K. Maity, and S. Das. "A Search for Good Pseudo-random Number Generators : Susrvey and Empirical Studies", arXiv:1811.04035, 2018.

[7] K. Javeed, D. Irwin, and X. Wang. " Design and Performance Comparison of Modular Multipliers Implemented on FPGA Platform, " International Conference on Cloud Computing and Security, 2016.

[8] B. Mohammed, C. Guyeux, J. Couchot and A. Oudjida, "Survey on hardware implementation of random number generators on FPGA: Theory and experimental analyses," Computer Science Review. 27. 135-153, 2018.

[9] N.Nedjah, L. Mourelle," Hardware Architecture for the Montgomery Modular Multiplication," 2002.