A Comparative Analysis 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 ASIC. 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. Comparisons between the two RNGs are based on energy efficiency, throughput, area, power consumption and quality of randomness.**

***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 touse 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 recurrece 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 impelementations, 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. 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.

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.

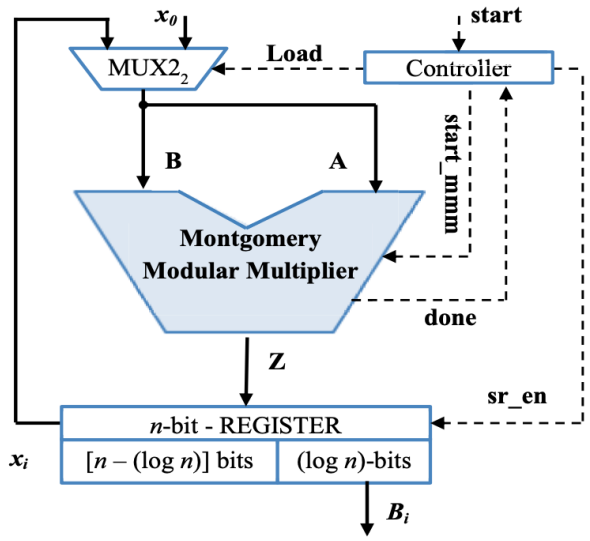

Fig. 1. BBS PRNG architecture using Montgomery Modular Multiplier

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 freqently 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.

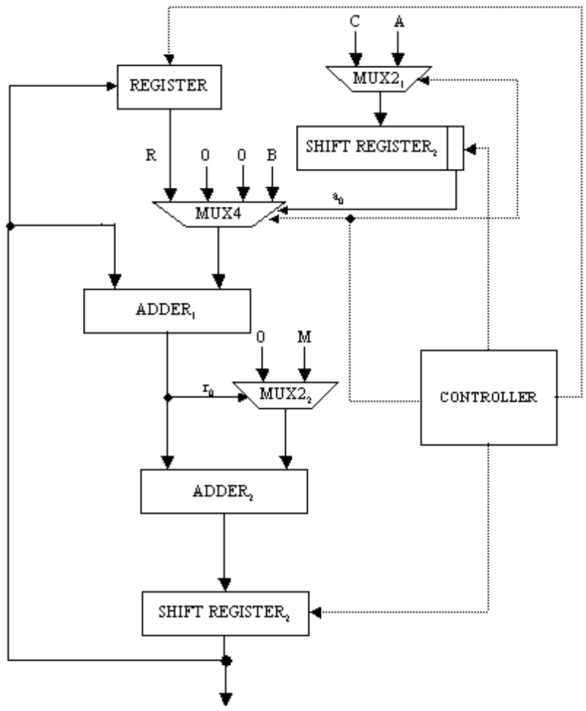

Fig. 2. Montgomery Modular Multiplier architecture

IV. TRNG with Integrated Post Processing

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 used 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.

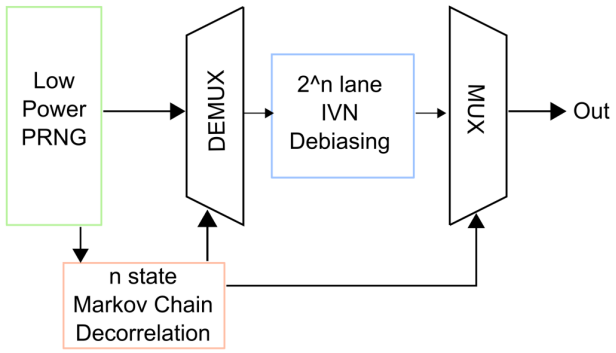

Fig. 3. Post processing integration with a lower quality PRNG. In this project the decorrelation, IVN debiasing, and muxing will be implemented in RTL. The low power PRNG will be constructed at the transistor level.

A low power option for the RNG is a strongARM latch with offset compensation to exploit meta-stability and noise resulting from mismatch on the input transistors(Figure 4). The strongARM latch is a good option because of 1) zero power consumption and 2) rail to rail output [3]. In this implementation, we will be holding the offset at a constant value, and relying on noise to cause the latch to swing to either of 1 or 0.

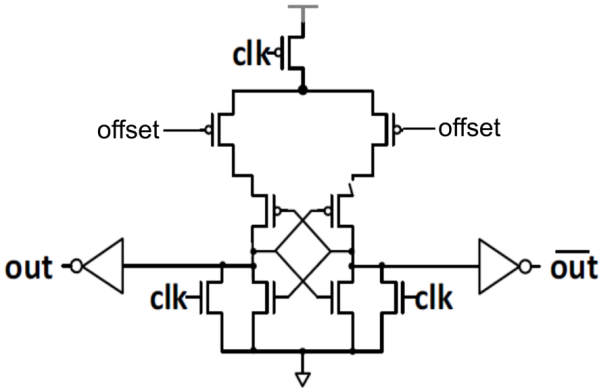

Fig. 4. StrongARM latch with offset control

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:

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

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

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

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.

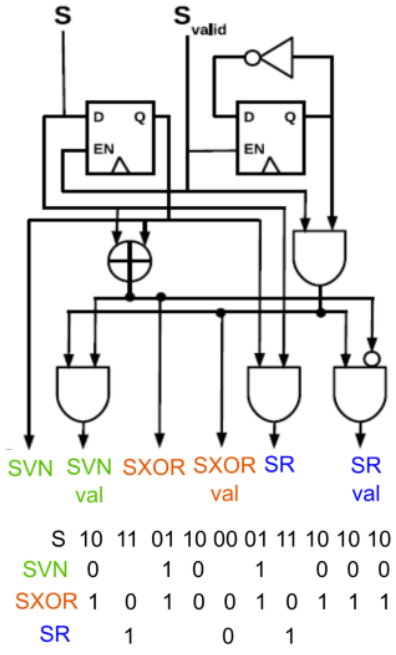

Fig. 5. IVN computation elements and example bit output

V. Methodology

Our evaluation methodology will consist of RTL implementation, synthesis, place-and-route, and simulation of the Blum Blum Shub PRNG and the TRNG with Markov Chain decorrelation and IVN debiasing. Both RNGs will generate one random bit per iteration. The two RNGs will be 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 plan to validate the quality of the two RNGs by running the NIST 800-22 statistical test suite for random and pseudorandom number generators for cryptographic applications. NIST 800-22 is a test suite frequently used in state of the art evaluations and is available as an open source platform.

VI. Hypothesis

As the main advantage of PRNGs is their bit throughput and efficiency, we would expect the Blum Blum Shub PRNG implementation to still have slightly better performance by these metrics than the integrated TRNG implementaion. Previous works have claimed both RNG implementations to be cryptographic quality, and thus we expect both to pass the NIST test suite.

VII. Conclusion

From this study, we expect to evaluate the efficiency and throughput, as well as quality of random numbers, of the Blum Blum Shub PRNG and an optimized TRNG. Both will be implemented, and we will determine if a TRNG can be made comparable in the metrics of interest to PRNGs while retaining random number quality. From the evaluations, we hope to draw conclusions on the best use cases for the two types of RNGs. 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.

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.