A TRNG Implemented using a Soft-Data Based Sponge Function within a Unified Strong PUF Architecture

Rachel Cazzola*, Cyrus Minwalla†, Calvin Chan‡, Jim Plusquellic*
*Department of Electrical and Computer Engineering, University of New Mexico, New Mexico, USA
†Cloud and Automation Technologies, Bank of Canada, Ontario, Canada
‡Department of Electrical, Computer and Energy Engineering, University of Colorado Boulder, Colorado, USA
E-mail: jimp@ece.unm.edu, minw@bank-banque-canada.ca, calvin.chan@colorado.edu

***Abstract*—Hardware security primitives including True Random Number Generators (TRNG) and Physical Unclonable Functions (PUFs) are central components to establishing a root of trust in microelectronic systems. In this paper, we propose a unified PUF-TRNG architecture that leverages a combination of the static entropy available in a strong PUF called the shift-register, reconvergent-fanout (SiRF) PUF, and the dynamic entropy associated with random noise present in path delay measurements. The SiRF PUF uses an engineered netlist containing a large number of paths as the source of static entropy, and a time-to-digital-converter (TDC) as a high-resolution, embedded instrument for measuring path delays, where measurement noise serves as the source of dynamic entropy. A novel data post-processing algorithm is proposed based on a modified duplex sponge construction. The sponge function operates on soft data, i.e., fixed point data values, to add entropy to the ensuing random bit sequences and to increase the bit generation rate. A post-processing algorithm for reproducing PUF-generated encryption keys is also used in the TRNG to protect against temperature-voltage attacks designed to subvert the random characteristics in the bit sequences. The unified PUF-TRNG architecture is implemented across multiple instances of a ZYBO Z7-10 FPGA board and extensively tested with NIST SP 800-22, NIST SP 800-90B, AIS-31, and DieHarder test suites. Results indicate a stable and robust TRNG design with excellent min-entropy and a moderate data rate.**

***Index Terms*—True Random Number Generator, Physical Unclonable Function, FPGA Implementation**

I. Introduction

Hardware security plays an increasingly important role in microelectronic systems, particularly those used to implement IoT resource-constrained applications and those used in unsupervised environments with heightened vulnerability to invasive attacks. The term "hardware security module" (HSM) is now widely used by security companies as an encapsulation of hardware security and trust primitives. HSMs build iron-clad security functions on top of a foundational module that is capable of generating random bit sequences, which are used either as keys for encryption, hashing, and authentication algorithms, as nonces for randomizing execution and authentication messages, or as initialization vectors for encryption.

Physical unclonable functions (PUFs) have emerged as hardware security primitives capable of serving as the root of trust within HSMs for key generation. The strong connection between PUFs and TRNGs has led to the proposal of several unified architectures. The primary distinction between PUFs and TRNGs is related to their sources of entropy. PUFs leverage a static source of entropy, i.e., baked-in manufacturing variations that are unique to each device and ideally remain stable throughout the lifecycle of the device. TRNGs target dynamic entropy, such as jitter, chaos, metastability, and sources of physical noise (e.g., thermal, shot, $1/f$).

Another important distinction relates to components of the architecture that are responsible for ensuring reproducibility of the bitstring. TRNGs have no such requirements, so these components are typically not needed. However, since TRNGs are subject to temperature-voltage attacks, we propose to leverage the PUF's reliability-enhancing module to add resilience against such attacks. A third distinction is related to the number of bits required and the bit generation rate, where TRNGs need to out-pace PUFs by orders of magnitude. Last, unlike TRNGs, the static source of entropy leveraged by PUFs is fixed and limited, significantly reducing the size of the entropy pool available to a TRNG. Therefore, TRNGs must be based on an unlimited dynamic source of entropy, or a combination of static and dynamic entropy, to be capable of meeting the demands of the HSM.

On the other hand, TRNGs and PUFs also have common requirements, namely the ability to produce random and unique bitstrings. Therefore, data post-processing components that improve the randomness of bitstrings, or improve their uniqueness across devices, can be used by both security functions. The common requirements of PUFs and TRNGs make a unified PUF-TRNG architecture attractive because a unified architecture can be smaller in size when compared to the combined sizes of the stand-alone versions. Moreover, both security functions are required in HSMs to provide a full range of security services to a wide range of application environments.

A unified PUF-TRNG hardware security primitive capable of serving the aforementioned roles is proposed and demonstrated in this paper. The key generation capability of the shift-register, reconvergent-fanout (SiRF) strong PUF is leveraged

here as a source of static entropy that is combined with a dynamic source of entropy to define a true random number generator (TRNG). The unified PUF-TRNG architecture reuses the data measurement and post-processing functions implemented within the SiRF PUF algorithm, adding only one additional module. The size of the expanded architecture is only approximately 5% larger.

A novel modification to the PUF data post-processing algorithm is proposed, introducing a functionality akin to the sponge function used in modern hashing algorithms (ex. SHA-3). However, instead of operating on bitstrings, the proposed sponge construction performs permutations on a set of soft-data values, i.e., fixed point values in the range of ± 64. The soft-data values are propagated from one iteration to the next in a chaining fashion, where each iteration updates the soft-data values based on a sample of the path delay measurements and a set of randomized parameters. This chaining approach has the advantage of eliminating all correlations that occur when reusing the path delay measurements over multiple iterations. Moreover, using soft-data expands the capacity of the internal state, thereby increasing the size of the entropy pool.

The specific contributions of this work include:

- A unified PUF-TRNG architecture that reuses more than 95% of the functionality of the stand-alone PUF architecture. The architecture includes a mode switch that changes the behavior of the challenge generation and the data post-processing operations associated with the PUF and TRNG functions.
- A novel soft-data based sponge construction that enables the use of both static and dynamic sources of entropy by the TRNG, while eliminating all traces of correlation from the static source.
- A Global Process and Environmental Variation (GPEV) post-processing module that adds resilience to temperature-voltage attacks.
- Experimental results validating the TRNG design and acceptance testing in four of the most popular random number test tools.

The remainder of this paper is organized as follows. Section II presents an overview of previously proposed unified PUF-TRNG architectures, and selected works describing stand-alone TRNG architectures. Section III describes the SiRF PUF-TRNG architecture and algorithm. Section IV presents the results of applying the statistical testing tools to the random bit sequences from multiple Xilinx Zynq SoC devices, and Section V presents our conclusions.

II. Related Work

A combined PUF-TRNG design using an array of ring oscillators (ROs) to generate entropy through jitter measurements was introduced in [1]. However, this architecture lacked mechanisms to counter temperature and voltage fluctuations and relies solely on noise as a source of entropy for the TRNG. Building on this foundation, the authors in [2] developed a calibration method to enhance performance, yet the underlying RO structure remained susceptible to attacks involving machine learning. In a different approach, the authors in [3] presented a unified PUF-TRNG architecture built on embedded flash memory, exploiting both spatial and temporal current variations. Fabricated using Global Foundries' 55 nm process, this design demonstrated resilience against machine learning-based intrusions. In more recent work, the authors in [4] implemented a unified PUF-TRNG architecture based on SRAM. Although this approach is scalable and offers high-speed operation, it had a significant LSB bit error rate, starting around 2.0% and increasing to 4.8% under temperature gradients. In contrast, the authors in [5] proposed a hybrid PUF-TRNG architecture that addresses environmental variability by testing 1-bit path delay differences, but the algorithm lacks a predictable runtime, and no bit rate metrics are provided. The SiRF PUF presented in [6] also uses delay differences, but it contrastingly computes the delay across an extensive set of uniquely designed paths. Their approach draws from both fixed and random entropy sources and applies a distribution-based adjustment to counter environmental influences on multi-bit digitized delays. Until now, there have been no investigations of the SiRF PUF's ability to generate true random numbers.

The authors of [7] presented a compact, energy-efficient TRNG and PUF design for securing IoT devices, featuring a reconfigurable ring oscillator structure with on-chip calibration to ensure high entropy and reliability, and an authentication protocol to resist various attacks. Liu et al. [8] introduced a memristive TRNG with an intrinsic two-dimensional PUF for tamper-resistance in nanoelectronics, using unique physical entropy sources analyzed by a neural network to enhance security against cloning. While the two-dimensional PUF's high sensitivity and randomness are beneficial for security, they might make the system more vulnerable to environmental variations (e.g., temperature or supply voltage changes), potentially impacting reliability. Pratihar et al. [9] presented a dual-mode PUF-TRNG design that leveraged both oscillation frequency and propagation delay, to provide both secure, instance-specific randomness for PUFs and high entropy for TRNGs. Although the design effectively integrated both functions with resistance to machine learning attacks, its reliance on specific Transition Effect Ring Oscillator (TERO) cell configurations may limit its adaptability to other circuit architectures or processes. The authors of [10] proposed a reconfigurable PUF-TRNG module achieving high hardware efficiency and robust cryptographic properties through ring oscillators and rigorous statistical testing. While the design improves hardware efficiency and scalability, the increased complexity of the configurable PUF-TRNG module may introduce potential challenges in terms of power consumption and processing overhead, especially in resource-constrained IoT environments. Satpathy et al. [11] designed a unified entropy generator combining a 512-bit CMOS entropy source with FPGA post-processing for secure IoT authentication. They achieved high throughput, low power, and resistance to power attacks, with 25% area savings over separate PUF and TRNG.

In other work, Vatajelu et al. [12] designed a unified

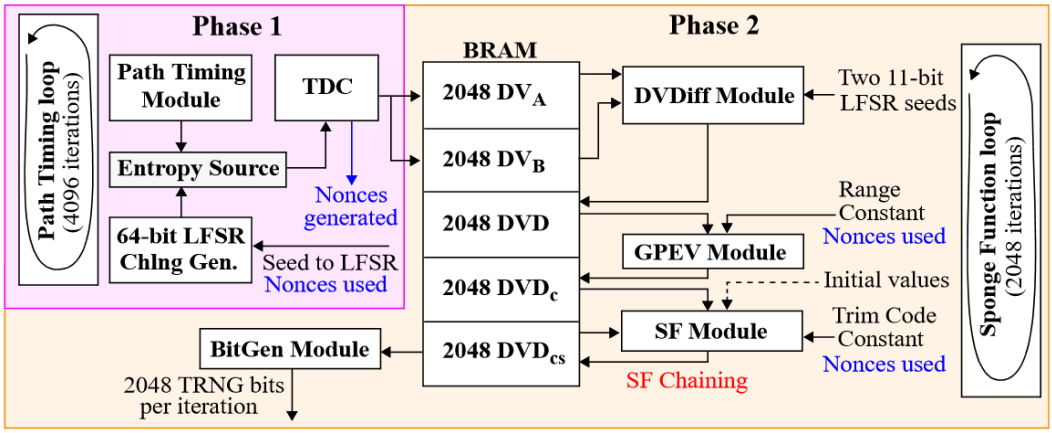

Fig. 1: SiRF TRNG algorithm.

PUF using magnetic RAM (MRAM) based on spin-transfer torque variations, which theoretically achieved zero-bit error rates. The PUF, however, was not physically realized, and its error rate remains untested in practice. Following this, Khan et al. [13] proposed a similar MRAM-based PUF and successfully fabricated and tested their model. Zalivako et al. [14] explored the use of a ring oscillator PUF as a source of randomness for generating true random numbers on an FPGA. Although the generated sequences demonstrated strong randomness, additional compression via an LFSR to improve statistical properties highlighted a limitation in the unprocessed PUF output. Sadr et al. [15] presented a true random number generator using an Arbiter PUF within an NFSR, achieving 10 million bits per second with high entropy and low resource usage. The reliance on Arbiter PUFs however, may limit stability in environments with significant temperature or voltage fluctuations. The author of [16] presented a method for designing TRNGs using memory-based ternary PUFs, where unpredictable state cells generate multiple sources of randomness, which were enhanced by an XOR compiler or modulo-3 addition. Although the approach demonstrated high-quality randomness, the effectiveness of the XOR compiler is limited when the initial data stream lacks randomness.

A stand-alone TRNG was proposed in [17], based on jitter noise within a ring oscillator based structure. The embedded carry chain within an Artix FPGA was used to obtain high-resolution measurements of three propagating edges launched simultaneously within the ring oscillator, which are processed into a random bit sequence. The authors run an extensive set of statistical tools to evaluate the quality of the random bit sequence. The authors of [18], also propose a stand-alone TRNG that used a cellular automata topology to implement an asynchronous circuit structure capable of generating random bit sequences. The NIST SP800-90B and AIS-31 statistical test suites were applied in both publications, allowing for direct comparisons between their TRNGs and the proposed PUF-TRNG architecture described in this paper.

III. System Overview

The proposed TRNG uses the SiRF PUF [6] static entropy source and data post-processing algorithms, as well as the dynamic entropy generated by the path delay measurement process. A flowchart of the operations carried out by the TRNG algorithm are shown in Fig. 1. The algorithm can be partitioned into a random pattern generation, path timing, and nonce bit generation phase (Phase 1) and a data post-processing and random bit generation phase (Phase 2). Operations in Phase 1 are dedicated to measuring a set of path delays (static entropy) while simultaneously generating a set of nonce bits (dynamic entropy). These operations provide digitized timing data and nonce bits for randomizing several parameters utilized by the state machine modules of Phase 2, and for the LFSR in Phase I during subsequent iterations.

A. Source of Static and Dynamic Entropy

The key building block of the static source of entropy for the SiRF PUF is shown in Fig. 2, which consists of a sequence of shift-registers, non-inverting logic gates, and MUXs. The elements within the module's netlist are not fixed to specific layout positions, as is true for identically designed PUF architectures, and instead are placed and routed according to the optimization algorithms within the physical synthesis tool. We use wire constraints to prevent the place-and-route tool from modifying the netlist structure, and we design the netlist to ensure glitch-free propagation of signals.

Challenges to the module, applied to the inputs along the left in the figure, are used to configure paths through the shift-registers and MUXs. The $TDChlng[0]$ bit of the challenge controls whether the rising transitions introduced by the **Launch FFs** on the module's inputs propagate through the module as rising (0) or falling (1) edges. This bit drives the low-order bit of the shift registers which determines the transition direction on the output of the shift registers. Since our design is based on the original SiRF netlist, additional details regarding its features are found in [6].

The number of testable paths through each module with 16 primary outputs, as shown in Fig. 1, is 512. The modules

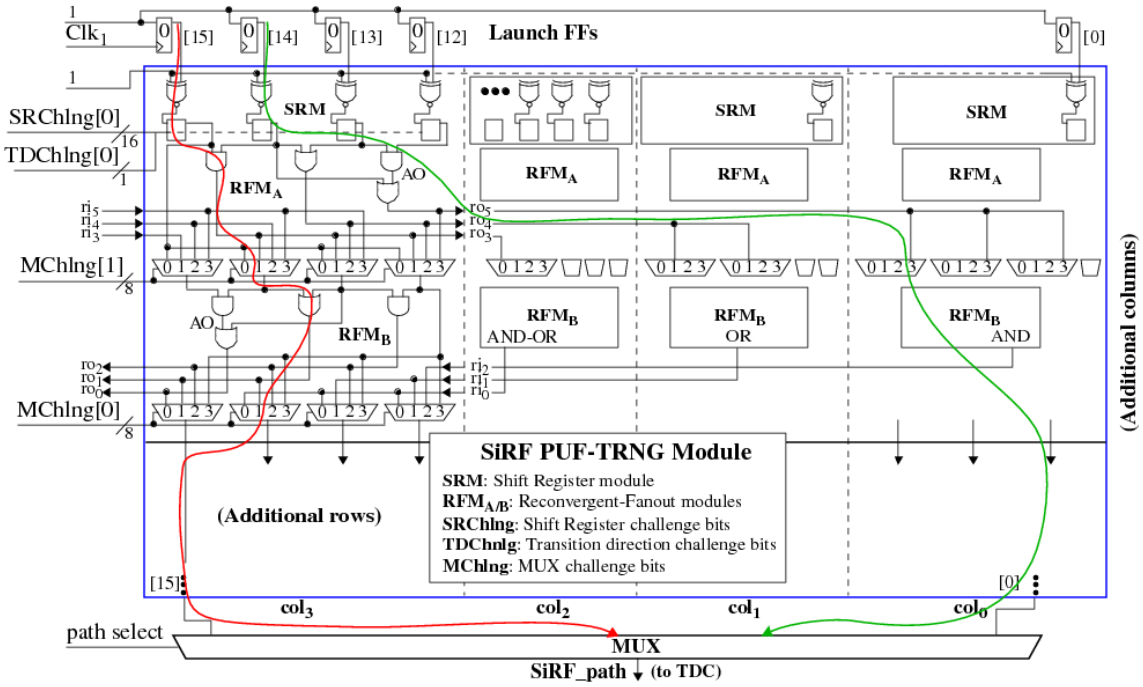

Fig. 2: Basic building block of the SiRF PUF-TRNG architecture.

can be stacked vertically and horizontally. Each vertically stacked module multiplies the number of testable paths by a factor of 80. For a composite netlist with three rows and two columns (3×2), the eight possible combinations of rising and falling transitions through each row result in 52,428,800 testable paths. The 3×2 configuration of modules is used as the netlist configuration in the experiments carried out in this work.

Dynamic entropy is represented as measurement noise in the path delay measurements, which is captured in the low-order bit of the delay value (DV). A full bit of dynamic entropy is obtained by XOR'ing the low-order bits of 12 consecutive path delay measurements. The need for 12 consecutive measurements was obtained from experiments carried out on FPGAs. We show in the Experimental Results section that this type of distillation process produces bitstrings that pass all statistical tests. During the path timing phase, a total of 341 nonce bits are obtained by measuring 4,096 path delays.

The 341 nonce bits, or approximately 42 bytes, are used for randomizing two parameters in post-processing operations carried out in Phase II, as described below. The 2048 iterations of the Sponge Function loop shown in Fig. 1 reuse the 42 nonce bytes every 20 iterations of the loop, and therefore, the nonce bytes are reused at least 102 times over the 2048 iterations. As we show, the contributions by other sources of entropy in Phase II eliminate the penalty that can occur when sources of entropy are reused for multiple purposes.

B. Phase 1 - Boot-Strap and DV Generation Operations

A boot-strap operation labeled as Phase 1 in Fig. 1 is executed to obtain an initial set of nonce bits. Here, the initial seed to the 64-bit LFSR used in the *Random Chlng Gen.* module to generate pseudo-random challenges is set to 1. The delay values (DV) measured during the boot-strap operation are then discarded, and only the nonce bits are retained.

The nonce bits are used to randomize several elements of the TRNG algorithm. The first 64 bits of the nonce are used to seed the 64-bit LFSR after the boot-strap operation to enable the *Random Chlng Gen.* module to generate a second set of pseudo-random challenges. The Path Timing operation is started a second time to measure the delays of 4,096 paths. But unlike boot-strap, the DV are now stored in the BRAM.

The SiRF PUF-TRNG incorporates an embedded instrument referred to as the time-to-digital converter (TDC), which is used to measure path delays at a resolution of approximately 18 ps. The *Random Chlng. Gen.* module generates a sequence of 128 random challenges using a 64-bit LFSR, where each challenge allows the measurement of exactly 32 path delays, one-at-a-time, for a total of 4,096 path delays. The TDC-digitized delays are referred to as delay values or DV, and can vary in value from approximately 300 to 1000 depending on the length of the path. The DV are stored as 12-bit integer values in an on-chip block RAM (BRAM) in two distinct groups of 2,048 elements each, labeled as DV_A and DV_B in Fig. 1. Operations are carried out on these values in Phase 2.

C. Phase 2 - Data Post-Processing and Random Bit Generation

The SiRF algorithm consists of a sequence of four data post-processing modules labeled *DVDiff*, *GPEV*, *SF* and *BitGen*, as shown in Fig. 1. The functions carried out by these

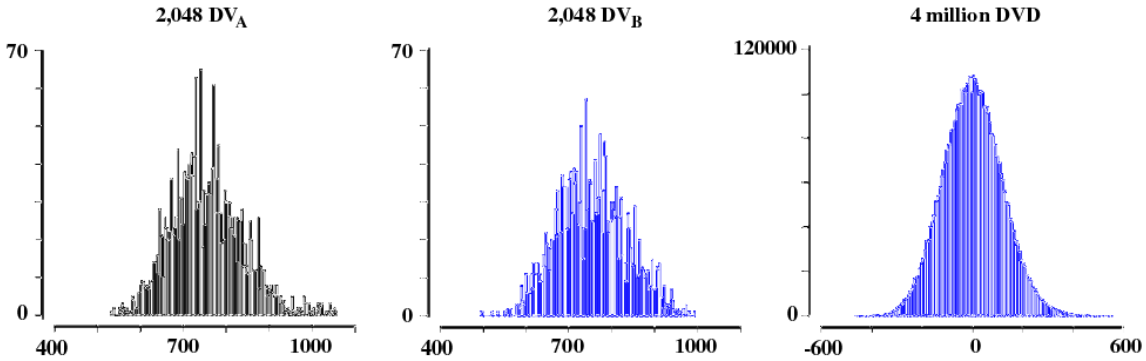

Fig. 3: Example DV_A, DV_B and DVD path delay distributions. The DVD are computed by creating differences using all combinations of elements in the DV_A and DV_B groups.

modules are described in the following with illustrations to show the effect that they have on the DV. Each data post-processing module is executed 2,048 times, with each iteration producing 2,048 random bits. In tandem, these steps capture the necessary properties of a sponge function in a novel way. As shown on the right of 1, we label the iteration of these post-processing modules as the *Sponge Function loop*.

The sampled delay values DV_A and DV_B are combinatorially expanded into a set of $2048 \times 2048 = 2^{22}$ or 4 million digital value differences (DVDs). DV_A and DV_B are reused to accelerate the bit-generation speed. A full analysis of bit generation rate and resource utilization is provided in the Experimental Results section.

1) Differencing: The first module, *DVDiff* module, uses two 11-bit LFSRs to pseudo-randomly select samples from the 2,048 DV_A and DV_B values. The 11-bit LFSRs are configured with a primitive polynomial enabling them to select all possible unique combinations of DV from the two sets. The seeds to the two LFSR at the beginning of each iteration are incremented and decremented, respectively, over the range from 0 to 2047 and from 2047 to 0, e.g., the first iteration assigns LFSR seeds 0 and 2047, the second iteration assigns 1 and 2046, etc. During each iteration, the selected elements in the DV_A and DV_B sets are pairwise subtracted to produce a set of 2,048 DVD, which are signed integers stored in the BRAM. A benefit of this approach is its simplicity, but, as we show, the drawback of full reuse is the existence of correlations between DVD sets. Example DV_A, DV_B and DVD distributions are shown in Fig. 3.

2) Temperature and Voltage Compensation: Next, the Global Process and Environmental Variation (GPEV) step reads the 2,048 DVD from the BRAM and applies two linear transformations. The first transformation removes delay variations introduced by both global process variations and temperature-supply voltage effects by effectively standardizing the DVD, while the second restores the DVD distribution to an integer value range. The horizontal spread (range) associated with the second transformation is controlled by a randomized *Range Constant* (RC) parameter. A 6-bit component of the nonces generated during boot-strap in Phase 1 is used to expand the range to values between 128 and 191, which adds unpredictability to the second transformation. The transformed DVD are stored in BRAM as DVD_c, where 'c' refers to compensated.

The first transformation is described in Eqs. 1 through 4. It defends against temperature attacks, and supply voltage attacks that change the DC level. Voltage glitching that is applied during the path delay measurement process will disrupt the compensation process, but the impact that the attack has on the DVD_c is unpredictable, and may in fact be defeated given the outlier avoidance method used in GPEV. The mean, μ, of the DVD, is computed in the standard fashion. The range is computed as the width of the distribution using Eq. 4, with offsets defined by Eqs. 2 and 3. Here, the range is measured at the limits given by -5% and +5% of the maximum and minimum values, respectively, of elements in the distribution. The offsets make the range measurement robust to the presence of outliers. The parameter $rand\_RC$ in Eq 6 refers to the randomized Range Constant from Fig. 1.

$$\mu = \frac{\sum_{j=1}^{|\text{DVD}|} \text{DVD}_j}{|\text{DVD}|} \tag{1}$$

$$\text{max} = \max(\text{DVD}) - 0.05 * \max(\text{DVD}) \tag{2}$$

$$\text{min} = \min(\text{DVD}) + 0.05 * \min(\text{DVD}) \tag{3}$$

$$\text{range} = \max_{\forall j \in |\text{DVD}|} \text{DVD}_j - \min_{\forall j \in |\text{DVD}|} \text{DVD}_j \tag{4}$$

$$DVD_N = \frac{(\text{DVD} - \mu)}{\text{range}} \tag{5}$$

$$\text{DVD}_c = DVD_N \times rand\_RC \tag{6}$$

3) Spread-Factor (SF) Chaining: The Spread-Factor module, labeled *SF* in Fig. 1, is responsible for removing correlations, which occur when the same DV_A and DV_B are subtracted under all combinations by the DVDiff module over the 2,048 iterations of the Sponge Function loop. Spread-Factors (SF) refer to sets of digital values defined over the range given by Eq. 7, that are subtracted from the DVD_c,

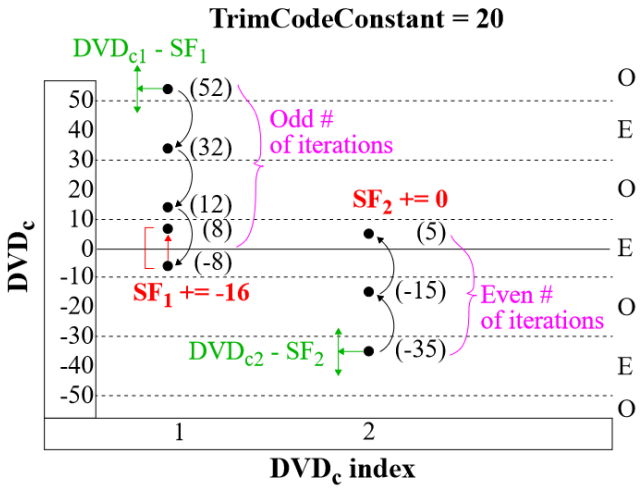

Fig. 4: The randomized Trim Code Constant (TCC) parameter partitions the DVD_c range into intervals shown by the dotted lines. The SF module processing operations are illustrated using two example DVD_c.

to produce DVD_{cs}. The SF are also updated and re-used over consecutive iterations of the Sponge Function loop, as described below.

$$SF := \pm 64 \tag{7}$$

The SF module defines a sequence of operations that are illustrated in Fig. 4 using two DVD_c. The SF module accepts an input parameter called the *Trim Code Constant* or TCC, whose value is randomized using a 3-bit nonce from the bootstrap operation in Phase 1. The 3-bit nonce is used to select an even-valued TCC between 8 and 22. The figure shows the operations carried out when the TCC selected is 20.

The algorithm works as follows: First, the incoming SF_x value is subtracted from the DVD_{cx} value. Second, the SF module iteratively adds or subtracts the TCC from the DVD_c until it falls with the region $\pm TCC/2$. And third, the number of subtractions or additions is used to determine if the current states of the DVD_{cx} and SF_x are updated. If the original DVD_c is located in the odd 'O' region (annotated on the right side of the figure), an offset is computed that moves the DVD_c to the symmetric position on the opposite side of the 0 line, and the offset is added to the SF_x. This occurs for DVD_{c1} in Fig. 4, where the computed offset is -16. Otherwise neither the DVD_{cx} nor SF_x are changed from their current values, as shown for the DVD_{c2} example in the figure.

The SF are set to zero on the first iteration of the Sponge Function loop (initialization is shown, labeled as *Initial values* in Fig. 1) and therefore, they have no effect on the DVD_{cx} processed during the first iteration. For successive iterations, the SF_x are randomly updated based on the selection of the TCC parameter and on the magnitude of the compensated difference values represented by the DVD_{cx}. The high order bit(s) of the SF_x are modified as needed to maintain the SF_x in the range of ± 64 as a means of bounding their minimum and maximum values.

The distributions of the SF over successive iterations of the Sponge Function loop illustrate an important characteristic of the SiRF TRNG algorithm. The triangular shape of the distribution, shown in Fig. 5, are typical of random distributions that are constructed by adding two discrete random numbers, e.g., the sums produced in experiments with two random die. The SF in the current iteration are the sum of the incoming DVD_c and the SF from the previous iteration. The graph plots the set of 208,896 SF produced over 102 iterations of the Sponge Function loop under the condition that the *Range Constant* and *Trim Code Constant* are the same (each iteration produces 2,048 SF, and the same RC and TCC are used every 20 iterations because of the nonce reuse discussed earlier). Fig. 5 is created by starting with iteration 19 of the Sponge Function loop. However, the other 19 distributions that can be created in this fashion are very similar.

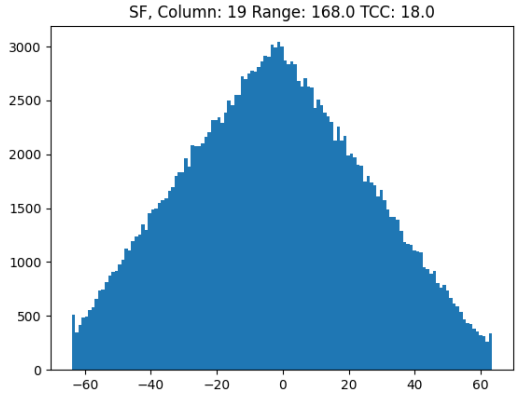

Fig. 5: Distribution of SF generated during multiple iterations of the TRNG algorithm, starting at iteration 19, and then every 20th iteration thereafter where the Range is 168.0 and the TCC is 18.0.

In contrast, the DVD_{cs} distributions generated as output in each iteration (and subsequently used to generate the random stream of bits) are uniformly distributed as shown by Fig. 6. Here, we divided each set of 2,048 DVD_{cs} by the TCC used during that iteration, i.e., they are *normalized* to values in the range between -0.5 and 0.5, to allow the underlying distribution for all 2^{22} $DVDcs$ to be illustrated. Our analysis reveals that the number of negative and positive elements are nearly balanced, with a difference of only 812 elements or 0.02%, across all 2^{22} DVD_{cs}. These two features of the TRNG function strongly support random statistical behavior, and explain the high statistical quality presented for the SiRF TRNG algorithm in the experimental results section.

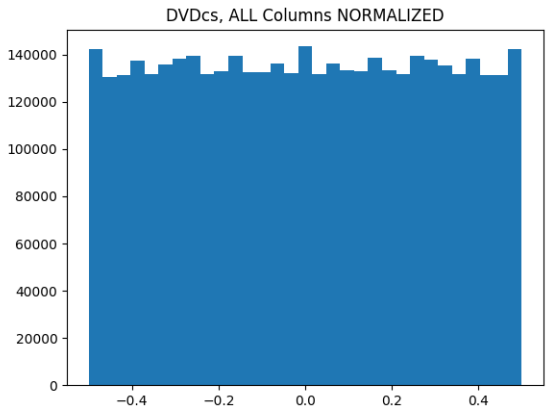

Fig. 6: Distribution of the 2^{22} *normalized* DVD_{cs} generated over all 2,048 iterations of the Sponge Function loop.

4) Bit Generation: Until now, the DVDiff, GPEV, and SF modules absorbed and transformed fixed point values. The final step of our Sponge Function loop, BitGen, squeezes bits from these values, i.e., the DVD_{cs} are processed into a sequence of 2,048 random bits by the *BitGen* module. The *BitGen* module generates a bit value of 0 if the DVD_{cs} is negative, a bit value of 1 if the DVD_{cs} is positive, and alternatives between generating a 0 and 1 in cases where the DVD_{cs} is 0.0. The Sponge Function loop in Phase 2 is repeated for 2,048 iterations before Phase 1 operations are carried out to measure a new set of DV_A and DV_B.

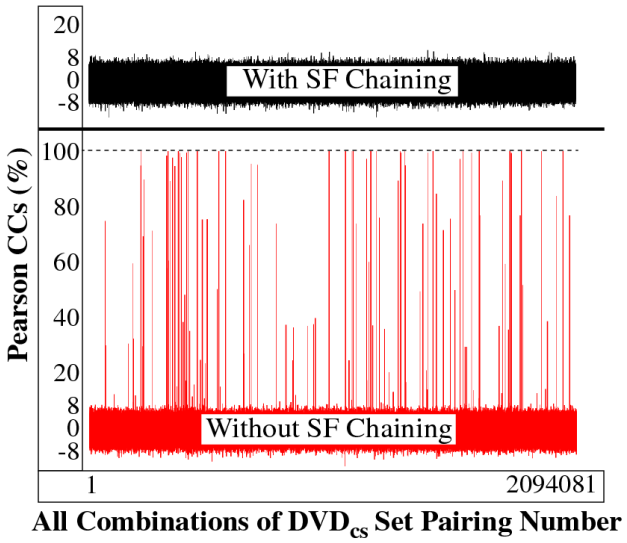

Fig. 7: Pearson's correlation coefficients computed using all combinations of 2,048 element sets of DVD_{cs} generated by Device C_1 over 2,048 iterations of the Sponge Function loop. The PCCs with SF Chaining are shown above those which do not use SF at all.

D. Analysis of Correlation With and without SF Chaining

The SF Chaining operation is critical to achieving high statistical quality in the TRNG bit sequences. We ran NIST and DieHarder statistical test suites on the bit sequences generated with and without SF Chaining enabled, and only the bit sequences with SF chaining passed the tests. The statistical test results with SF Chaining are provided in Section IV.

To better understand why SF Chaining is beneficial to the statistical quality of the bit sequences, we use a standard correlation test metric. The Pearson's Correlation Coefficient (PCC) measures the degree of similarity between two waveforms, and expresses that level over a range between -100% to +100%, where 0% represents no correlation. We use PCC to determine if correlations exist in the 2,048-element DVD_{cs} data sets. In order to comprehensively evaluate all possible sources of correlations, we compute PCCs using all pairing combinations of the DVD_{cs} data sets. The PCCs are plotted in Fig. 7, where it is clear that there are many instances of maximum correlation at 100% without SF Chaining (bottom). In contrast, with SF Chaining enabled (top), no pairing exceeds the value of $\pm$ 10%, which indicates that SF Chaining is effective at removing all correlations that occur when the DV_A and DV_B are reused under all combinations by the DVDiff module.

The cases of 100% correlation occur because some of the pairing sequences controlled by the two 11-bit LFSRs produce DVD sequences that are vertically shifted copies of each other. An intuitive example is fabricated as follows. Assume one DV from the DV_A set is selected, and then the DVD are created by subtracting all of the elements of DVD_B from this DV_A element. Also, assume a second DVD sequence is created in the same fashion but using a different DV_A element. The shapes of two DVD curves are identical and are only different by a vertical offset equal to the difference in the two DV_A set elements. After GPEV is applied, the shift (DC offset) is removed, making the two curves identical, and 100% correlated. Although this fabricated example is not possible when using two LFSRs to select elements, the LFSR pairing algorithm proposed cannot guarantee that all possible differences are generated and therefore, identical but shifted DVD sequences occur.

E. Sponge Construction

The post-processing steps that are chosen and implemented are deliberate in modeling the behavior of a cryptographic sponge construction [19]. Specifically, the proposed approach is a random-permutation sponge based on the iterative nature of SF-chaining. An ideal sponge construction consists of absorption and squeezing phases, where absorption consists of one or more bit-wise operations while squeezing is a pseudo-random bit-space transformation. The two phases can be interleaved across subsets of the bitstring in a *duplex* operating mode. It can be shown that the proposed TRNG achieves both absorption and squeezing properties via careful selection and ordering of the post-processing steps.

The first step which calculates the DV_{Diff} differences pseudo-randomly selects pairs of DV elements from the available sets in the BRAM. While this is a soft-data operation, we can treat them as analogous to one or more bitwise operations (AND, XOR, etc.) applied in series. Note that

bitwise operators do not preclude the presence of collisions, as witnessed in Fig. 7. The GPEV step is an additional absorption step that corrects for temperature and voltage variations. The SF Chaining step is a pseudo-random transformation that consumes values generated during the absorption phase. It is initialized with the original DVD_c values that are subsequently permuted over multiple iterations. In each cycle, the function transforms each DVD by shifting it an arbitrary amount based on a randomly varying offset. The number of iterations, which is determined by the high-order bits of the SF-shifted DVD_{cx}, cannot be predicted *a priori* because of the randomized TCC parameter. This crucial property ensures that the behavior of SF chaining over multiple iterations is equivalent to a pseudo-random permutation over the bit-space of DVD_{cx}. This process completely exhausts the underlying entropy of the stored delay values, while maximizing the throughput of the random bit generation process.

IV. Experimental Results

In this section, we present a statistical analysis using commonly used statistical tests, including NIST SP 800-22 [20], NIST SP 800-90B [21], AIS 31 [22], and DieHarder [23] test suites. The data analyzed is collected from a set of five Zynq 7010 SoCs, installed on Digilent ZYBO boards [24]. For the NIST SP 800-22 tests, we ran the TRNG repeatedly until each board generated 40 one-million bit (40 Mbit) sequences. For the NIST SP 800-90B and AIS 31 tests, we collected 10 MByte sequences from the five boards, while for Dieharder, the amount of data is unknown but in the range of 250 GigaBytes.

A. NIST SP 800-22 Statistical Test Results

The NIST SP 800-22 test suite consists of 15 distinct tests that measure, e.g., the frequency of 0's and 1's across the entire bitstring and locally over blocks of bits in the bit sequence, the length of the runs of 0's and 1's, the longest runs of 0's and 1's, and others referred to as cumulative sums, serial, compression, approximate entropy, etc. The 40 one-million bit TRNG bit sequences collected from each of the five Zynq devices pass all 15 tests, including all but one of the p-value-of-the-p-value tests. There was one instance of a failed non-overlapping template test, where only 36 of the 40 bitstrings passed, which is one less than the required number of 37.

B. NIST SP 800-90B Statistical Test Results

The NIST SP 800-90B test suite is a more recent addition that determines the pass-fail status of the random bit sequence. The 90B test suite evaluates the statistical quality of TRNG bit sequences using a conservative measure of entropy called **min-entropy**, which is traditionally defined as the amount of uncertainty in predicting the most-likely outcome from an entropy source. This NIST test suite estimates min-entropy using tests from two different tracks, the IID-track and the non-IID track, where TRNGs that sample from an Independent and Identically Distributed (IID) distribution employ the IID metrics. To determine the track, NIST recommends applying a set of tests to the TRNG bit sequence to determine if evidence can be found that the samples are not IID. If no evidence is found, then IID can be assumed.

TABLE I: NIST SP 800-90B IID Test Details

Test Name	Evaluation Criteria
Excursion	Measures how far the running sum of sample values deviates from its average value at each point in the bit sequence
Number of Directional Runs	Counts the number of runs of 0s and 1s across consecutive samples
Length of Directional Runs	Computes the length of the longest run across consecutive samples
Number of Increases and Decreases	Counts the maximum number of increases or decreases between consecutive sample values
Number of Runs Based on the Median	Counts the number of runs that are constructed with respect to the median of the input data
Length of Runs Based on Median	Determines the length of the longest run that is constructed with respect to the median of the input data
Average Collision	Counts the number of successive sample values until a duplicate is found
Maximum Collision	Counts the maximum number of successive sample values until a duplicate is found
Periodicity	Determine the number of periodic structures in the data
Covariance	Measures the strength of the lagged correlation
Compression	Length of the compressed sequence (bit sequence is first encoded)

The IID tests are pass-fail, and use a methodology called Permutation Testing. Permutation testing tests a statistical hypothesis in which the test statistic computed on a permuted version of the TRNG bit sequence is compared to the initial (un-permuted) sequence. The assumption is that permuting the initial bit sequence should produce similar test statistics. If *t* represents the value of the test statistic on the original (un-permuted) sequence and *t'* represents the test statistic computed on a permuted version of the original bit sequence, the IID tests count the number of times, over 10,000 permutations, that the value of *t'* is less than *t* (represented as C_0), and the number of times they are equal (represented as C_1). The IID test fails if the sum $C_0 + C_1 <= 5$ or if $C_0 >= 9,995$, indicating a significant difference exists between the permuted sequences and the original sequence. NIST repeats this evaluation using eleven IID statistical tests, listed and briefly explained in Table I. A bit sequence passes the IID tests if and only if all eleven tests pass the count metric, otherwise the bit sequence is deemed non-IID. We subjected bit sequences of length of 10 MBytes for each of the five Zynq devices to the IID test suite and determined that all bit sequences pass the IID tests.

The NIST SP 800-90B test suite additionally estimates the min-entropy, using two distinct sets of tests, one for TRNGs with IID and one for non-IID sources of entropy. For TRNGs with IID outputs, min-entropy is estimated using the most common value estimate. Here, the most common value is determined and then its proportion in the bit sequence-under-test is computed. The upper bound of the corresponding confidence interval is used as the min-entropy per sample estimate. For binary data, the most common value is either 0 or 1 and the proportion is simply the larger fraction of 0s or 1s in the bit sequence. The values obtained for the five Zynq devices are shown in the first row of Table II, which shows

TABLE II: IID and Non-IID Test Results of the NIST SP 800-90B Entropy Estimation

Estimator	Min Entropy				
	C58	**C60**	**C61**	**C62**	**C63**
Most Common Value	0.99949	0.99954	0.99949	0.99947	0.99958
Collision	0.97489	0.97731	0.97237	0.95915	0.97095
Markov	0.99986	0.99995	0.99988	0.99964	0.99984
Compression	0.94808	**0.94156**	0.94419	0.96217	0.94891
t-Tuple	**0.94499**	0.94499	**0.94219**	**0.94358**	**0.94643**
LRS	0.99119	0.99895	0.97301	0.99707	0.99845
Multi MCW Prediction	0.99989	0.99978	0.99979	0.99992	0.99958
Lag Prediction	0.99947	0.99964	0.99961	0.99926	0.99926
MultiMMC Prediction	0.99970	0.99984	0.99954	0.99953	0.99941
LZ78Y Prediction	0.96743	0.99983	0.99956	0.99938	0.99948

TABLE III: NIST SP 800-90B Non-IID and AIS 31 Test Descriptions

Test Name	Evaluation Criteria
NIST SP 800-90B	
Most Common Value	Measures the frequency of the most common value
Collision	Measures the frequency of repeated values (collisions)
Markov	Measures the degree of predictability based on past output values
Compression	Measures the degree of compression possible
t-Tuple	Evaluates how often sequences of t consecutive symbols repeat
LRS	Detects redundancy by finding the longest repeated substring
MultiMCW Prediction	Evaluates predictability by analyzing symbol patterns using multiple "Most Common in Window" predictors
Lag Prediction	Analyzes the ability to predict output based on previous outputs at different lags
MultiMMC Prediction	Considers multiple Markov chains and evaluates how well they can predict future values
LZ78Y Prediction	Uses the LZ78 compression method to measure predictability of the sequence
AIS-31	
Disjointness	Ensures outputs from the entropy source are independent across multiple tests
Monobit	Verifies the balance of 0s and 1s in the bitstream
Poker	Analyzes frequency distributions to identify patterns in the data
Runs	Checks the randomness of bit sequences by analyzing runs of consecutive 0s and 1s
Long Run	Identifies overly long runs of consecutive 0s or 1s
Autocorrelation	Evaluates the dependency between bits separated by fixed lags
Uniform Distribution	Checks if output values follow a uniform distribution
Homogeneity	Tests consistency of symbol distributions across subsets of data
Entropy Estimation	Measures the unpredictability of the source output

that nearly a full bit of entropy is contained in each output bit.

We also ran the non-IID test suite to estimate min-entropy, despite the fact that the bit sequences pass the IID tests. The non-IID test suite conservatively estimates min-entropy using a diverse battery of tests, which are listed and briefly explained in the top portion of Table III. The smallest min-entropy estimate obtained from one of these tests is used as the estimate for the bit sequence. The results of applying the non-IID test suite to the 10 MByte bit sequences obtained from the five Zynq devices are shown in the remaining rows of Table II, where we have highlighted in bold font the worst-case min-entropy test results. The worst-case values vary between 0.941 to 0.946, which suggests the SiRF TRNG produces a high-quality random bit sequence.

TABLE IV: Results of the AIS 31 Statistical Test Suite

Test	Pass rate	Result
T0 - Disjointness test	1/1	Pass
T1 - Monobit test	257/257	Pass
T2 - Poker test	257/257	Pass
T3 - Runs test	257/257	Pass
T4 - Long run test	257/257	Pass
T5 - Autocorrelation test	257/257	Pass
Test	**Test statistic / Pass condition**	**Result**
T6 - Uniform distribution test	$\lvert\mathbb{P}(1) - 0.5\rvert = 0.00235 / < \mathbf{0.025}$ $\lvert\mathbb{P}(01) - \mathbb{P}(11)\rvert = 0.00306 / < \mathbf{0.020}$	Pass
T7 - Test for homogeneity	$T[0] = 3.329; T[1] = 2.165; / < \mathbf{15.13}$ $T[00] = 1.670; T[01] = 3.281;$ $T[10] = 0.077; T[11] = 0.022; / < \mathbf{15.13}$	Pass
T8 - Entropy estimation	$H_1 = 8.00169 / \geq \mathbf{7.976}$	Pass

C. AIS 31 Statistical Test Results

As part of our evaluation process, we applied the AIS 31 test suite as additional validation of the SiRF TRNG. A brief description of the AIS 31 tests is provided in the lower portion of Table III.

The results obtained after applying the AIS 31 tests to the 10 MByte bit sequence generated from one of the Zynq devices are shown in Table IV, which shows that all tests passed. The pass-fail results for the other boards are identical. Similar to the claims made using the NIST SP 800-90B test suite, these results also validate that the SiRF TRNG produces a high-quality random bit sequence.

D. DieHarder Statistical Test Results

For completeness, we generate random bit sequences from the five Zynq board and use them as input to the DieHarder test suite [23]. The DieHarder tests are applied by redirecting the SiRF PUF-TRNG bit sequences directly to the DieHarder test tool. Given the bit generation rate is 2.67 Mbits per second, the five Zynq devices used in the experiments ran for more than 20 days before enough bits were generated to run all 116 DieHarder tests. No test failures occurred for any test and for any of the five bit sequences generated by the Zynq devices, and only 18 'WEAK' results were observed. For comparison, we also subjected the random bit sequences generated by the OpenSSL pseudo-random number generator to the DieHarder test suite in five separate experiments and observed no failures, and only 19 'WEAK' results. Therefore, the quality of the SiRF TRNG bit sequences is similar to those produced by OpenSSL.

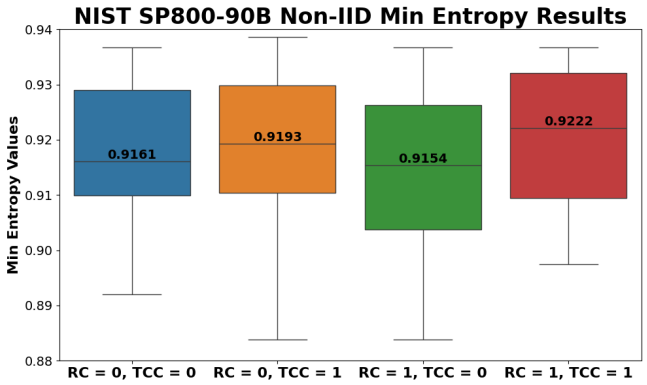

Fig. 8: Box plots showing the NIST SP 800-90B Non-IID min-entropy values from 25 boards. The horizontal axis represents the different configurations of Range Constant (RC) and Trim Code Constant (TCC), indicating whether each is turned ON or OFF. The median value for each configuration is displayed above the corresponding boxplot.

E. Statistical Analysis of Dynamic Entropy

As indicated earlier, the nonces used to randomize parameters to the 64-bit LFSR Challenge generator and Sponge Function modules represent the dynamic entropy component of the TRNG. We collected 10 sequences of 100,000 bits from the five devices and ran the NIST SP 800-22 test suite on the sequences. All applicable NIST statistical tests passed, as well as the p-value-of-the-p-value tests. Therefore, the distillation operation, which XORs 12 consecutive low-order bits of the path delay measurements, is capable of generating bit sequences of high statistical quality.

F. Analysis of RC and TCC SiRF PUF-TRNG Parameters

We carry out a special set of tests that evaluate the benefit of randomizing two parameters used by the Sponge Function loop shown in Fig. 1, namely, the Range Constant (RC) in the GPEV module and the Trim Code Constant (TCC) in the SF module. The DV used in these evaluations are held constant across the experiments while the RC and TCC parameters are either randomly varied, e.g., RC = 1 or held constant, e.g., RC = 0, as shown by the labels along the bottom of the box plot graph shown in Fig. 8. The box plot characterizes the min-entropy results from the NIST SP 800-90B test suite for bit sequences generated by the SiRF PUF-TRNG algorithm using DV collected from 25 devices. The best result is obtained when both RC and TCC are randomly varied, as exemplified by the larger median value for the right-most box plot when compared with the others. From the results, it is clear that the improvement is marginal, suggesting that most of the min-entropy is obtained by the SF Chaining operation. Despite the small improvement, randomizing these parameters adds uncertainty to the processing operations within the Sponge Function loop, and therefore, improves attack resilience.

TABLE V: Comparison with Other FPGA TRNG Designs

Design	Device	Entropy Est.	Area	ThrPut [Mbps]	Power [mW]
SiRF	Zynq 7010	≥ 0.999	5842 LUTs 4377 FFs 32 CARRY4s	2.5	27
TROT [17]	Zynq 7000	≥ 0.999	32 LUTs 55 FFs 17 CARRY4s	12.5	9.5
CHAOS [18]	Virtex-6	≥ 0.983	53 LUTs 22 FFs	1600	2.05

G. Comparison with other TRNGs

In this section, we compare the statistical test results of the SiRF PUF-TRNG with two stand-alone TRNGs proposed in [17] and [18]. These particular TRNGs are referenced because the authors apply the NIST SP 800-90B and AIS 31 test suites to their bit sequences. We were not able to find any unified PUF-TRNG architecture papers that provided a comprehensive evaluation using these test suites. The comparison in Table V shows that the SiRF PUF-TRNG has a smaller bit generation rate, but compares favorably in terms of the other metrics. The larger footprint associated with the SiRF PUF-TRNG is due to the inclusion of the PUF security function.

H. SiRF PUF-TRNG Bit Generation Rate and Resource Utilization

The path timing operation carried out in Phase 1 (from Fig. 1) takes approximately 50 milliseconds using a 50 MHz clock frequency, while the linear operations carried out in Phase 2 by the four post-processing modules are able to execute in approximately 600 microseconds per iteration, yielding a bit generation rate of approximately 2.67 Mbits per second. Note that increasing the clock frequency would increase the bit generation rate proportionally, e.g., 100 MHz would double the rate.

The SiRF PUF-TRNG implementation on the Zynq FPGAs utilizes 5,842 LUTs and 4,377 FFs, and requires two DSP primitives to implement multiplication in two modules of the PUF-TRNG algorithm. Resource utilization is nearly identical (within 5%) to the utilization required for the SiRF PUF stand-alone. The BRAM utilization is 24 KBytes, which is 4,096 bytes larger than that required by SiRF PUF stand-alone. The additional 4,096 bytes are needed to accommodate the SF Chaining operation.

V. Conclusions

In this paper, we present a unified SiRF PUF-TRNG architecture that utilizes static entropy from a strong PUF and dynamic entropy derived from path delay noise. The novel inclusion of a soft-data sponge function enhances randomness and efficiency, while resilience against temperature-voltage attacks is achieved through the GPEV post-processing module. Extensive evaluation using the NIST SP 800-22, NIST SP 800-90B, AIS 31, and DieHarder test suites demonstrates the TRNG's robust statistical performance and high min-entropy.

The architecture demonstrates a compact and resource-efficient approach, reusing over 95% of the SiRF PUF's components, with a bit generation rate of 2.67 Mbps and minimal resource overhead. The proposed approach advances the integration of PUF and TRNG capabilities, providing a reliable and scalable solution for secure hardware systems.

REFERENCES

[1] A. Maiti, R. Nagesh, A. Reddy, and P. Schaumont, "Physical unclonable function and true random number generator: A compact and scalable implementation," in *GLSVLSI*, 2009, p. 425–428.

[2] C. Martínez-Gómez and I. Baturone, "Calibration of ring oscillator PUF and TRNG," in *ECCTD*, 2020, pp. 1 – 4.

[3] S. Larimian, M. R. Mahmoodi, and D. B. Strukov, "Lightweight integrated design of puf and trng security primitives based on eflash memory in 55-nm cmos," *Trans. on Electron Devices*, vol. 67, no. 4, pp. 1586–1592, 2020.

[4] S. Taneja, V. K. Rajanna, and M. Alioto, "36.1 unified in-memory dynamic trng and multi-bit static puf entropy generation for ubiquitous hardware security," in *ISSCC*, vol. 64, 2021, pp. 498–500.

[5] G. E. S. Charles W. O'Donnell and S. Devadas, "Puf-based random number generation," in *MIT CSAIL CSG TM 481*, 2004, pp. 1 – 4.

[6] J. Plusquellic, "Shift Register, Reconvergent-Fanout (SiRF) PUF Implementation on an FPGA," *Cryptography*, vol. 6, no. 4, 2022. [Online]. Available: https://www.mdpi.com/2410-387X/6/4/59

[7] Y. Cao, W. Liu, Y. Zheng, S. Chen, J. Ye, L. Qian, and C.-H. Chang, "A new reconfigurable true random number generator and physical unclonable function unified chip with on-chip auto-calibration." *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS I-REGULAR PAPERS*, 2023.

[8] B. Liu, J. Ma, H. H. Tai, D. Verma, M. Sahoo, Y.-F. Chang, H. Liang, S. Feng, L.-J. Li, T.-H. Hou, and C.-S. Lai, "Memristive true random number generator with intrinsic two-dimensional physical unclonable function." *ACS APPLIED ELECTRONIC MATERIALS*, 2023.

[9] K. Pratihar, U. Chatterjee, M. Alam, D. Mukhopadhyay, and R. S. Chakraborty, "A tale of twin primitives: Single-chip solution for PUFs and TRNGs," Cryptology ePrint Archive, Paper 2021/1067, 2021.

[10] S. Sánchez-Solano, L. F. Rojas-Muñoz, M. C. Martínez-Rodríguez, and P. Brox, "Hardware-Efficient Configurable Ring-Oscillator-Based Physical Unclonable Function/True Random Number Generator Module for Secure Key Management." *Sensors*, vol. 24, no. 17, p. 5674, 2024.

[11] S. K. Satpathy, S. K. Mathew, R. Kumar, V. Suresh, M. A. Anders, H. Kaul, A. Agarwal, S. Hsu, R. K. Krishnamurthy, and V. De, "An all-digital unified physically unclonable function and true random number generator featuring self-calibrating hierarchical von neumann extraction in 14-nm tri-gate CMOS." vol. 54, pp. 1074 – 1085, 2019.

[12] E. I. Vatajelu, G. Di Natale, and P. Prinetto, "Security primitives (puf and trng) with stt-mram," in *VTS*, 2016, pp. 1–4.

[13] M. N. I. Khan, C. Y. Cheng, S. H. Lin, A. Ash-Saki, and S. Ghosh, "A morphable physically unclonable function and true random number generator using a commercial magnetic memory," in *ISQED*, 2020, pp. 197–197.

[14] S. S. Zalivako and A. A. Ivaniuk, "The use of physical unclonable functions for true random number sequences generation." *Automatic Control and Computer Sciences*, vol. 47, no. 3, pp. 156 – 164, 2013.

[15] A. Sadr and M. Zolfaghari-Nejad, "Physical unclonable function (puf) based random number generator." *Advanced Computing: An International Journal*, vol. 3, pp. 139 – 145, 2012.

[16] B. F. Cambou, "Design of true random numbers generators with ternary physical unclonable functions." *Advances in Science, Technology and Engineering Systems Journal*, vol. 3, pp. 15 – 29, 2018.

[17] M. Grujić and I. Verbauwhede, "TROT: A Three-Edge Ring Oscillator Based True Random Number Generator With Time-to-Digital Conversion," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 6, pp. 2435–2448, 2022.

[18] Y. Luo, W. Wang, S. Best, Y. Wang, and X. Xu, "A High-Performance and Secure TRNG Based on Chaotic Cellular Automata Topology," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 12, pp. 4970–4983, 2020.

[19] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Sponge-based pseudo-random number generators," in *Cryptographic Hardware and Embedded Systems, CHES 2010*, S. Mangard and F.-X. Standaert, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 33–47.

[20] A. Rukhin, S. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, N. Heckert, J. Dray, S. Vo, and L. Bassham, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," National Institute of Standards and Technology, Special Publication 800-22 Revision 1a, 2010. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-22r1a

[21] M. Turan, E. Barker, J. Kelsey, K. McKay, M. Baish, and M. Boyle, "Recommendation for the entropy sources used for random bit generation," National Institute of Standards and Technology, Special Publication 800-90B, 2018. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-90B

[22] W. Killman and W. Schindler, "A proposal for: Functionality classes for random number generators," Bundesamt für Sicherheit in der Informationstechnik (BSI), Technical Guideline, Sept. 2011.

[23] R. G. Brown, D. Eddelbuettel, and D. Bauer, "DieHarder: A Random Number Test Suite," https://webhome.phy.duke.edu/ rgb/General/dieharder.php.

[24] *ZYBO Reference Manual*, 2014. [Online]. Available: https://digilent.com