Towards Programmable All-Digital True Random Number Generator

Rashmi Agrawal
Adaptive and Secure Computing Systems Laboratory
Department of Electrical and Computer Engineering
Boston University - rashmi23@bu.edu

Lake Bu
The Charles Stark Draper Laboratory, Inc.
Cambridge, MA
lbu@draper.com

Eliakin Del Rosario
Adaptive and Secure Computing Systems Laboratory
Department of Electrical and Computer Engineering
Boston University - edelrosa@bu.edu

Michel A. Kinsy
Adaptive and Secure Computing Systems Laboratory
Department of Electrical and Computer Engineering
Boston University - mkinsy@bu.edu

ABSTRACT

Random number generator (RNG) is a core component in many applications such as scientific research, testing and diagnosis, gaming, and cryptosystems (e.g., obfuscation, encryption, and authentication). Although, there are various RNG designs targeting specific application goals such as low-power, high-throughput, stronger security guarantees, a universal programmable RNG design has remained elusive. Indeed, it is a challenge to have only one RNG unit in a system with multiple compute modules with different randomness requirements. In this work, we aim to provide a practical solution to this design challenge by proposing a multi-purpose true random number generator (TRNG), which can be configured in real time to generate random sequences with different requirements. Such a programmable TRNG is able to supply random bits to multiple modules with different demands. The proposed TRNG is a highly convenient multi-purpose hardware primitive that can be deployed in many designs as it provides a tunable physical entropy source and a dynamic cost-performance trade-off.

CCS CONCEPTS

• **Security and privacy → Hardware-based security protocols**.

KEYWORDS

True random number generation, chaotic maps, Lorenz systems, FPGA.

ACM Reference Format:
Rashmi Agrawal, Lake Bu, Eliakin Del Rosario, and Michel A. Kinsy. 2020. Towards Programmable All-Digital True Random Number Generator. In *Proceedings of the Great Lakes Symposium on VLSI 2020 (GLSVLSI '20), September 7–9, 2020, Virtual Event, China.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3386263.3406922

GLSVLSI '20, September 7–9, 2020, Virtual Event, China

ACM ISBN 978-1-4503-7944-1/20/09…$15.00
https://doi.org/10.1145/3386263.3406922

1 INTRODUCTION

Modern cryptographic approaches like the Advanced Encryption Standard (AES) achieve stronger security guarantees by making its algorithm open/public. This approach runs counter to earlier cryptographic algorithms which relied on closed designs or algorithmic obscurity for their security, e.g., Data Encryption Standard (DES). For open cryptographic techniques, the security of the cryptosystem resides not only in specific secret vectors such as keys, obfuscation masks, or one-time pads, but also the strength of those vectors. A secret key or vector is considered secure, if it is hard to guess/predict by a malicious party. In other words, it needs to be random, and preferably with high degree of physical entropy.

Different applications usually require different level of randomness [**? ?**]. For example, in computational tasks like sampling or reordering, true random numbers of moderate quality are acceptable. On the other hand, random masking to protect a system against a cold boot attack requires a large amount of random bits in a short period of time. Similarly, cryptographic applications whose strength depends on a strong secret key, e.g., data encryption, benefit from high entropy and high-quality random bits to generate a reliable key. Moreover, there are cases where within a single system, multiple compute modules need random numbers of different characteristics/requirements, e.g., high throughput for random masking, low power for frequent nonces, or high quality random bits for secret keys. Therefore, in order to tackle this issue, we propose a programmable multi-purpose TRNG design that can be used to generate different qualities of random numbers at run-time. The key contributions of this work are:

- A design to provide real-time programmable framework for TRNG. Users can tune a number of parameters in order to generate a random sequence fitting a specific demand: entropy, throughput, power, and quality;
- A TRNG cost-performance trade-off analysis. The capability to tune the TRNG's parameters enables users to gauge optimal level of performance for specific applications where less energy is consumed with low quality or throughput demand, and high energy is consumed with high quality or throughput demand;
- A TRNG design that provides better performance than other existing works in terms of the randomness quality, throughput, and energy cost per bit, when configured to work in the corresponding mode.

The proposed programmable multi-purpose TRNG uses chaotic maps to amplify the randomness of the seeds extracted from the physical source of entropy. Chaotic maps have the property that once a proper set of system parameters are selected, the output of the function will be highly sensitive to their initial state. Various types of pseudo or true random number generators using chaotic maps (e.g., logistic, Bernoulli shift, and dissipative quantum maps, etc.) have been proposed [**?**], [**?**], [**?**] where many of these RNG works passed the National Institute of Standards and Technology (NIST) random bit test [**?**]. We use these designs to perform the comparative study of the proposed Lorenz chaotic maps design. In addition, we also compare the proposed design against a number of representative TRNGs of various types, in terms of output quality, throughput, and energy cost, etc.

2 MOTIVATION AND PRELIMINARIES

2.1 Possible Applications of the Proposed Design

The proposed design is a programmable TRNG element that can provide random bits of varying qualities to different applications, i.e., a multi-purpose module that can be used in most designs and has the trade-off properties depicted in Fig. 1.

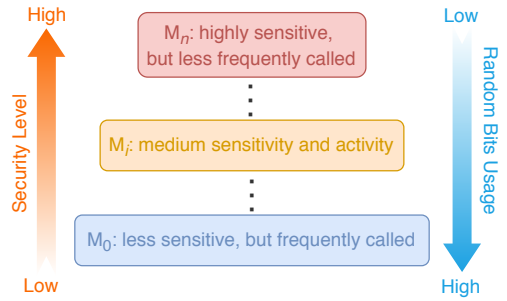

Figure 1: Different modules in a system may require different security levels and random bits usages. The random sequences for each module therefore have different characteristics.

The illustration in figure 1 consists of multiple modules $\{M_0 \cdots, M_i, \cdots, M_n\}$ with different properties. For example, M_0 carries out less sensitive tasks, but is called frequently and thus consumes lots of random bits. In this case, a TRNG with a small energy/bit cost having moderate randomness quality may suffice. Meanwhile, M_n processes highly sensitive tasks and is activated less frequently. Therefore, a high quality TRNG which is cryptographically secure may be a better option. Under this usage scenario, it is reasonable to spend a slightly larger energy/bit cost.

The system designer can either equip each module with its own TRNG or use one universal TRNG which can be customized to satisfy the randomness requirements of all the modules through real time configuration. In this work, we introduce such a programmable multi-purpose TRNG design. In addition to the examples listed above, other concrete application scenarios are:

2.1.1 Information Storage Systems. In these systems, there are many security and privacy concerns in managing how information is treated from creation to deletion. Based on the application needs (e.g., configuration frequency, analysis processes, provenance tracing, migratory behaviors, etc.), the associated information may have different levels of data sensitivity and validity periods. For example, in a secure multilevel memory system, the strength of encryption and the associated randomness between the processor and the first level cache is different from the encryption and the quality of the random number generator used between the main memory and off-chip storage. In this type of system, data can exist with either short validity periods where the data get processed and dumped frequently (e.g., data in first level cache) or with longer validity periods that needs to be protected with high security (e.g., data in off-chip storage). Additionally, there are cases where data can be both short-lived and long-lived depending on the context. All of these systems can benefit from a programmable TRNG to generate different levels of random masks or keys.

2.1.2 Distributed Systems. In a distributed system, an edge node may communicate frequently with other nodes in the same local group, similar to the M_0 case in Fig. 1. Its interactions with nodes in neighboring groups or with intermediate base stations may be less frequent (i.e., M_i), and its communication with a cloud server may be highly sporadic but require the highest security as in the M_n case. Such a distributed system also fits well in our use-case scenario [**?**].

2.1.3 Randomness Functions in operating system (OS). Many Linux users use `/dev/urandom` to generate random bits for most general purpose programs and `/dev/random` for highly confidential programs that demand more entropy and security. While `/dev/urandom` has high throughput and does blocks output (M_0 case), `/dev/random` (M_n case) blocks whenever the entropy pool is empty, i.e., collected random bits from the system's physical noise are depleted. Although regular computers can feed the entropy pool with system artifacts like keyboard strokes or noises, it is not feasible for computing systems without those peripherals like web servers. This creates potential vulnerabilities for these systems. For example, web server secure sockets layer (SSL) that heavily rely on secret keys. While software-based solutions have been proposed, e.g., `HAVEGE` [**?**], a programmable multi-purpose TRNG could provide a robust lightweight hardware-level solution.

2.2 Preliminaries of the Lorenz Chaotic Systems

Chaotic maps are a type of nonlinear and unpredictable systems which are highly sensitive to the initial condition. In such a system, any slight difference in the initial state will produce rapid escalating and compounding variations in the system's future behavior. These phenomena capture the infinite complexity of the nature, which is often described by fractal mathematics and thus serve as a potential candidate for randomness generation.

There are many types of chaotic systems, such as one-dimension (1D) logistic map, two-dimension (2D) Van der Pol system, and three-dimension (3D) Chua circuit. In this work we focus on the three dimensional Lorenz system. The Lorenz system was originally invented to describe and model the consequent bidirectional convection of thermally induced fluid, which is uniformly heated from below and cooled from above. However, because of its chaotic properties, it has also been used for a number of cryptographic

purposes. The most common application is obfuscation such as block cipher and image encryption. Another popular usage is in key agreement protocols. Furthermore, researchers have shown that the divergence and convergence properties of Lorenz systems can be used to design lightweight authentication techniques [?].

For the representation of the Lorenz system and the TRNG design, we introduce the following notations:

- α, β, γ: represent the parameters of Lorenz system's function;
- $p_n = (x, y, z)$: denotes the output point of a three-dimension (3D) Lorenz function. n stands for the number of iterations a Lorenz function ran before generating an output, and (x, y, z) are the coordinates;
- $LF_i(p_{0\text{-}i}, n)$: the i^{th} Lorenz function characterized by $\{\alpha_i, \beta_i, \gamma_i\}$, with $p_{0\text{-}i}$ being the initial condition, $LF_i(p_{0\text{-}i}, n) = p_{n\text{-}i}$, and where $p_{n\text{-}i} = (x_i, y_i, z_i)$ stands for the output of the i^{th} Lorenz function.

The discrete form of the Lorenz functions is given below:

$$\begin{cases} x_{n+1} = x_n + \alpha(x_n - y_n)\triangle t \\ y_{n+1} = y_n + (\gamma x_n - x_n z_n - y_n)\triangle t, \\ z_{n+1} = z_n + (x_n y_n - \beta z_n)\triangle t \end{cases} \quad (1)$$

where $\triangle t$ determines the resolution of the map.

Fig. 2 shows the shape of [Eq. 1] with system parameters $\alpha = 10, \beta = 2.6667, \gamma = 28$, which are the original values chosen only for fluid convection modeling, although, other values can be used to construct different Lorenz maps.

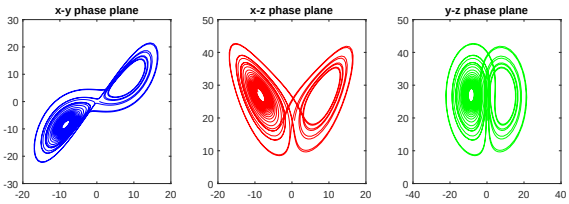

Figure 2: The 3d trajectory of a Lorenz system projected onto $x-y$, $x-z$ and $y-z$ planes, usually in a butterfly or "8" pattern.

The major properties of Lorenz functions are:

1. *Stationary points*: In [Eq. 1], when $\gamma > 1$, there are two distinct stationary points, which are:
$$C1, C2 = (\pm\sqrt{\beta(\gamma-1)}, \pm\sqrt{\beta(\gamma-1)}, \gamma - 1) \quad (2)$$
Although $C1$ and $C2$ are not physically on the trajectory, they serve as the *attractors* to balance out the initial transients, and drive the system towards its typical behavior.
2. *Convergence*: The attractors bring in the convergence property of a chaotic system. In other words, even if the initial state p_0 is not a point on the trajectory, it will converge to the orbit within limited iterations. In addition, although the placement of an exact point on the trajectory is highly unpredictable, over time the points do conform to the butterfly pattern statistically.
The convergence attribution can be described by Hausdorff dimension $dim_H K$ bounded by [?]:
$$dim_H K \leq 3 - \frac{2(\alpha+\beta+1)}{\alpha+1+\sqrt{(\alpha-1)^2+4\gamma\alpha}} \quad (3)$$
3. *Divergence*: A key property leveraged for the construction of the proposed TRNG. Intuitively, the divergence comes from the high randomness of the location and timing that a point $p_n = (x, y, z)$ appears on a 3D Lorenz map. With a tiny variation of the initial condition p_0, after n iterations the final output p_n will largely deviate. Theoretically, the Lyapunov exponent can be used to measure the rate of divergence of a chaotic system: $|\delta(p)| \approx |\delta(0)|e^{\lambda p}$, (4)
where for a trajectory $T(p)$'s nearby orbit $T(p) + \delta(p)$, $\delta(p)$ is a vector with infinitesimal initial length. The maximal λ is known to be approximately 0.9056.

3 A PROGRAMMABLE TRUE RANDOM NUMBER GENERATOR BASED ON LORENZ SYSTEMS

Here, we introduce the architecture of the proposed programmable multi-purpose true random number generator (TRNG). First, a digital physical entropy source named Asynchronous STopwatch-controlled Ring Oscillator (ASTRO) provides true random seeds for the TRNG. Then a group of Lorenz functions work as randomness amplifiers of the seeds. For randomness amplification, any number of functions can be used, e.g., ciphers or hash functions. In the proposed design, we choose chaotic functions due to their implementation simplicity and high throughput properties. The design passes the NIST SP 800-90A test which is a standard for cryptographically secure random number generators.

The outputs of the Lorenz functions are quantified and combined to further improve their quality before being sent out as the TRNG output. The architecture of the TRNG is shown in Fig. 3. The main components of the TRNG design are: (1) a **Strict Avalanche Criterion (SAC) network** - used to shuffle user's input and configure the parameters $\{\alpha, \beta, \gamma\}$ of the Lorenz functions, (2) the **ASTRO** - serves as the physical true randomness source, which dynamically configures the initial condition of the Lorenz functions, (3) **Lorenz Function group** - a group of Lorenz functions whose $\{\alpha, \beta, \gamma\}$ and p_0 are to be dynamically configured by the user input and ASTRO respectively, (4) a **Quantification (QNTF) module** - each Lorenz function's output is truncated by the QNTF module, in order to enhance its randomness, and (5) a **Blending (BLD) module** - used to permute and combine the truncated Lorenz function outputs into the final random bit string.

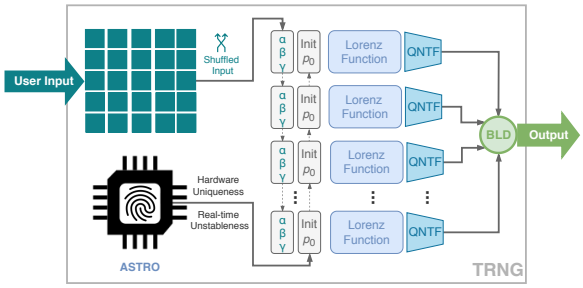

Figure 3: The inputs ($\{\alpha, \beta, \gamma\}$ and p_0) of the Lorenz Functions group are run through the SAC network and the ASTRO. Outputs from those units are then passed through the QNTF and BLD modules. The architecture features six tunable parameters for the tuning of the cost-performance trade-off.

In the implemented instance of the proposed design, all the vectors ($\{\alpha, \beta, \gamma\}$ and $\{x, y, z\}$) are 64-bit, where the 8 most significant bits (MSBs) are the integer part, and the 56 least significant bits (LSBs) represent the decimal part.

There are six user inputs that can be used to tune the TRNG to a desired working mode (e.g., high physical entropy, high quality, high throughput, energy saving etc.) as shown in Fig. 4. These parameters are introduced briefly in subsections 3.1 to 3.5, and further explored in detail with experimental data in Section 5.

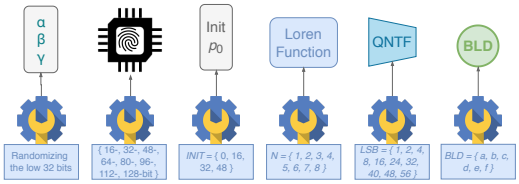

Figure 4: The six user inputs of the TRNG to tune and achieve the desired setting for different applications.

3.1 The SAC Network and Its $\{\alpha, \beta, \gamma\}$ Configurations

The introduced TRNG design supports the option of having user configure $\{\alpha, \beta, \gamma\}$ of the Lorenz function. Although user input is not mandatory since all $\{\alpha, \beta, \gamma\}$ can be preset, this feature adds more flexibility and runtime customization capabilities to the system. According to [Eq. 2], the change of $\{\alpha, \beta, \gamma\}$ will result in the relocation of the two attractors, which ultimately leads to a new chaotic map. Fig. 5 shows how the two attractors drift away when $\{\alpha, \beta, \gamma\}$ are changed.

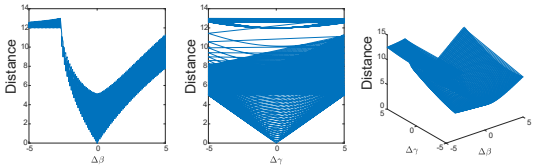

Figure 5: When β (left) or γ (middle) or both of them (right) fluctuate, the attractors will drift away according to the fluctuation magnitude. α on the other hand is related to the size of the trajectory. Every point with distance $\neq 0$ stands for a new chaotic map due to the change made to $\{\alpha, \beta, \gamma\}$.

Given a user input (a binary vector), a binary Strict Avalanche Criterion (SAC) network/matrix [**?**] will shuffle it before it is used for the $\{\alpha, \beta, \gamma\}$ configuration. In a SAC network, whenever a single input bit is changed, each output bit should have a flipping probability of 0.5. Thus, any two user inputs with a small difference will result in largely different configurations. For example, the S-Box in AES is a function satisfying SAC.

However, while the shuffled user input can be any arbitrary value, the Lorenz parameters $\{\alpha, \beta, \gamma\}$ cannot be arbitrarily configured. Otherwise the resultant trajectory may lose its chaotic property. Fig. 6 shows the results of changing the whole number part (first 8 bits), the decimal part (56 bits), and only the last 32 bits of the decimal part of a Lorenz map with the original parameters $\{\alpha = 10, \beta = 0.6, \gamma = 28\}$. It is worth noting that the user input is an external additive from programmability, it does not provide security and unpredictability to the TRNG. That task is accomplished by ASTRO.

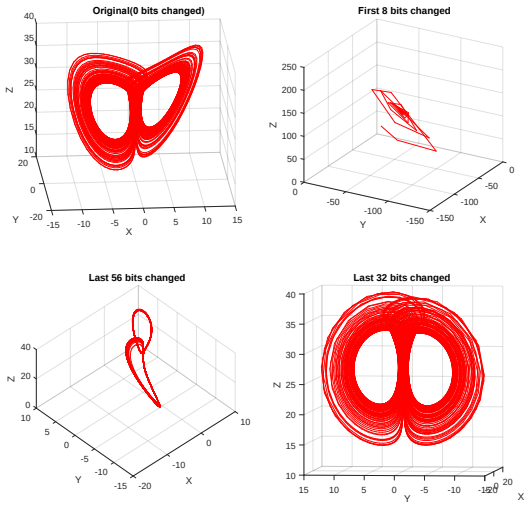

Figure 6: When the integer bits or all the 56 decimal bits are arbitrarily changed (up-right), the results trajectory could be no longer chaotic. Only the 48 LSBs or less (bottom-right for 32 bits) can be arbitrarily configured while maintaining the chaotic property.

3.2 Asynchronous STopwatch-controlled Ring Oscillator (ASTRO)

The Asynchronous STopwatch-controlled Ring Oscillator (ASTRO) module serves as the physical entropy source to provide seeds of true randomness to the TRNG. ASTRO is a variant of ring oscillator-based physical entropy generator, as proposed in [**?**]. Besides providing high physical entropy as conventional RO does, ASTRO is able to achieve a larger throughput by its design. ASTRO provides 16- to 128-bit true random seeds (per user's customization) for the initial condition p_0, which has three coordinates $\{x, y, z\}$. This feature fits perfectly into Lorenz function's divergence property, that a small variation in p_0 will lead to drastic deviation in p_n. The ASTRO module has a configurable parameter, $INIT \in \{48, 32, 16, 0\}$, to determine the number of bits to modify in the coordinate values. The design and implementation details of ASTRO are presented in Section 4.

3.3 Lorenz Function Group

All Lorenz functions in the group have 32 MSBs of their $\{\alpha_i, \beta_i, \gamma_i\}$ fixed with different values. Their 32 LSBs of $\{\alpha_i, \beta_i, \gamma_i\}$ and $\{p_{0\text{-}i}\}$ are arranged by the SAC and ASTRO. By [Eq. 1], a Lorenz function can be implemented on an FPGA using fixed-point adders and multipliers. This module manages a configurable parameter $N \in \{1, 2, 3, 4, 5, 6, 7, 8\}$, namely the number of Lorenz function.

3.4 QNTF Module

The quantification (QNTF) module does the truncation of the Lorenz functions' outputs. It has been proved in [**?**] that using the entire output vector as the random bit string may not pass the NIST test. This is because the MSBs of the output (especially the first 8) change very slowly as the number of iterations increases. Therefore, the LSBs should be used to maintain both good randomness and high throughput. The QNTF module has a configurable parameter, $LSB \in \{1, 2, 4, 8, 16, 24, 32, 40, 48, 56\}$, to determine the number of LSBs to use for the random string.

3.5 BLD Module

The blender (BLD) module blends all the $\{p_{n\text{-}i}\}$ from the Lorenz function group into the final TRNG output. The BLD module has a configurable parameter, $BLD \in$ {a, b, c, d, e, f}, to select which one of the following six formulas to use on the QNTF-truncated $\{p_{n\text{-}i}\}$ to form the final random bit string. The six formulas below are designed to shuffle and combine the outputs of the N Lorenz functions by XORing, permuting, reversing, and interleaving in multiple ways. More formulas can be explored and added to this module.

a. $(x_0 \oplus y_0 \oplus z_0)||\cdots||(x_{N-1} \oplus y_{N-1} \oplus z_{N-1})$;

b. $(x_0 \oplus \cdots \oplus x_{N-1})||(y_0 \oplus \cdots \oplus y_{N-1})||(z_0 \oplus \cdots \oplus z_{N-1})$;

c. $\bigoplus_{i=0}^{N-1}((i \text{ is even})?x_i : z_i) \,||\, \bigoplus_{i=0}^{N-1}((i \text{ is even})?y_i : x_i) \,||\, \bigoplus_{i=0}^{N-1}((i \text{ is even})?z_i : y_i)$;

d. $\bigoplus_{i=0}^{N-1}((i \text{ is even})?x_i : \overleftarrow{x_i}) \,||\, \bigoplus_{i=0}^{N-1}((i \text{ is even})?y_i : \overleftarrow{y_i}) \,||\, \bigoplus_{i=0}^{N-1}((i \text{ is even})?z_i : \overleftarrow{z_i})$;

e. $\bigoplus_{i=0}^{N-1}((i \text{ is even})?x_i : \overleftarrow{z_i}) \,||\, \bigoplus_{i=0}^{N-1}((i \text{ is even})?y_i : \overleftarrow{x_i}) \,||\, \bigoplus_{i=0}^{N-1}((i \text{ is even})?z_i : \overleftarrow{y_i})$;

f. $\bigoplus_{i=0}^{N-1}(x_i \oplus y_i \oplus z_i)$,

where $\oplus$ stands for bitwise XOR, $||$ concatenation, and $\leftarrow$ the bit order reverse operator.

4 THE PHYSICAL ENTROPY SOURCE: ASYNCHRONOUS STOPWATCH-CONTROLLED RING OSCILLATOR (ASTRO)

In this section, we describe the physical entropy source of the TRNG named Asynchronous STopwatch-controlled Ring Oscillator (ASTRO). Unlike many chaotic map-based TRNGs that use analog circuits as the physical entropy source, ASTRO can be conveniently programmed and instantiated as a digital circuit by Hardware Description Language (HDL) and implementation constraints on FPGAs.

4.1 The ASTRO Architecture

The micro-architecture of ASTRO is shown in Fig. 7. It consists of two five-stage ring oscillators (RO) and each clock is a counter with one serving the other as a stopwatch. The ASTRO's true randomness comes from the RO's unpredictable frequency fluctuation. Due to the manufacturing variation of each gate, the two ROs will have different frequencies. In addition, according to the measurements, the RO frequency is highly sensitive to the surrounding environment and varies from time to time. Thus, the timing of the faster RO's counter reaching overflow and triggering the "Stop" signal varies at each run, and the incrementing speed of the slower RO's counter also changes every time. These two interactive uncertainties together make it possible for true randomness. The RO frequency is usually around 350 MHz and is much higher than the 100 MHz FPGA clock. To clearly reflect the frequency difference (usually < 3MHz) between the two ROs, the counter size has to be no smaller than 28 bits. Through statistical analysis, the 12 MSBs of the slower RO's counter are relatively stable. However, the 16 LSBs always demonstrate adequate unpredictability, and can thus serve as a physical entropy source.

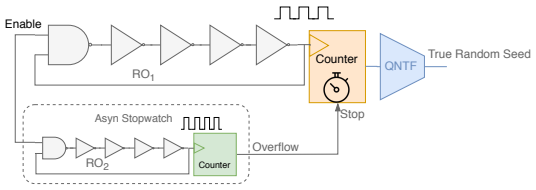

Figure 7: Whichever RO reaches to the counter overflow first, will send an asynchronous "Stop" signal to pause the other RO's counter.

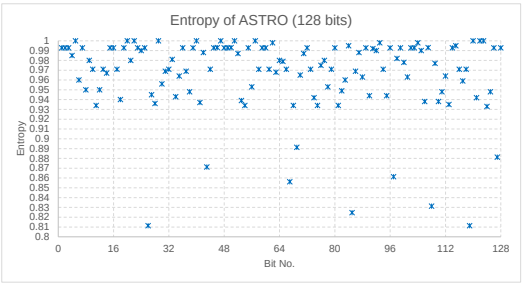

Figure 8: The entropies of most bits produced by ASTRO are located between the [0.93, 1] bit window, which is already a good quality. It can be further improved to a [0.997, 1] bit window by the proposed TRNG microarchitecture (cf. Fig. 11).

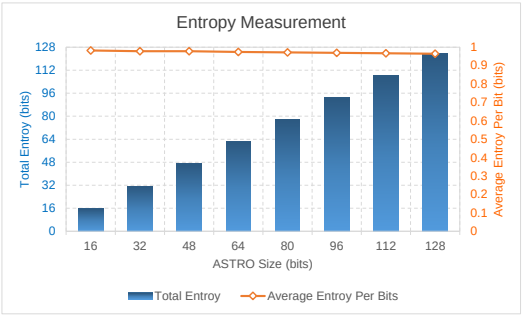

Figure 9: The total entropy in each case is very close to the output size. The average entropy provided by each bit of ASTRO is 0.964.

The proposed TRNG is equipped with eight ASTROs. Therefore, it is able to output up to $8 \times 16 = 128$ bits of true random seeds. The entropy of each bit is calculated in Fig. 8 based on over 50,000 sets of ASTRO output data. The total and average entropies produced by eight different sizes of ASTRO outputs (ranging from 16 to 128 bits) are shown in Fig. 9. It is notable that in order to provide strong cryptographic keys, information security standards [**?**] require at least 112-bit of security strength from physical entropy (equivalent to seven ASTROs turned on). Since every individual ASTRO costs negligible power (0.005W), we suggest all eight ASTROs to be turned on for any security-oriented applications.

5 PROGRAMMABILITY AND EXPERIMENTS

In this section, we present a summary of the behavioral study of the proposed TRNG design. We evaluate the TRNG's output entropy and the key sensitivity. We perform a comparative study between the proposed TRNG and the existing chaotic map-based RNGs. We show that the introduced design is able to achieve higher output quality than most existing works using comparative resources.

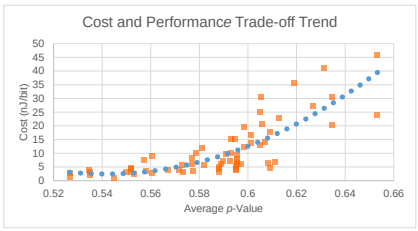

Figure 10: The blue dotted trend shows that the quality of the random sequences rises with the energy (nJ) consumed per bit.

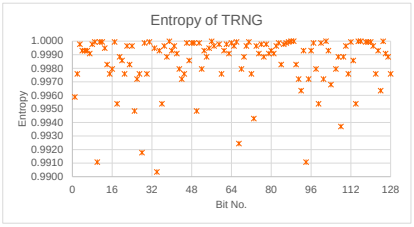

Figure 11: The entropy of the TRNG final outputs, which on average is 0.998 bit per output bit. To be comparable with Fig. 8, the TRNG is made to work under 128 bits/cycle throughput.

In Fig. 12, a comparison on output quality (evaluated by NIST random test) is made between the proposed TRNG and 7 other works(referred to by their citation indexes [? ? ? ? ? ? ?]).

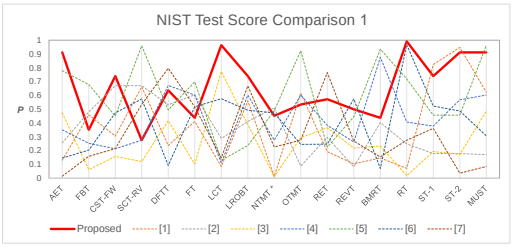

Figure 12: If a competitor has more than one design of RNG, the one with the best p-value is adopted in the figure.

Fig. 12 and Fig. 13 show the NIST random test scores by the p-values of each sub-test. The other RNGs (dual-metastability-based, RO-based, hash-based, and open-loop-based etc.) are referred to by their citation indexes [? ? ? ? ? ? ?].

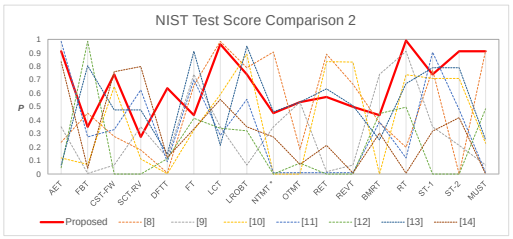

Figure 13: If a competitor has more than one design of RNG, the one with the best p-value is adopted in the figure.

6 CONCLUSION

In this work, we propose the design of a programmable multi-purpose true random number generator (TRNG) based on Lorenz chaotic systems. The proposed TRNG is able to generate any desired complexity of randomness. It lends itself well to computing systems with multiple modules and different demands of randomness, e.g., budget-limited, high throughput, high quality and security.