



# OPEN Integration of multiple coinflip devices for high-quality random sampling

Brady Taylor<sup>1,2</sup>✉, J. Darby Smith<sup>1</sup>, Shashank Misra<sup>1</sup>, James B. Aimone<sup>1</sup> & Christopher R. Allemang<sup>1</sup>✉

Artificial intelligence, scientific computing, and probabilistic computing use random sampling to approximate solutions to various problems, with larger models requiring a substantial quantity of random numbers. To generate the required vast quantity of random numbers at high rates, we explore so-called “coinflip” devices, which are stochastic microelectronic devices ideally capable of independently generating random bits with a tunable weight at a high rate. However, coinflip devices are inherently analog and demonstrate nonidealities, like temperature dependence and drift, that can introduce determinism into the outputs. We present important considerations for building systems of multiple coinflip devices to produce high-quality bitstreams with low error and little dependency on previous bits. Using tunnel diodes as coinflip devices, we implement a control loop to adapt to temperature dependence and generate fair bitstreams with each device. While this can lead to dependencies between bits in a single bitstream, we demonstrate that combining results generated in parallel with individual tunnel diodes can produce fair and unpredictable bitstreams. The suitability of these bitstreams for use in probabilistic computing is then demonstrated through a Monte Carlo approximation of  $\pi$ .

Random sampling is an often overlooked, yet pervasive, component of many computing workloads. In machine learning, random sampling is used in initialization of network parameters<sup>1,2</sup>, regularization of learning<sup>3,4</sup>, and stimulating generative models<sup>5,6</sup>. In stochastic computing, streams of random bits are sampled with different probabilities of 1 and combined through digital operations to produce a stream with a probability that is a function of the input probabilities<sup>7</sup>. In biology<sup>8</sup>, finance<sup>9</sup>, and physics<sup>10</sup>, random sampling is used to approximate solutions using Monte Carlo methods. Generating large amounts of random numbers for sampling more complex applications can be a computationally intensive process. For example, in scientific computing when simulating high-energy particle collisions, generating uniform random numbers can consume between 30–50% of CPU-time<sup>11</sup>. Due to the pressing need for efficient random number generators, we look towards systems of single-device true random number generators (TRNGs), called coinflip devices, to generate random bits efficiently, quickly, and in parallel<sup>11</sup>.

Coinflip devices use physical noise or stochastic quantum behavior to switch between two states, such as high and low resistances or voltages. Ideal coinflip devices are compact, consume little energy per bit, switch at high speeds, and are unpredictable but with a weight that can be tuned to different probabilities of 1. Notable examples of potential coinflip devices include magnetic tunnel junctions (MTJs)<sup>12,13</sup>, diffusive memristors<sup>14</sup>, single photon avalanche diodes<sup>15–18</sup>, and tunnel diodes<sup>19–21</sup>. MTJs, for example, can generate random bits for as little as 0.1 pJ per sample<sup>12</sup> by switching between high and low resistive states. Devices are usually either physically stimulated by some current or electric field to randomly adopt one of two states before being evaluated (e.g. tunnel diodes<sup>19</sup>) or the state of the device flips occasionally over time and is sampled intermittently (e.g. MTJs<sup>13</sup>). These sequences of analog values are interpreted as digital states, i.e. 1's and 0's (heads and tails). Bitstreams can then be partitioned into multi-bit slices to form integers by taking  $N$  sequential bits from a bitstream for an  $N$ -bit integer or used directly in probabilistic computing. These integers are expected to be uniformly distributed for a fair coin with a probability  $p$  of heads (also written as  $P(H)$ ) of  $p = 0.5$ <sup>22</sup>. We call this probability the “weight” of a coin or coinflip device. From a uniform distribution, algorithms such as rejection sampling can be used to generate more complicated distributions such as a Gaussian distribution<sup>23</sup>.

<sup>1</sup>Sandia National Laboratories, Albuquerque, NM 87123, USA. <sup>2</sup>Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708, USA. ✉email: btaylor@sandia.gov; crallem@sandia.gov

Whereas here we focus on the TRNGs for use in otherwise conventional computations, coinflip devices have much in common with p-bits and probabilistic computing. Many p-bits are built from coinflip devices<sup>24–27</sup>, which can be processed through analog or digital means for probabilistic computing<sup>28,29</sup> or can even augment low quality PRNGs to generate high quality samples<sup>30</sup>.

However, while they generate digital outputs, coinflip devices are analog by nature and subject to nonidealities and variation. For example, the stochastic behaviors of diffusive memristors (random delay time) and tunnel diodes (random high or low voltage) can vary with temperature<sup>14,19</sup>. Sampling bits from devices with a fair weight despite variation requires compensation or control electronics. Furthermore, it is important to understand the practical implications of integrating multiple coinflip devices into a single system. If the behavior of one device can deterministically affect other devices, this can have deleterious effects on the ability of the system to generate high quality random samples with high throughput. In this work, we examine the random bit generation capabilities of multiple coinflip devices integrated into a single system, using a tunnel diode (TD) as our representative coinflip device. We use a control loop to adapt to variation with temperature that causes deterministic behavior and maintain a fair coin, which can be generalized to adapt to other forms of variation. Furthermore, we examine dependencies of the probability of 1 on previous results and other devices, finding that sampling devices out-of-phase and combining results from multiple devices with an XOR operation (similarly to previous works<sup>31–34</sup>) can substantially improve sampling quality. Finally, using experimentally collected bits from our system of coinflip devices, we demonstrate performance in a probabilistic computing application with a Monte Carlo approximation of  $\pi$ . While it is vital for TRNGs used in cryptography to maintain certain strict standards such as those outlined in the NIST Statistical Test Suite<sup>35</sup>, the requirements for probabilistic computing may differ significantly. For example, certain PRNGs that fail these NIST statistical tests may still produce statistically identical results to TRNGs that pass all tests<sup>36</sup>. It is for this reason that we examine the ability of our coinflip devices to generate uniform distributions and perform accurate Monte Carlo simulations. Results for the NIST Statistical Test Suite and the NIST SP 800-90B IID track Most Common Value Estimate test for estimating entropy can be found in the supplementary materials for context. Our work demonstrates that multiple TRNGs can be controlled and processed to overcome both device-to-device variation and changes in operating conditions to produce high-quality random samples for a probabilistic computing application.

## Results

Tunnel diodes are degenerately-doped PN junctions with a very narrow depletion region, allowing carriers to tunnel through at low forward biases<sup>37</sup>. Tunneling current increases with voltage initially, but peaks and begins decreasing as fewer and fewer states are available for the carriers to tunnel into, creating a region of negative differential resistance. The current then increases with voltage again as thermionic emission takes effect, demonstrating standard diode behavior<sup>38</sup>. Driving a current through the device near the peak tunneling current (PTC) will cause the voltage across the device to either remain low or stochastically flip to a higher bias, representing tunneling current and thermionic emission current, respectively<sup>19</sup>. A low voltage is interpreted as a logical 0 or “tails” and a high voltage as a logical 1 or “heads”. This stochastic behavior is attributed to trap filling in the band gap<sup>19</sup>. The probability of a high voltage increases for currents approaching the PTC and is 1 above the PTC. This behavior is seen in  $P(H)$  vs. current pulse magnitude in Fig. 1a.

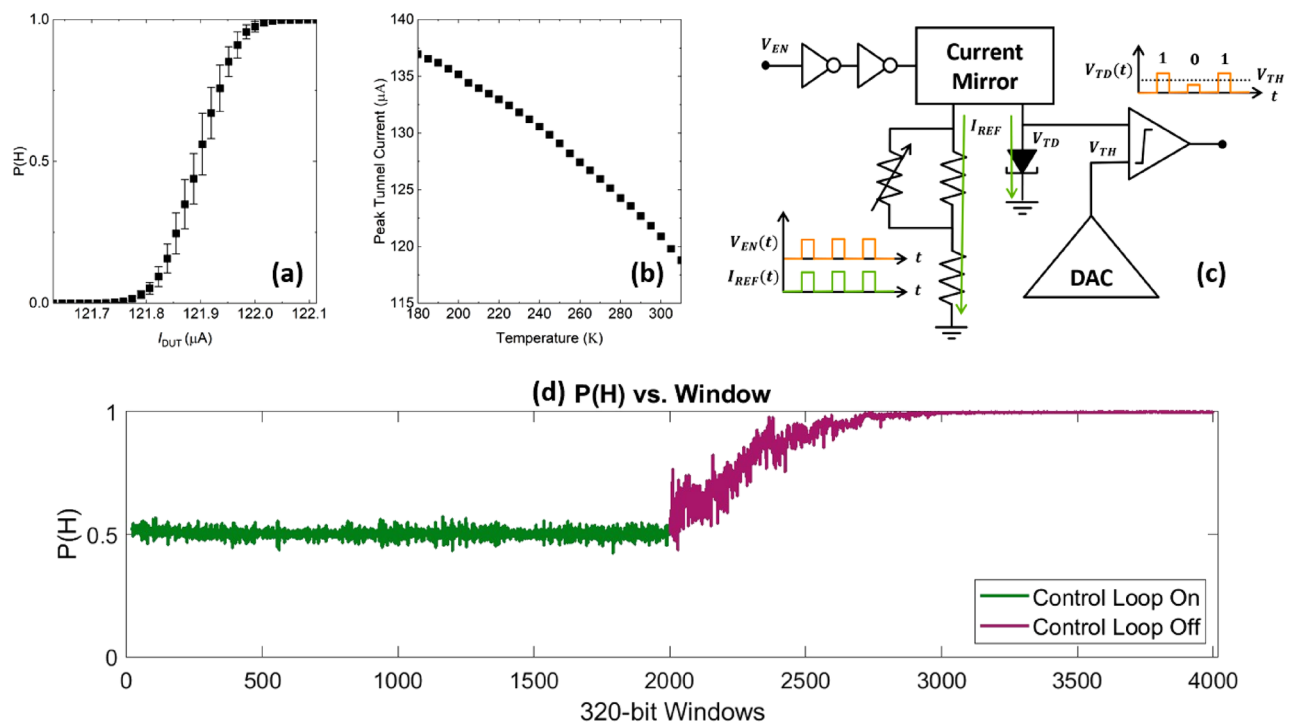
We designed a system for generating random bitstreams with discrete, off-the-shelf TDs on a battery-powered printed circuit board (PCB). A diagram of the system is shown in Fig. 1c. Full details on circuit operation are outlined in the “Methods” section. This system pulses the TDs with current that has a magnitude corresponding to a  $P(H)$  of 0.5 in the middle of the sigmoidal curve in Fig. 1a.  $P(H)$  can be tuned by increasing or decreasing current magnitude along this curve.

However, despite calibrating the TD to a fair weight initially, current pulses of a constant magnitude very quickly lead to deterministic behavior, producing all 1’s or 0’s as shown in Fig. 1d. Change in temperature causes the PTC to vary over time, affecting  $P(H)$ <sup>19</sup>. The change in PTC over a range of temperatures for a TD is shown in Fig. 1b. PTC can vary by up to around 200 nA per degree K, and  $P(H)$  is very sensitive to current, with about 500 nA difference between  $P(H) = 1$  and  $P(H) = 0$ . Therefore, a very small change in temperature can cause a radical shift in  $P(H)$  for the TD. Although a feedback mechanism is recommended in Bernardo-Gavito et al. (2017)<sup>19</sup>, it is not fully analyzed in the text. We compensate for the shifting PTC by implementing a digital control loop in the data collection system, adjusting the current magnitude based on calculated  $P(H)$  of the bitstream. We call the process of controlling the TD to be fair “retuning”. The details of this control loop are outlined in the “Methods” section.

### Retuning single TDs can lead to deterministic behavior

We collect 32M bits with the PCB, sampling continuously. We calculate  $P(H)$  for retuning by averaging together bits in disjoint windows of 32 bits as they are generated, but we calculate  $P(H)$  from disjoint windows of 320 bits (or 10 retuning windows) when plotting.  $P(H)$  for a single, retuned TD is shown in Fig. 2a. Compared to an untuned TD, this bitstream’s  $P(H)$  is tightly distributed around 0.5 despite variations due to temperature or noise. This indicates that our bitstream produces an approximately even number of 1’s and 0’s over the span of several retuning windows. However, this is insufficient evidence of randomness, as predictable patterns can appear in bitstreams with even numbers of 1’s and 0’s, such as a simple square wave that flips each cycle. A fair and random coin should have a 50% chance of heads or tails at any time, regardless of previous results<sup>35</sup>.

To determine how random our fair bitstream is, we start by checking for a dependency on previous bits. Compactly, if  $b_m = 1$  is a bit in our stream, we want to know what the probability that a future bit is also heads, or  $P(b_{m+n} = 1 | b_m = 1)$  for  $n \in \{1, \dots, N\}$ . To calculate this for each  $n$ , we iterate through a section of our bitstream, and whenever we encounter a 1, we look  $n$  bits forward in the stream for  $n \in \{1, \dots, 250\}$  (although 250 is an arbitrary choice and any value of  $N$  would work). If any of these bits are also 1, we increment the



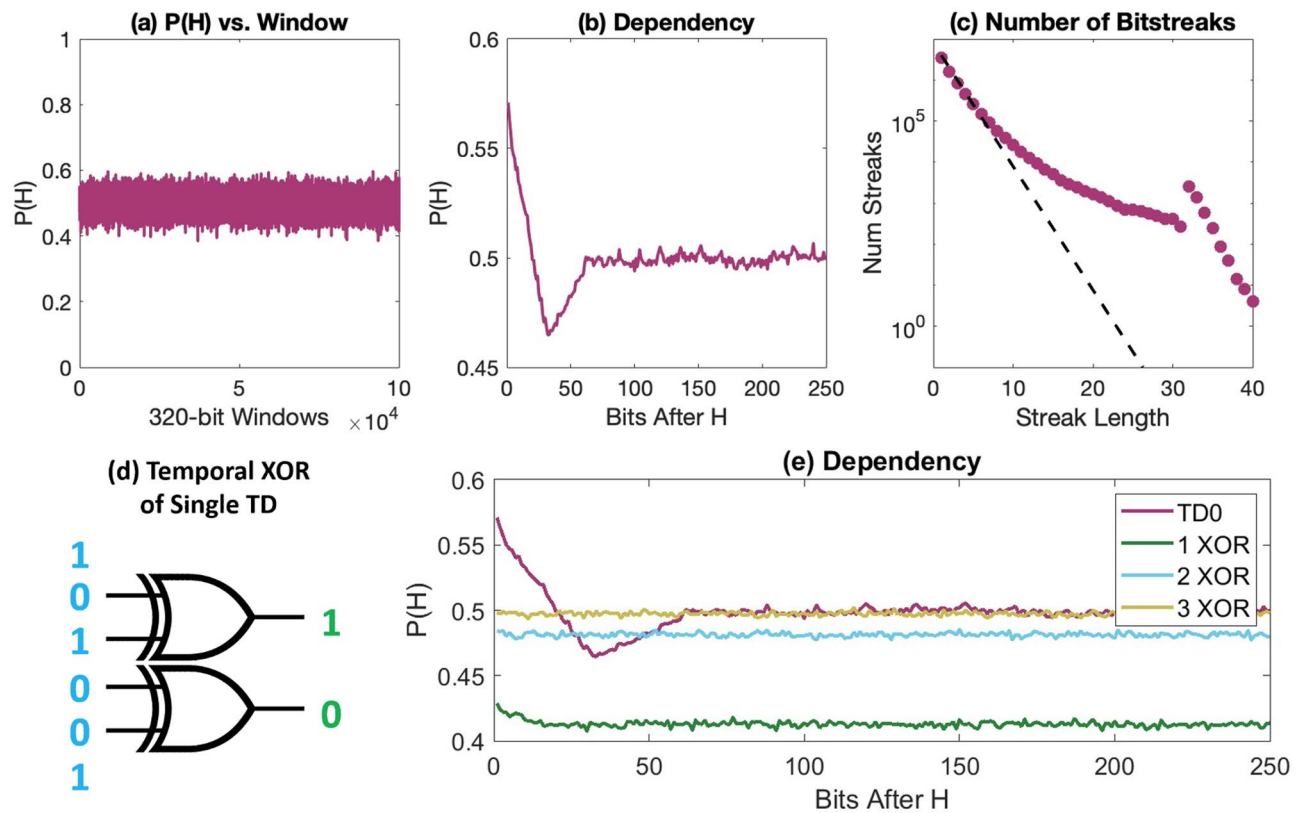
**Fig. 1.** (a)  $P(H)$  measured empirically over 10,000 flips vs. the amplitude of the current pulse through the TD at room temperature. Error bars here represent the standard deviation over 100 trials, which are notably wider near 0.5. (b) Peak tunnel current of a TD vs. the ambient temperature. (c) Our system for stimulating TDs and detecting stochastic results.  $V_{EN}$  enables the current mirror, which drives a  $I_{REF}$  through the load and reflects it through the tunnel diode.  $V_{TD}$  is compared to  $V_{TH}$  to digitize the result into a 1 or 0. (d)  $P(H)$  for a single TD while the control loop is on (green) and after it is turned off (magenta).  $P(H)$  begins to drift before maxing out at 1.

corresponding counter. Once we have gone through the bitstream, we divide the counters by the total number of 1's encountered. More information on this computation can be found in the “Methods” section.

We calculate this probability, which we call the “dependency”, for a single TD with retuning and plot the results in Fig. 2b.  $P(H)$  is shown to be higher than 0.5 given an initial result of heads and decreases linearly with each subsequent flip until 32 bits after the initial result.  $P(H)$  is below 0.5 at this point and abruptly begins increasing again before eventually settling near 0.5. Note that this abrupt change in trajectory for  $P(H)$  happens a full retuning window of 32 bits after the initial heads. TD collection is retuned after a retuning window, which causes this change in slope. Further experiments (shown in the supplementary material) show that changing the retuning window length  $L_{retune}$  results in a corresponding change to the position of this point. These results indicate that our bitstream is not only more predictable than a random coin, but that the retuning algorithm plays a part in this dependency.

The dependency of our TD, which shows that a result of heads is more likely to be followed by another heads, is manifested in the length and number of streaks of 1's and 0's that appear in the bitstream. Intuitively, if a coin is more likely to produce the same result than it is to switch, we can expect longer streaks of both states. We count all streaks of 1's exactly of length  $N$  for  $N$  from 1 to 40. A streak of 1's exactly of length  $N$  is any section of the bitstream that is comprised of a 0, followed by  $N$  consecutive 1's, and then another 0. An exception to this would be a streak that starts at the beginning or finishes at the end of the bitstream, which would not have a starting 0 or ending 0, respectively. We then compare these counts for each  $N$  to their expected values, which are dependent on the total length of the bitstream. Figure 2c shows the counted streaks for our bitstream, which are overwhelmingly higher than expected. Notably, a jump at 32, the value of  $L_{retune}$ , suggests that many retuning windows are full of only 1's. We speculate that this could be attributed to retuning steps that are too large, causing subsequent windows to be filled with all 1's or all 0's.

To combat dependencies and suppress “streaky” behavior, previous works<sup>32,33,39</sup> have used XOR gates in TRNGs. For a bitstream with an unfair weight, XOR operations on sequential bits, also known as a temporal XOR, have been shown to reduce the error<sup>40</sup>. Unfortunately, while this works well for bitstreams with no dependencies and unfair weight, bitstreams with dependencies can result in larger error<sup>40</sup>, disqualifying this method for increasing the unpredictability of our bitstreams. Nevertheless, we compute the temporal XOR throughout the bitstream as in Fig. 2d and plot the dependency after one, two, and three operations in Fig. 2e. After the temporal XOR, the error is quite large, but the dependency is also eliminated. While subsequent temporal XOR operations gradually reduce the error, each one divides the length of the bitstream in half.



**Fig. 2.** A single TD with retuning: (a)  $P(H)$  over time is controlled to stay around 0.5. (b) The dependency, or  $P(H)$  for  $n$  bits given an initial result of heads. (c) Counted streaks of 1's (magenta points), compared to the expected number of streaks of exactly length  $N$  (black dashed line). (d) A temporal XOR of a single TD, computed by taking the XOR of pairs of sequential bits as they are produced and creating a new bitstream of half the length. (e)  $P(H)$  over time given an initial result of heads. Results are shown for a single TD as well as the TD after one, two, and three temporal XOR operations.

### Multiple devices sampled simultaneously can be correlated

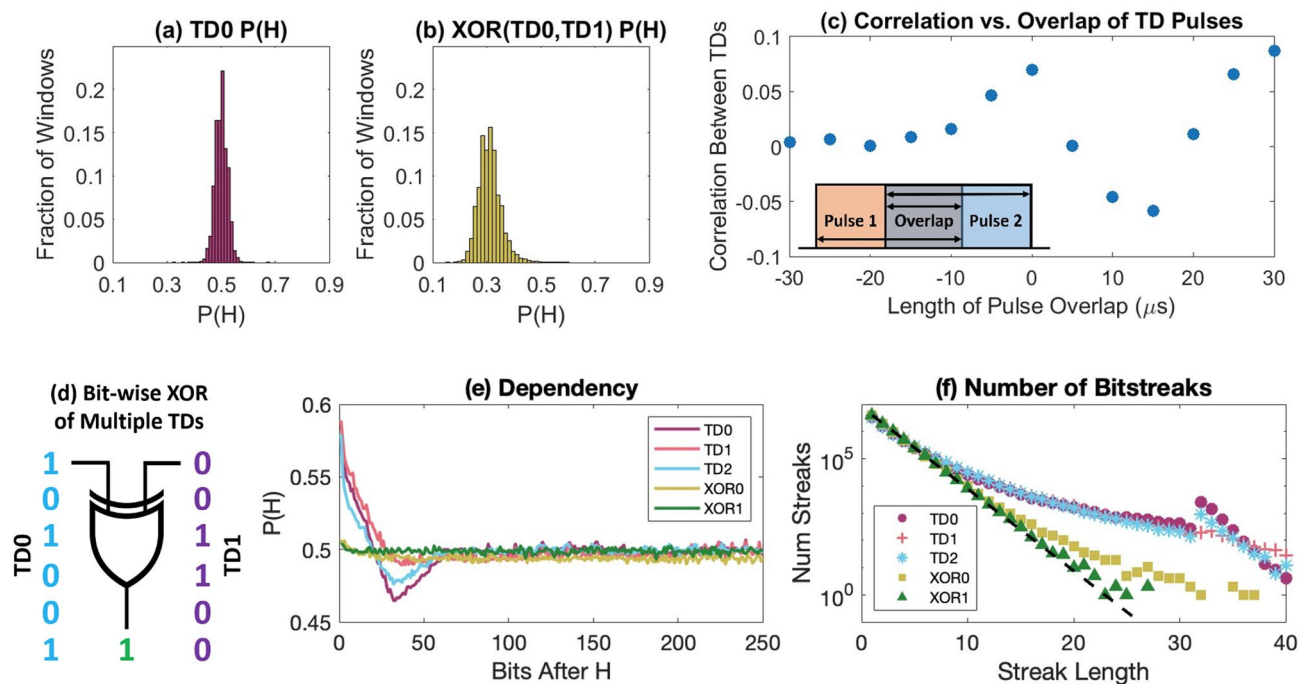
Instead of a temporal XOR, other works have used the XOR of parallel TRNGs, also known as a bit-wise XOR<sup>31–34</sup>. We again take 32M bits, but this time from each of three TDs, labelled TD0, TD1, and TD2, which are pulsed simultaneously. Rather than computing temporal XOR operations like we did with the individual TD bitstreams, we perform bit-wise XOR operations on bitstreams from multiple TDs as in Fig. 3d. Our expectation of the results would be that the individual TDs are producing bits independently, and their bit-wise XOR should remove the dependency as occurred in Fig. 2e without a substantial increase in error due to having no correlation.

Figure 3a and b show the distributions of  $P(H)$  for a single TD and the bitstream resulting from the XOR of two TDs, respectively. The distributions show that the average value of  $P(H)$  is significantly lower than 0.5 and closer to 0.3 instead, indicating that the individual TDs are correlated<sup>40</sup>, causing substantial error.

To illustrate the cause of this correlation, we collect random bits from two TDs and incrementally delay the pulse to one TD (TD1) with respect to the other TD (TD0). Initially, both TD0 and TD1 are pulsed simultaneously for 30  $\mu$ s, and we collect 500k bits in this manner, computing the correlation between the two bitstreams at the end. We then add a 5  $\mu$ s delay to TD1's pulse such that the two pulses overlap in time for 25  $\mu$ s. Another 500k bits are collected, and the process is repeated, delaying TD1 by an additional 5  $\mu$ s such that the two pulses overlap in time for 20  $\mu$ s and computing the correlation between the bitstreams. We repeat an additional 10 times until TD1's pulse starts a full 30  $\mu$ s after TD0's pulse has ended and we have 12 correlations recorded.

The median of six trials are plotted in Fig. 3c, and the inset figure visually shows the overlap mentioned. When the rising or falling edges of the pulses occur approximately concurrently, such as when the pulses completely overlap or one pulse ends as the other begins, the correlation's magnitude is higher. When the falling edge of the first pulse occurs in the middle of the second pulse, this causes a large negative correlation. These results could indicate that the correlation is as a result of cross-talk in the PCB causing one TD's behavior to affect the other. Another possibility is that the overlap of the pulses in time increases the likelihood that shared noise in the PCB, such as in the power rail, might cause the TDs to exhibit similar behavior. Moreover, as the delay between the pulses increases such that they have "negative" overlap (a gap between the pulses), the correlation converges towards 0.





**Fig. 3.** (a)  $P(H)$  distribution for a single TD. (b)  $P(H)$  distribution for the bit-wise XOR of two TDs, pulsed simultaneously. (c) Correlation versus overlap between individual pulses to two TDs. A negative value for overlap indicates a gap between the two pulses. The results indicate that a gap between pulses consistently results in lower correlation (magnitude) than pulsing TDs simultaneously. The insert demonstrates pulse 1 to TD0 (orange) and pulse 2 to TD1 (blue) with an overlap in time (gray). (d) A bit-wise XOR of multiple TDs, computed by taking the XOR of a single bit from multiple TDs in parallel to produce a new bitstream of the same length. (e) Dependency curves for each TD and bit-wise XOR operations on multiple TDs. (f) Counted streaks for each TD and bit-wise XOR operations on multiple TDs.

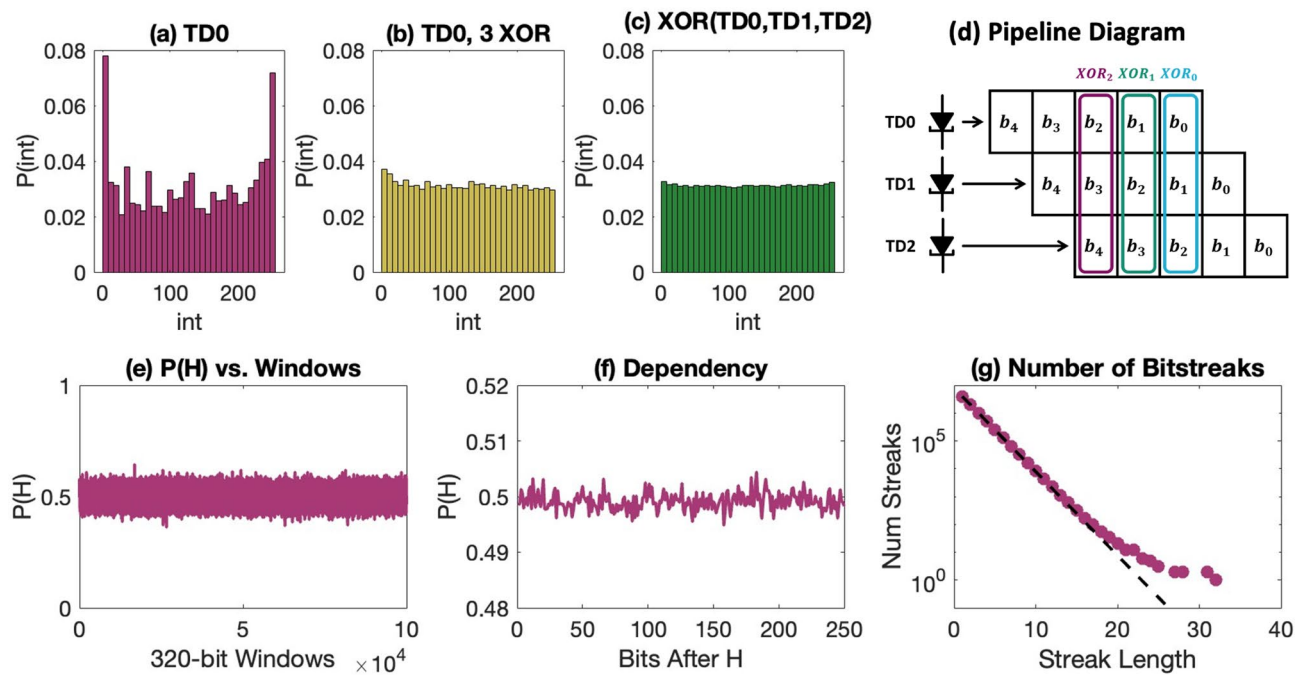
### Cross-device XOR operations lead to high-quality TRNG behavior

We take 32M bits from each of the three TDs, now pulsed out-of-phase with a gap of 15  $\mu\text{s}$  between pulses to different TDs, and compute a bit-wise XOR between two TDs and all three TDs. We note that the data shown in Fig. 2 was from TD0 in this collection. The weight of TD0, TD1, and TD2 are all exactly 0.5 with a negligible error. Previously, temporal XOR operations on TD0 had weights of 0.4421, 0.4814, and 0.4934 for one, two, and three XOR operations, respectively. Comparatively, the weight of the bit-wise XOR of TD0 and TD1 when pulsed simultaneously was 0.311, which increased to 0.5086 after an additional bit-wise XOR with TD2. Now, with the TDs pulsed out-of-phase with a gap of 15  $\mu\text{s}$  between pulses to each TD, the weights from two and three combined TDs are 0.4902 and 0.4997, respectively. While the first two methods consumed three TD bits to generate one high quality random bit, the latter method achieves a comparable weight by consuming two bits to generate one high quality bit. Consuming an additional bit gives the resulting bitstream a lower error than the other two methods. Looking at the dependency curves in Fig. 3e, both bitstreams produced from the bit-wise XOR of multiple TDs have eliminated dependency on previous bits, with a  $P(H)$  given an initial heads that is relatively flat. Furthermore, the error is minimal in the case of the XOR of two TDs and nearly nonexistent in the case of the XOR of three TDs.

The counted streaks of exactly length  $N$  for each individual TD and the XOR of two and three TDs are plotted in Fig. 3f. The bit-wise XOR of two TDs is sufficient to significantly reduce the amount of overcounted streaks of 1's. The additional XOR with TD2 reduces these further, conforming very closely to the line of expected values. As these data points are plotted with a logarithmic y-axis, the cluster of points at the bottom of the plot are off by just a few streaks, for example counting two streaks of 27 in a bitstream of millions of bits when much less than one are expected. While this method of pulsing individual TDs one at a time can be inefficient, decreasing the throughput of high quality bits by a factor of 3, we address this issue by recommending a pipelined pulsing method later in this work.

### A system of TDs is suitable for Monte Carlo approximations

To see these bitstreams in action, we partition them into  $N$ -bit integers and generate distributions of 500k integers each. These distributions are expected to be uniform for an ideal coin<sup>22</sup>. We choose  $N = 8$  and plot the distributions for three distributions: one from a single TD's bitstream, one from a single TD's bitstream after three temporal XOR sweeps, and one from the bit-wise XOR of three individual TDs.  $P(int)$  is the probability of sampling a given integer. These distributions are shown in Fig. 4a–c. The spikes at 0 and 255 for the single TD's distribution can be attributed to its streaky behavior, as many 8-bit samples are all 0's or all 1's. However, the



**Fig. 4.** The distribution of 8-bit numbers generated from: (a) TD0. (b) TD0 after 3 temporal XOR operations. (c) the bit-wise XOR of TD0, TD1, and TD2. Distributions for TD1 and TD2 can be found in the supplementary materials. (d) Diagram of pipelined TDs. Each  $b_x$  represents a bit collected during cycle  $x$ , which are buffered until an XOR operation with bits collected by other TDs during other cycles. Results of pipelining bits from TDs pulsed simultaneously: (e)  $P(H)$  over time is distributed around 0.5. (f)  $P(H)$  for  $n$  bits after an initial heads. (g) Counted streaks of 1's, compared to the expected number of streaks of exactly length  $N$ .

Bitstream	Mean approx. of $\pi$	Std. dev. of approx.
TD0	2.8635	0.1680
TD0, 1 XOR	3.4201	0.1393
TD0, 2 XOR	3.2610	0.1234
TD0, 3 XOR	3.2027	0.1099
XOR(TD0,TD1)	3.1619	0.0702
XOR(TD0,TD1,TD2)	3.1461	0.0541
Pipelined	3.1499	0.0541

**Table 1.** Approximations of  $\pi$ .

distributions for the TD after three temporal XOR operations and the bit-wise XOR of three TDs are both very uniform. The former has a slight bias to lower numbers, which reflects the bitstream's slightly lower weight of 0.4904, indicating more 0's. The latter, with a weight of 0.4997, is very flat, with almost equal probability at each number.

For a more robust test of these distributions, we use them in a Monte Carlo approximation of  $\pi$ . We sample pairs of numbers from a distribution, square and add them, and divide the number of pairs  $> 255^2$  by the total number of pairs<sup>41,42</sup>. This should approximate  $\pi/4$ , which we multiply by 4 to compare to  $\pi$ .

500k 8-bit integers were sampled from each bitstream. These were split into groups of 2k integers. We then ran 250 trials with 1k pairs of 8-bit integers used to approximate  $\pi$ . The mean and standard deviation of these results are recorded in Table 1.

Based on the results in the table, the bit-wise XOR of three TDs has the closest approximation of  $\pi$  as well as the smallest standard deviation. The bit-wise XOR of two TDs also achieves a closer approximation than the three temporal XOR operations on a single TD. Moreover, this is accomplished by consuming only two TD bits for every output bit, compared to the temporal XOR operations which consume three TD bits for every output bit. This implies that combining bitstreams could be a more efficient method of generating high quality bitstreams than post-processing a bitstream from a single source.

## Pipelining XOR reduces correlation without sacrificing throughput

A system where no two devices can be pulsed simultaneously to avoid correlation would be limited in throughput. Crucially, avoiding an XOR of bits collected simultaneously may significantly reduce correlation. If we were to pipeline bit collection with coinflip devices, we could consider the XOR of bits collected simultaneously a data hazard due to their correlation in the same way that we consider simultaneously executing dependent instructions in a CPU a data hazard. In Fig. 4d, we portray a pipeline of three TDs, each producing a bit at every cycle. This allows the system as a whole to continue generating a random bit every cycle rather than every three cycles.

Bits can be buffered in a shift-register after being measured from a TD. On cycle 2, three bits are available for an XOR operation:  $b_0$  from TD0,  $b_1$  from TD1, and  $b_2$  from TD2. In the next cycle, the XOR of  $b_1$  from TD0,  $b_2$  from TD1, and  $b_3$  from TD2 is computed. This continues each cycle, and a new bit is computed from the XOR of bits taken from three TDs on different cycles. Pipelining allows TDs to be pulsed simultaneously while computing an XOR on bits from different cycles to prevent correlation. Thus, we are able to both minimize correlation between devices and eliminate self-dependency of the bitstream through XOR operations.

Taking bits collected from three TDs simultaneously, we simulate this pipeline by circular-shifting one bitstream by one and another bitstream by two before computing a bit-wise XOR. The resulting bitstream's  $P(H)$  over time, counted streaks, and dependency curve are shown in Fig. 4e–g. These show similar results to the XOR of three TDs pulsed out-of-phase. The correlation between TDs is not only reduced through pipelining, but the XOR of the three TDs mitigates the self-dependency as well, as shown in Fig. 4f. Furthermore, using the same uniform distribution sampling and  $\pi$  approximation technique with 250 trials of 1k integer pairs, this bitstream approximates  $\pi$  as 3.1499 with a standard deviation of 0.0541.

While some systems, including stream-based stochastic computing systems<sup>7</sup>, may desire coinflip devices tuned to other values of  $P(H)$ , our work shows that control loops can introduce dependency and other pathologies. Furthermore, while XOR operations reduce bias from a fair coin, there is not an equivalent digital operation currently known for reducing bias from other values of  $P(H)$ . One potential method of sampling a coin with a different value of  $P(H)$  is to sample from a uniform distribution, as we did in our Monte Carlo approximations, and compare the sample to the desired  $P(H)$ , producing a 1 when the sample is  $< P(H)$ . Sampling an  $N$ -bit integer from a uniform distribution and comparing to  $P(H)$  could be accomplished by sampling  $N$  fair coins in parallel to minimize any sacrifices to throughput. Therefore, although a coin cannot be tuned to an unfair weight and improved through a digital operation as a fair coin can, our efforts toward achieving a high quality fair coin can also benefit efforts to design systems of coins with unfair weights.

## Discussion

In this work, we explored integration of stochastic devices into a high quality TRNG, using TDs as representative devices. We show that a control loop can compensate for non-idealities like thermal drift in PTC. This control loop is able to maintain a bias near that of a fair coinflip but introduces dependency into the bitstream that results in long streaks of 1's and 0's. Through bit-wise XOR operations of independent TDs, we are able to mitigate dependency without sacrificing desired weight of the bitstream. Furthermore, while potential nonidealities in our system introduce correlation between individual devices pulsed simultaneously, we found that pulsing TDs out-of-phase results in sufficient independence. We recommend pipelining so that devices can be sampled in parallel while performing XOR operations only on uncorrelated samples.

Some of these concepts, such as the use of a feedback mechanism for maintaining fair coinflips<sup>32,43</sup> and performing bit-wise XOR operations on many lower quality TRNGs<sup>31–34</sup> have been used in TRNG designs before. However, this work shows that an active control loop can even assist simple single-device TRNGs in producing high quality and fair bitstreams. We have also shown experimentally that bit-wise XOR operations can overcome second-order nonidealities in device statistics other than weight, such as dependency on previous results that cause long bitstreaks.

While other coinflip devices such as the MTJ are not as sensitive to temperature variations as TDs (depending on collection method<sup>13</sup>), control loops could potentially be utilized to adapt to other sources of variation such as device-to-device variation or unexpectedly noisy operating conditions. Alternatively, they can be used to compensate for relaxed fabrication or operating constraints, such as fabricating MTJs that are less robust to temperature or using stochastic reading instead of stochastic writing for lower power consumption<sup>13</sup>. Furthermore, systems experiencing a high degree of correlation between coinflip devices might consider pipelining to improve fairness without sacrificing throughput.

Ultimately, we built a true random number generator from nonideal, off-the-shelf components and found that these measures were sufficient to generate random samples suitable for a Monte Carlo approximation of  $\pi$ . The use of control loops and digital operations between devices was shown to be capable of sampling from a uniform distribution, a vital step toward many powerful Monte Carlo simulation techniques. We look to the future and envision large arrays of integrated coinflip devices with control logic, buffers, and XOR gates that facilitate low power, high throughput generation of trillions of random bits per second for use in artificial intelligence, stochastic computing, scientific computing, and other vital applications.

## Methods

### Data collection system

Our data was collected with a custom PCB designed in KiCAD. We used MP1103 tunnel diodes from M-Pulse Microwave Inc. To sample a TD, a Microchip Technology ATmega328P microcontroller sends a pulse ( $V_{EN}$ ) to a Texas Instruments CD74HC4050 buffer, which drives the pulse across a resistive load to generate a reference current ( $I_{REF}$ ). This load consists of an Analog Devices AD5175 rheostat (or potentiometer) in parallel and

series with SMD resistors. An Analog Devices ADL5315 current mirror reflects the reference current through the TD. The voltage across the TD ( $V_{TD}$ ) is compared to an intermediate voltage between the high and low states (the threshold voltage, or  $V_{TH}$ ) by a Analog Devices LT1116 comparator to determine the state of the TD. The microcontroller reads and then writes the result to an SD card on the PCB. The entire module is placed in a large metal box during collection to mitigate the influence of external noise sources. Our ability to collect data in noisier environments is limited by the dynamic range of our potentiometers and number of TDs integrated on the PCB, but a system with sufficient range should be able to comfortably adapt to sources of noise such as wireless signals and vibrations. For details regarding robustness of TRNGs to more intense noise like ionizing radiation, see Martín et al.<sup>44</sup>. Bits are collected continuously, and  $P(H)$  is determined by averaging together bits in disjoint windows of 32 as they are produced.

To compensate for the shifting PTC, the potentiometer's resistance can be controlled by the microcontroller using a proportional control loop. If  $P(H) > 0.5$  or  $< 0.5$ , the potentiometer can be set to higher or lower resistance, respectively. In the proportional controller, the process variable is  $P(H)$  for the TD and the control variable is the magnitude of the current pulses. The output bits of the TD are averaged over 32 flips to produce the empirical  $P(H)$ , which is passed through the controller equations

$$e = 0.5 - P(H), \quad (1)$$

$$i = i + k_p e, \quad (2)$$

where  $e$  is the error signal and  $i$  is the magnitude of the current for sampling the TD. The parameter  $k_p$  is the proportional feedback gain, set to  $8 \times 10^{-9}$  in our controller. The length of the window over which the  $P(H)$  is estimated ( $L_{\text{retune}}$ ), as well as the constant  $k_p$ , were chosen through a sweep of the search-space.

### Tuning the TD control loop

We searched  $L_{\text{retune}} \in \{16, 32, 64, 128\}$  and  $k_p \in \{1 \times 10^{-9}, 2 \times 10^{-9}, 4 \times 10^{-9}, 8 \times 10^{-9}, 16 \times 10^{-9}\}$  to find the optimal controller parameters. After collecting 500k bits with each configuration of  $L_{\text{retune}}$  and  $k_p$ , we plotted  $P(H)$  and dependency curves for each TD.  $P(H)$  and dependency curves for TD0, TD1, and TD2 are shown in the supplementary materials. We identified a trade-off between fairness (weight near 0.5) and dependency (deviation of  $P(H)$  from 0.5 given an initial heads). While  $P(H)$  seems to improve with increased  $k_p$  and decreased  $L_{\text{retune}}$  for all TDs, it can increase dependence in some cases. Notably, TD2 demonstrates this the most for  $k_p = 16 \times 10^{-9}$  and  $L_{\text{retune}} = 16$ , where  $P(H)$  is tightly distributed around 0.5, but the dependency curve oscillates significantly around 0.5. We chose  $L_{\text{retune}} = 32$  and  $k_p = 8 \times 10^{-9}$  as these values resulted in an acceptable trade-off between fairness and dependency.

### Data selection and analysis

When pulsing multiple TDs simultaneously, the current required for a fair weight decreased compared to that of the TDs pulsed sequentially. Consequently the resistance required to maintain a fair coin went outside of the operating range of the potentiometers, causing  $P(H)$  to increase to 1. Eventually, the control variable returned again to the potentiometer's operating range, and  $P(H)$  returned to its narrow margin around 0.5 (refer to the potentiometer value and  $P(H)$  over time shown in the supplementary materials). Our analysis used the 32M bits collected after the control variable returned to the potentiometer's operating range. To ensure fair comparison between this data and data collected from sequentially pulsed TDs, we used the same slice of 32M bits from the sequentially pulsed bitstream. Data was read from the SD card and analyzed by a MATLAB script.

### Calculation of expected streaks of 1 and dependency of a bitstream

Assume we have a bitstream of  $M$  bits and want to find the number of streaks of 1's of exactly length  $N$ . This value, which we will call  $Y$ , is calculated by

$$Y_N = \delta_1 + \delta_M + \sum_{i=2}^{M-N-1} \delta(b_i, \dots, b_{i+N}), \quad (3)$$

where  $\delta$  is 1 if the given window of bits is all 1's with a 0 on either side, written as  $b_{i-1} = 0$ ,  $b_{i+N+1} = 0$ , and  $b_i = \dots = b_{i+N} = 1$ , but is 0 otherwise. The term  $\delta_1$  represents the corner case of the first  $N$  bits, which have no preceding bit, and it is 1 if  $b_1 = \dots = b_N = 1$  and  $b_{N+1} = 0$ , but is 0 otherwise. The term  $\delta_M$  represents the corner case of the last  $N$  bits, which have no succeeding bit, and it is 1 if  $b_{M-N} = \dots = b_M = 1$  and  $b_{M-N-1} = 0$ , but is 0 otherwise. Assuming independence of the  $b_i$ 's, the expected value of each term in the summation is equal to the probability that the term is 1. Therefore, the expected value of  $Y_N$  is

$$\mathbb{E}(Y_N) = 2(1-p)p^N + (M-N-2)(1-p)^2p^N. \quad (4)$$

The first term represents the two corner cases, and the second term represents the summation. We can calculate the expected value of  $Y_N$  for all  $N$  from 1 to the length of the bitstream,  $M$ , but this value quickly decreases to be much lower than 1, even for a very large  $M$ . For 32M bits, the expected value for streaks of exactly length 23 is slightly less than 1 and continues to decrease to approximately 0 for streaks of 30 or longer.

To calculate the dependency of a bitstream, we examined windows of length  $n$  after every 1 appearing in a bitstream. The bit-wise average of all such windows produces the effective probability of encountering a 1 within



$n$  bits of an initial 1. For the sake of a manageable data array size, we took a slice of 100K bits from the bitstream and chose  $n = 250$  for our analysis.

### Data availability

Data from the tunnel diodes will be made available upon reasonable request to the corresponding authors and pending organizational approval.

### Code availability

Code, including the MATLAB script used for processing bitstream data and relevant microcontroller code, is available upon reasonable request to the authors and pending organizational approval.

Received: 29 October 2024; Accepted: 30 May 2025

Published online: 01 July 2025

### References

- Daniely, A., Frostig, R. & Singer, Y. Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity. *Adv. Neural Inf. Process. Syst.* **29** (2016).
- Glorot, X. & Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 249–256 (JMLR Workshop and Conference Proceedings, 2010).
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **15**, 1929–1958 (2014).
- Wan, L., Zeiler, M., Zhang, S., Le Cun, Y. & Fergus, R. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, 1058–1066 (PMLR, 2013).
- Goodfellow, I. et al. Generative adversarial nets. *Adv. Neural Inf. Process. Syst.* **27** (2014).
- Karras, T., Laine, S. & Aila, T. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 4401–4410 (2019).
- Alaghi, A. & Hayes, J. P. Survey of stochastic computing. *ACM Trans. Embed. Comput. Syst.* **12**, 1–19 (2013).
- Wierling, C. et al. Prediction in the face of uncertainty: A Monte Carlo-based approach for systems biology of cancer treatment. *Mutat. Res./Genet. Toxicol. Environ. Mutagen.* **746**, 163–170 (2012).
- Deng, L.-Y., Guo, R., Lin, D. K. & Bai, F. Improving random number generators in the Monte Carlo simulations via twisting and combining. *Comput. Phys. Commun.* **178**, 401–408 (2008).
- Carlson, J. et al. Quantum Monte Carlo methods for nuclear physics. *Rev. Mod. Phys.* **87**, 1067 (2015).
- Misra, S. et al. Probabilistic neural computing with stochastic devices. *Adv. Mater.* **35**, 2204569 (2023).
- Rehm, L. et al. Stochastic magnetic actuated random transducer devices based on perpendicular magnetic tunnel junctions. *Phys. Rev. Appl.* **19**, 024035 (2023).
- Liu, S. et al. Random bitstream generation using voltage-controlled magnetic anisotropy and spin orbit torque magnetic tunnel junctions. *IEEE J. Explor. Solid-State Comput. Devices Circuits* **8**, 194–202 (2022).
- Jiang, H. et al. A novel true random number generator based on a stochastic diffusive memristor. *Nat. Commun.* **8**, 882 (2017).
- Stanco, A. et al. Efficient random number generation techniques for CMOS single-photon avalanche diode array exploiting fast time tagging units. *Phys. Rev. Res.* **2**, 023287 (2020).
- Zhou, H., Li, J., Pan, D., Zhang, W. & Long, G. Quantum random number generator based on quantum tunneling effect. *arXiv preprint arXiv:1711.01752* (2017).
- Zhou, H., Li, J., Zhang, W. & Long, G.-L. Quantum random-number generator based on tunneling effects in a Si diode. *Phys. Rev. Appl.* **11**, 034060 (2019).
- Tontini, A., Gasparini, L., Massari, N. & Passerone, R. SPAD-based quantum random number generator with an  $n^{\text{th}}$ -order rank algorithm on FPGA. *IEEE Trans. Circuits Syst. II Express Briefs* **66**, 2067–2071 (2019).
- Bernardo-Gavito, R. et al. Extracting random numbers from quantum tunnelling through a single diode. *Sci. Rep.* **7**, 17879 (2017).
- Aungskunsiri, K. et al. Random number generation from a quantum tunneling diode. *Appl. Phys. Lett.* **119** (2021).
- Aungskunsiri, K. et al. Multiplexing quantum tunneling diodes for random number generation. *Rev. Sci. Instrum.* **94** (2023).
- Billingsley, P. *Probability and Measure* 3rd ed. (Wiley, 2012).
- Thomas, D. B., Luk, W., Leong, P. H. & Villasenor, J. D. Gaussian random number generators. *ACM Comput. Surv. (CSUR)* **39**, 11–es (2007).
- Camsari, K. Y., Salahuddin, S. & Datta, S. Implementing p-bits with embedded MTJ. *IEEE Electron Device Lett.* **38**, 1767–1770 (2017).
- Faria, R., Camsari, K. Y. & Datta, S. Low-barrier nanomagnets as p-bits for spin logic. *IEEE Magn. Lett.* **8**, 1–5 (2017).
- Kim, J. et al. Fully CMOS-based p-bits with a bistable resistor for probabilistic computing. *Adv. Funct. Mater.* 2307935 (2024).
- Woo, K. S. et al. Probabilistic computing using CuO. 1TeO<sub>9</sub>/HfO<sub>2</sub>/Pt diffusive memristors. *Nat. Commun.* **13**, 5762 (2022).
- Chowdhury, S. et al. A full-stack view of probabilistic computing with p-bits: Devices, architectures, and algorithms. *IEEE J. Explor. Solid-State Comput. Devices Circuits* **9**, 1–11 (2023).
- Kaiser, J. & Datta, S. Probabilistic computing with p-bits. *Appl. Phys. Lett.* **119** (2021).
- Singh, N. S. et al. CMOS plus stochastic nanomagnets enabling heterogeneous computers for probabilistic inference and learning. *Nat. Commun.* **15**, 2685 (2024).
- Xu, X. & Wang, Y. High speed true random number generator based on FPGA. In *2016 International Conference on Information Systems Engineering (ICISE)*, 18–21 (IEEE, 2016).
- Kinniment, D. & Chester, E. Design of an on-chip random number generator using metastability. In *Proceedings of the 28th European Solid-State Circuits Conference*, 595–598 (IEEE, 2002).
- Kohlbrener, P. & Gaj, K. An embedded true random number generator for FPGAs. In *Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays*, 71–78 (2004).
- Sunar, B., Martin, W. J. & Stinson, D. R. A provably secure true random number generator with built-in tolerance to active attacks. *IEEE Trans. Comput.* **56**, 109–119 (2006).
- Rukhin, A. et al. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications* vol. 22 (US Department of Commerce, Technology Administration, National Institute of, 2001).
- Ghera, D., Parakh, A. & Mezei, M. Comparison of a quantum random number generator with pseudorandom number generators for their use in molecular monte carlo simulations. *J. Comput. Chem.* **38**, 2713–2720 (2017).
- Esaki, L. New phenomenon in narrow germanium p-n junctions. *Phys. Rev.* **109**, 603 (1958).
- Hall, R. Tunnel diodes. *IRE Trans. Electron Devices* **7**, 1–9 (1960).
- Fischer, V. & Drutarovský, M. True random number generator embedded in reconfigurable hardware. In *International Workshop on Cryptographic Hardware and Embedded Systems*, 415–430 (Springer, 2002).

40. Davies, R. B. Exclusive or (xor) and hardware random number generators.
41. Strbac-Savic, S., Miletic, A. & Stefanović, H. The estimation of pi using Monte Carlo technique with interactive animations. In *8th International Scientific Conference "Science and Higher Education in Function of Sustainable Development-SED"*, vol. 2015 (2015).
42. Metropolis, N. & Ulam, S. The Monte Carlo method. *J. Am. Stat. Assoc.* **44**, 335–341 (1949).
43. Tokunaga, C., Blaauw, D. & Mudge, T. True random number generator with a metastability-based quality control. *IEEE J. Solid-State Circuits* **43**, 78–85 (2008).
44. Martín, H., Vaskova, A., López-Ongil, C., San Millán, E. & Portela-García, M. Effect of ionizing radiation on TRNGs for safe telecommunications: Robustness and randomness. In *2014 IEEE 20th International On-Line Testing Symposium (IOLTS)*, 202–205 (IEEE, 2014).

## Acknowledgements

The authors acknowledge support from the DOE Office of Science (ASCR/BES) Microelectronics Co-Design project COINFLIPS. This work was performed, in part, at the Center for Integrated Nanotechnologies, an Office of Science User Facility operated for the U.S. DOE Office of Science. Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

## Author contributions

CRA conceived of and implemented the tunnel diode sampling circuit. BT collected bitstream data from the tunnel diodes and performed analysis on the data. CRA and SM provided guidance on data analysis. JDS consulted on the mathematical analysis of data. JBA assisted in outlining the scope of the project. BT drafted the manuscript. All authors reviewed the results and contributed to the manuscript.

## Declarations

### Competing interests

The authors claim no competing interests.

## Additional information

**Supplementary Information** The online version contains supplementary material available at <https://doi.org/10.1038/s41598-025-05171-1>.

**Correspondence** and requests for materials should be addressed to B.T. or C.R.A.

**Reprints and permissions information** is available at [www.nature.com/reprints](http://www.nature.com/reprints).

**Publisher's note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

**Open Access** This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License, which permits any non-commercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if you modified the licensed material. You do not have permission under this licence to share adapted material derived from this article or parts of it. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by-nc-nd/4.0/>.

© The Author(s) 2025