Jitter-based Adaptive True Random Number Generation Circuits for FPGAs in the Cloud

XIANG LI, University of Massachusetts Amherst, MA, USA
PETER STANWICKS, University of Massachusetts Amherst, MA, USA
GEORGE PROVELENGIOS, University of Massachusetts Amherst, MA, USA
RUSSELL TESSIER, University of Massachusetts Amherst, MA, USA
DANIEL HOLCOMB, University of Massachusetts Amherst, MA, USA

In this paper, we present and evaluate a true random number generator (TRNG) design that is compatible with the restrictions imposed by cloud-based Field Programmable Gate Array (FPGA) providers such as Amazon Web Services (AWS) EC2 F1. Because cloud FPGA providers disallow the ring oscillator circuits that conventionally generate TRNG entropy, our design is oscillator-free and uses clock jitter as its entropy source. The clock jitter is harvested with a time-to-digital converter (TDC) and a controllable delay line that is continuously tuned to compensate for process, voltage, and temperature variations. After describing the design, we present and validate a stochastic model that conservatively quantifies its worst-case entropy. We deploy and model the design in the cloud on 60 EC2 F1 FPGA instances to ensure sufficient randomness is captured. TRNG entropy is further validated using NIST test suites, and experiments are performed to understand how the TRNG responds to on-die power attacks that disturb the FPGA supply voltage in the vicinity of the TRNG. After introducing and validating our basic TRNG design, we introduce and validate a new variant that uses four instances of a linkable sampling module to increase the entropy per sample, and improve throughput. The new variant improves throughput by 250% at a modest 17% increase in CLB count.

ACM Reference Format:
Xiang Li, Peter Stanwicks, George Provelengios, Russell Tessier, and Daniel Holcomb. 2022. Jitter-based Adaptive True Random Number Generation Circuits for FPGAs in the Cloud. *ACM Trans. Reconfig. Technol. Syst.* 1, 1, Article 1 (January 2022), 20 pages. https://doi.org/10.1145/3487554

1 INTRODUCTION

Random numbers are fundamental to cryptographic systems and widely used for generating keys, nonces, and initialization vectors. The quality of randomness required in these applications necessitates the use of TRNGs. TRNGs exploit the inherent physical properties of the system in which they are embedded to generate statistically random and unpredictable numbers. This characteristic makes the outputs of a TRNG unpredictable even to an adversary that knows the current state of the circuit. Physical sources of entropy commonly used in FPGAs by on-chip TRNGs include thermal noise and clock or oscillator jitter. The randomness of the numbers created by TRNGs is typically evaluated using stochastic models and statistical tests [1].

Authors' addresses: Xiang Li, University of Massachusetts Amherst, MA, USA, xiang@umass.edu; Peter Stanwicks, University of Massachusetts Amherst, MA, USA, pstanwicks@umass.edu; George Provelengios, University of Massachusetts Amherst, MA, USA, gprovelengio@umass.edu; Russell Tessier, University of Massachusetts Amherst, MA, USA, tessier@umass.edu; Daniel Holcomb, University of Massachusetts Amherst, MA, USA, dholcomb@umass.edu.

TRNGs are widely used in real-world applications with differing throughput requirements, ranging from one-time pads [2] and keys for web authentication which have low throughput requirements to initialization vector (IV) generation for block ciphers in storage drives or internet protocol packets which can have higher throughput requirements. For example, data passed from a host to a self-encrypting storage drive needs encryption/decryption before being read/written [3]. In this case, the throughput and quality of the TRNG affects system performance, because the latency will increase if the IV generation speed can not keep up with the read/write speed of the drive. To support applications such as these which require random numbers, several works have studied TRNG on FPGAs [4][5][6][7][8] in recent years.

FPGAs are increasingly being used in cloud-based systems for prototyping and acceleration, and to support secure soft processors [9] which require a source of random numbers. Yet to protect their infrastructure from malicious voltage attacks [10], cloud providers such as AWS impose restrictions on the types of circuits that are allowed on their FPGAs. Circuits that deviate from standard digital design flows, including logic-driven clocks and combinational loops as found in ring oscillators (ROs), are detected during bitstream compilation and disallowed from being loaded onto the FPGA [11]. This restriction causes difficulty in creating and characterizing jitter-based TRNG circuits for cloud applications. Despite this unique restriction on cloud-FPGAs, TRNGs on FPGAs must nonetheless be designed to work correctly when deployed across multiple instances, must be supported by a stochastic model to validate the entropy claims, and should be robust against external perturbations.

In this work, we extend our previous conference manuscript from FPT 2020 [12] to present a TRNG design and validation procedure that is tailored around the restrictions of cloud-based FPGAs. Our design is able to harvest jitter without creating oscillators, applicable to multiple cloud-FPGA instances, and adaptable to differences in clocks. The design adjusts to changing environmental conditions and can be characterized without requiring ground truth delay measurements that are commonly obtained by counting oscillations. We make several specific contributions in this work:

- A TRNG for Virtex UltraScale+ FPGAs used in the cloud is detailed, implemented, and analyzed across numerous AWS EC2 F1 instances. The design, which is based on tunable delay chains and a TDC that harvests entropy from clock jitter, avoids primitives such as combinational loops that are common in TRNGs but disallowed by AWS and other cloud providers.
- A novel procedure is proposed for computing the min-entropy per sample using a stochastic model. The model empirically relates component delays to clock jitter by least-squares fitting. The delays that are independently computed by the model during entropy evaluation strongly correlate to the delays from the FPGA's own timing report, which supports the validity of the delay values that are inferred by our approach.
- The robustness of our TRNG is evaluated by implementing a voltage attack against it on F1 and showing how the TRNG adjusts in response to the attack without compromising its ability to create random numbers. Demonstrating resilience to environmental changes is important for a TRNG that will be used in cloud settings, and is a novel feature of the work.
- Beyond our basic TRNG design, we introduce a new approach based on linkable sampling modules, that can be instantiated and connected together to increase the entropy per sample without requiring additional tuning. The new variant of the TRNG improves throughput and min-entropy according to the stochastic model by 250% and 270% respectively at a modest 17% increase in slice count by sharing the control unit and tunable delay elements across the linkable sample modules.

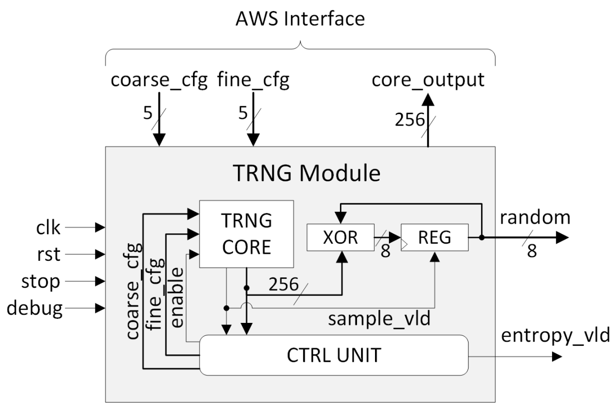


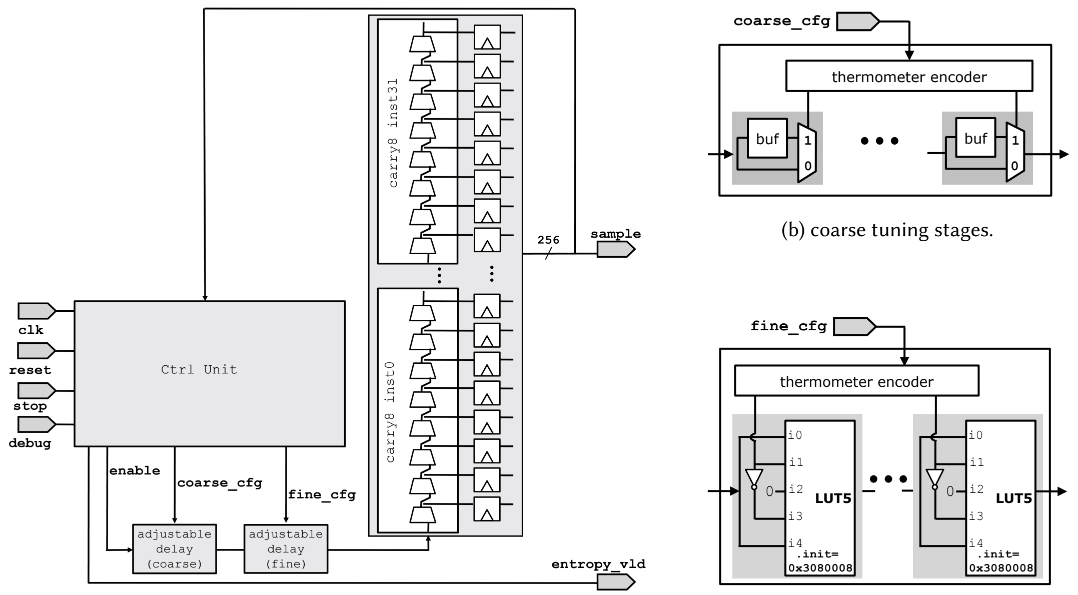
Fig. 1. Structure of TRNG design and interface.

The remainder of this paper is structured as follows. Section 2 provides background on previous FPGA TRNG approaches. Section 3 describes the structure of our TRNG and modeling is discussed in Section 4. Sections 5 and 6 evaluate and discuss the TRNG entropy, resilience, and costs. Section 7 introduces and evaluates the TRNG with linkable sampling module. Section 8 concludes the paper.

2 BACKGROUND AND RELATED WORK

Several works on cloud security have shown the demand for TRNGs on cloud-FPGAs. Zeitouni et al. [13] propose a scheme to protect clients' IPs while checking rogue circuits for cloud-FPGA vendors. Their scheme requires that the trusted shell on an FPGA has a TRNG to generate a nonce that is sent to the client to compute a proof of authenticity. Wolfe et al. [14] perform secret sharing Multi-Party Computation (MPC) on FPGAs in the datacenter. In the implementation of their MPC protocol on AWS FPGAs, a random key for each party needs to be generated and shared with one other party. The implementation of TRNGs in FPGAs has been widely studied, although none of the prior approaches comprehensively address the unique challenge of cloud FPGAs as well as the general TRNG requirements listed in Section 1. A large majority of these previous implementations rely on ROs to generate high-frequency signals that exhibit significant jitter. For example, Kohlbrenner and Gaj [15] use two ROs and a sampling circuit to measure jitter and Maiti et al. [16] deploy up to 128 ROs to amplify uncertainty. Some TRNGs augment ROs with delay paths to increase timing sensitivity. Like our approach, Rozic et al. [17] and Yang et al. [4] use carry logic-based delay chains to assist with entropy extraction. These approaches do not include tunable delays to combat environmental factors and an RO is used to excite the delays.

Several non-RO based TRNGs have been built for FPGAs, but they also have limitations that make them inappropriate for cloud deployment. Majzoobi et al. [6] use programmable delay lines built from lookup tables (LUTs) that can be difficult to characterize on a per-FPGA basis. Deák et al. [18] use the jitter from an on-FPGA phase locked loop (PLL) to create a TRNG using clock settings that are a challenge to replicate in a cloud setting. Perhaps the most similar TRNG approach to ours [19] uses a standard clock input, tunable delay buffers, and a delay path. However, unlike our approach, the delay path is made from a chain of LUTs which have variable logic and routing delays across stages.



(a) TDC chain.

(b) coarse tuning stages.

(c) fine tuning stages.

Fig. 2. Components of TRNG core and control unit.

3 STRUCTURE OF PROPOSED TRNG

Fig. 1 shows the top-level view of our TRNG design. It is a hardware module that serially generates 8-bit random numbers. We instantiate the TRNG module within a hardware testbench for analysis. The design is created for AWS EC2 F1 instances, which contain Xilinx Virtex UltraScale+ VU9P FPGAs. The bitstream is generated using Amazon's Hardware Development Kit (HDK), and then converted to an Amazon FPGA Image (AFI) that is reused for deployment across F1 instances. Amazon provides a runtime tool to interact with the deployed design by reading and writing 32-bit data to or from user-defined registers using AXI4 over PCIe. We make the signals at the top of Fig. 1 accessible to the runtime tool only when the TRNG hardware is set to debug mode. The debug mode allows us to control the TRNG and observe sample values from the TRNG core, which is useful for data collection and analysis in the cloud, but would be insecure if enabled in production.

Internally, the TRNG module gets entropy from the TRNG core, and hashes it into a local entropy pool by XOR operation. The control unit keeps a conservative estimate of the current entropy in the pool by counting the number of valid samples provided to it. Once enough entropy is collected, a signal from the control unit is asserted and the 8-bit random value in the registers can be read out, upon which the entropy count is reset to 0. We now describe in more detail the components of the TRNG core (Fig. 2).

3.1 Carry Chain Description

The TDC in our circuit (Fig. 2a) consists of 32 8-bit carry stages, and each output bit from the carry chain is the data input to a D flip-flop in the same slice. The controller repeatedly generates a single rising edge that propagates into the carry chain with appropriate delay such that it will be propagating through the carry chain when the next rising clock edge occurs. The number of 1-values captured in the 256 flip-flops, i.e. the Hamming weight of the sample, is an indication of how far up the chain the rising edge has propagated by the time of the rising clock edge. The Hamming weight of samples will fluctuate slightly in each trial due to clock jitter, which is our source of randomness.

3.2 Tunable Delay Elements and Feedback Control

Our circuit uses tunable delay elements and feedback to ensure that the rising edge from the delay line is within the TDC chain when the clock arrives. The control unit measures Hamming weight by summing the sampled values captured in the flip-flops. Increasing the propagation delay will cause the rising edge to reach fewer TDC stages and thereby will reduce the Hamming weight of samples. Similarly, decreasing the delay will increase the Hamming weight. In this way the tunable delay circuits are the knob used for adjusting the Hamming weight. The control unit uses the delay knob to position the rising edge in the TDC chain during clock arrival, as is required to generate randomness.

Ideally, the Hamming weight of the samples should be centered at around 128 in a 256-stage delay chain, which gives a maximum margin against delay changes in either direction that could detune the circuit. The control unit adjusts the coarse-tuning and fine-tuning settings for the next sample based on the Hamming weight of the current sample using the simple feedback scheme of Eq. 1, where (c, f) and (c', f') are the current and next values of the coarse and fine tuning settings, and HW is the Hamming weight of the current sample; note in Eq. 1 that f_{mid} represents the middle setting for fine tuning, which in our case is 15. Furthermore, with each sample, the control unit credits entropy to the entropy pool only when the Hamming weight is between 30 and 225 so that samples are not counted as random if the circuit becomes detuned and the rising edge is approaching either end of the carry chain where jitter may not be captured. Once enough samples are collected, the control unit asserts a signal to indicate the generation of an 8-bit random number is complete, which will be further explained in Section 3.3. During testing, the control unit is able to configure the TRNG tuning manually using values provided through AWS interface, and to return the resulting samples via the same interface.

$$(c', f') = \begin{cases} (c+1, f_{mid}) & \text{if } 208 \leq HW \\ (c, \quad f+1) & \text{if } 158 \leq HW < 208 \\ (c, \quad f-1) & \text{if } \;\, 48 < HW \leq 98 \\ (c-1, f_{mid}) & \text{if } \qquad\;\; HW \leq 48 \end{cases} \tag{1}$$

The coarse and fine tuning stages are implemented as follows. Each coarse tuning stage adds or bypasses a LUT1 primitive that implements a logical buffer, as shown in Fig. 2b. Each fine-tuning stage selects between a shorter and longer pin-to-pin delay of a LUT5, where the enabled path through the LUT is set by the control input, as shown in Fig. 2c. The stages are controlled using thermometer encoding, so that incrementing or decrementing their configuration settings will change only one stage along the delay line, which helps ensure predictable control but has higher area cost than a binary-encoded tunable delay in which each stage has twice the delay of the next. Fig. 3 shows the Hamming weight of samples for all combinations of tuning; note that debug mode is used to generate this plot, as it overrides the feedback of the controller, and allows the samples from the TRNG core to be logged.

3.3 Post-processing Circuit

The 256-bit samples from the TRNG core are hashed into the 8-bit state of the entropy pool using a simple scheme as shown in Fig. 4. Each update includes a 1-bit circular shift of the 8-bit entropy pool, which ensures that randomness

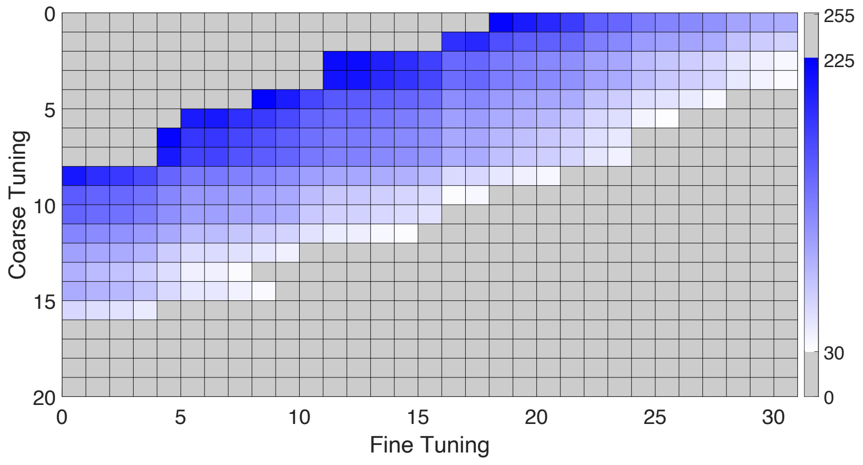


Fig. 3. Heatmap showing the Hamming weight of samples on a single F1 instance for all possible tuning settings. Increasing the coarse or fine tuning reduces the Hamming weight. Sampled values indicative of a poorly-tuned TDC, colored gray, would not cause the entropy count to be incremented.

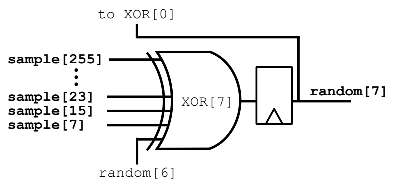


Fig. 4. 256-bit samples are hashed into the 8-bit random signal which is the entropy pool. The hashing uses eight XOR gates, all configured in the same manner as the one that is shown.

will get distributed through the 8 bits even if always coming from the same position in the 256-bit sample. We have used this particular scheme for simplicity, but it could be replaced with any number of other hash functions for the same effect. A counter tracks the number of valid samples produced by the TRNG core, and requires that 80 valid samples are hashed into the entropy pool before it is considered to be random, which assumes 0.1 bits of entropy per sample. The actual value of entropy per sample should exceed the assumed entropy per sample, but the specific value of 0.1 bits is chosen somewhat arbitrarily as a conservative assumption. If the entropy per sample is assumed to be higher, then the circuit will need fewer valid samples before deciding that the TRNG has accumulated enough entropy to produce an 8-bit random output. This assumption therefore has an impact on the throughout of the TRNG. Section 4 of the paper will show that the actual entropy per sample exceeds this conservative estimate by a factor of more than 2 using both a stochastic model and NIST tests.

4 MODELING OF TRNG

The randomness of the TRNG comes from the samples of the delay line in the TDC. Even if the propagation delay of the delay line does not change across trials, the TDC can produce different samples if it has fine enough time resolution and its sampling clock has sufficient jitter. The time resolution on the TDC is a consequence of the low propagation

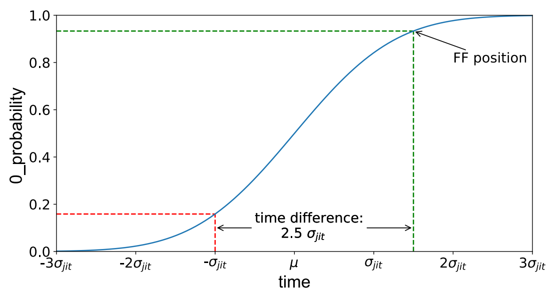


Fig. 5. TDC position in normal distribution CDF for modeling.

delay of the stages in the hard carry chains of Xilinx Ultrascale+ devices. A larger clock jitter or finer time resolution will both have the same effect of making the TDC samples more random because both changes will make the jitter relatively larger in comparison to the TDC time resolution. Accordingly, the relevant consideration for modeling a TRNG such as ours is to quantify how the amount of jitter compares to the time resolution of the TDC.

In our stochastic model of the TRNG, we relate jitter to TDC time resolution without relying on conservative timing reports or jitter estimates. We rely in modeling only on the simplifying assumption that jitter is normally distributed, which is in particular consistent with the jitter component caused by thermal noise [20] [21]. We also assume its standard deviation is invariant with respect to the tuning settings of the delay line. From this we calculate the time resolution of each TDC stage in terms of the standard deviation of jitter, which we use as the unit delay in our model. After calculating the time resolution of each stage, we use the same model to calculate a lower bound on min-entropy per sample. The next subsection describes our modeling approach.

The bits produced by any TRNG design should be as random as the bits produced by any other, unless one of the designs is fundamentally flawed. The stochastic model assures the randomness by validating the claimed entropy of the source. If a source has lower entropy, its TRNG will still produce fully random outputs, but it will require more samples from the source to accumulate the required amount of entropy for each random output value.

4.1 Empirical Model Relating TDC Delay to Jitter

In a given trial, the flip-flop associated with each TDC stage will sample a 0 value if its clock arrives before its rising data input from the delay line, and will sample a 1 otherwise. If the delay tuning settings are held constant across samples, the 0-probability (1-probability) of the stage indicates the proportion of trials in which its clock arrives before (after) its rising data input from the delay line.

If the flip-flop of stage i samples 0 with probability 0.159, then 15.9 percent of clock edges arrive there before the rising data input, and 84.1 percent of clock edges arrive after. Under the assumption that jitter is normally distributed, the observation that 15.9 percent of clock edges arrive before the rising data input reveals that rising data input coincides with the clock being $-1.0 * \sigma_{jit}$ away from its mean value, because $\Phi^{-1}(0.159) = -1.0$, where Φ^{-1} is the inverse CDF of a normal distribution. This scenario is depicted graphically in Fig. 5.

In these same trials, if the flip-flop of stage j samples 0 with probability 0.933, then we similarly conclude that its rising data input coincides with its clock being $+1.5 * \sigma_{jit}$ away from its mean because $\Phi^{-1}(0.933) = 1.5$. If there is no clock skew between the flip-flops of i and j, then these two findings together indicate that the time difference between the rising data inputs on i and j is equal to $2.5 * \sigma_{jit}$. If clock skew is allowed, then we generalize the claim slightly to more formally conclude only the difference in criticality (i.e. timing slack) between the two flops is equal to $2.5 * \sigma_{jit}$, although for our purposes it is actually the criticality that matters so we need not worry about skew. The delay or criticality difference between any two stages can therefore be estimated from their 0-probabilities in a set of trials. Because the estimate is noisy when the associated 0-probabilities are close to 0 or close to 1, we apply it only when the 0-probabilities indicate that both stages are within $\pm 2\sigma_{jit}$ of their means. Note that the delay difference is being calculated in units of σ_{jit} even though the value of σ_{jit} is not known in absolute terms. Using this approach, two flip-flops i and j have arrival times (denoted T_i and T_j) that are related as shown in Eq. 2, where $\widehat{P_i}$ and $\widehat{P_j}$ are their respective 0-probabilities for a particular tuning setting.

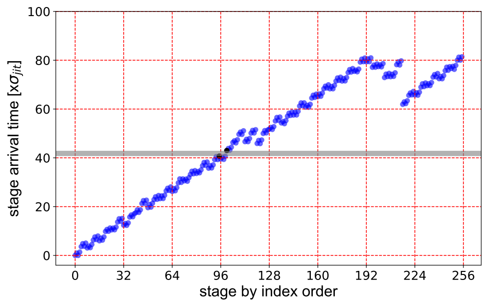
$$T_i - T_j = \left(\Phi^{-1}(\widehat{P_i}) - \Phi^{-1}(\widehat{P_j})\right) \sigma_{jit} \tag{2}$$

4.2 Calculating Stage Arrival Times

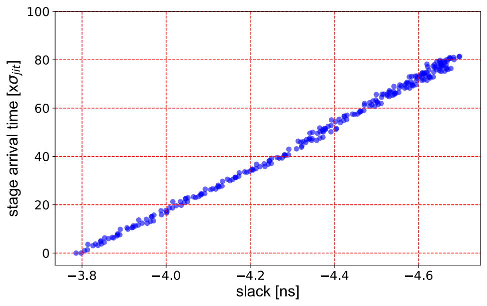
Given that 0-probabilities from each tuning setting will relate the arrival times of some of the TDC stages, and that the same stages can be related to each other by multiple different tunings, we can solve a set of equations to obtain the arrival time T_i for each stage i . The set of equations is as described in Eq. 3 where T is the n-element column vector of unknown arrival times, A is the m-by-n matrix of coefficients in which all entries are 0 except for a single +1 and single -1 in each row for the two stages that are related, and B is an m-by-1 column vector of the arrival time differences, calculated as shown on the right hand side of Eq. 2. We then find the least-squares solution to $AT = B$ (Eq. 3) which gives stage arrival times in terms of σ_{jit}. Because the formulation deals with differences in arrival times, we adopt the convention that T_0, the arrival time of the first stage, is 0; other T_i values therefore represent the arrival time of stage i relative to first stage.

$$AT = B\sigma_{jit} \tag{3}$$

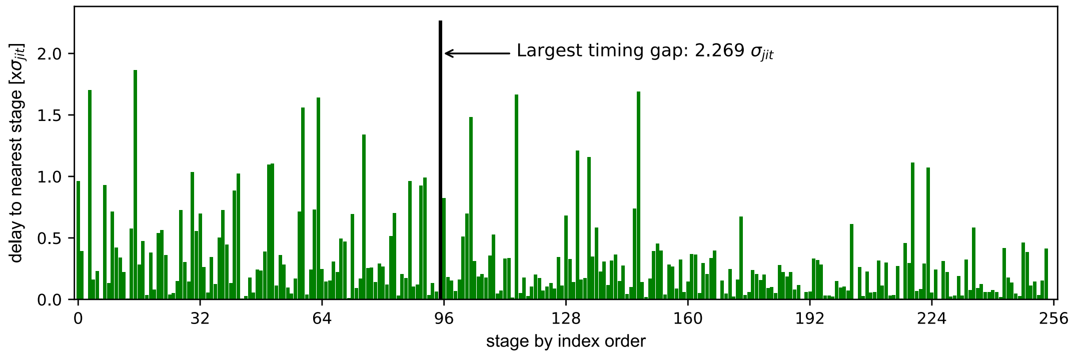
Fig. 6a shows, for one instance of the TRNG, the stage arrival times $(T_0, \dots, T_{255})$ obtained by solving Eq. 3. The arrival times across the 256 stages cover a span of approximately 84 times σ_{jit}. While the arrival time generally increases while moving up the TDC chain, note that the trend is not smooth. Although the rising edge does propagate through the carry chains in sequential order, the anomalies in this trend imply that it does not reach the d input of the flip-flops in sequential order. This can occur because the delays between carry chain and flip-flop of each stage are not uniform, and because clock skew at the flip-flop aliases to delay on its data input; both of these artifacts are captured in the timing report so it is instructive to compare the modeled arrival times to the slack from the timing report. Fig 6b shows the same modeled arrival times from Fig. 6a, but now plotted against the reported timing slack for the corresponding flip-flop which accounts for all delays and clock skew. Because a span of 84 times σ_{jit} corresponds to slightly above 800ps of reported slack, we can estimate σ_{jit} to be around 10ps in absolute terms per the timing report, although the timing report itself uses a conservative delay with built-in safety margin, so the average-case delay may be somewhat less. There is a high correlation ($r = -0.997$) between the arrival times from the model and the reported slack. Given that the model was fitted to data measured on the board, the high correlation between the two quantities supports the validity of the model for correctly resolving on-chip delays, and hence also for capturing the difference in criticality between stages. Now that the empirical timing model is validated, we use it as the basis for estimating the worst-case entropy of our TRNG.



(a) Modeled arrival time vs stage index.



(b) Modeled arrival time vs slack from timing report.



(c) Precision of each stage in TDC chain from one FPGA instance, obtained from characterization procedure.

Fig. 6. TDC characterization. The largest timing gap between any two neighboring stage arrival times is highlighted and annotated in Fig. 6a and Fig. 6c

4.3 Stochastic Model for Entropy Estimation

Based on the precision of each stage in Fig. 6c, we obtain the largest timing gap ($2.268 * \sigma_{jit}$) between any two neighboring arrival times as highlighted in Fig. 6, which can be used to estimate a lower bound on min-entropy of the samples. The worst-case min-entropy corresponds to the sampled value that can be produced with highest probability. This would occur when the mean arrival time of the clock coincides with the center of the largest timing gap of any stage, which we denote here as Δ_{max}. This is illustrated in Fig. 7, which depicts clock jitter assumed as normal distribution. In the worst-case min-entropy, the mean of jitter is in the middle of the two stages with $\Delta_{max} = 2.268 * \sigma_{jit}$. The shaded region represents all the clock arrival times that would result in the same sample being produced. The probability of producing this sample would then be the probability associated with the shaded region, which we denote as P_{max}, and calculate from the normal CDF as in Eq. 4. The min-entropy of an outcome with probability P_{max} is given by Eq. 5. For the specific instance used to generate these results, the largest interval is 2.268 $*\sigma_{jit}$ shown in Fig. 6c, which corresponds to a shaded area in Fig. 7 with P_{max} of 0.743, and hence min-entropy of 0.429 bits.

$$P_{max} = \Phi\left(\frac{\Delta_{max}}{2}\right) - \Phi\left(\frac{-\Delta_{max}}{2}\right) \tag{4}$$

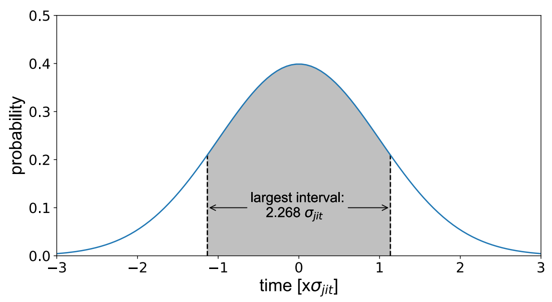


Fig. 7. Largest share of clock arrival times that will cause TDC to sample the same value.

$$entropy_{min} = \log_2\left(\frac{1}{P_{max}}\right) \tag{5}$$

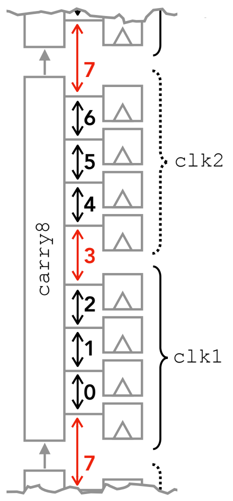
4.4 Impact of Routing and Clock Skew on Entropy

The previous subsection explains that worst-case min-entropy is limited by the largest timing gap among all the stages. It is therefore desirable to make all of the timing gaps uniform so that none are unusually large. We now present further results and discussion to explain why routing makes this objective difficult to accomplish in practice.

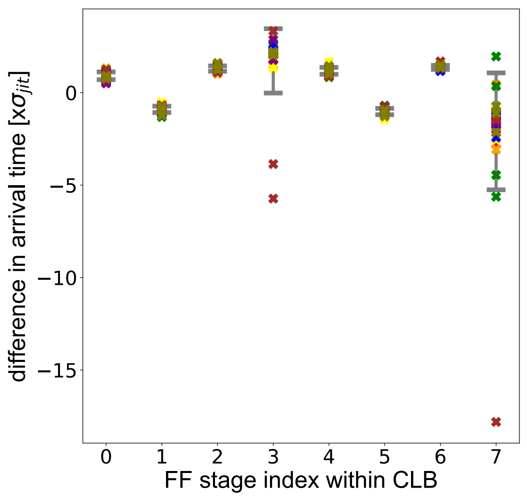
Fig. 8b shows the difference in arrival time between the FF of each stage and that of the next stage by index. Here, instead of using indices from 0 to 255 to represent the stages across all 32 CLBs as was done in Fig. 6a, we use indices 0-7 for each CLB as annotated in Fig. 8a, and have occurrences of each index from all 32 CLBs. Fig. 8b shows, for each stage index, the 32 differences in arrival time between that stage and the next. The differences in arrival time are predictable for stages 0,1,2 and for stages 4,5,6. In stages 3 and 7, the arrival time difference is inconsistent from CLB to CLB. This inconsistency occurs because Ultrascale+ uses different clock inputs for the upper and lower halves of the CLB, which causes stages 3 and 7 to span two different clock leaf nodes (clk1 and clk2 in Fig. 8a); the skew between the clock leaf nodes aliases to arrival time as discussed in Sec. 4.2. For the TRNG design, one must therefore be careful to avoid large positive clock skew at these points as it can reduce worst-case entropy by causing highly probable outcomes for certain unlucky conditions. Negative skew causes no such problem, as can be observed in Fig. 6a, so the one-sided restriction on skew is easy to satisfy in practice. In fact, note that the negative skew at stage 208 in Fig. 6a can improve the quality of the TRNG, because it causes the rising edge to be captured twice in the same sample, at different positions in the chain. In Sec. 7 we build on this principle to design a TRNG that samples the rising edge multiple times.

5 TRNG QUALITY EVALUATION

In this section, we use three different techniques to test the quality of the random numbers produced by our design. As described in the following three subsections, the results support (1) that our design exceeds the 0.1 bits of min-entropy per trial that was assumed as a security parameter; (2) that our stochastic model gives a reasonable estimate of min-entropy; and (3) the random numbers generated pass tests for statistical randomness.

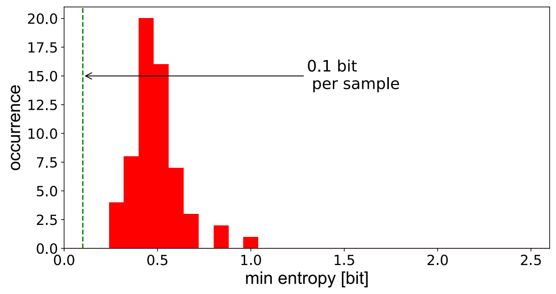


(a) CLB stage indices.

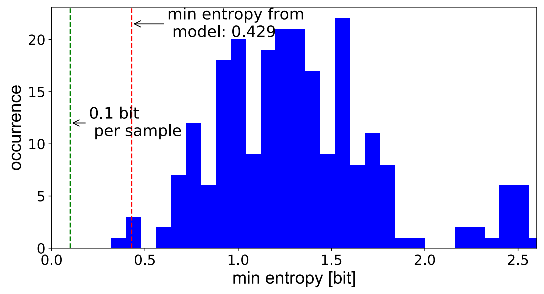


(b) Differences in arrival times by index.

Fig. 8. Across the 32 CLBs in the TDC, the difference in arrival time between one index and the next is predictable for indices 0,1,2 and 4,5,6. Indices 3 and 7 are each followed by a stage that is on a different clock leaf, and there the difference in arrival time is inconsistent due to clock skew. Error bars extend one standard deviation from the mean.



(a) Entropy per stochastic model for 60 FPGA instances.



(b) Entropy per NIST SP800-90B for 234 tunings on one instance.

Fig. 9. Min-entropy by both stochastic model and NIST SP800-90B suite exceeds 0.1 bits per sample.

5.1 Stochastic Model Applied Across EC2 F1 Instances

The stochastic model from Section 4.3 is our primary strategy for estimating the worst-case min-entropy for each single instance of the TRNG. To test across FPGAs, we load the same bitstream onto 60 different EC2 F1 instances, and on each machine apply our characterization procedure to evaluate the worst-case min-entropy. The distribution of calculated min-entropy values (Fig. 9a) ranges from 0.250 to 0.972. These values indicate that across all 60 instances our design exceeds, by at least a margin of 2.5×, the 0.1 bits of min-entropy per sample that was assumed.

Statistical tests	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-value	%
Frequency	10	12	6	12	11	4	11	14	4	16	0.091	98
BlockFrequency	15	17	8	7	7	7	12	9	6	12	0.163	99
CumulativeSums	13	6	10	7	10	15	13	9	9	8	0.596	99
CumulativeSums	11	8	13	8	12	7	15	8	11	7	0.637	98
Runs	9	8	8	8	16	6	15	15	7	8	0.172	98
LongestRun	9	12	8	11	16	6	9	10	11	8	0.657	98
Rank	9	12	13	10	12	5	11	9	13	6	0.637	100
FFT	9	12	12	15	8	5	11	10	12	6	0.494	100
NonOverlap. Template	8	4	7	11	13	9	15	13	10	10	0.401	100
Overlapping Template	8	9	11	8	8	15	12	9	8	12	0.817	98
Universal	8	8	8	12	14	12	14	6	8	10	0.616	99
Approximate Entropy	18	10	12	6	8	14	6	12	6	8	0.109	99
Serial	5	10	8	9	11	12	10	9	16	10	0.616	99
Serial	6	8	10	11	6	13	11	10	12	13	0.740	100
LinearComplexity	13	8	4	14	13	7	15	9	8	9	0.249	100

Table 1. Results from applying NIST statistical test suite to 100M generated bits shows that the TRNG outputs are evaluated as consistent with being random.

5.2 Stochastic Model vs. NIST Entropy Assessment

Next, to check our stochastic model, we apply the NIST SP800-90B entropy assessment suite [22] to obtain an independently calculated estimate of min-entropy. To generate data for the NIST assessment, we apply on one instance all tuning settings and collect 1,000,000 samples from the TDC with each setting applied [23]. We keep the data from any settings in which the average Hamming weight of samples is in our allowed range of 30 to 225, which corresponds to a total of 234 tuning settings. The NIST assessment is applied separately to each of these 234 datasets, and the distribution of results is shown as a histogram in Fig. 9b. The NIST estimate of min-entropy for most of the tuning settings fall above the estimate of 0.429 bits per sample from the stochastic model on this instance. Because the NIST entropy values tend to exceed our estimated worst-case, we gain some confidence that our stochastic model is not overestimating entropy.

5.3 End-to-End NIST Statistical Tests

Although the evaluation of the entropy source in the prior subsections is the primary validation for a TRNG, we also apply statistical tests to the post-processed 8-bit values produced by the TRNG as a further validation. The NIST Statistical Test Suite [24], which is widely used with random number generators, applies a collection of statistical tests and for each test reports whether the sequences of bits are consistent with being random. The report shows how often the P-values from each test fall within uniformly sized bins C1 through C10, and should tend toward being uniformly distributed when enough random data is tested. The test suite is applied to a dataset comprising 100 sequences of 1,000,000 bits from the TRNG and the results are displayed in Tab. 1. The final column of the table shows the proportion of sequences that pass the test, indicating that the sequences have statistical properties consistent with being random.

5.4 Empirical Min-entropy with Respect to Lag

Our stochastic model provides an upper bound on the ability to predict the response of the TRNG circuit, but the model assumes that all variation in the data is explained by temporally uncorrelated jitter. If variation in the characterized data actually has temporal correlation due to a physical cause like temperature or voltage, then the model may overestimate jitter. As an additional validation of entropy, we evaluate empirically whether an attacker that knows the value of a

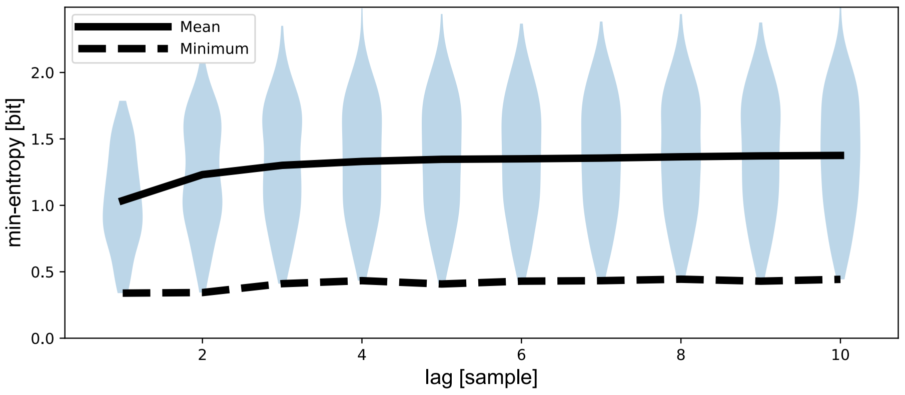


Fig. 10. Distribution of min-entropy values, with annotations for mean and minimum, when guessing a lagging sample based on the value of a leading sample. There is only a minor temporal relationship, and min-entropy remains above 0.33 even in the worst case.

leading sample can predict the value of a lagging sample a few cycles later. We perform this evaluation for all valid tuning settings, and collect a dataset of 100,000 samples for each. From each dataset, for all Hamming weight values a and b, we find the conditional probability of observing b as the value of the lagging sample, given that a was observed as the value of the leading sample. The maximum conditional probability corresponds to the best lagging sample guess by the attacker, when leading sample has the value that most benefits the attacker. Min-entropy is calculated from this probability, and this is the empirical lower bound on min-entropy for the tuning setting that generated the 100,000 sample dataset. We obtain one such min-entropy for each tuning, and vary the lag from 1 to 10 samples, and show the results in Fig. 10. Across all the tuning settings, the minimum min-entropy never drops below 0.33 bits which still surpasses our security assumption of 0.1 bits per sample.

6 TRNG PERFORMANCE AND COST EVALUATIONS

Aside from requirement of avoiding circuits such as oscillators that are disallowed in certain clouds, the large capacity of cloud FPGAs implies that the TRNG must also be resilient to any noise, voltage, or temperature fluctuations that are caused by high-powered circuitry around the TRNG.

6.1 Resilience to Environmental Fluctuations

We subject the TRNG design to intentional environmental disruptions to check that its feedback is able to adapt appropriately. Specifically, we build a configurable power consumption circuit that is next to the TRNG on the F1 instance. The power waster consists of 32 different levels of power consumption that can be enabled. Each level turns on one instance of a circuit comprising four combinational rounds of the Advanced Encryption Standard (AES) block cipher, with additional feed-forward paths added to increase glitching [25]. The power consumption of the circuit is measured as the average power reported by the fpga-describe-local-image command provided in the AWS management tools. Because the reported power updates only once per minute, we perform separate experiments to characterize the consumption of the power wasters instead of measuring their power in real-time when using them to disturb the TRNG. The baseline power consumption of the instance is 8W, and each enabled level of power waster consumes an additional 3W. Turning on the power wasters can disrupt the TRNG by causing heating and voltage droop.

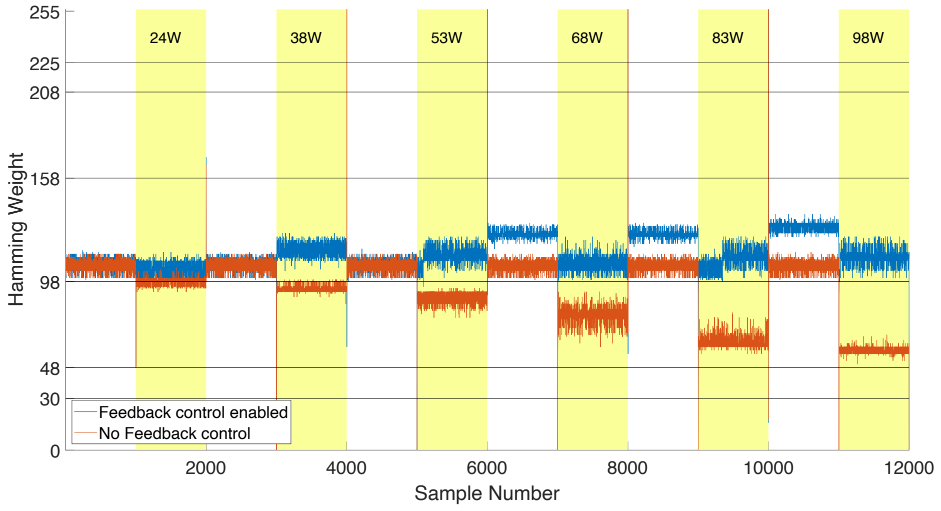


Fig. 11. Control loop adapts to changes in localized power consumption on the FPGA in order to keep the TRNG tuned.

Fig. 11 shows the power wasting circuit is toggled on and off every 1,000 samples, and each time it is switched on an additional five of the 32 power waster instances are enabled, which corresponds to around 15W of additional power consumption. The blue line shows the Hamming weight of the samples when feedback is enabled, and the orange line shows the Hamming weight when feedback is disabled. Both voltage droop and increasing temperature increase propagation delay between the controller and the TDC, which can explain the drop in Hamming weight. The feedback allows the TRNG to compensate for this. When the feedback is disabled, we can see by the Hamming weight that the magnitude of power consumption has a direct relation on the delay of the circuit. Therefore, the controller uses feedback to adapt, and is able to keep the TRNG tuned and operating correctly.

For an end-to-end validation of the TRNG under disturbance from power wasters, we repeat the analysis of Section 5.3. As before, the NIST test suite is applied to 100 sequences of 1,000,000 bits, and now 32 power waster instances are running during the data collection. The power wasting circuitry toggles between on and off with every 1,000,000 TRNG bits collected. Similar to Table 1, the TRNG again passes the tests, which indicates that these environmental fluctuations are not observed to compromise the TRNG quality.

6.2 Comparison to Prior Work

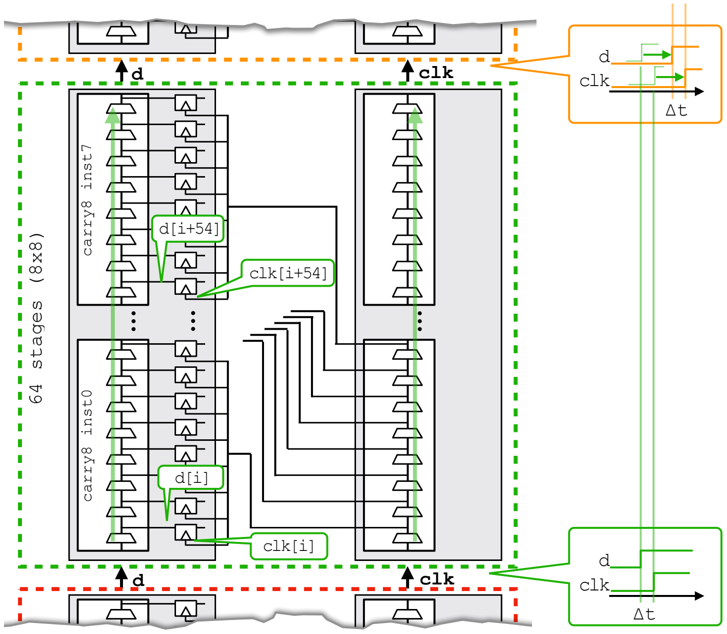
The distinguishing feature of our work is its suitability for, and deployment on, cloud FPGAs. As we have described, this imposes limitations on the types of circuitry that can be used, and increases the importance of the TRNG being robust to environmental changes. Despite these challenges, the costs of our TRNG are found to be reasonable for a large cloud FPGA. Table 2 compares the throughput, logic utilization, efficiency (throughput/slice), testing methods, resistance to attack and entropy of our TRNG to other recently published TRNGs that are implemented on Xilinx FPGAs. Our TRNG design (Fig. 1) consumes 791 LUTs (0.067% of available), 33 CARRY8s, and 559 flip-flops (0.024%) across a total of 184 slices. Among these resources, the controller logic that configures the coarse- and fine-tuning consumes 92 LUTs and 34 flip-flops, while the remainder of the resources are consumed by the TRNG core itself. Our design generates random

Work	FPGA type	Throughput (Mbps)	Utilization (slice)	Efficiency (Mbps/slice)	Approach	Testing method	Analysis of attacks resisted	Entropy validation Entropy value Entropy type
[26]	Spartan 6	100	46	2.17	self-timed ring	NIST SP800-22	-	-
[18]	Spartan 6	14.3	67	0.213	ring oscillator	TestU01 DIEHARD NIST SP800-22 ENT	-	ENT 7.99998/byte (postproc) unknown
[4]	Spartan 6	1.15	3	0.383	ring oscillator	NIST SP800-90B AIS-31	-	NIST SP800-90B 0.76/sample (raw) min-entropy
[27]	Virtex-4	12.5	580	0.022	RS latches metastability	DIEHARD NIST SP800-22	-	-
[5]	Virtex-6	50	224	0.223	timing non-uniformity	DIEHARD NIST SP800-22	-	-
[6]	Virtex-5	2	32	0.063	metastability	NIST SP800-22	Unspecified external perturbations	-
[7]	Spartan 6	3.3	27	0.122	ring oscillator	AIS-31	-	stochastic model >0.91/bit (raw) min-entropy
[8]	Spartan 6	1.1	128	0.122	ring oscillator	AIS 20/31	-	AIS-20/31 7.998265/byte (raw) Shannon entropy
[28]	Spartan 6	0.76	1	0.76	latched ring oscillator	NIST SP800-22 AIS-31	Voltage	AIS-31 7.99834/byte (raw) Shannon entropy
[29]	Spartan 3	6	270	0.022	ring oscillator	NIST SP800-22 AIS-31	-	AIS-31 7.9946/byte (raw) Shannon entropy
[30]	Virtex-6	-	9	-	self-timed ring	NIST SP800-22 NIST SP800-90B AIS-31	Power and Thermal Attacks	NIST SP800-90B 7.8869/samp (postproc) min-entropy
our basic TRNG [12]	Virtex UltraScale+	2.43	184	0.013	clock jitter	NIST SP800-22 NIST SP800-90B	PVT	NIST SP800-90B 0.37/sample (raw) min-entropy
TRNG w/ linkable modules	Virtex UltraScale+	6.08	216	0.028	clock jitter	NIST SP800-22 NIST SP800-90B	PVT	NIST SP800-90B 1.45/sample (raw) min-entropy

Table 2. Comparison with related TRNGs implemented on Xilinx FPGAs.

numbers at a rate of 2.43Mbps, which is sufficient for most applications, but could be increased through parallelization if needed.

Here we listed the entropy comparisons between our work and other works in Table 2. It is worth noting that NIST SP800-20B does not list a minimum value of what constitutes a usable amount of min-entropy per sample.



(a) Linkable sampling module for TRNG.

	Arrival time [ns]		
Stage	data	clock	skew
i+54	0.368	0.300	-0.068
i+48	0.327	0.332	0.005
i+40	0.286	0.318	0.032
i+32	0.245	0.296	0.051
i+24	0.204	0.262	0.058
i+16	0.163	0.255	0.092
i+8	0.122	0.221	0.099
i	0.081	0.227	0.146

(b) Arrival time relative to module inputs

Fig. 12. Schematic and timing of 64-stage linkable sampling module used in the TRNG

7 TRNG BASED ON LINKABLE SAMPLING MODULE

In this section, we extend our basic TRNG to increase entropy per sample while still retaining the stochastic model of randomness. The key idea of the approach is to make a modular version of the sampling chain that can be arbitrarily extended by abutment to increase entropy. We choose to make each sampling module 64 stages in length. The new module is shown within the dashed box of Figure 12a; we instantiate and link four such modules to create the sampling chain for the TRNG. Each module has inputs for a rising data edge and a sampling clock (shown at bottom), and then propagates those signals through parallel carry chains to the outputs at the top of the module. In the first instance, the data input is the rising edge from the tunable delay line as in Fig. 2a, and the clock input is attached to the 125 MHz system clock.

7.1 Timing Analysis

The simple timing diagrams shown at right in Fig. 12a illustrate the operating principle of the design. The matched paths taken by data and clock through the module ensure that whatever timing difference exists between them at module input, is preserved at module output, which is the input to the next module. Therefore, neglecting small imbalances, each module performs the same sampling experiment with the same relative timing. Effectively, the same rising edge is being sampled within each module. We first consider the behavior of a single module and then how the four modules interact.

7.1.1 Single module. The 64 stages of a sampling module span 8 rows. The sampling clock for the 8 FFs in each CLB comes from one tap on the clock path. Relative to the d and clk module inputs, the relative arrival time at the data and clock inputs of sampling FFs are shown in the table of Fig. 12b. The position of the clock tap, and its routing to the sampling FFs, is fixed but not optimized. The sampling clock has a different arrival time at each CLB, unlike the original design where the clock tree ensures low skew. Recall from Sec. 4.2 that skew aliases to delay; therefore, even though the propagation delay of data through the 64 stages is reported to be 368 ps per the timing report, the difference in skew is only 214 ps because the clock arrives later at upper stages. That skew is likely the reason that the empirical delay difference through 64 stages is equivalent to around $20 \times \sigma_{jit}$, which is less than was observed in the original design where the clock was synchronous to all stages. Plotting arrival time of each stage against reported timing slack (Fig. 13b), we can see that the inferred arrival times of each module are correlated to reported slack with correlations of (0.97, 0.96, 0.97, 0.96), which are slightly lower than the correlation of 0.997 reported in Sec. 4.2.

7.1.2 *Multiple modules.*

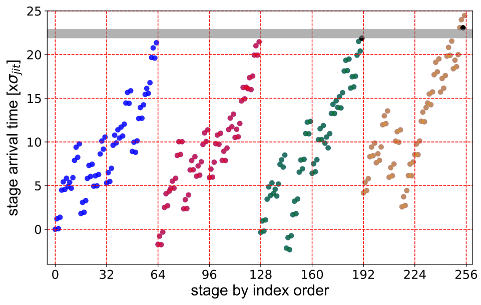
We now consider the timing behavior of all 256-stages, comprising four identical 64-stage modules. When the rising edge is sampled four times, it lands between two stages in each of the four modules. The samples of each module overlap each other, which therefore increases sensitivity and tends to reduce the bin widths as shown in Fig. 13c. The worst-case min-entropy calculated by applying the stochastic model to Fig. 13c is 1.152 bits, which is 2.7× larger than the min-entropy calculated for our basic TRNG design. Accordingly, we increase our assumption of entropy from 0.1 bits per sample in the basic TRNG to 0.25 bits per sample in the TRNG that uses four linkable sampling modules, noting that our assumption remains highly conservative relative to worst-case entropy indicated by the model. When considering empirical arrival time from our model against slack, the expected correlation within each module can be observed in Fig. 13b, but there is an offset across modules. The reason is as follows. The relative timing of data and clock is the same at the input of each module due to the matched paths, yet timing analysis is conservative and considers adding the same delay to data and clock as making the path more critical. To preserve the same amount of negative slack, the timing analysis would require the added delay on the data path to be larger than the added delay on the clock path.

When the feedback control is implemented based on the total Hamming weight, there can be unpredictable changes if one instance entirely misses a rising edge due to poor tuning, so we instead control the delay based on the Hamming weight of a single 64-bit module instance, which also simplifies the control logic. Although we use four sampling modules to keep the total number of stages to 256, in principle an arbitrary number of sampling modules can be instantiated and connected by abutment. If there is systematic variation between data and clock paths, then it could perhaps eventually become difficult to use a common tuning for many instances. There are no indications of any such problems with four modules.

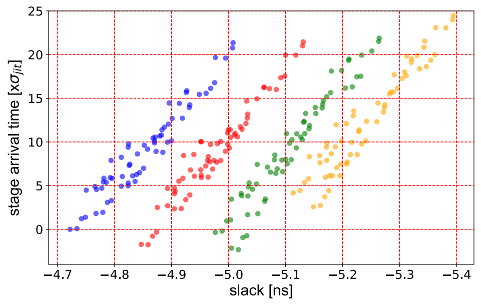
7.2 Entropy and Performance

Similar to Section 5.1 for the basic TRNG, we load the modified TRNG onto 60 EC2 instances and apply the stochastic model to calculate the min-entropy on each instance. As shown in Fig. 14a, the worst case min-entropy of any instance is 0.93 bits. As before, the entropy assumption is conservative; the actual entropy from the model (0.93 bits/sample) is 3.72× higher than what is assumed (0.25 bits/sample). As in Section 5.2, we also collect 1,000,000 samples from each delay tuning on one instance, and apply the NIST SP800-90B entropy assessment to the data. The distribution in Fig. 14b shows the min-entropy for all tunings on this instance, all of which exceed the 1.152 bits calculated by stochastic model for the same instance.

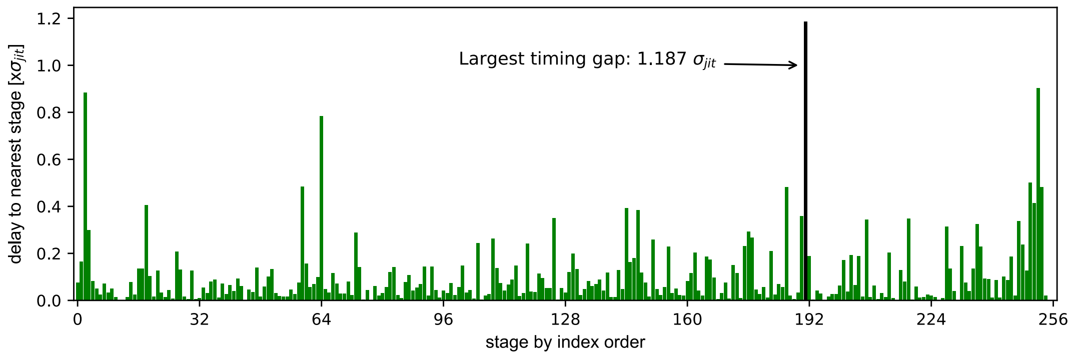
Our justified assumption of 0.25 bits of entropy per sample enables a 250% increase in throughput relative to the basic TRNG from FPT'20 [12]. An 8-bit random number is now generated using only 32 samples instead of 80. Due



(a) Arrival time at each stage.



(b) Arrival time vs slack (correlation=0.96, 0.97, 0.97, 0.96).



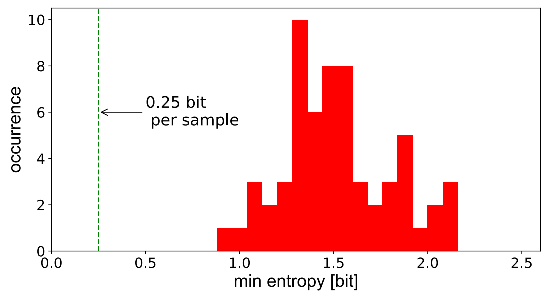
(c) Bin sizes, which determine entropy.

Fig. 13. The use of four linkable sampling modules causes the samples to overlap in time (in a and b), so that the same edge is sampled once in each of the four modules. This reduces the bin size (in c) compared with the original TRNG. The largest timing gap between two neighboring stages in terms of arrival time is highlighted and annotated in Fig. 13a and Fig. 13c

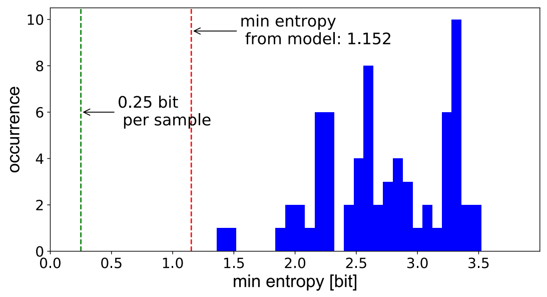
to the 32 extra CLBs used to route the clock through the four instances of the linkable sampling module, the 250% increase in throughput comes at only 17% resource cost. The linkable sampling modules create an attractive cost vs performance tradeoff. Table 2 compares the performance of our TRNGs against eleven other works. The throughput of a linkable sampling TRNG is able to support low-throughput applications aforementioned in Section 1. The throughput of any FPGA TRNG can be increased to accommodate high throughput applications by adding more instances, so an important metric to consider is the efficiency, in terms of throughput of random numbers per slice of area. We therefore list the metric of Mbps/slice as well in the table. Finally, we list some of the attacks that each design is claimed to resist, and describe the entropy metrics that are provided for each. Notably, given that most FPGA TRNGs are based on oscillators which are forbidden, the only designs in the table that can be implemented on EC2 F1 are ours and [6][5].

8 CONCLUSION

Cloud FPGAs are commonly used for accelerating computationally expensive cryptographic operations that rely on the generation of random numbers. In this paper, we introduced and evaluated a TRNG design that is compatible with the design restrictions imposed by cloud-based FPGA providers. The TRNG oscillator-free design that we impose uses a controllable delay and harvests clock jitter as an entropy source using a circuit that is similar to a TDC. The



(a) Min-entropy of 60 instances, from stochastic model.



(b) Entropy assessment of 70 tunings, on one instance.

Fig. 14. Entropy according to stochastic model, and from NIST assessment, both show that the modified TRNG with linkable sampling modules produces around 1 bit of entropy per sample.

effectiveness of the design is supported by NIST test results and a stochastic model of the entropy source. Furthermore, the design is shown to be able to compensate for voltage droop that may occur during a power attack, and its entropy is not compromised in this scenario. Future work can consider further increases in entropy-per-sample and the impact of advanced clocking features.

ACKNOWLEDGEMENT

This research was funded in part by NSF grants CNS-1749845 and CNS-1902532. The authors also gratefully acknowledge Gradient Technologies and Amazon for their support and contributions to this work.

REFERENCES

[1] J. Soto, “Statistical testing of random number generators,” in *the 22nd National Information Systems Security Conference*, vol. 10, no. 99. NIST Gaithersburg, MD, 1999, p. 12.

[2] G. Zheng, G. Fang, R. Shankaran, and M. A. Orgun, “Encryption for implantable medical devices using modified one-time pads,” *IEEE Access*, vol. 3, pp. 825–836, 2015.

[3] C. Meijer and B. Van Gastel, “Self-encrypting deception: weaknesses in the encryption of solid state drives,” in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 72–87.

[4] B. Yang, V. Rožic, M. Grujic, N. Mentens, and I. Verbauwhede, “ES-TRNG: a high-throughput, low-area true random number generator based on edge sampling,” *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 267–292, 2018.

[5] X. Yang and R. C. Cheung, “A complementary architecture for high-speed true random number generator,” in *2014 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2014, pp. 248–251.

[6] M. Majzoobi, F. Koushanfar, and S. Devadas, “FPGA-based true random number generation using circuit metastability with adaptive feedback control,” in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2011, pp. 17–32.

[7] A. Peetermans, V. Rozic, and I. Verbauwhede, “A highly-portable true random number generator based on coherent sampling,” in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2019, pp. 218–224.

[8] J. Balasch, F. Bernard, V. Fischer, M. Grujić, M. Laban, O. Petura, V. Rožić, G. Van Battum, I. Verbauwhede, M. Wakker, and Y. Bohan, “Design and testing methodologies for true random number generators towards industry certification,” in *2018 IEEE 23rd European Test Symposium (ETS)*. IEEE, 2018, pp. 1–10.

[9] L. Gaspar, V. Fischer, L. Bossuet, and R. Fouquet, “Secure extensions of FPGA soft core processors for symmetric key cryptography,” in *6th International Workshop on Reconfigurable Communication-Centric Systems-on-Chip (ReCoSoC)*. IEEE, 2011, pp. 1–8.

[10] G. Provelengios, D. Holcomb, and R. Tessier, “Characterizing power distribution attacks in multi-user FPGA environments,” in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, 2019, pp. 194–201.

[11] I. Giechaskiel, K. B. Rasmussen, and J. Szefer, “Measuring long wire leakage with ring oscillators in cloud FPGAs,” in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2019, pp. 45–50.

[12] X. Li, P. Stanwicks, G. Provelengios, R. Tessier, and D. E. Holcomb, “Jitter-based adaptive true random number generation for FPGAs in the cloud,” in *2020 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2020, pp. 112–119. [Online]. Available:

https://doi.org/10.1109/ICFPT51103.2020.00024

[13] S. Zeitouni, J. Vliegen, T. Frassetto, D. Koch, A.-R. Sadeghi, and N. Mentens, “Trusted configuration in cloud FPGAs,” in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM).* IEEE, 2021, pp. 233–241.

[14] P.-F. Wolfe, R. Patel, R. Munafo, M. Varia, and M. Herbordt, “Secret sharing MPC on FPGAs in the datacenter,” in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL).* IEEE, 2020, pp. 236–242.

[15] P. Kohlbrenner and K. Gaj, “An embedded true random number generator for FPGAs,” in *Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays.* ACM, 2004, pp. 71–78.

[16] A. Maiti, R. Nagesh, A. Reddy, and P. Schaumont, “Physical unclonable function and true random number generator: a compact and scalable implementation,” in *Proceedings of the 19th ACM Great Lakes Symposium on VLSI*, 2009, pp. 425–428.

[17] V. Rozic, B. Yang, W. Dehaene, and I. Verbauwhede, “Highly efficient entropy extraction for true random number generators on FPGAs,” in *2015 52nd ACM/IEEE Design Automation Conference (DAC).* IEEE, 2015, pp. 1–6.

[18] N. Deák, T. Györfi, K. Márton, L. Vacariu, and O. Cret, “Highly efficient true random number generator in FPGA devices using phase-locked loops,” in *2015 20th International Conference on Control Systems and Computer Science.* IEEE, 2015, pp. 453–458.

[19] J.-L. Danger, S. Guilley, and P. Hoogvorst, “High speed true random number generator based on open loop structures in FPGAs,” *Microelectronics Journal*, vol. 40, no. 11, pp. 1650–1656, 2009.

[20] T. J. Yamaguchi, K. Ichiyama, H. X. Hou, and M. Ishida, “A robust method for identifying a deterministic jitter model in a total jitter distribution,” in *2009 International Test Conference.* IEEE, 2009, pp. 1–10.

[21] L. Xu, Y. Duan, and D. Chen, “A low cost jitter separation and characterization method,” in *2015 IEEE 33rd VLSI Test Symposium (VTS).* IEEE, 2015, pp. 1–5.

[22] M. S. Turan, E. Barker, J. Kelsey, K. A. McKay, M. L. Baish, and M. Boyle, “Recommendation for the entropy sources used for random bit generation,” *NIST Special Publication*, vol. 800, no. 90B, p. 102, 2018.

[23] C. Celi, “NIST SP800-90B entropy assessment,” https://github.com/usnistgov/SP800-90B_EntropyAssessment, 2019.

[24] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, and E. Barker, “A statistical test suite for random and pseudorandom number generators for cryptographic applications,” National Institute of Standards and Technology, Tech. Rep., 2010.

[25] G. Provelengios, D. Holcomb, and R. Tessier, “Power wasting circuits for cloud FPGA attacks,” in *International Conference on Field Programmable Logic and Applications (FPL)*, 2020, pp. 231–235.

[26] J.-Y. Choe and K.-W. Shin, “A self-timed ring based TRNG with feedback structure for FPGA implementation,” in *2020 International Conference on Electronics, Information, and Communication (ICEIC).* IEEE, 2020, pp. 1–4.

[27] H. Hata and S. Ichikawa, “FPGA implementation of metasability-based true random number generator,” *IEICE Transactions on Information and Systems*, vol. 95, no. 2, pp. 426–436, 2012.

[28] R. Della Sala, D. Bellizia, and G. Scotti, “A novel ultra-compact FPGA-compatible TRNG architecture exploiting latched ring oscillators,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2021.

[29] N. N. Anandakumar, S. K. Sanadhya, and M. S. Hashmi, “FPGA-based true random number generation using programmable delays in oscillator-rings,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 3, pp. 570–574, 2019.

[30] 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.