FPGA PUF using programmable delay lines

Conference Paper · January 2011

DOI: 10.1109/WIFS.2010.5711471 · Source: IEEE Xplore

Farinaz Koushanfar
University of California, San Diego
345 PUBLICATIONS **21,512** CITATIONS
SEE PROFILE

Sahana Devadas
Bangalore Medical College and Research Institute
552 PUBLICATIONS **31,253** CITATIONS
SEE PROFILE

FPGA PUF using Programmable Delay Lines

Mehrdad Majzoobi[†,1], Farinaz Koushanfar[†,2], Srinivas Devadas[‡,3]

[†] *Electrical and Computer Engineering Department, Rice University*
6100 Main Street, Houston, TX 77005 United States
[1] mehrdad.majzoobi@rice.edu
[2] farinaz@rice.edu

[‡] *Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology*
77 Massachusetts Avenue, Cambridge, MA 02142 United States
[3] devadas@mit.edu

***Abstract*—This paper proposes a novel approach for efficient implementation of a real-valued arbiter-based physical unclonable function (PUF) on FPGA. We introduce a high resolution programmable delay logic (PDL) implemented by lookup table (LUT) internal structure. Using the PDL, we perform fine tuning to cancel out delay skews caused by asymmetries in routing and systematic variations. We devise a symmetric switch structure that can be easily implemented on FPGA. To mitigate the arbiter metastability problem, we present and analyze methods for majority voting of responses. Lastly, a method to classify and group challenges into different robustness sets is introduced, to further increase the corresponding responses' stability in the face of environmental variations. Experimental evaluations show that the responses to robust challenges have an average error rate of less than 2% under temperature variations from -10^{o}C to 75^{o}C.**

***Index Terms*—physical unclonable functions, programmable delay line, FPGA, majority voting, tuning**

I. Introduction

Field programmable gate arrays (FPGA) provide a generic substrate of interconnected blocks that can be (re)programmed several times. The inherent flexibility of FPGAs compared to ASICs together with their lower time-to-market and availability of third party IPs, have made them the platform of choice for many applications. Like other systems, FPGAs demand security and resilience to attacks. In addition, techniques for ensuring IP security are necessary for prevention against piracy and unauthorized access.

A common denominator for many security protocols is the concept of a *secret*. For example, in public- and private-key cryptography, there is a secret key known by a limited set of parties. However, permanent FPGA key storage is not straightforward, as FPGAs often do not include nonvolatile on-chip memory. Even when the keys are externally powered or hidden in the bitstream, many side channel attacks for extracting the keys has been reported.

Physical unclonable functions (PUFs) aim at addressing the shortcomings of the digital key storage by relying on the secrets generated by the inherent and unclonable unique meso-scopic characteristics (signatures) of the physical phenomena [1, 2]. The physical properties of each device determine a specific mapping between a set of *challenges* (inputs) to a set of *responses* (outputs). *Challenge-response (CR) protocols* take advantage of this unique mapping to authenticate the device and/or its components.

To date, a number of possible implementations of PUFs on FPGAs based on the unique variations of silicon has been reported [3–5]. Applications of FPGA PUFs include securing programs and data, IP protection, RFIDs, and secure key generation, remote activation and IC entablement [3–9]. However, key limitations of the existing FPGA PUFs include the polynomial number of CR pairs, high power consumption, arbiter metastability, and/or the delay imbalances dictated by the routing constraints [7, 10].

This paper introduces a practical and stable implementation of an arbiter-based PUF on FPGA. An arbiter-based PUF works by comparing path timings for two routes with the same nominal delay (by design) but with differing actual delays (caused by manufacturing variations). To achieve equal nominal delays and to avoid bias, the two routes must be symmetric in signa routing. We introduce a low-overhead high-resolution programmable delay line (PDL) implemented by a single lookup table (LUT) on the FPGA. The new PDL is used to tune and calibrate the delay bias caused by asymmetries in signal routing. Furthermore, a symmetric PDL-based switch structure is introduced that can be implemented on FPGA. To mitigate arbiter metastability and to obtain more stable results, we introduce redundancy and majority voting on the responses. Lastly, we present a new method to classify and group challenges into different robustness sets. The challenge classification further increases the corresponding responses' resilience to environmental variations.

II. Background

A PUF utilizes the inherent specific properties of a physical device to define a unique mapping from a set of challenges (inputs) to a set of responses (outputs). The delay variations of CMOS logic components can be exploited to produce unique responses. In the PUF structure introduced in [2], the analog delay difference between two parallel timing paths is compared. The paths are built identically and their delays must be equal by construction, but the physical device imperfections

WIFS'2010, December 12-15, 2010, Seattle, WA, USA.
978-1-4244-9080-6/10/\$26.00 ©2009 IEEE.

make them different. The architecture of the arbiter-based PUF racing parallel paths is demonstrated in Figure 1. A step input simultaneously triggers the two paths. At the end of the two parallel (racing) paths, an arbiter is used to convert the analog delay difference between the paths to a digital value. The two paths can be divided into several smaller subpaths by inserting path swapping switches. Each set of inputs to the switches act as a challenge set (denoted by C_i), defining a new pair of racing paths whose delays can be compared by the arbiter to generate a one-bit response.

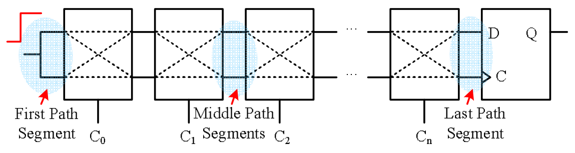


Fig. 1. Arbiter-based PUF with path swapping switches.

III. Related work

The authors in [2] were the first to exploit the unique and unclonable silicon process variations in nanometer scales for PUF formation. Their PUF used the analog differences between the delays of two parallel paths that are equal in design, but the physical device imperfections make the delays different. An arbiter inserted at the end of the paths generates binary responses indicating a comparison between the delays. To generate many CR pairs, the paths are divided into multiple subpaths and multiplexed by challenges. It is shown in [7, 10] that implementation of delay-based PUFs on FPGAs is problematic because of the routing constraints and arbiter inaccuracy. Ring oscillator (RO) PUFs rely on the specific and unique delay of an oscillating path on each device [5]. The presently known PUFs of this type contain many RO's so there are many possible pairs to compare. One can only have a quadratic number of challenges with respect to the number of RO's on FPGAs. Another disadvantage of RO PUF is the continuous dynamic power dissipation due to oscillation.

Another class of candidate FPGA PUFs are SRAM-PUFs and butterfly PUFs [3, 4]. Each FPGA SRAM cell would naturally tend to one logic state (either zero or one) upon startup. There are only a polynomial number of challenges with respect to the number of SRAM cells. An FPGA-based PUF along with a suite of time-bounded authentication protocols is introduced in [11]. The PUF produces binary responses based on the difference between the clock speed and some combinational circuit delay. Some instances of analog and digital PUFs that attempt to implement public cryptography are presented in [12–15].

IV. Arbiter PUF on FPGA

One of the major problems in implementation of PUFs on FPGAs, particulary the arbiter-based PUFs, is in signal routing. Unlike ASICs where hand-drawn custom layout is possible, routing on FPGA is constrained by its rigid fabric and interconnect structure. As a result, performing completely symmetric routing is physically infeasible in most cases. The PUF designer may do his/her best to constrain and guide the placement and routing software to achieve the highest degree of symmetry in the PUF layout. However, due to physical constraints of the FPGA fabric, the designer may still not be able to achieve complete symmetry on some routes. Asymmetries in routing when implementing PUFs can lead to bias in delay differences leading to predictable responses, lack of randomness, and decreased response entropy [7, 10].

The PUF routing can be divided into four different sections; the routing (1) before the first switch, (2) inside the switches, (3) between switches, and (4) after the last switch or before the arbiter (see Figure 1). As we will show later, by placing the logic components on symmetric sites and locations on the FPGA, the routing between switches will automatically follow a symmetric route. However, maintaining a *complete* symmetry between the top and bottom path routes before the first switch and after the last switch is structurally infeasible. To alleviate this problem, we introduce and exploit accurate PDLs to tune and remove the bias delay differences caused by asymmetries in net routing. We further introduce a new switch structure that by construction has a symmetric implementation.

A. Tuning with Programmable Delay Lines

In this section, we introduce a low overhead and high precision PDL with *pico*-second resolution. The introduced PDL is implemented by a single LUT. Figure 2 shows the internal structure of an example 3-input LUT. An n-input LUT can be configured to implement any n-input logic function. The LUT in Figure 2 is configured so that the inputs A_2 and A_3 act as *don't-care* bits. The LUT output is inverted A_1 and is not a function of A_2 and A_3. However, if we look closely, the inputs A_2 and A_3 determine the signal propagation path inside LUT. For instance, if $A_2A_3 = 00$, the signal propagates through the solid path (red), whereas if $A_2A_3 = 11$, the signal propagates through the path marked with the dashed-lines (blue). The lower dashed path is slightly longer than the upper solid path which results in a larger propagation delay. Xilinx Virtex 5

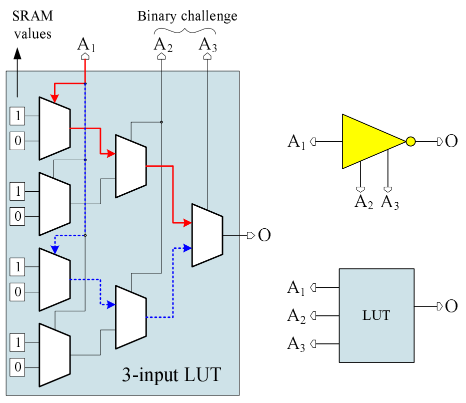


Fig. 2. The internal structure of a 3-input LUT.

FPGAs have 6-input LUTs which can implement a PDL with 5 control bits - there are 4 LUTs in each Slice and two Slices per each CLB. Similar to the above example, the first LUT input, A_1, is the inverter input and the rest of the LUT inputs control the delay of the inverter. For, $A_2A_3A_4A_5A_6$=$A_{[2:6]}$=00000, the inverter has the smallest delay (shortest internal propagation path) and for $A_2A_3A_4A_5A_6$=$A_{[2:6]}$=11111, the inverter has the maximum delay. In general if $A_{[2:6]} > A'_{[2:6]}$ then $D_{LUT}(A) > D_{LUT}(A')$, where $D_{LUT}(A)$ and $D_{LUT}(A')$ are the delay of the inverter with A and A' as the control inputs respectively.

We measured the changes in LUTs' propagation delays under different inputs. For delay measurements, we used the timing characterization circuit shown in Figure 3. The characterization circuit consists of a *launch* flip-flop, *sample* flip-flop, and *capture* flip-flop, an XOR gate, and the *Circuit Under Test (CUT)* whose delay is to be measured.

At the rising edge of the clock a signal is sent through the CUT by the launch flip-flop. At the falling edge of the clock, the output of the CUT is sampled by the sample flip-flop. If the signal arrives at the sample flip-flop well before sampling takes place, the correct value is sampled. The XOR compares the sampled value with steady state output of the CUT and produces a zero if they are the same. Otherwise, the XOR output rises to '1', indicating a timing violation. If the signal arrives almost simultaneously with when sampling occurs, the sample flip-flop enters into a metastable condition and produces a non-deterministic output. By sweeping the clock frequency and monitoring the rate at which timing errors happen, the CUT delay can be measured with a very high accuracy. For further details on the delay characterization method the reader is referred to [11, 16].

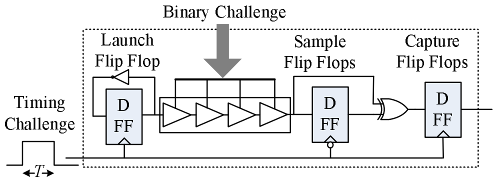


Fig. 3. Delay characterization circuit.

The measurements performed on Xilinx Virtex 5 FPGAs suggest that the maximin delay difference (i.e. A=00000, and A'=11111) achieved by each inverter is 9ps on average.

B. PDL-based Symmetric Switch

The first arbiter-based PUF introduced in [2] (see Figure 1) uses path swapping switches as shown in Figure 4 (a). The switch, based on its selector bit, provides a straight or cross connection. Figure 4 (b) shows the equivalent circuit implementation and delays. The path swapping switch structure does not lend itself to FPGA implementation, since it is extremely difficult to equalize the nominal delays of the top and bottom paths due to routing constraints, i.e., ***a*** and ***d*** (or the diagonal paths ***b*** and ***c***). To alleviate the issue, we propose a new non-swapping switch structure as shown in Figure 4 (c). The yellow triangles in the figure represent two PDLs. Figure 4 (d) shows its equivalent circuit where the nominal delay values of ***a*** and ***d*** (or the diagonal paths ***b*** and ***c***) must be the same.

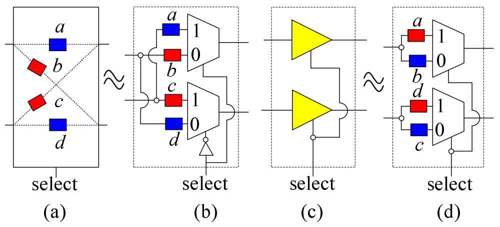


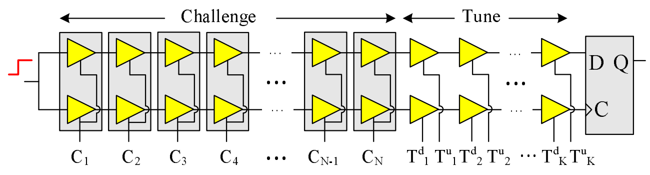
Fig. 4. (a),(b) path swapping switch and its delay abstraction (c),(d) PDL-based switch and its delay abstraction.

The complete PUF circuit that uses the new switch structure and the tuning blocks is shown in Figure 5. The presented system consists of N switches and K tuning blocks. The tuning blocks insert extra delays into either the top or bottom path based on their selector inputs to cancel out the delay bias caused by routing asymmetry. The only difference between a tuning block and a switch block is that in the former, the selectors to the top and bottom PDLs are controlled independently but in the latter, the same selector bit drives both PDLs. Also note that the tuning blocks do not necessarily have be placed at the end of the PUF. As a matter of fact, they can be placed anywhere on the PUF in between the switches.

Similar to the arbiter-based PUF with path swapping switches, the new PUF structure is a linear system. The PUF response will be '1' if the sum of the delay switch differences along the path is greater than zero, and '0' otherwise:

$$\sum_{i=1}^{N} C_i \times (a_i - d_i) + (1 - C_i) \times (b_i - c_i) + \Delta \underset{R=1}{\overset{R=0}{\lessgtr}} 0, \quad (1)$$

where a_i, b_i, c_i, d_i are the i-th switch delays as shown in Figure 4 (d), $C_i \in \{0, 1\}$ is the i-th challenge bit, and R is the response. Also, Δ is a constant delay difference from first and last path segments and tuning blocks lumped together. The security aspects of the linear PUF structures against machine learning attacks can be boosted by insertion of feed forward arbiter and attaching input/output XOR logic networks to multiple rows of PUFs [17, 18]. The work in analyzing the complexity of machine learning and model attacks against different classes of PUFs [19].


Challenge
Tune
D Q
C
C_1 C_2 C_3 C_4 … C_{N-1} C_N T^d_1 T^u_1 T^d_2 T^u_2 … T^d_K T^u_K
Fig. 5. The new arbiter-based PUF structure.

V. Precision Arbiter

Arbiters in practice are implemented by D flip-flops. As a result, an arbiter has a limited resolution meaning that if the absolute delay difference of the arriving signals is smaller than its setup and hold time, it enters a metastable state and its output becomes highly sensitive to circuit noise and will be unreliable. The probability of flip-flop output being equal to ‘1’ is a monotonically decreasing function of the input signal timing difference (Δ_T). Such probability in fact follows a Gaussian CDF curve as shown in [7, 16]:

$$P_{O=1}(\Delta_T) = Q(\frac{\Delta_T}{\sigma}) \qquad (2)$$

where $Q(x) = \frac{1}{\sqrt{2\pi}}\int_x^\infty \exp\left(-\frac{u^2}{2}\right)$ is the Q function. For an infinitely precise arbiter, σ is infinitesimal i.e. $\sigma \to 1/\infty$, and $P_{O=1}(\Delta_T) \to 1 - U(\Delta_T)$ where U is the step function.

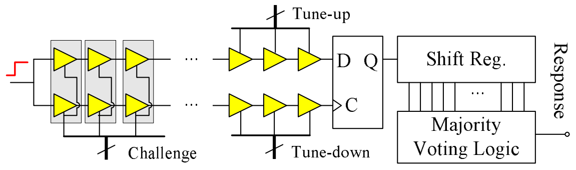


Fig. 6. Reducing the response instability due to arbiter metastability by using majority voting.

To increase the arbiter accuracy, we propose multiple evaluations of the same challenge to the PUF and running a majority vote on the output responses as shown in Figure 13. The repetitive challenge evaluation combined with majority voting is equivalent to having an arbiter with effectively smaller σ. We will quantify the reduction in σ as a function of the number of repetitions in the experimental results section.

VI. Robust Responses

Fluctuations in operational conditions such as temperature and supply voltage can cause variations in device delays. The impact on delays may not be equal on all devices. As an example, the signal propagation delay on the PUF top and bottom paths is represented in Figure 7 by solid and dashed lines respectively. In this example, the path delays increase with temperature at different rates. In the diagram in Figure 7 (a), the delay difference Δ_d at the end of the PUF for a given applied challenge at nominal temperature is small, whereas Δ_d in Figure 7 (b) is larger for another challenge. The response to the challenge in Figure 7 (a) changes as temperature varies because the delays change their order (cross). However, in Figure 7 (b) the PUF response remains the same. As demonstrated by this example, the responses to those challenges that cause large delay differences are unlikely to be affected by temperature or supply voltage variations [5].

In this paper, we use the PDL-based tuning blocks to distinguish those challenges that cause a larger delay difference and mark them as robust challenges. For each challenge, the PUF is evaluated with (i) an extra delay inserted to the top path by tuning upward (ii) an extra delay inserted to the bottom path by tuning downward (iii) equal tuning on both the top and bottom paths. If the response to these three cases are the same, we conclude that the resulting delay difference by the applied challenge is large enough not to be affected by

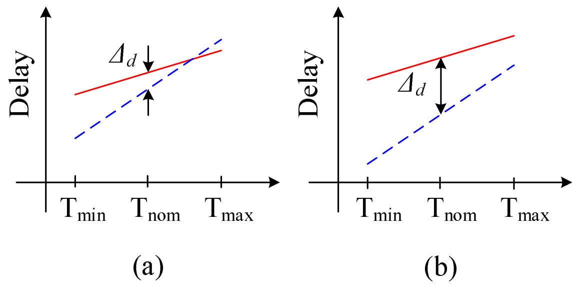


Fig. 7. Signal propagation delay as a function of temperature.

small additions and subtractions. However, it’s critical to note that from information theoretical point of view, the responses from more robust challenges are more likely to have lower entropy. In other words, ultimate robustness exists when the responses are absolutely biased toward either zero or one, however, the entropy is zero and the responses are not distinct enough for identification. Quantification of such a trade-off between response robustness and entropy in PUFs is not being addressed in this paper and is a subject of future study.

VII. Experimental Evaluation

In this section, we present the results of experiments performed on a Xilinx Virtex 5 XC5VLX110T FPGA device. The implemented PUF has 16 rows whose challenge input bits are connected together and placed in parallel on the FPGA to produce 16 bits of responses per challenge. Each PUF consists of 64 stages of PDLs, where the PDL is implemented by 2 LUTs each acting as an inverter. Figure 8 shows the placement and routing of one of the PUF rows. As it can be seen, except for the routing at the beginning and end of the PUF, the rest follows a completely symmetric pattern. A TCL script is developed to control the PUF challenges to the PDLs and read back the responses using ChipScope Pro VIO Core.

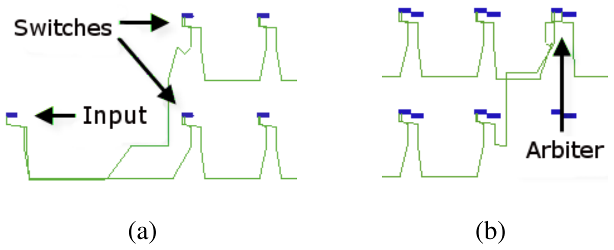


Fig. 8. Routing and placement of the PUF (a) first segment (b) last segment.

Before using the PUF and in order to see any changes in the responses, it must be tuned to remove the delay bias from routing asymmetry. In the first experiment, we look at all 16 responses to observe at what tuning level their response to a fixed challenge begins to transition. At first, the selectors to PDLs are all set to zero, and then more delay is added to the top path by raising their selectors bits to ‘1’. We refer to the difference in the number of ‘1’s in the top and bottom PDL selector bits as the *tuning level*. This quality can be either positive or negative indicating insertion of delays to the top and bottom path respectively. The response for each tuning level is repeated 100 times, and the number of '1's in the responses are counted and normalized to find the probability of '1' response as shown in Figure 9. The probability of the response being equal to one is shown for 4 PUF response bits. For instance, the third response bit goes from one to zero as the tuning is increased from 8 to 9. We average the transition points for all 16 bits and find the best tuning level to be 14.

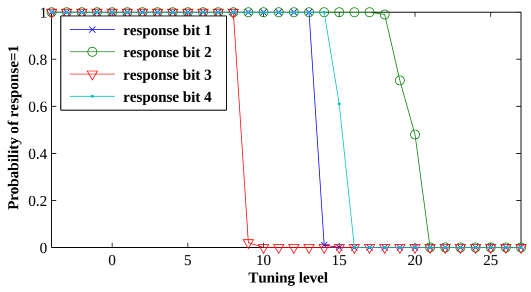


Fig. 9. Probability of observing a 1 response versus different tuning levels for four of the PUF response bits.

In the previous experiment, the challenges to the PUF were fixed. Next, we apply 1000 random challenges to the PUF repeating each 10 times. If more than half of the responses are ones, the responses is considered one, and zero otherwise. Figure 10 shows the frequency of '1' responses obtained from 1000 challenges for all 16 bits. Each boxplot represent 16 points and each point is derived by dividing the number of ones in the 1000 responses. As it can be seen the median of the distribution is biased toward '1' as tuning is increased. Again, for the tuning level of 14 the median is closest to 50%. From now on, we choose the tuning level of 14 as the optimum tuning level. The tuning level is determined once from statistics across a few chips and it will be permanently set to that level. Note that from the PUF definition, the circuit must always be the same on every chip and the response must be only a function of the input challenges and manufacturing variations. Having different tuning levels for each circuit on different ICs contradicts with the standard definition of the PUF. Nevertheless, the tuning bits can be treated as a separate set of challenges.

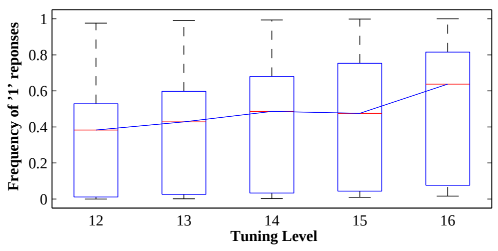


Fig. 10. Probability of observing a '1' response for 1000 challenges versus different tuning levels.

Figure 11 shows the percentage of '1' responses in each response bit to 1000 random challenges and tuning level of 14. Similar to the previous study, each challenge is being repeated 10 times to obtain a stable response. To ensure uniqueness of the responses, the hamming distance between each pair of output response bits to 1000 random challenges is calculated and shown in Figure 12. The low and high hamming distances correspond to the skewed response bits.

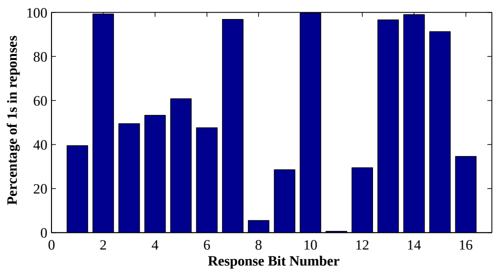


Fig. 11. The percentage of 1 responses in each response bit to 1000 random challenges and tuning level of 14.

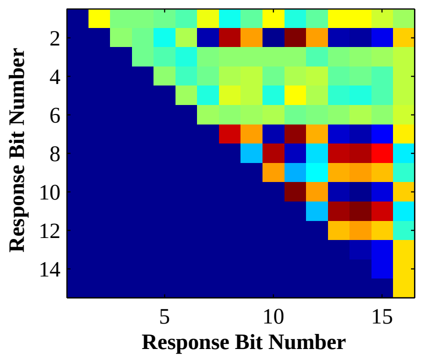


Fig. 12. Hamming distance of pairs of output response bits.

As discussed in the paper, repeating the challenges to the PUF and running majority voting on the obtained responses can help improve the precision of the arbiter. In this section, we quantify this effect. Figure 13 shows the probability of observing a '1' output from a flip-flop as a function of the input signals delay difference. This characteristic has been measured on Xilinx Virtex 5 FPGAs [7, 16]. The width of the transition region (3σ) gets narrower as evaluation is repeated and more statistics is gathered.

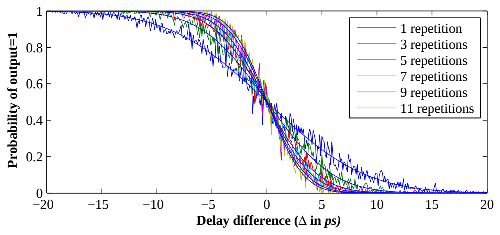


Fig. 13. The probability of majority voting system output being equal to 1 as a function of the delay difference.

The equivalent σ which represents the width of the metastable window (i.e. 3σ) is calculated for different number of repetitions as shown Figure 14. The reduction in the metastable window width is logarithmic with respect to the number of repetitions. For 10 repetitions, $\sigma = 2.5\ ps$.

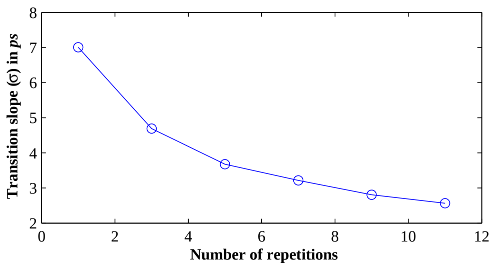


Fig. 14. The sharpness (σ) of the transition slope versus the number of repetitions for majority voting.

Finally, we study the effect of robust challenge classification on PUF error rate in presence of temperature variations. In order to separate and classify challenges into different robustness groups, we perform the following three experiments. We apply a 1000 sets of random challenges to the PUF while the tuning level is once set to 14, once 13 and once to 15 and we obtain 1000×16 responses for each case. Then the challenges that produce those responses that do not change across the three cases are marked as *low* robust challenges. These challenges are a subset of all 1000 *normal* challenges. Next, the same experiment is repeated but this time with a wider range of tuning level variations from 12 to 16. The challenges that produce responses that remain the same over this interval are marked as *high* robust. To find the response error rate for these three different sets of challenges, they are applied to the PUF (with tuning level fixed to 14) and the operational temperature is set to -10^{o}C and 75^{o}C in two experiment settings. Then the responses from the three group of challenges are compared to capture the error rate as shown in Figure 15. Each boxplot consists of 16 points where each point corresponds to one response bit. As it can be observed the error rate is considerably reduced for low and high robust challenges.

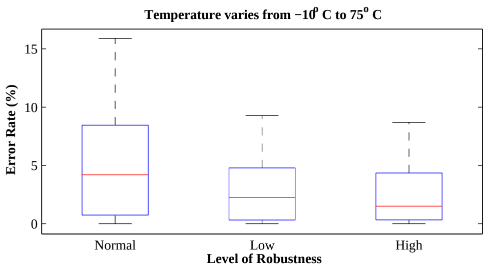


Fig. 15. The response error rate for sets challenges with different robustness level under temperature variations.

VIII. Conclusion

We introduced a new arbiter-based PUF structure that exploits programmable delay lines (PDL) to tune and cancel out the delay skews caused by asymmetries in routing on FPGAs. A low-overhead high-resolution PDL based on LUT internal propagation paths was introduced and used for tuning. A PDL-based symmetric switch structure was further introduced to resolve the routing issues. To mitigate the arbiter metastability problem, we presented and analyzed majority voting of responses. Lastly, a method to classify challenges into different robustness groups was introduced, to increase the response stability in the presence of environmental variations. Experimental evaluations demonstrated that the responses to robust challenges have an average error rate of less than 2% under temperature variations from -10^{o}C to 75^{o}C.

References

[1] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, "Physical one-way functions," *Science*, vol. 297, pp. 2026–2030, 2002.
[2] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon physical random functions," in *CCS*, 2002, pp. 148–160.
[3] J. Guajardo, S. Kumar, G. Schrijen, and P. Tuyls, "FPGA intrinsic PUFs and their use for IP protection," in *CHES*, 2007, pp. 63–80.
[4] S. Kumar, J. Guajardo, R. Maes, G.-J. Schrijen, and P. Tuyls, "The butterfly PUF protecting IP on every FPGA," in *HOST*, 2008, pp. 67–70.
[5] G. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *DAC*, 2007, pp. 9–14.
[6] G. Suh, C. O'Donnell, I. Sachdev, and S. Devadas, "Design and implementation of the AEGIS single-chip secure processor using physical random functions," in *ISCA*, 2005, pp. 25–36.
[7] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Techniques for design and implementation of secure reconfigurable PUFs," *TRETS*, vol. 2, no. 1, pp. 1–33, 2009.
[8] Y. Alkabani, F. Koushanfar, and M. Potkonjak, "Remote activation of ICs for piracy prevention and digital right management," in *International Conference on Computer Aided Design (*ICCAD*)*, 2007, pp. 674–677.
[9] Y. M. Alkabani and F. Koushanfar, "Active hardware metering for intellectual property protection and security," in *USENIX Security Symposium*, 2007, pp. 1–16.
[10] S. Morozov, A. Maiti, and P. Schaumont, *An Analysis of Delay Based* PUF *Implementations on* FPGA. Springer, 2010, pp. 382–387.
[11] M. Majzoobi, A. Elnably, and F. Koushanfar, "FPGA time-bounded unclonable authentication," in *IH*, 2010.
[12] U. Rührmair, "SIMPL system: on a public key variant of physical unclonable function," *Cryptology ePrint Archive*, 2009.
[13] N. Beckmann and M. Potkonjak, "Hardware-based public-key cryptography with public physically unclonable functions," in *IH*, 2009, pp. 206–220.
[14] G. Csaba, X. Ju, Q. Chen, W. Porod, J. Schmidhuber, U. Schlichtmann, P. Lugli, and U. Rührmair, "On-chip electric waves: An analog circuit approach to physical uncloneable functions," *Cryptology ePrint Archive*, 2009.
[15] C. Jaeger, M. Algasinger, U. Ruhrmair, G. Csaba, and M. Stutzmann, "Random pn-junctions for physical cryptography," *Applied Physics Letters*, vol. 96, no. 17, pp. 172 103 –172 103–3, 2010.
[16] M. Majzoobi, E. Dyer, A. Elnably, and F. Koushanfar, "Rapid FPGA characterzation using clock synthesis and signal sparsity," in *ITC*, 2010.
[17] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Testing techniques for hardware security," in *ITC*, 2008, pp. 1–10.
[18] L. Daihyun, J. Lee, B. Gassend, G. Suh, M. van Dijk, and S. Devadas, "Extracting secret keys from integrated circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 10, pp. 1200 – 1205, 2005.
[19] U. Rhrmair, F. Sehnke, J. Slter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling attacks on physical unclonable functions," in *Conference on Computer and Communications Security*, 2010.