FPGA-based true random number generation using circuit metastability with adaptive feedback control

Citation: Majzoobi, Mehrdad, Farinaz Koushanfar, and Srinivas Devadas. "FPGA-Based True Random Number Generation Using Circuit Metastability with Adaptive Feedback Control." Cryptographic Hardware and Embedded Systems – CHES 2011. Ed. Bart Preneel & Tsuyoshi Takagi. LNCS Vol. 6917. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. 17–32.

As Published: http://dx.doi.org/10.1007/978-3-642-23951-9_2

Publisher: Springer Berlin / Heidelberg

Persistent URL: http://hdl.handle.net/1721.1/73860

Version: Author's final manuscript: final author's manuscript post peer review, without publisher's formatting or copy editing

FPGA-based True Random Number Generation using Circuit Metastability with Adaptive Feedback Control

Mehrdad Majzoobi[1] and Farinaz Koushanfar[1] and Srinivas Devadas[2]

[1] Rice University, ECE
Houston, TX 77005
{mehrdad.majzoobi,farinaz}@rice.edu
[2] Massachusetts Institute of Technology, CSAIL
Cambridge, MA 02139
{devadas}@mit.edu

Abstract. The paper presents a novel and efficient method to generate true random numbers on FPGAs by inducing metastability in bi-stable circuit elements, e.g. flip-flops. Metastability is achieved by using precise programmable delay lines (PDL) that accurately equalize the signal arrival times to flip-flops. The PDLs are capable of adjusting signal propagation delays with resolutions higher than fractions of a pico second. In addition, a real time monitoring system is utilized to assure a high degree of randomness in the generated output bits, resilience against fluctuations in environmental conditions, as well as robustness against active adversarial attacks. The monitoring system employs a feedback loop that actively monitors the probability of output bits; as soon as any bias is observed in probabilities, it adjusts the delay through PDLs to return to the metastable operation region. Implementation on Xilinx Virtex 5 FPGAs and results of NIST randomness tests show the effectiveness of our approach.

1 Introduction

True Random Number Generators (TRNG) are important security primitives that can be used to generate random numbers for various essential tasks including the generation of (i) secret or public keys, (ii) initialization vectors and seeds for cryptographic primitives and pseudo-random number generators, (iii) padding bits, and (iv) nonces (numbers used once). Since modern cryptographic algorithms often require large key sizes, generating the keys from a smaller sized seed will significantly reduce the entropy of the long keys. In other words, by performing a brute-force attack only on the seed that generated the key, one could break the crypto system. In addition, for applications that demand a constant high-speed and high-quality generation of keys, e.g. secure web servers, algorithmic approaches to pseudo-random number generation are typically inefficient, and hardware accelerated mechanisms are highly desired. True random numbers also find applications in gaming, gambling and lottery drawings.

To date, numerous TRNG designs have been proposed and implemented. Each design uses a different mechanism to extract randomness from some underlying physical phenomena that exhibit uncertainty or unpredictability. Examples of sources of randomness include thermal and shot noise in circuits, secondary effects such as clock jitter and metastability in circuits, Brownian motion, atmospheric noise, nuclear decay, and random photon behavior.

Because of its flexibility and fast time to market, FPGA has become a popular platform for implementing many cryptographic systems that include TRNGs as an essential block. It is important to develop new FPGA TRNG solutions because: (i) not all hardware TRNG methods available for ASICs or other platforms are amenable to FPGA implementation; (ii) the existing FPGA TRNGs have limitations in terms of the throughput-per-unit-area and could be improved; and (iii) active adversarial attacks as well as variations in operational conditions such as fluctuations in temperature and voltage supply may bias and disturb the randomness of TRNGs output bitstream. Since most of the state-of-the-art TRNGs operate in an open-loop fashion, it is important to incorporate a mechanism to constantly provide a feedback signal to adaptively adjust the TRNG system parameters to increase its output bit randomness.

In this work, we propose a novel technique to generate true random numbers on FPGA using the flip-flop metastability as a source of randomness. The introduced TRNG core operates within a closed-loop feedback system that actively monitors the output bit probabilities over windows of bit sequences and generates a proportional feedback signal based on any observed bias in the bit probabilities. The feedback mechanism is made possible by performing fine delay tuning using high precision PDLs with picosecond resolution. The delay tuning ensures that the signals arrive simultaneously at the flip-flop to drive it into a metastable state. Our contributions are as follows.

- We introduce an FPGA-based TRNG system that utilizes flip-flop metastability as the source of randomness.
- A novel feedback mechanism is introduced that performs auto-adjustment on delays in order to make the metastability condition more likely to happen.
- We demonstrate the use of a PDL to perform fine tuning with a precision of higher than a fraction of a pico-second.
- Highly accurate delay measurement results for PDL are demonstrated.
- The proposed TRNG system is implemented on Xilinx Virtex 5 FPGA; the hardware evaluation results demonstrate the high throughput-per-area and the high quality (i.e., true randomness) of the produced output bits.

2 Related work

The work in [15] uses sampling of phase jitter in oscillator rings to generate a sequence of random bits. The output of a group of identical ring oscillators are fed to a parity generator function (i.e., a multi-input XOR). The output is constantly sampled by a D-flipflop driven using the system clock. In absence of noise and identical phases, the XOR output would be constant (and deterministic). However, in presence of a phase jitter, glitches with varying non-deterministic lengths appear at the output. An implementation of this method on Xilinx Virtex II FPGAs was demonstrated in [12].

Another type of TRNG is introduced in [11] that exploits the arbiter-based Physical Unclonable Function (PUF) structure. PUF provides a mapping from a set of input challenges to a set of output responses based on unique chip-dependent manufacturing

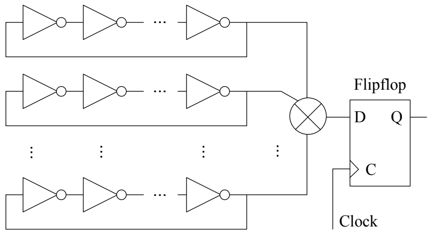


Fig. 1: TRNG based on sampling the ring oscillator phase jitter.

process variability. The arbiter-based PUF structure introduced in [3], compares the analog delay difference between two parallel timing paths. The paths are built identically, but the physical device imperfections make their timing different. A working implementation of the arbiter-based PUF was demonstrated on both ASICs [5] and FPGA [8, 13]. Unlike PUFs where reliable response generation is desired, the PUF-based TRNG goal is to generate unstable responses by driving the arbiter into the metastable state. This is essentially accomplished through violating the arbiter setup/hold time requirements. The PUF-based TRNG in [11] searches for challenges that result in small delay differences at the arbiter input which then cause unreliable response bits.

To improve the quality of the output TRNG bitsteam and increase its randomness, various post-processing techniques are often performed. The work in [15] introduces resilient functions to filter out deterministic bits. The resilient function is implemented by a linear transformation through a generator matrix commonly used in linear codes. The hardware implementation of resilient function is demonstrated in [12] on Xilinx Virtex II FPGAs. The TRNG after post processing achieves a throughput of 2Mbps using 110 ring oscillators with 3 inverters in each. A post-processing may be as simple as von Neumann corrector [10] or may be more complicated such as an extractor function [1] or even a one-way hash function such as SHA-1 [4].

Besides improving the statistical properties of the output bit sequence and removing biases in probabilities, post-processing techniques increase the TRNG resilience against adversarial manipulation and variations in environmental conditions. An active adversary may attempt to bias the output bit probabilities to reduce their entropy. Post-processing techniques typically govern a trade-off between the quality (randomness) of the generated bit versus the throughput. Other online monitoring techniques may be used to assure a higher quality for the generated random bits. For instance, in [11], the generated bit probabilities are constantly monitored; as soon as a bias in the bit sequence is observed, the search for a new challenge vector producing unreliable response bits is initiated. A comprehensive review of hardware TRNGs can be found in [14]. The TRNG system proposed in this paper simultaneously provides randomness, robustness, low area overhead, and high throughput.

3 Programmable delay lines

Programmable delay lines (PDLs) alter the signal propagation delay in a controlled fashion. The common mechanisms used to change the delay includes (i) varying the effective load capacitance, (ii) modifying the device current drive (by increasing/decreasing the effective threshold voltage by body biasing), or (iii) incrementally altering the length of the signal propagation path. The first two methods are often employed in either analog fashion and/or in application specific integrated circuits (ASICs) and are not amenable to FPGA implementation.

On reconfigurable digital platforms such as FPGAs, PDLs can be implemented by only changing the signal propagation path length or by altering the circuit fanout that modifies the effective load capacitance. The latter is only feasible if dynamic reconfiguration is available. In other words, changing circuit fanout requires topological changes to the circuit which in turn needs a new configuration. In [2], a technique is proposed to alter the propagation path length by letting the signal bounce a few times inside the switch matrices of FPGA instead of a direct and straight connection. The concept is illustrated in Figure 2. In the switch matrix on the left side, the signal bounces three times off the switch edges before it exits the switch. In the right switch, the signal only bounces once and as a result a shorter propagation path length and a smaller delay is achieved. However, changing the switch connections points and routings require a new configuration, and doing so during the circuit operation is only possible by dynamic reconfigurability.

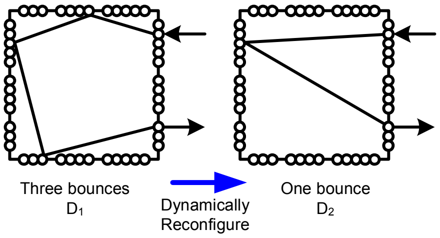


Fig. 2: A PDL implemented by altering the signal routing inside FPGA switch matrix.

In this paper, we use a novel technique to vary the signal propagation path length in minute increments/decrements by only using a single lookup table (LUT). The technique changes the propagation path inside the LUT. We use an example to illustrate the concept. Figure 3 shows a 3-input lookup table. The LUT consists of a set of SRAM cells that store the intended functionality and a tree-like structure of multiplexers (MUXs) that enables selection of each individual SRAM cell content. The inputs to the MUXs serve as an address that points to the SRAM cell whose content is selected to appear at the output of LUT. The LUT in Figure 3 is programmed to implement an inverter, where the LUT output is always an inversion of its first input (A_1). The other inputs of LUT, namely A_2 and A_3 are functionally “don’t-cares”, but their value affect the signal proposition path from A_1 to the output. For instance, as shown in Figure 3, for $A_2A_3 = 00$ and $A_2A_3 = 11$ the signal propagation path length (and thus the propagation delay) from A_1 to O are the shortest and the longest respectively. Xilinx Virtex 5, Virtex 6, and Spartan 6 devices utilize 6-input LUTs. Therefore, by using one single LUT, a programmable delay inverter/buffer with five control inputs can be implemented. The five inputs provide $2^5 = 32$ discrete levels for controlling the delay. The measurement data presented in Section 6 obtained from Xilinx Virtex 5 FPGAs suggest that the maximum delay difference from each LUT is approximately 10 pico seconds.

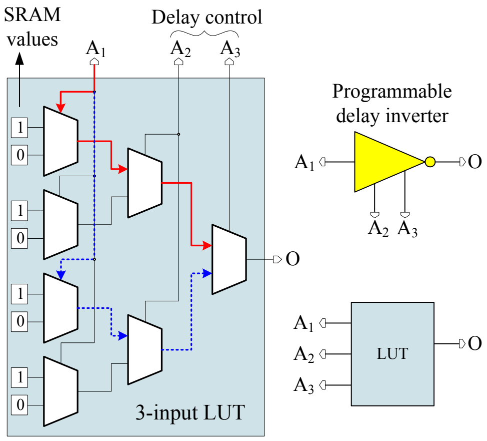


Fig. 3: Precision PDL using a single LUT.

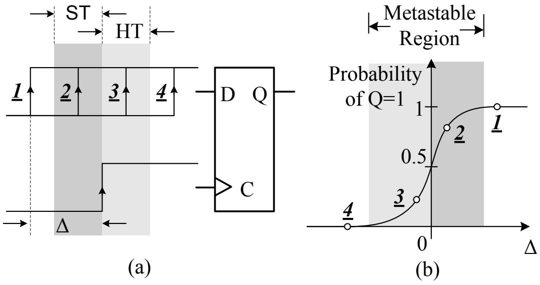
4 Metastability

The proposed TRNG induces metastable conditions in bi-stable logic circuit elements, i.e., flip-flops and latches. The metastable state eventually resolves to a stable state, but the resolution process is extremely sensitive to operational conditions and circuit noise, rendering the result highly unpredictable.

A 'D' flip-flop samples its input at the rising edge of the clock. If sampling takes place within a narrow time window before or after the input signal transitions, a race condition occurs. The race condition takes the flip-flop into a metastable oscillating state. The time window around the sampling moment is typically referred to as setup/hold time. The oscillation eventually settles onto a stable final state of either one or zero. This phenomenon is demonstrated in Figure 4. Note that the probability of settling onto '1' is a monotonic function of the time difference (Δ) between the moment sampling happens and the moment transition occurs at the input. In fact, as shown in [16, 9, 7], the probability can be accurately modeled by a Gaussian CDF. If the delay difference of the arriving signals is represented by Δ and σ is proportional to the width of the setup/hold time window, then the probability of the output being equal to one can be written as:

$$\text{Prob}\{Out = 1\} = Q(\frac{\Delta}{\sigma}), \tag{1}$$

where $Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^{\infty} exp(\frac{-u^2}{2})du$. This model can be explained by Central Limit Theorem. Figure 4 demonstrates four scenarios for different signal arrival times. The corresponding probabilities for the scenarios are marked by the scenario number on the probability plot. For instance, in scenarios 1 and 4, since the delay difference is larger than the setup/hold time of the flip-flop, the output is completely deterministic. In order to obtain completely non-deterministic and unpredictable output bits with equal probabilities (Prob$\{Output{=}1\}$ = Prob$\{Output{=}0\}$ = 0.5), our method forces the flip-flop into metastability by tuning sampling and signal arrival times so they occur as simultaneously as possible (driving $\Delta \to 0$) using the PDLs.



(a)

(b)

Fig. 4: (a) Flip-flop operation under four sampling scenarios, (b) probability of output being equal to '1' as a function of the input signals delay difference (Δ). The numbers on the probability plot correspond to each signal arrival scenario.

5 TRNG System Design

To drive the flip-flop into its metastable state, we use an at-speed monitor-and-control mechanism that establishes a closed loop feedback system. The monitor module keeps track of the output bit probabilities over repeated time intervals. It then passes on the information to the control unit. The control unit based on the received probability information decides to add/subtract the delay to/from top/bottom paths to calibrate the delay difference so that it gets closer to zero. For instance, if the output bits are highly skewed towards 1, then the delay difference (Δ) must be decreased by increasing the top path delay to balance the probabilities. Figure 5 (a) demonstrates this concept.

A straightforward implementation of the monitoring unit can be realized by using a counter. The counter value is incremented every time the flip-flop outputs '1' and is decremented whenever the flip-flop generates a '0'. This is analogous to performing a running sum over the sequence of output bits where zeros are replaced by '-1'. If zeros and ones are equally likely, the value of the counter will stay almost constant. A feedback signal is generated proportional to any deviation from this constant steady state value. The generated error signal is fed back to the signal-to-delay transducer, i.e., the PDL. The delay difference (Δ) is updated/corrected based on the feedback signal.

The described system is in effect a proportional-integral (PI) controller. The system is depicted in Figure 5 (b). In this figure, Δ_b is the constant bias/skew in delays caused

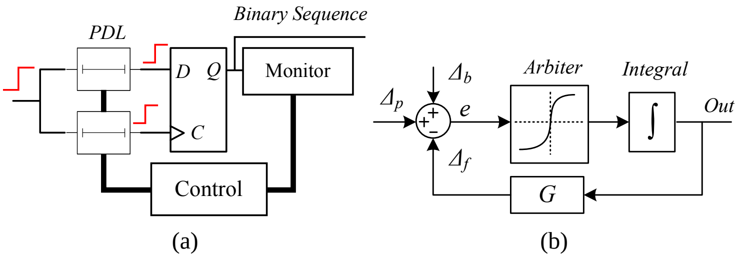


Fig. 5: The TRNG system model.

by the routing asymmetries. Δ_p is the delay difference induced by changes in environmental and operational conditions such as temperature and supply voltage, and/or delay difference imposed by active adversarial attacks. Δ_f is the correction feedback delay difference injected by the PDL based on the counter value. Equation 2 expresses the total delay difference at the input of the flip-flop. G represents transformation carried out by the PDL from the counter binary value to an analog delay difference. The arbiter and integrator refer to the flip-flop and counter respectively. Therefore, the following relationship holds;

$$\Delta = \Delta_p + \Delta_b - \Delta_f. \tag{2}$$

An example PDL-based implementation of the TRNG system is shown in Figure 6.

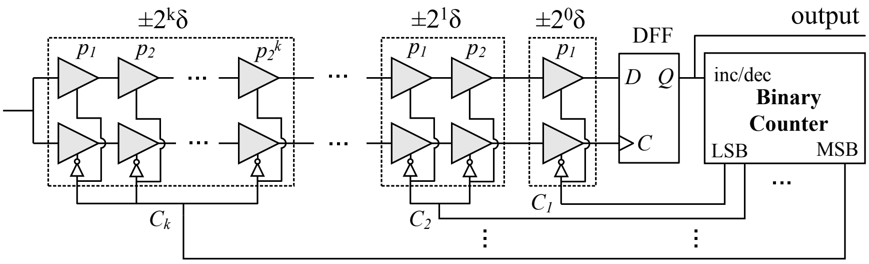


Fig. 6: The TRNG system implementation with a PI controller on FPGA.

The PDLs are depicted as gray triangles which provide the finest and most granular level of control over the delays. If the resulting delay difference from one PDL is equal to δ, the effective input/output delay of a PDL, $D(i)$, for the binary input i would be:

$$D(i) = i \times d_c + (1 - i) \times (d_c + \delta). \tag{3}$$

where d_c is a constant delay value. Each programmable delay block consists of two PDLs. The control input of top PDL inside each block is the complement of the bottom PDL control input in order to make a differential programmable delay structure. Based on Equation 4, the differential delay is:

$$D_{diff}(i) = (1 - 2i) \times \delta = (-1)^i \delta, \qquad i = 0 \text{ or } 1. \tag{4}$$

In this example, the programmable delay blocks are packed in groups with sizes of multiples of two to efficiently generate any desirable delay difference using a binary control input. In other words, the first programmable delay block consists of two PDLs, the second one contains 4 PDLs, and so on. With this arrangement, the total incurred delay difference can be written as:

$$\varDelta_f = G(\mathbf{C}) = \sum_{i=0}^{K}(-1)^{C_i}2^i\delta, \tag{5}$$

where $C_i \in \mathbf{C}$ is the i^{th} counter bit with $i = 0$ being the least significant bit (LSB) and $i = K$ being the most significant bit (MSB), and $\mathbf{C}$ represents the counter value. δ is the smallest possible delay difference produced by one PDL. Let us assume that in the beginning the counter is reset to zero. The resulting feedback delay difference is $\varDelta_f = (2^{(K+1)} - 1) \times \delta$ according to Equation 5. This large delay difference skews the output of flip-flop toward ‘1’. This keeps raising the counter value, lowering the delay difference ($\varDelta$). As $\varDelta$ approaches zero, the flip-flop begins to output ‘0’s more frequently and lowers the rate at which the counter value was previously increasing. At the steady state, the counter value will settle around a constant value with a slight oscillatory behavior. Any outside perturbation on delays will cause transient fluctuations in bit probabilities; however, the automatic adjustment mechanism brings the system back to the equilibrium state.

Counter	I^t	I^b	w
111	1111	0000	$+4$
110	0111	0000	$+3$
101	0011	0000	$+2$
100	0001	0000	$+1$
000	0000	0001	-1
001	0000	0011	-2
010	0000	0111	-3
011	0000	1111	-4

Fig. 7: Decoding operation.

Although the performance of the system in Figure 6 seems ideally flawless, a straightforward hardware implementation was not successful. This is because the design is based on the assumption that δs from PDLs are equal. However, due to manufacturing process variability, the δs slightly vary from one PDL to another. As a result, it is not feasible to generate any desirable delay difference, because the intended weights are not exactly multiples of two anymore. In particular, the input to the largest programmable delay block dominates the system’s output behavior.

Instead, we took an alternative approach and used two sets of fine and coarse delay tuning blocks as shown in Figure 8. With n fine tuning delay lines with a resolution of δ_{fn}, and m coarse tuning delay line with resolution of δ_{cs}, any delay difference in the range of $R = [n\delta_{fn} + m\delta_{cs}, -n\delta_{fn} - m\delta_{cs}]$ that satisfies Equation 6 can be produced.

$$\varDelta_f = w_{fn}\delta_{fn} + w_{cs}\delta_{cs} \tag{6}$$

where w_{fn} and w_{cs} are integer weights (or levels) such that $-n < w_{fn} < n$ and $-m < w_{cs} < m$. By carefully selecting n,m, δ_{fn}, and δ_{cs}, any delay difference with a resolution of δ_{fn} can be produced within the range R.

The system in Figure 8 is designed such that the weights (or tuning levels) in Equation 6 are a function of the difference in the total number of ‘1’s at PDL inputs on the top and bottom paths;

$$w_{fn} = \sum_{i=1}^{n} I^t[i] - \sum_{i=1}^{n} I^b[i], \quad w_{cs} = \sum_{i=1}^{m} I^t[i] - \sum_{i=1}^{m} I^b[i] \tag{7}$$

where $I^t[i] \in \{0, 1\}$ and $I^b[i] \in \{0, 1\}$ are the input signals to PDLs as demonstrated in Figure 8. Thus, decoder block in Figure 8 needs to perform a mapping from the counter value to the number of '1's at PDL inputs. For example, if $n = 4$, the counter value of '111' corresponds to -4 and '000' corresponds to +4. Table 7 shows an example of decoding operation and corresponding tuning weights for a 3-bit counter. The conversion from the counter value to the effective tuning weight is expressed by Equation 8.

$$w_{fn} = (-1)^{C_K} \times \left(1 + \sum_{i=0}^{K-1} C_i 2^i\right), \quad K = \lfloor log_2 n \rfloor. \tag{8}$$

The fundamentals of the system's operation shown in Figure 8 are the same as the system in Figure 6 with the only difference lying in how the feedback signal is generated based on the counter states.

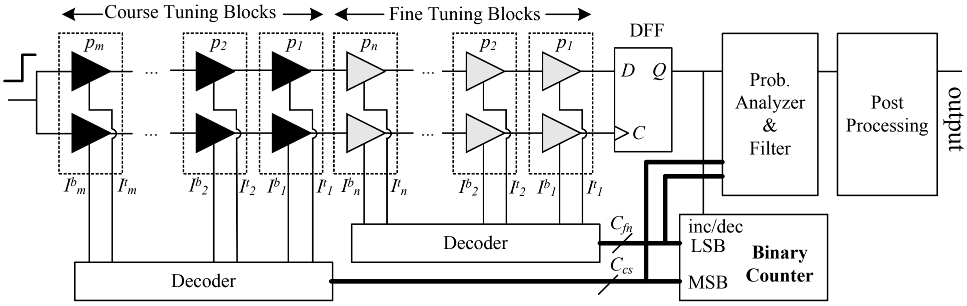


Fig. 8: The complete TRNG system.

Notice that the controller type determines the response time to changes in delays as well as the error in the steady state response. Proportional integral (PI) controllers as opposed to proportional integral derivative (PID) controller due to the lack of derivative function can make the system more stable in the steady state in the case of noisy data. This is because derivative action is more sensitive to higher-frequency terms in the inputs. Additionally, a PI-controlled system is less responsive to inputs (including noise) and so the system will be slower to respond to quick perturbations on the delays than a well-tuned PID system.

The following two observations are important from a security standpoint. First, in the steady state, the counter value oscillates around a constant center value (C_{center}). Let us define the oscillation amplitude as the peak-to-peak range of the oscillations, i.e. the maximum counter value minus the minimum counter value ($C_{max} - C_{min}$). The oscillation is not as periodic as one might think. It is rather a random walk around the center value. Each step in the random walk involves going from one counter value to a one lower or higher value:

$$\text{Step} : C_{current} \rightarrow C_{current} \pm 1$$

The probability of each step (move) is a function of the current location. Intuitively the probability of going outside the range is almost zero:

$$\begin{aligned} \text{Prob}\{C_{max} \rightarrow C_{max} + 1\} &\simeq 0 \\ \text{Prob}\{C_{min} \rightarrow C_{min} - 1\} &\simeq 0 \end{aligned} \tag{9}$$

Also assuming a smooth monotonically increasing probability curve as shown in Figure 4 for the flip-flop, the farther the current counter value is from the center (C_{center}), the lower the probability of moving farther away from the center:

$$\begin{aligned} Prob\{C_i \rightarrow C_i + 1\} < Prob\{C_j \rightarrow C_j + 1\} \text{ for } C_j < C_i \\ Prob\{C_i \rightarrow C_i - 1\} < Prob\{C_j \rightarrow C_j - 1\} \text{ for } C_j < C_i \end{aligned} \tag{10}$$

Each generated output bit corresponds to a counter value. The probability of the output being to '1' is a function of the feedback counter value. The maximum counter value almost always results in a '0' output, since a '0' value decrements the counter value. Based on Equation 9, transition $C_{max} \rightarrow C_{max} + 1$ is unlikely, thus $r(C_{max})$ can almost never be '1'. The following deductions can be explained similarly:

$$\begin{aligned} Prob\{r(C_{center}) = 1\} &\simeq 0.5 \\ Prob\{r(C_{min}) = 1\} &\simeq 1 \\ Prob\{r(C_{max}) = 1\} &\simeq 0 \end{aligned} \tag{11}$$

In other words, during the random walk only those steps that pass close at the center point will result in high entropy and non-deterministic responses. A smaller error in the steady state response means oscillations happen closer to center of the probability transition curve which in turn leads to higher randomness in generated output bits.

In addition, it is desired that the system responds as quickly as possible to external perturbations since the during the recovery time the TRNG generates output bits with highly skewed probabilities.

6 Experimental results

In this section, we present the LUT-based PDL delay measurement evaluations and TRNG hardware implementation results obtained from Xilinx Virtex 5 LX50T FPGA.

Before moving onto the TRNG system performance evaluation, we shall first discuss the results of our investigation on the maximum achievable resolution of the PDLs. We set up a highly accurate delay measurement system similar to the delay characterization systems presented in [9, 7, 6].

The circuit under test consists of four PDLs each implemented by a single 6-input LUT. The delay measurement circuit as shown in Figure 9 consists of three flip-flops:

launch, sample, and capture flip-flops. At each rising edge of the clock, the launch flip-flop successively sends a low-to-high and high-to-low signal through the PDLs. At the falling edge of the clock, the output from the last PDL is sampled by the sample flip-flop. At the last PDL's output, the sampled signal is compared with the steady state signal. If the signal has already arrived at the sample flip-flop when the sampling takes place, then these two values will be the same; Otherwise they take on different values. In case of inconsistency in sampled and actual values, XOR output becomes high, which indicates a timing error. The capture flip-flop holds the XOR output for one clock cycle.

To measure the absolute delays, the clock frequency is swept from a low frequency to a high target frequency and the rate at which timing errors occur are monitored and recorded. Timing errors start to emerge when the clock half period (T/2) approaches the delay of the circuit under test. Around this point, the timing error rate begins to increase from 0% and reaches 100%. The center of this transition curve marks the point where the clock half period (T/2) is equal to the effective delay of the circuit under test.

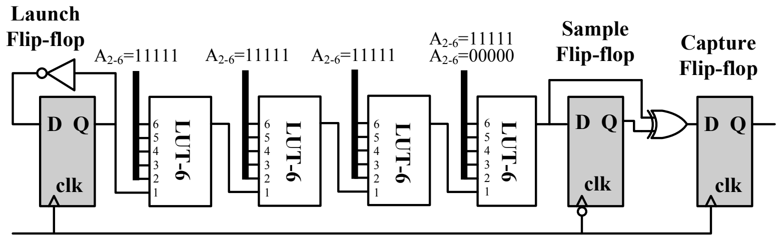
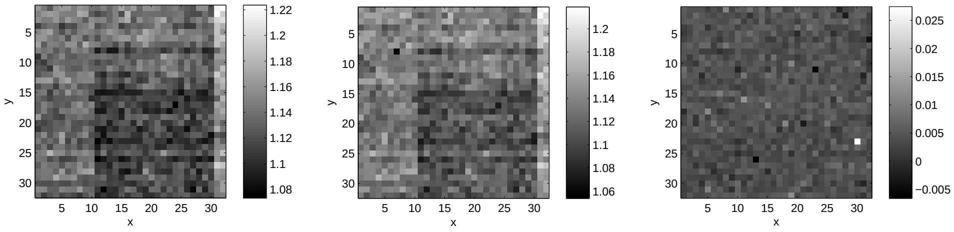


Fig. 9: The delay measurement circuit. The circuit under test consists of four LUTs each implementing a PDL.

To measure the delay difference incurred by the LUT-based PDL, the measurement is performed twice using different inputs. In the first round of measurement, the inputs to the four PDLs are fixed to $A_{2-6} = 11111$. In the second measurement the inputs to the last PDL are changed to $A_{2-6} = 00000$. In our setup, a 32×32 array of the circuit shown on Figure 9 is implemented on a Xilinx Virtex 5 LX110 FPGA, and the delay from our setup is measured under the two input settings. The clock frequency is swept linearly from 8MHz to 20MHz using a desktop function generator and this frequency is shifted up by 34 times inside the FPGA using the built-in PLL.

The results of the measurement are shown on Figure 10. Each pixel in the image corresponds to one measured delay value across the array. The scale next to the color-map is in nano-seconds. Figure 10 (c) depicts the difference between the measured delays in (a) and (b). As can be seen, the delay values in (b) are on average about 10 pico-seconds larger than the corresponding pixel values in (a). This is in fact equal to the amount of delay difference caused by the coarse PDLs, i.e., δ_{cs}. The delay difference induced by the fine PDL of Figure 11 (a), δ_{fn} is approximately equal to 1/16 of δ_{cs}.

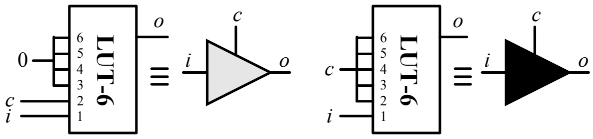
To evaluate the performance of the TRNG system, we implement the system shown in Figure 8 using 32 coarse and fine programmable delay lines ($n = m = 32$). A 12-bit counter performs the running sum operation on the output generated bits. The first



(a) Delay for $A_{2-6} = 00000$ (b) Delay for $A_{2-6} = 11111$ (c) Delay difference

Fig. 10: The measured delay of 32×32 circuit under tests containing a PDL with PDL control inputs being set to (a) $A_{2-6} = 00000$ and (b) $A_{2-6} = 11111$ respectively. The difference between the delays in these two cases is shown in (c).

six (LSB) bits control the finely tunable PDLs, and the next six (MSB) bits control the coarsely tunable PDLs. Both fine and coarse PDLs are implemented by using one LUT as shown in Figure 11. As illustrated in Figure 11, to implement the fine PDL, the LUT inputs A_3 to A_6 are fixed to zero and the only input that controls the delay is A_2. For the coarse PDL, all of the LUT inputs are tied and controlled together.



(a) Fine PDL (b) Coarse PDL

Fig. 11: Coarse and fine PDLs implemented by a single 6-input LUT.

In the first experiment, we only examine the forward system, which consists of the PDLs, the flip-flop, and the decoders. The tuning weights/levels are swept from the minimum to maximum, and the probability of the flip-flop producing a ‘1’ output is measured at each level. This probability is measured by repeating each experiment over 100 times and counting the number of times the flip-flop outputs a ‘1’. Since $n = m = 32$, both the fine and coarse tuning levels can go from -32 to 32. Recall that the tuning level represents the difference in the total number of ones at PDL inputs on the top path minus those on the bottom path (see Equation 7). As can be observed from Figure 12, increasing both the coarse and fine tuning levels increase the probability of output being equal to ‘1’. The non-smoothness of the probability curve is due to variability in the manufacturing process which creates local non-monotonicity. With these observations, we expect the feedback system behavior to stabilize somewhere close to the center of the transition point. Next, we close the feedback loop and initialize the operation. At the beginning, the counter is loaded with all ‘1’s (which results in a

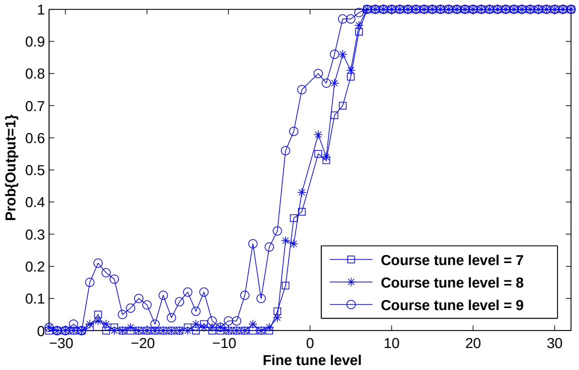


Fig. 12: The probability of flip-flip generating a '1' output as a function of the fine and coarse tuning levels.

decimal value of 2^{12}-1 = 4095). Figure 13 shows the counter value as the operation progresses. The x-axis is the number of clock cycles. Once the operation starts, the counter value keeps decreasing until it reaches the value of approximately 700 after about 3,400 clock cycles. From this point further, the counter value reaches a steady state with a slight oscillatory behavior around a constant value. A close-up of the steady state behavior is depicted in the lower plot of Figure 13. The close-up zooms into the segment between 25,000 to 30,000 clock cycles. As can be observed in the steady state, the counter value oscillates between 559 and 564.

Next, we investigate the frequencies at which counter values appear in the steady state. In this experiment, we collect 1,000,000 counter values in the steady state and plot the histogram of the observed values as shown in the middle plot (b) in Figure 14. The normalized histogram suggests that the counter holds the value of 561 more than 40% of the time. Next, it is critical to investigate the probabilities associated with each courter value. In other words, we would like to know for the given counter values − which produce a feedback input to the TRNG core − the probability of the flip-flop output being equal to '1'. The top plot (a) in Figure 14 presents this result. It is interesting to see that most of the counter values produce highly skewed probabilities. Among these counter values, 561 leads to a '1' output slightly more than 40% of the time. We define a metric which is the multiplication of the counter values' frequency of occurrence with the probability of output being equal to one for each counter value. This metric represents the contribution of each counter value to the total number of '1' in the output sequence. The metric values are shown in bottom plot (c) in Figure 14.

To remove the bias in the output sequence in a systematic way as well as to eliminate predictable patterns, we propose a filtering mechanism based on the steady state counter values. The filter unit analyzes the output bit probabilities for each counter value within a window of specific size and flags the counter values that lead to outputs bits with skewed probabilities. Next, it filters out the output bits associated with the flagged counter values. For example, in our implementation, the filter only allows output bits

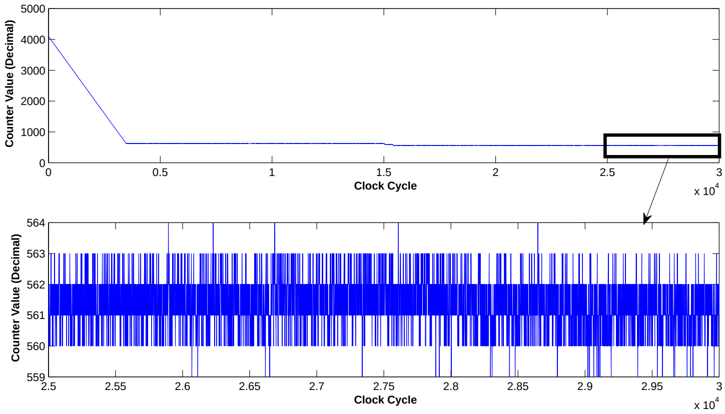


Fig. 13: The transient counter value (decimal) versus the clock cycles.

associated with the counter value of 561 to pass through. As a result, the bit-rate is lowered to almost half of the original bit-rate. However, the output bits may still suffer from bias in the bit probabilities. Therefore, a post-processing unit after the filter unit is used to remove any localized biases from the bitstream. In our implementation, we use a von Neumann corrector to perform the post-processing task. The results of the NIST randomness test from running on megabytes of data is shown in Table 1. The comprehensive test results are available online at http://www.ruf.rice.edu/ mm7/trng/.

Table 1 includes the results of the NIST statistical test suite on megabytes of collected data after counter-based filtering and von Neumann correction are performed on the TRNG output bitstream. Due to the large bias in the probabilities, most of the randomness failed when the test was run on the output bitstream before the filtering and correction were carried out.

Finally, according to the ISE Synthesis report, the propagation delay through the TRNG core is equal to 61.06ns which achieves a bit-rate of 16Mbit/sec. The bit-rate drops to 1/8 of the original bit-rate (to 2Mbit/sec) after filtering and von Neumann correction. The TRNG core consumes 128 LUTs that are packed into 16 Virtex 5 CLBs. Note that in practice multiple TRNG cores can run in parallel to offer a higher bit-rate.

7 Conclusion

A novel FPGA-based technique to generate true random numbers through flip-flop metastability was introduced. The presented method took advantage of highly precise programmable delay lines (PDL) to accurately equalize the signal arriving times to flip-flops, thus enforcing a metastable behavior. PDLs as demonstrated in the paper are capable of adjusting signal propagation delays with sub pico-second resolution. With the help of a closed-loop proportional integral (PI) control system, the output bit probabilities are constantly monitored and as soon as any skews in probabilities are observed,

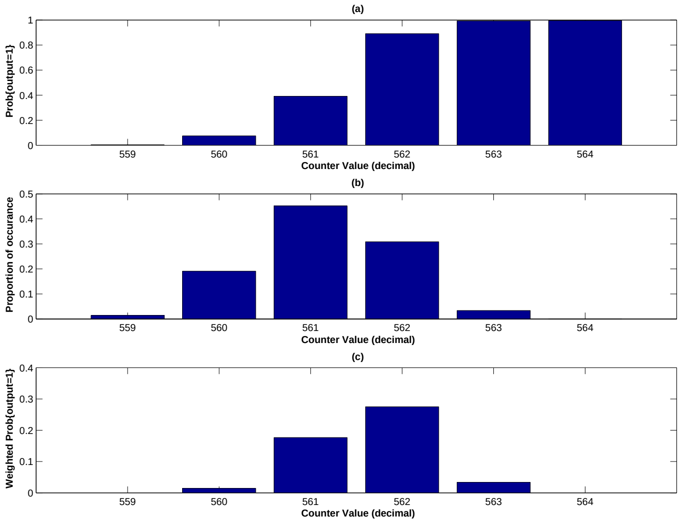


Fig. 14: Distribution of the steady state counter values and associated bit probabilities.

feedback signal instantly adjusts the delay taps to revert to the metastable condition. The feedback systems provides resilience against fluctuations in environmental conditions, as well as robustness against active adversarial attacks. Implementation on Xilinx Virtex 5 FPGAs and results of NIST randomness tests show the effectiveness of our true random number generator. The proposed TRNG is capable of producing a throughput of 2 Mbit/sec after post-processing and filtering with a low overhead, using only 5 CLBs.

References

1. Barak, B., Shaltiel, R., Tromer, E.: True random number generators secure in a changing environment. In: Cryptographic Hardware and Embedded Systems workshop (CHES). pp. 166–180. Springer-Verlag (2003)
2. Bergeron, E., Feeley, M., Daigneault, M.A., David, J.: Using dynamic reconfiguration to implement high-resolution programmable delays on an FPGA. In: NEWCAS-TAISA. pp. 265 –268 (2008)
3. Gassend, B., Clarke, D., van Dijk, M., Devadas, S.: Silicon physical random functions. In: CCS. pp. 148–160 (2002)
4. Jun, B., Kocher, P.: The Intel random number generator. In: CRYPTOGRAPHY RESEARCH, INC. (1999)
5. Lee, J., Lim, D., Gassend, B., Suh, G., van Dijk, M., Devadas, S.: A technique to build a secret key in integrated circuits for identification and authentication applications. In: Symposium on VLSI Circuits. pp. 176 – 179 (2004)
6. Majzoobi, M., Dyer, E., Elnably, A., Koushanfar, F.: Rapid FPGA characterization using clock synthesis and signal sparsity. In: International Test Conference (ITC) (2010)

Table 1: NIST Statistical Test Suite results.

Statistical Test	Block/Template length	Lowest success ratio
Frequency	-	100%
Frequency within blocks	128	100%
Cumulative sums	-	100%
Runs	-	100%
Longest run within blocks	-	100%
Binary rank	-	100%
FFT	-	100%
Non-overlapping templates	9	90%
Overlapping templates	9	100%
Maurer's universal test	7	100%
Approximate entropy	10	100%
Random excursions	-	100%
Serial	16	100%
Linear complexity	500	90%

7. Majzoobi, M., Koushanfar, F.: FPGA time-bounded authentication. IEEE Transactions on Information Forensics and Security PP(99), 1 (2011)
8. Majzoobi, M., Koushanfar, F., Devadas, S.: FPGA PUF using programmable delay lines. In: IEEE Workshop on Information Forensics and Security (WIFS). pp. 1–6 (2010)
9. Majzoobi, M., Elnably, A., Koushanfar, F.: FPGA time-bounded unclonable authentication. In: Information Hiding (IH). pp. 1–16 (2010)
10. von Neumann, J.: Various techniques used in connection with random digits. von Neumann Collected Works 5, 768–770 (1963)
11. Odonnell, C.W., Suh, G.E., Devadas, S.: PUF-based random number generation. In: In MIT CSAIL CSG Technical Memo 481. p. 2004 (2004)
12. Schellekens, D., Preneel, B., Verbauwhede, I.: FPGA vendor agnostic true random number generator. In: Field Programmable Logic and Applications (FPL). pp. 1 –6 (2006)
13. Suh, G., Devadas, S.: Physical unclonable functions for device authentication and secret key generation. In: Design Automation Conference (DAC). p. 914 (2007)
14. Sunar, B.: Cryptographic engineering, chap. True Random Number Generators for Cryptography. Springer (2009)
15. Sunar, B., Martin, W.J., Stinson, D.R.: A provably secure true random number generator with built-in tolerance to active attacks. IEEE Transactions on Computers 58, 109–119 (2007)
16. Wong, J.S.J., Sedcole, P., Cheung, P.Y.K.: Self-measurement of combinatorial circuit delays in FPGAs. ACM Transactions on Reconfigurable Technology and Systems 2(2), 1–22 (2009)