An Energy-Efficient PUF Design: Computing While Racing

Hongxiang Gu, Teng Xu, and Miodrag Potkonjak
Computer Science Department
University of California, Los Angeles
{hxgu, xuteng, miodrag}@cs.ucla.edu

ABSTRACT

Physical unclonable functions (PUFs) take advantage of the effect of process variation on hardware to obtain their unclonability. Traditional PUF design only focuses on the analog signals of circuits. An arbiter PUF, for example, generates responses by racing delay signals. Implementations of such PUFs usually employ large area and power consumption while providing very low throughput.

To address this problem, we propose an energy efficient PUF design in such a way that it races analog signals and computes digital logic simultaneously. More importantly, the analog portion of the circuit (racing) shares a large amount of hardware resources with the digital portion of the circuit (computing) by introducing only small overhead in terms of area and power. Our test results on Spartan-6 field-programmable gate array (FPGA) platforms indicate that by combining the two outputs, our design enables much larger PUF output throughput, better randomness and less power consumption compared to traditional PUFs.

CCS Concepts

•**Hardware** → *Power and energy;*

Keywords

PUF; random number generator; FPGA;

1. INTRODUCTION

There are two major concerns for modern logic designs: power and security. The prevailing portable devices such as mobile phones, tablets, and laptops impose high requirements on the low power design and applications due to the highly constrained power supply. Protecting mobile devices is more challenging than securing non-portable devices not only because of the emergence of novel attack methods such as malicious mobile software, mobile phone trojans, and even electromagnetic side-channel attacks, but also due to the fact that traditional security approaches usually demand high power cost, and thus are not applicable on mobile devices due to limited power supply. Therefore, the desire for lightweight security primitives is stronger than ever.

A PUF, as a unique type of hardware security primitive, has excellent low power, high speed, and unclonablity properties. An individual PUF is a piece of hardware implementing a one-way function which takes advantage of the inevitable process variation to guarantee uniqueness of the function. The input-output mapping function of an individual PUF is deterministic but unpredictable, due to the fact that process variation is not predictable. As a hardware security primitive, PUF employs lower overhead comparing to traditional software-based cryptographic approaches. More importantly, a PUF itself is unclonable which provides protection at the physical level.

However, PUFs suffer from significant drawbacks as well. Two of the most important concerns are related to output throughput and randomness. As an example, an arbiter PUF has an n-bit challenge vector and only a single output bit. When used to encrypt and decrypt messages in real applications, n is usually set to be at least 64, in which case the ratio between the PUF output and the PUF input is $1 : 64$ or lower. The other concern is associated with the randomness of PUF outputs. The unpredictable and uncontrollable nature of process variation creates unbalanced delay path routing in PUFs. The frequency of $0s$ and $1s$ in the outputs are usually not equal which compromises the security of PUFs. Other randomness problems such as repeated patterns in outputs and strong PUF input-output correlations can also be observed in some implementations.

We propose a novel PUF design which avoids the above PUF problems while keeping it low-delay, low-power and physically unclonable. We focus specifically on arbiter PUFs. Our design is motivated by the following observation. Traditional arbiter PUFs focus on the analog properties of circuits, for example, delays in the case of arbiter PUFs. Two signals are sent as inputs to the two paths of the PUF, and depending on which path has a longer delay, one signal will arrive at the arbiter first, thus producing a 1 or 0 accordingly. However, the PUF circuit itself is capable of serving beyond racing delay signals; it can also be used to convey meaningful digital information. In arbiter PUFs, each delay component is made of transistors/gates. Therefore, when connecting these gates in a particular manner, the overall design can compute specific functions.

Our key idea is to use the same piece of hardware to compute digital logic while racing analog signals at the same time. With only small extra area overhead, our circuit de-

ISLPED '16, August 08-10, 2016, San Francisco Airport, CA, USA
© 2016 ACM. ISBN 978-1-4503-4185-1/16/08...$15.00
DOI: http://dx.doi.org/10.1145/2934583.2934604

sign will generate two types of outputs, respectively analog outputs from the PUF, and digital outputs from the digital logic. Moreover, both outputs are generated in the same clock cycle. By combining the above two outputs, we expect to keep the unclonability of PUFs while gaining the advantage of digital circuits.

The digital portion of the circuit can be designed to implement any functionality. In our design, we have specifically chosen leap-forward linear feedback shift register (leap-forward LFSR) for the following reasons. First, a leap-forward LFSR is a pseudo-random number generator; by combining PUF outputs with leap forward LFSR outputs (e.g., XOR the two outputs), the combined result will be highly random. Second, a leap-forward LFSR generates large outputs in a compact area, thus, without introducing extra hardware or delay, the output throughput is highly boosted.

To summarize our contributions, we propose a power efficient PUF design by combing traditional arbiter PUFs with leap forward LFSRs. Our design only introduces a small area overhead as the arbiter PUF and the leap forward LFSR share a majority of hardware and signal resources. With a single execution of the circuit, both outputs are generated simultaneously. By combining the arbiter PUF outputs with the leap forward LFSR outputs, the system gains higher throughput as well as better randomness. Our design and implementation is based on, but not limited to, FPGA platforms. We have described our detailed implementation in Section 6.

2. RELATED WORK

2.1 Physical Unclonable Functions

PUF was first proposed by Pappu et al. using mesoscopic optical systems [1]. Gassend et al. developed the first silicon PUFs through the use of intrinsic process variation in deep submicron integrated circuits [2]. A variety of other types of PUFs have since been proposed, including arbiter PUFs [2], ring oscillator PUFs [3], SRAM PUFs [4], and butterfly PUFs [5]. Majzoobi et al. proposed to use FPGA based programmable delay lines to build delay PUFs [6]. Numerous traditional protocols can be interpreted using PUFs, ranging from the traditional security key communication and authentication [7] to more sophisticated public key communication [8] with the key idea of employing the high unpredictability of PUF responses to secure the information. More recently, Xu et al. created PUF-based recursive inverse function [9] and digital bidirectional function [10] to protect sensitive data with ultra low area, latency and energy overhead.

2.2 Hardware Random Number Generators

Random number generators are widely used in many security applications. James has reviewed a majority of commonly used random number generators [11]. In the past, random number generation was mostly done by software. However, as hardware systems become cheaper, faster, and more lightweight, it is feasible and more power efficient to implement the random number generators directly in hardware. A number of HRNGs are proposed based on different technologies. Using PUFs to build an HRNG was first proposed by O'Donnell from MIT [12]. Our work implements the design of leap-forward LFSRs which is an extension of the standard LFSR by allowing all shifts in the standard design to be applied in a single clock cycle [13].

3. DESIDERATA

Our work is the first effort to take advantage of both analog properties as well as digital properties of a circuit to create a security primitive. The analog properties are used to build an arbiter PUF while the digital properties are realized by creating a leap-forward LFSR. Before discussing our detailed design and implementation, we identify the architectural, operational, and security desiderata of our work.

- In terms of architecture, we have created a "one circuit, two outputs" design. The PUF portion and the LFSR portion of our designed circuit share almost the same hardware resources, consequently, it saves area and lowers the power consumption.

- In terms of operation, our design achieves high throughput by employing only small delay since the digital outputs and the analog outputs are generated in the same clock cycle without timing overhead.

- In terms of security, by combining the arbiter PUF output with the LFSR output, the final system output has kept the unclonability of the original PUF while the randomness is enhanced from LFSR.

4. PRELIMINARIES

4.1 PUF Model

The PUFs we use for signal racing are standard arbiter PUFs. Figure 1 shows the schematic diagram of the PUF model. The basic structure of an n-bit PUF consists of n delay segments. The two propagation delays in the ith segment are denoted as d_i^0 and d_i^1 respectively. The two delays are designed to be nominally equal to each other, but after manufacturing, the effect of process variation will cause unpredictable delay difference between them. When built on an FPGA, the upper delay and the lower delay in each segment are directly implemented using LUTs with the same size. Two identically designed paths are generated by connecting delay components from each segment, and an arbiter is placed at the end of the two paths. The two paths can be modified using the control bit of each segment. When the control bit is 0, the two paths will not shuffle. When the control bit is 1, the two paths swap. For example, in Figure 1, if the control bit of the ith segment is 0, then d_i^0 stays with the upper path and d_i^1 stays with the lower path. However, if the control bit is 1, d_i^0 shuffles to the lower path and d_i^1 shuffles to the upper path. Note that when the shuffling happens, all the delays that connect prior to $d_i^0(d_i^1)$ will be shuffled at the same time.

The vector consisting of all control bits is denoted as the PUF challenge. When an n-bit challenge $(c_1c_2...c_{n-1}c_n)$ is provided to the PUF, two identically designed paths are generated. To retrieve a response, an impulse signal is fed into the system to excite both paths simultaneously. Because of process variation, the signal traveling along one of the two paths will reach the arbiter earlier, generating a corresponding arbiter output denoted as the PUF response.

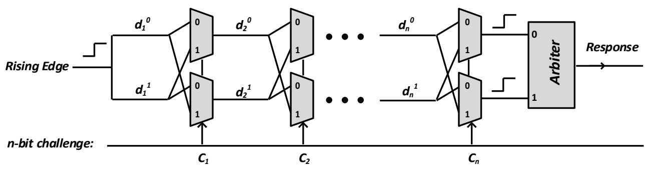

Figure 1: The model of arbiter PUFs with an n-bit challenge.

4.2 Leap-Forward LFSR

A LFSR is a commonly seen pseudo-random number generator (PRNG). The leap-forward LFSR method utilizes only one LFSR and shifts out several bits. This method is based on the observation that a LFSR is a linear system and the register state is expressed as $Q(i+1) = A * Q(i)$. $Q(i+1)$ and $Q(i)$ are the initial values at $(i+1)th$ and ith steps; A is the transition matrix.

To calculate the content in shift registers after k steps, the equation transforms into: $Q(i+1) = A^k * Q(i)$. We can compute A^k and determine the XOR structure accordingly. The new circuit leaps k steps in one clock cycle while the circuit uses identical shift registers.

To illustrate the idea, Chu et al. proposed a motivational example of a 4-bit leap-forward LFSR [13]. The derived new transitional matrix A^4 is calculated from A, which is obtained from a single-bit LFSR random number generator.

$$\mathrm{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}, A^4 = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

After performing the operations, we can derive the feedback equation for each signal as:

$$q0\_next = q0 \oplus q3 \tag{1}$$

$$q1\_next = q0 \oplus q1 \oplus q3 \tag{2}$$

$$q2\_next = q0 \oplus q1 \oplus q2 \oplus q3 \tag{3}$$

$$q3\_next = q0 \oplus q1 \oplus q2 \tag{4}$$

Note that the final transition matrix depends only on the initial transition matrix, thus, different initialization values in shift registers will lead to a different number of XOR logics required in a multi-bit leap-forward LFSR. A 64-bit leap-forward LFSR takes at most 125 XOR gates to build.

5. ARCHITECTURE

5.1 Observations

We observed some opportunities that can be taken to further improve a conventional FPGA-based arbiter PUF implementation.

5.1.1 Signal Encoding

In a conventional arbiter PUF implementation, the pair of racing clock signals clk_1^i and clk_2^i after the ith segment is always synchronous with the original input signals clk_1^{orig} and clk_2^{orig}. One observation is that the synchronization is not strictly enforced, as long as clk_1^i and clk_2^i are in phase, the arbiter appended at the end of the PUF is able to accurately catch the faster signal. This observation enables us to encode racing signals to digital signals. For example, if clk_1^i and clk_1^{orig} are in phase, we encode clk_1^i as a 0 and if they are in antiphase we encode it as a 1.

5.1.2 LUT Utilization

FPGA implementation of arbiter PUFs utilizes LUT6_2 to race signals. An LUT6_2 is a 6-input, 2-output look-up table that is able to act as two LUT5s that shares the same inputs or just a single LUT6 depending on the control bit. Among the six inputs bits, the control bit (most significant bit) is set to 1 in order to transform the LUT6_2 into two LUT5s. The second most significant bit is usually used as a selection bit that decides the traveling path inside the LUT. This bit serves as a challenge bit of the arbiter PUF. The next two bi ts are fixed to constant 1s. The two least significant bits are used to take two racing clock signals. We observe that a conventional arbiter PUF implementation does not fully utilize all the possible resources of LUT6_2. If we allow antiphased signals between PUF segments, we could change the 3rd and 4th bits from static 1s to user defined values. The conventional design of enforcing two static bits is equivalent to leaving a LUT2 unused for every LUT6_2 in the delay chain.

5.2 Overall Design

We propose producing logical output and signal racing results simultaneously on the same hardware. Our design takes a pair of impulse signals as input, and because of process variation, two racing signals do not arrive at the finish line (arbiter) at the same time. Meanwhile logical computations are performed based on the logical input and the result is carried on the bypassing racing signals using encoding. Thus, we generate two outputs, each of which is completely uncorrelated. Consequently, an advantage of our design is that we are able to reduce overall power and area by sharing hardware and wires.

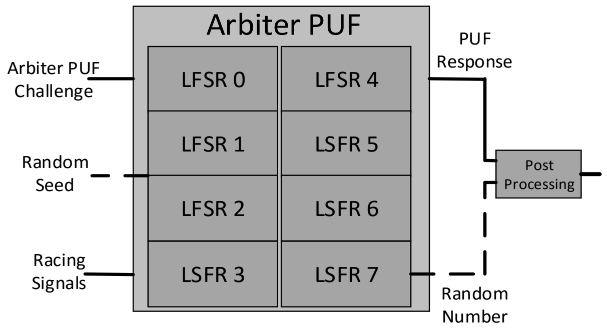

Figure 2: High level illustration of proposed design.

We propose to load leap-forward LFSRs on top of a conventional arbiter PUF as shown in Figure 2. The arbiter PUF and multiple LFSRs share the same hardware but generate different outputs. The arbiter PUF takes PUF challenges to configure internal paths for the delay signals to race. Meanwhile, leap-forward LFSRs are designed on top of the arbiter PUF to generate digital random numbers.

Eventually, the leap-forward LFSR output and the arbiter PUF response are combined using a post process module (in our case we use XORs and a Von Neumann corrector). The final output leverages advantages from both sources: it inherits high throughput and high randomness from leap-forward LFSRs and physical unclonablity from arbiter PUFs.

6. IMPLEMENTATION

We utilize LUT6_2 on FPGAs to demonstrate the feasibility of our design. Figure 4 shows our overall implementation. Our design maintains the LUT chain structure and some LUT input pin assignments from the conventional arbiter PUF implementation. However, the third and fourth input bit of each LUT6_2 are no longer restricted to constant 1s. Instead, they are now open to any user-defined values. We use these two bits along with unconstrained initialization vector (INIT) bits to implement an additional XOR gate on top of the LUT6_2. We then use these XOR gates to implement leap-forward LFSRs. The output of the XOR gate is encoded from the racing signal using a flip-flop. Depending on if the signal is in phase or antiphase with the clock signal, the result of XOR gate is encoded as 0 or 1.

6.1 Implementation of Arbiter PUF

The arbiter PUF retains the same structure as a conventional FPGA-based design. Each LUT6_2 implements a single segment of an arbiter PUF. We have made modifications to the implementation to allow racing signals to carry additional digital information. Note that our modification still maintains signal synchronization by keeping two racing signals in phase throughout the racing process.

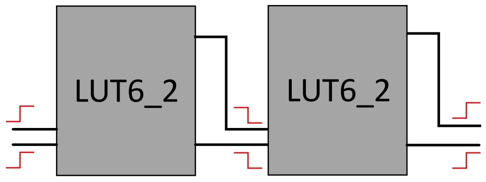

Figure 3: The two output signals from LUT6_2 are required to be in phase, but can be inverted simultaneously.

6.1.1 Allowing Antiphase Signals

In our design, we no longer require signals traveling between LUTs to be synchronous with the clock signal. Figure 3 shows a possible scenario where the signals are inverted after traveling through the first LUT and inverted again after the second LUT. Our encoding scheme (in-phase encoded as 0 and antiphase encoded as 1) allows us to carry computational output on racing signals that are traveling through the arbiter PUF chain.

Additional constraints on the INIT value of LUTs need to be applied to allow such functionality. All INIT values must fulfill both requirements below:

- The most significant 32 bits must be mirrored to the least significant 32 bits in units of 4 bits. For example, the mirror image of 1100_1010 would be 1010_1100.
- $INIT[4k] \oplus INIT[4k+3] = 1$ for k ($k \in \{0, 1, ..., 15\}$). The value of $INIT[4k]$ and $INIT[4k+3]$ are decided by the computational logic.

The first constraint is applied to guarantee the upper and lower LUT5s within the LUT6_2 to be the same, thus creating theoretically identical racing paths. The second constraint is enforced to create racing paths that signals can propagate through. These constraints together guarantee that the two outputs of each LUT6_2 are two in-phase clock signals instead of static outputs or out of phase signal pairs. All bits that are not restricted by the rules can be customized to program arbitrary logic.

6.1.2 Arbiter

The arbiter used to capture the faster-racing signal is as simple as a $\overline{SR}$ latch. However, since we allow the signals to be inverted, the phase of the final output of each chain now relies on the implemented logic instead of staying unchanged. Whether the racing signals at the "doorstep" of the arbiter are in phase with the clock signal or not, our original $\overline{SR}$ latch-based arbiter is still able to produce the correct result. Thus, the original PUF functionality is not compromised.

6.2 Implementation of leap-forward LFSRs

Leap-forward LFSRs are implemented using XOR gates and flip-flops. A 64-bit leap-forward LFSR requires only 64 flip-flops and 125 XOR gates. However, the wiring complexity of a 64-bit leap-forward LFSR is nontrivial. As the size of leap-forward LFSRs grows, the wiring complexity grows dramatically. To avoid the wiring overhead, we implement multiple 64-bit leap-forward LFSRs on top of our arbiter PUF design.

6.2.1 XOR Gates

According to our modification on the arbiter PUF design, it is possible to implement an additional XOR gate on the LUT6_2. Table 1 shows the rules needed to implement the XOR logic in addition to the constraints described in section 6.1.1. For example, a valid assignment of INIT[31:0] is $8518C3EA$, the mirrored INIT[63:31] would be $AE3C9158$.

The XOR results are then encoded according to rules described in section 5.1.1. The two outputs of LUT6_2 are guaranteed to be identical based on the constraints we set, so the XOR result can be retrieved by encoding either of them.

Position in INIT[0:31]	Init value
0,7,11,12,16,23,27,28	0
3,4,8,15,19,20,24,31	1
Other	-

Table 1: INIT value rules for implementing XOR gates.

6.2.2 Flip-Flops

The flip-flops serve two purposes. First they are used to extract digital information from the racing signal. Second, they are used to store the results of leap-forward LFSRs. In our implementation, we reuse some flip-flops for both purposes to save area and power. Each leap-forward LFSR requires at most 125 flip-flops.

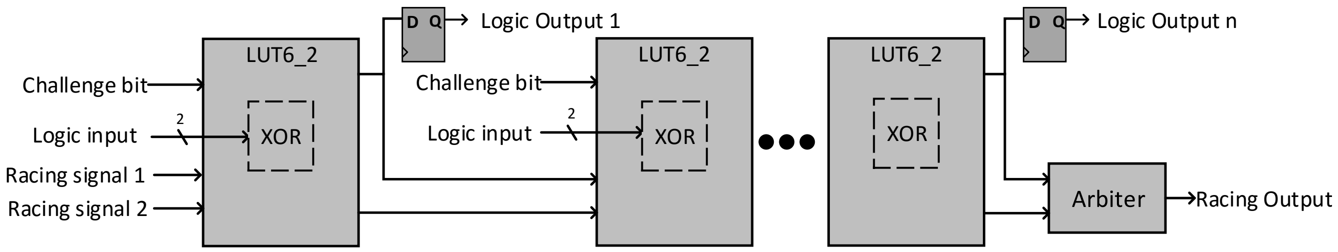

Figure 4: Overall implementation using LUT6_2 on FPGA.

6.3 Post Process

64 LUT6_2s and an $\overline{SR}$ latch are needed to implement a 64-bit arbiter PUF while a 64-bit leap-forward LSFR implementation uses 125 XOR gates and 125 flip-flops. This means that we can load four 64-bit leap forward LSFRs on top of eight 64-bit arbiter PUFs. The throughput ratio of arbiter PUF and LFSR is then 1:32 ($8 : 4 \times 64$). We claim that simple XOR operations to combine both outputs provide sufficient randomness. The XOR operation is done between the concatenation of all leap forward LFSR outputs ($4 \times 64bits$) and a string consists of self-concatenation of arbiter PUF outputs (32 times, which makes it $8 \times 32bits$). By combining the two outputs, we are able to boost the system throughput by 32× compared to conventional eight 64-bit arbiter PUFs. Note that the randomness can be further improved by applying Von Neumann correction on the arbiter PUF results before the XOR operations. The result randomness is evaluated in section 7.2.

7. EXPERIMENTAL RESULTS

In our implementation, we combine leap-forward LFSRs with an arbiter PUF by sharing hardware and wires. Our motivation is to create a security primitive that inherits the advantages of both while staying free of their drawbacks. We carefully evaluate area, power and output randomness of our proposed implementation in this section.

7.1 Area and power

We claim that by sharing hardware and signals, we are able to utilize area and power more efficiently. To evaluate our design, we implement four 64-bit leap forward LFSRs and eight 64-bit arbiter PUFs using the same hardware resources on a Spartan-6 XC6SLX45 FPGA. In each clock cycle, our design generates 256-bit post processed output. We compare our design with standalone leap-forward LFSRs and arbiter PUFs that generate the same output throughput (256 bits). We have also compared our hardware sharing design with non-sharing designs on four 64-bit leap forward LFSRs and eight 64-bit arbiter PUFs. The comparison result is shown in Table 2.

Even though we spend more power and area than leap-forward LFSRs, our design gains the advantage of physical unclonability. When compared to arbiter PUFs, we have reduced LUT cost by 30.1× and reduced power by 16.9×.

When compared to four independent 64-bit leap-forward LFSRs and eight 64-bit arbiter PUFs that do not share hardware as shown in the last column of Table 2, our design also does better in terms of both area and power.

	Our design	LFSR	PUF	Non-share
Throughput	256+8	256	256	256+8
Flip-flops	532	256	512	320
LUTs	544	250	16,384	764
Slices	288	135	8192	402
Unclonable	yes	no	yes	-
Power(mW)	6.92	3.39	117	7.38
Power/bit	0.026	0.013	0.457	0.028

Table 2: FPGA resource and power characteristics: our design (four 64-bit leap forward LFSRs loaded on eight 64-bit arbiter PUFs) vs. four standalone 64-bit leap forward LFSRs vs. 256 64-bit arbiter PUFs vs. four 64-bit leap forward LFSR and eight 64-bit arbiter PUFs that do not share hardware resources. Power per bit unit: mW/bit.

The result shows our design use more flip-flops compared to the non-sharing scheme. This is due to the fact in order to capture the digital information on the racing signals, we use additional flip-flops for signal encoding. However, we are able to save 40.4% of LUTs and 39.58% of occupied slices. Considering flip-flops are much smaller than LUTs in size on FPGA, we claim our design improves in terms of area.

As of power consumption, our sharing scheme reduces overall power consumption and power per bit by 7.69% comparing to the non-sharing scheme. This number is relatively small considering the large area improvement. This is caused by the larger number of flip-flops used in the implementation. Flip-flops tend to dissipate more switching power than LUTs while in our sharing scheme the majority of shared hardware resources are LUTs.

7.2 Randomness

We quantify the statistical randomness of our design by applying the industry-standard National Institute of Standards and Technology (NIST) Statistical Test Suite to post-processed outputs [14]. An output stream is generated in such a way that the output of the system in the current clock cycle is fed back to the design as a challenge in the next clock cycle. We repeat this stream production process until all output bits ($1,000 \times 10,000$) are collected. Figure 5 displays the lowest success ratio of outputs generated by leap forward LFSRs, arbiter PUFs with Von Neumann correction, and our design. Three conclusions can be drawn from the figure:

- The output of our design shows excellent randomness, passing all tests in the test suite.

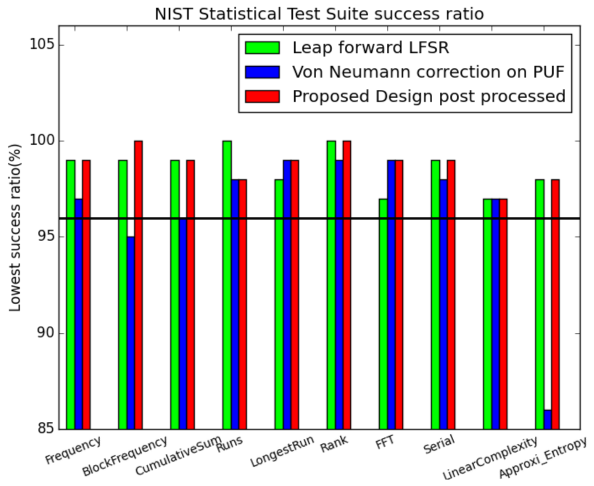

Figure 5: NIST Statistical Test Suite success ratio, one thousand 10,000 bit-streams are passed to each test. The test passes for p-value $\geq \sigma$, where σ is 0.05. The black line indicates a threshold of success ratio of 96%. All test results below this line are considered test failure. Arbiter PUF results without Von Neumann correction have success rate below 5% in most tests, thus are not shown in the figure.

- Our results is at least as random as leap-forward LF-SRs.

- Our results outperform arbiter PUF and Von Neumann correction in block frequency and approximate entropy. In all, our proposed method provides better randomness while maintaining physical unclonability.

8. CONCLUSION

We propose a mechanism to combine signal racing and logic computation through signal and hardware sharing. Employing such a mechanism greatly reduces the area overhead and power consumption. We illustrate our idea by combining leap-forward LFSRs and arbiter PUFs on a Spartan-6 FPGA. The evaluation shows that our sharing design saves 40.4% LUTs while achieving 7.69% of power improvement compared to the non-sharing scheme. Our design maintains the physical unclonability inherited from arbiter PUFs while, as suggested by NIST statistical test suite, achieving much better randomness. We conclude that by racing signals and computing logical operations simultaneously, we are able to create a power- and area-efficient PUF design with unclonability and high randomness.

9. ACKNOWLEDGMENT

This work was supported in part by the NSF under Award CNS-0958369, and Award CNS-1059435.

10. REFERENCES

[1] Ravikanth Pappu, Ben Recht, Jason Taylor, and Neil Gershenfeld. Physical one-way functions. *Science*, 297(5589):2026–2030, 2002.

[2] Blaise Gassend, Dwaine Clarke, Marten Van Dijk, and Srinivas Devadas. Silicon physical random functions. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 148–160. ACM, 2002.

[3] G Edward Suh and Srinivas Devadas. Physical unclonable functions for device authentication and secret key generation. In *Proceedings of the 44th annual Design Automation Conference*, pages 9–14. ACM, 2007.

[4] Jorge Guajardo, Sandeep S Kumar, Geert-Jan Schrijen, and Pim Tuyls. *FPGA intrinsic PUFs and their use for IP protection*. Springer, 2007.

[5] Sandeep S Kumar, Jorge Guajardo, Roel Maes, Geert-Jan Schrijen, and Pim Tuyls. The butterfly PUF protecting IP on every FPGA. In *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on*, pages 67–70. IEEE, 2008.

[6] Mehrdad Majzoobi, Farinaz Koushanfar, and Srinivas Devadas. FPGA PUF using programmable delay lines. In *Information Forensics and Security (WIFS), 2010 IEEE International Workshop on*, pages 1–6. IEEE, 2010.

[7] Leonid Bolotnyy and Gabriel Robins. Physically unclonable function-based security and privacy in RFID systems. In *Pervasive Computing and Communications, 2007. PerCom'07. Fifth Annual IEEE International Conference on*, pages 211–220. IEEE, 2007.

[8] Ulrich Rührmair. SIMPL Systems: On a Public Key Variant of Physical Unclonable Functions. *IACR Cryptology ePrint Archive*, 2009:255, 2009.

[9] Teng Xu, Hongxiang Gu, and Miodrag Potkonjak. Data protection using recursive inverse function. In *Field Programmable Logic and Applications (FPL), 2015 25th International Conference on*, pages 1–4. IEEE, 2015.

[10] Teng Xu and Miodrag Potkonjak. The digital bidirectional function as a hardware security primitive: Architecture and applications. In *Low Power Electronics and Design (ISLPED), 2015 IEEE/ACM International Symposium on*, pages 335–340. IEEE, 2015.

[11] Frederick James. A review of pseudorandom number generators. *Computer Physics Communications*, 60(3):329–344, 1990.

[12] Charles W O'Donnell, G Edward Suh, and Srinivas Devadas. PUF-based random number generation. *In MIT CSAIL CSG Technical Memo*, 481, 2004.

[13] Pong P Chu and Robert E Jones. Design techniques of FPGA based random number generator. In *Military and Aerospace Applications of Programmable Devices and Technologies Conference*, volume 1, pages 28–30. Citeseer, 1999.

[14] Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, and Elaine Barker. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, DTIC Document, 2001.