Fault Tolerant Implementations of Delay-based Physically Unclonable Functions on FPGA

Durga Prasad Sahoo*, Sikhar Patranabis†, Debdeep Mukhopadhyay‡ and Rajat Subhra Chakraborty§
SEAL/CSE, Indian Institute of Technology, Kharagpur, INDIA - 721302
Email:{dpsahoo.cs*,sikharpatranabis†}@gmail.com & {debdeep‡,rschakraborty§}@cse.iitkgp.ernet.in

***Abstract*—Recent literature has demonstrated that the security of Physically Unclonable Function (PUF) circuits might be adversely affected by the introduction of faults. In this paper, we propose novel and efficient architectures for a variety of widely used delay-based PUFs which are robust against high precision laser fault attacks proposed by Tajik et al. in FDTC-2015. The proposed architectures can be used to detect run-time modifications in the PUF design due to fault injection. In addition, we propose fault recovery techniques based on either logical reconfiguration or dynamic partial reconfiguration of the PUF design. We validate the robustness of our proposed fault tolerant delay-based PUF designs on Xilinx Artix-7 FPGA platform.**

***Index Terms*—Arbiter PUF (APUF), PUF, laser fault attacks and countermeasures, robust PUF architectures.**

I. Introduction

The advent of new technology such as the Internet of Things (IoT) has led to a growing demand for reconfigurable and efficient devices with embedded security for a variety of applications. One such technique is the use of programmable logic devices (PLDs) such as Field Programmable Gate Arrays (FPGAs) and Complex Programmable Logic Devices (CPLD) for designing cryptographic primitives. Traditional cryptographic applications built using FPGAs, such as secure key storage and random key generation, have been subjected to a number of side channel attacks, leading to the need for alternative solutions in this regard. Physically Unclonable Functions (PUFs) [1], [2] are believed to be pertinent solutions in this regard. A PUF is a physical entity that is easy to evaluate but hard to predict, and is practically infeasible to clone, even with the knowledge of the exact manufacturing process that produced it. PUFs with intrinsic randomness have gradually grown to be an effective and more secure cryptographic substitute for traditional methods of random key generation and secure key storage. Unlike traditional approaches, where the cryptographic key is generated and stored in secure non-volatile storage, PUFs provide a method to generate random keys [3], [4] via a unique sequence of challenge-response pairs (CRPs) that is hard to model or reproduce exactly. In addition, several classes of PUFs such as delay-based PUFs [1], [5] are efficiently implementable on FPGAs, making them an ideal choice for device authentication and random key generation. Some such PUF constructions include the XOR Arbiter PUF (APUF) which are popularly used for authentication [6], [7], and the Ring Oscillator PUF (ROPUF) family that is suitable for random number generation [3].

While traditionally PUFs have been subjected to a number of machine learning based attacks that attempt to model the PUF in terms of its challenge-response behavior [8], recent literature also reports several instances of physical attacks [9]–[14] on PUF instances. We next discuss the nuances of fault attacks on PUFs and argue why it is different from normal fault attack techniques on other standard cryptographic primitives.

A. Faults Attacks on PUFs

A fault is an aberration in a device/circuit arising from either manufacturing defects or (post-manufacturing) external interferences. In cryptography, fault injections attacks are invasive/semi-invasive in nature, and are aimed at revealing some secret information about the system such as the secret key. Fault attacks can be performed using a variety of techniques, namely - voltage spikes, clock glitches, EM radiations, temperature variations and even optical attacks by exposing a depackaged chip to photo flash or laser beam sources [15].

PUFs have been subjected to a number of different fault attacks in the recent literature [14]. Unlike in traditional cryptographic primitives where an attack is carried out with the intention of recovering the secret key, PUFs may be attacked to compromise one of more of its properties, including randomness, reliability and unclonability [16], so as to reduce the cryptographic

strength of the primitive designed using the PUF module. We describe a few such attack objectives below:

1) **Entropy reduction.** The PUF responses with reduced entropy are easier to predict and should not be used in security applications. For instance, in [14], a laser beam applied on the backside of the IC is used to convert the inverting stage of a RO to a non-inverting buffer, thus hindering the oscillation at the output of RO. This leads to a reduction in the output entropy. ROPUF based applications such as key-generator or random number generator are vulnerable to this type of fault attack.

2) **Accelerating ML based modeling.** Fault attacks are often used to expedite machine learning (ML) based modeling attacks on PUFs that are otherwise expensive or computationally infeasible. For instance, ML based modeling of x-XOR APUF [17], [18] is computationally infeasible with $x \geq 10$. However, in [14], the authors use laser fault injection to modify the XOR APUF structure such that they can access the CRPs of individual APUFs and eventually, combine these to model the whole XOR APUF. Alternatively, in [19], authors observe the correlation between the reliability of XOR PUF and its component APUFs, and they employ this fact to model the individual APUFs based on the reliability information of the XOR APUF. Some fault attacks change the operating temperature of the PUF to generate unreliable CRPs that leak side channel information and make modeling attacks easy to mount [19]. Another instance of a semi-invasive attack uses EM radiations to characterize RO PUFs from the IC backside [20].

3) **Denial of service.** These attacks are commonly mounted on PUF instances used for authentication, e.g. Loop PUF (LPUF) [21] and Bistable Ring PUF (BRPUF) [22]. The adversary aims to inject faults in the PUF instance so as to cause a authentication failure with the verifier maintaining a database of the original fault-free PUF device. In case of LPUF, converting an inverting element to non-inverting (or vice versa) using a precise laser beam stops the oscillation at the loop output. Similarly, an adversary could maliciously modify a BRPUF instance with even number of inverters to one with odd number of inverters, such that the output never reaches a stable state. In case such faults are remnant, the PUF would revert to its fault-free state after reconfiguration, but not before the service of the PUF-enabled device has been disrupted for a certain time interval.

In view of such threats, suitable measures must be taken to prevent them. However, there exist some fundamental challenges in applying traditional countermeasure techniques such as spatial and temporal redundancy [23] for the design of fault tolerant circuits for PUFs. The challenge arises from the two major properties of PUFs, namely randomness and physical unclonability, as described next:

- **Randomness.** A PUF instance is characterized by a uniquely random set of CRPs, implying that there is no golden or reference CRP that can be used to detect the presence of a fault. There exist no guarantees as to whether a randomly injected fault shall improve or degrade the randomness of a PUF instance, making it very hard to detect the occurrence of fault, if any.
- **Physical Unclonability.** The very fact that PUF instances cannot be physically duplicated implies that spatial redundancy is not a viable option for detecting faults. Temporal redundancy, on the other hand, may be rendered unproductive by an adversary who introduces a *remnant fault* [15] in the PUF circuit. Such a fault, once introduced during the configuration of a FPGA, lingers till the next reconfiguration phase, and causes the circuit to produce the same faulty output in both the original and redundant operation phases.

B. Contributions

The main contribution of our paper is the design of novel fault tolerant PUF circuits for a variety of delay PUFs, that are robust against high precision fault attacks using laser pulse injection [14]. Our contributions in this paper are summarized as follows:

1) The paper proposes fault detection circuits for APUF design that can detect any architectural changes in the PUF circuit due to fault injection. The fault detection logic (FDL) in the circuit outputs '0' or '1' to indicate the presence or absence of faults. We further use these fault tolerant APUF constructions as building blocks to design robust XOR APUF circuits. In addition, we protect the final XOR gate of the XOR APUF to prevent the adversary from gaining access to the responses

of the individual APUF instances. This in turn prevents the modeling attacks against XOR APUF proposed in [14].

2) The paper also proposes a fault tolerant design for the Ring Oscillaotr (RO), which is a basic building block for the ROPUF. Our attack can detect any modifications to the individual delay stages of the RO that could potentially prevent the oscillations at the output of the RO. We thus successfully thwart attacks that could attempt to bias the output of the ROPUF and degrade the security of the cryptographic primitive designed using the same.
3) The paper explores possible fault attacks on two additional varieties of delay PUFs: Loop PUF and Bistable Ring PUF, and proposes fault tolerant design techniques to thwart these attacks.
4) Two possible fault recovery techniques for PUFs - *Rollback* and *Random-sliding* are discussed in details in the paper. The pros and cons of the two approaches are compared in terms of their efficiency and implementation overhead.
5) The proposed countermeasures are validated using high precision fault injection simulations on PUF circuits implemented on a Xilinx Artix-7 FPGA platform. The overhead for the fault detection logic is also reported for each countermeasure. The results establish that our proposed countermeasures successfully thwart a wide variety of fault attacks on popularly used PUF instances with low overhead costs for implementation.

C. Organization

The rest of the paper is organized as follows. In Section II, we provide a brief overview of Xilinx FPGA architecture and delay PUF designs considered in this paper. The possible fault attacks and their countermeasures for APUF and ROPUF are discussed in Sections III and IV, respectively. Section V explains possible fault attacks and the corresponding countermeasures for the LPUF and BRPUF. Two fault recovery schemes are illustrated in Section VI. Experimental results and hardware resource overhead of fault tolerant PUF designs are reported in Section VII. Finally, the paper is concluded in Section VIII.

II. Preliminaries

A. Notations

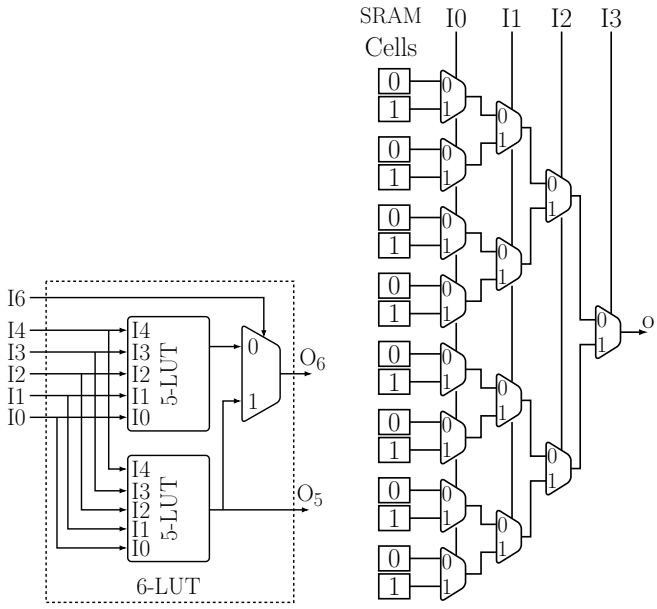
We use following notation system in the rest of the paper. A letter in bold font refers to a vector, e.g $\mathbf{a}$. A vector with m-components is represented as $\mathbf{a} = (\mathbf{a}[0], \ldots, \mathbf{a}[i], \ldots, \mathbf{a}[m-1])$, where $\mathbf{a}[i]$ is the ith component of the vector. We use $\mathbf{a}[i{:}j]$ to denote a sub-vector $(\mathbf{a}[i], \ldots, \mathbf{a}[j])$. A scalar is denoted by a lower-case letter, e.g. n.

B. Fundamentals of Xilinx FPGA Architecture

FPGAs are a special class of PLDs that are based on gate array technology and are preferred over CPLDs for their sophisticated architecture with Look-Up Tables (LUTs) and arbitrarily programmable interconnects, that lead to greater performance at lesser hardware footprint for large designs. Although originally used as a hardware prototyping platform, the modern FPGA has evolved into a mainstream platform for digital system designs owing to the addition of several reconfigurable features. State-of-the-art FPGAs are equipped with DSP blocks, multi-port block RAMs, and hard processor cores that make them suitable for the design and realization of complex systems. In the PUF literature, a number of PUF designs focus specifically on deployment in secure FPGA based designs [24]–[26]. All the PUF constructions reported in this paper are implemented on the Artix-7 FPGA, which is a Xilinx 7-series FPGA. We briefly discuss the typical architecture of Xilinx 7-series FPGA, which is referred to extensively in the remainder of the paper.

An FPGA may be viewed as a sea of configurable logic blocks (CLBs) interconnected via programmable switching components. Each CLB in a 7-series FPGA consists of two slices, which may in turn be of two categories: SLICEL (logic slice) and SLICEM (memory slice). SLICELs are used solely as function generators, whereas SLICEMs can be used as function generators, shift registers, and distributed RAMs. Each slice in a CLB contains four 6-input LUTs (in short 6-LUTs) and eight flip-flops (FF). These LUTs constitute the main programmable component to realize any custom logic. LUTs in 7-series FPGAs have two outputs O_5 and O_6, that allow a single LUT to realize either a single 6-input function (using output O_6 only), or two 5-input functions (using both outputs O_5 and O_6). An architectural view of a 6-LUT is shown in Fig. 1a. Typically, an LUT (aka function generator) comprises of an array of SRAM cells to store the function outputs and a tree of 2:1 MUXes to establish an input-output mapping, as shown in Fig. 1b. For a give values of its inputs, a specific SRAM cell is selected and its content propagates to the output o (cf. Fig. 1b).

It can be observed from Fig. 1b that the initialized content of the SRAM cells results in a programmable delay buffer, with I0 as the input and o as the output.



(a) A 6-input LUT (6-LUT) (b) Internal view of 4-LUT

Fig. 1: Structural view of lookup table (LUT) in Xilinx 7-series FPGA. The internal structure of the 6-input LUT is similar to that of the 4-input LUT as shown in Fig. 1b.

The remaining inputs I1,...,I3 can be used to program the input-to-output propagation delay of the buffer. For instance, for I0 = 0, each combination of the remaining inputs I1,...,I3 select a unique SRAM cell with '0' content, resulting in a different propagation delay. This idea is used in the development of the Programmable delay line (PDL) based APUF [25]. Yet another PUF design, known as the Anderson PUF [27], exploits the carry-chain logic of the FPGA slice as the basic component.

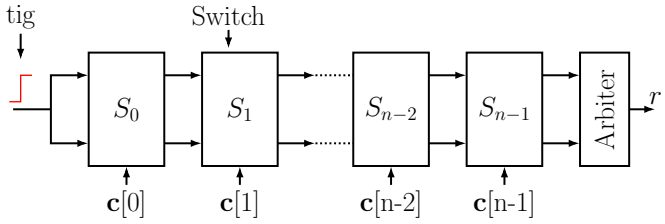
C. Overview of Delay PUFs

In this section we provide a brief overview of the digital silicon PUF. The CMOS silicon PUF exploits the intrinsic random process variations in the CMOS devices due to submicron CMOS fabrication [28]. There are two classes of silicon PUFs based on this response generation technique: delay PUF and memory PUF. In delay PUF, the response to a challenge is generated by comparing the delays of two symmetrical delay paths, whereas in memory PUF the response depends on the random stable state attained by bistable memory elements left from an initial unstable state. In this paper, we focus only on the delay PUFs and hence we discuss a few relevant delay PUF designs that are used in the paper.

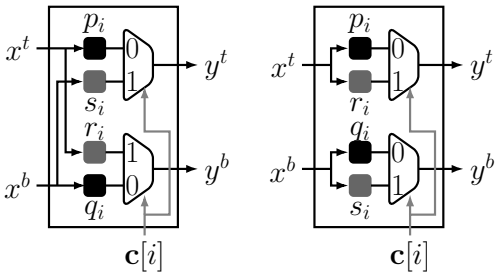
Figure 2a shows the architectural overview of APUF design [1], with a cascade of 2-by-2 switches and an *arbiter* at the end. From a common point, a voltage impulse ('tig') propagates along two symmetrical paths defined by the applied control bits to the switches. The individual stage delays are randomly determined by device-level process variation effects, and are unpredictable. The accumulated delay difference along the two paths determines the output of the arbiter circuit, which is usually an edge-triggered D flip-flop (DFF) or a SR latch. The control bit string is considered as the applied *challenge*, and the output of arbiter is considered as *response*.

There currently exist two different proposals for APUF switching components: *swapping path switch* [1] and *non-swapping path switch* [25]. Both the swapping and non-swapping switches can be implemented using a pair of 2:1 multiplexers as shown in Figs. 2b and 2c, respectively. A discussion on the difference in the quality (uniformity, uniformity, reliability) of APUF for these two types of switches is presented in [29]. Furthermore, APUF with swapping path switches exhibits better quality (uniformity and uniqueness) than one with non-swapping path switches. In this paper, we focus more on fault tolerant APUF designs, and use both the aforementioned structures in our discussions.

Although a standalone APUF is often vulnerable to machine learning based modeling attacks [17], its XOR



(a) APUF architecture



(b) Classic switch (c) PDL based switch

Fig. 2: Arbiter PUF architecture with two types of switches. (b) Classic switches is also known as path swapping switch, and (c) PDL switch is known as non-swapping path switch.

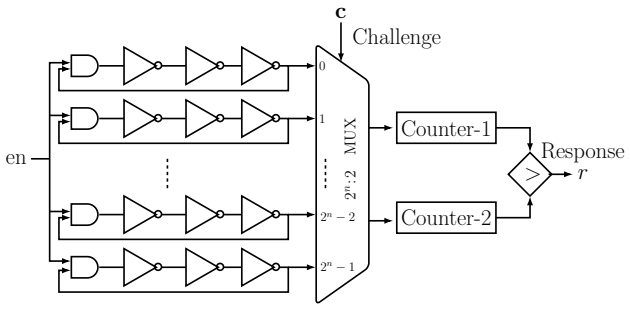


Fig. 3: Architectural overview of ROPUF.

composition, popularly known as the XOR APUF, is robust against modeling provided that the number of APUF instances being XORed exceeds 10 [18]. However, as mentioned previously, in [14], Tajik et al. proposed a poly-time modeling fault attack on XOR APUF, that uses laser injections to model each individual APUF component separately. In this paper, we aim to develop a countermeasure to prevent such fault attacks on XOR APUF circuits.

Another delay PUF, widely used as a random key-generator, is the Ring Oscillator PUF (ROPUF). It exploits ring oscillators (ROs) as the delay components, and the response to a given challenge is generated by comparing the frequencies of a pair of identically laid-out ROs. In Xilinx FPGA, the Hard Macro feature of the Xilinx FPGA Editor CAD software tool is used to instantiate identically laid-out ROs. Figure 3 depicts the architectural description of classic ROPUF. We consider this ROPUF design to discuss the fault tolerant ROPUF design against laser fault attack [14].

III. Countering Fault Attacks on XOR APUF

XOR APUFs with more than 10 APUF instances are, to the best of our knowledge, secure against traditional machine learning attack. However, two recently proposed fault attacks [14] on the XOR APUF using precise laser injections manage to model a XOR APUF by converting it into a simple APUF design. For an x-XOR APUF, the adversary repeats this process x times to get the responses of the x individual APUF instances, leading to a linear time modeling of an x-XOR APUF for any arbitrary x. Based on the exact location of fault injection, two attacks are described in [14]:

- **Attack-I.** The target circuit component in this attack is the last switch (S_{n-1}) of APUF (cf. Fig. 4a) which is a component of XOR APUF. The adversary modifies the switching logic of S_{n-1} such that CLK input of the arbiter circuit is always '0', resulting in a fixed value at the output of arbiter, regardless of the input challenge. She needs to perform this modification for all APUFs except the one APUF for which she wants to collect CRPs.
- **Attack-II.** In this attack scenario, the final XOR circuit of the XOR APUF is the target component, which is implemented by a LUT on a FPGA. The adversary uses a laser pulse to change the content of SRAM cells associated with the target LUT so as to transform the XOR gate into a buffer gate for its ith input. Thus, the response of ith APUF will be available as the XOR response, and this is a serious security issue as the basic assumption of the XOR APUF (that the internal inputs of XOR are secret) is violated in this case.

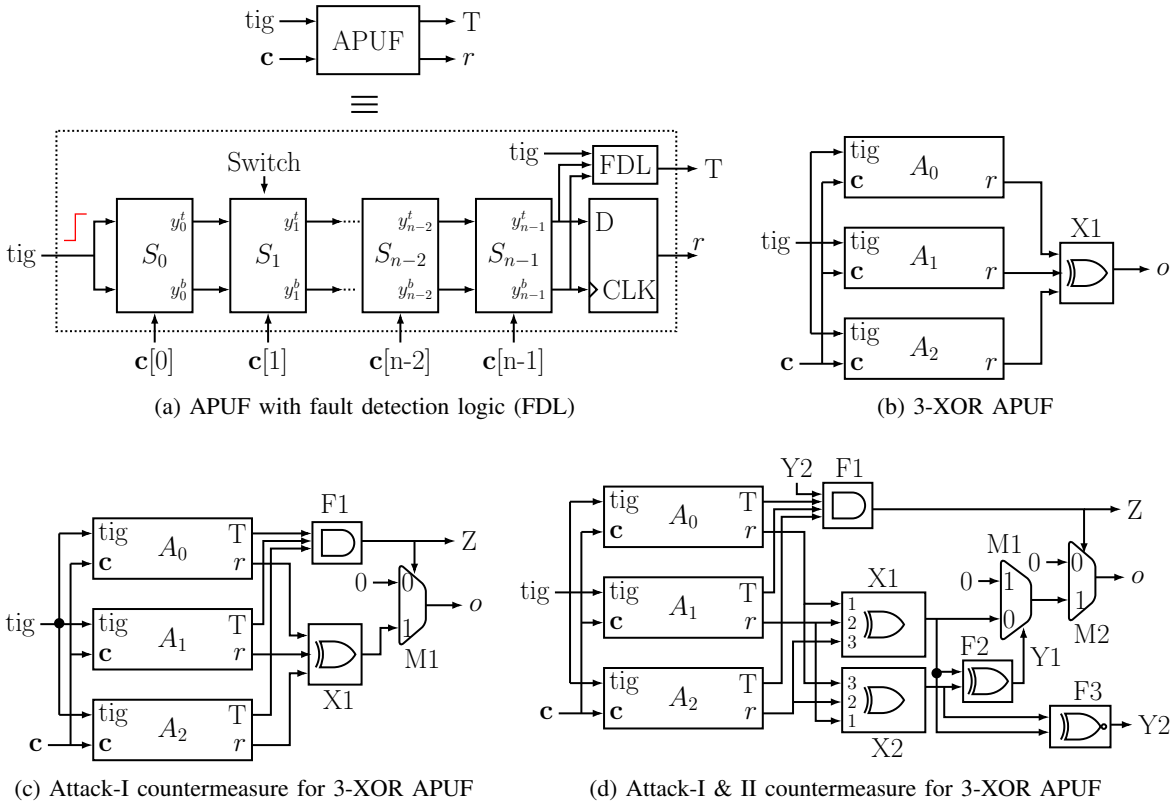
Note that both Attack-I and Attack-II share a common underlying philosophy - the adversary inject faults to change the configuration memory of one or more programmable logic cells, and forcing its functionality to differ from its normal functionality. We propose two countermeasure against both the aforementioned attacks on the XOR APUF. The basic approach is to include additional fault-checking circuit in the existing design of XOR APUF to make the fault injection detectable. The output (o) of XOR APUF is set to '0' on detection of the fault; otherwise it represents the normal XOR APUF output.

A. Countermeasure for Attack-I

The basic philosophy of Attack-I in [14] is the injection of a fault in the lower part of the PDL switch of stage S_{n-1} of an individual APUF instance, such that the CLK is set to '0'. Symmetrically, yet another possible attack scenario is to set the upper D signal to either 0 or 1, thus forcing the arbiter circuit to always sample the same value. We now propose a simple yet effective countermeasure to detect either of these modifications in the PUF circuit. Our strategy is to employ a 3-input fault detection logic (FDL) with output T, as shown in Fig. 4a. The corresponding functional behavior of the FDL is summarized in Table I. We state the following theorem:

TABLE I: Truth table of 3-input FDL in Fig. 4a

Inputs			Output
tig	y^t_{n-1}	y^b_{n-1}	T
0	0	0	1
1	1	1	1
x	0	1	0
x	1	0	0



(a) APUF with fault detection logic (FDL)

(b) 3-XOR APUF

(c) Attack-I countermeasure for 3-XOR APUF

(d) Attack-I & II countermeasure for 3-XOR APUF

Fig. 4: APUF and 3-XOR APUF with fault detection option.

Theorem 1. *The output of each individual APUF circuit is correct if and only if T=1.*

Proof. Observe that if there is no blockage (stuck-at-0 or stuck-at-1) on the path from S_0 to S_{n-1} (that is each of the intermediate switching stages are correctly functional), then the 'tig' signal (either 0 or 1) propagates to the outputs of S_{n-1}, namely y^t_{n-1} and y^b_{n-1}. Otherwise, the 'tig' signal cannot propagate to y^t_{n-1} and y^b_{n-1}. Thus, the circuit functions correctly if and only if:

$$\text{tig=0, } y^t_{n-1} = y^b_{n-1} = 0$$
$$\text{tig=1, } y^t_{n-1} = y^b_{n-1} = 1$$

with 'T=0' for the other conditions. This precisely corresponds to the given truth table for 'T' in Table I. This completes the proof of Theorem 1. □

Finally, the overall XOR APUF combines the 'T' values output by each component APUF instance, and a fault is detected if any of these instances output 'T = 0'. An illustration of our proposed countermeasure is illustrated in Figs. 4b and 4c that depict two 3-XOR APUF designs without FDL and with FDL, respectively. Attack-I is not detectable for the design in Fig. 4b, whereas it is detectable for the design in Fig. 4c provided that there is no modification in the AND gate (F1) which is used to combine T outputs of all component APUFs. If Z is not '1', then output o of XOR PUF will be '0' irrespective of the applied challenge; otherwise, the correct output of the XOR APUF is available at o.

B. Countermeasure for Attack-II

Fault Attack-II on the XOR APUF circuit proposed in [14] attacks the XOR gate (X1) used to combine outputs of the component APUF circuits. The LUT corresponding to the XOR function is modified to a buffer by precise laser injections. Our proposed countermeasure consists of detecting this modification by employing a redundant XOR gate (X2) with *different input connection patterns* from X1, as depicted in Fig. 4d. Note that to defeat this countermeasure, the adversary needs to now modify the LUTs of both the XOR gates such that they both behave as buffers for the same APUF instance, which is extremely difficult due to their different input patterns. In addition, to prevent attacks on either X1 or X2, another XOR gate (F2) is used to ensure that both X1 and X2 produce the same outputs. If output Y1 of F2, which is the control input of MUX M1, is '0', then output of M1 is the output of XOR APUF,

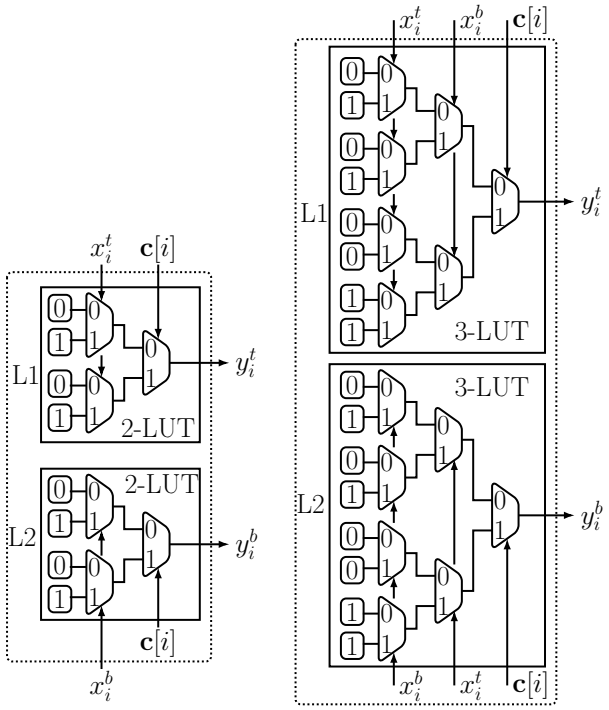
otherwise output of M1 is always '0' irrespective of the input challenge.

Finally, the essence of the XNOR gate F3 in Fig. 4d may be explained as follows. Observe that in the absence of F3, the adversary could modify X1 and F2 (X2 is untouched), such that X1 behaves as a buffer and F2 always outputs '0'. However, the presence of F3 would detect such an attack, since F3 is the complement gate of F2 and the output of F3 (Y2) is one of the inputs of the fault detection logic F1. Thus the only way the adversary could still attack the modified circuit is to alter both F2 and F3, which seems difficult due to their non-identical functionalities. The fault tolerance may be further fortified by keeping redundant copies of each individual gate, that are in turn multiplexed appropriately to generate the final output signal Z.

C. A Self-testable APUF Switch Design

We next introduce a generic countermeasure that modifies the APUF switch design in a manner that allows it to detect any changes in the expected switching behavior. Figures 5a and 5b depicts the LUT based realization of a PDL switch (cf. Fig. 2c) and classic switch (cf. Fig. 2b) of APUF, respectively. For clarity of presentation, we focus here on the necessary modifications to the PDL switch to make it self-testable. A similar approach may be employed for the self-testable classic switch design as well.

It is quite evident from Fig. 5a that the LUTs L1 and L2 have identical content. Hence, given values for inputs x_i^t, x_i^b and $\mathbf{c}[i]$, outputs y_i^t and y_i^b of L1 and L2, respectively, must be equal. Any change made to L1, that is not also made to L2, can be detected by an XNOR logic. However, although the XNOR ensures that L1 and L2 have the *same* content, it cannot possibly ensure that they have the *correct* content as per the depiction in Fig. 5a. This motivates us to present a new countermeasure, presented in Fig. 6 that allows each APUF switch to detect any alteration in its expected functionality automatically. Observe from Fig. 5a that the contents of the SRAM cells selected by x_i^t and x_i^b are the same as the values of x_i^t and x_i^b themselves, and these are precisely values that propagate to y_i^t and y_i^b, respectively. Hence, in our proposed self-testable design of APUF switches, a logic module G1 is required in addition to the XNOR gate (G2) to check the equality of x_i^t, x_i^b and y_i^t. A functional description of G1 is provided in Table II. If both G1 and G2 are '1', then the content of L1 and L2 are guaranteed to be correct. Otherwise, the circuit detects an unwanted modification, and outputs



(a) 2-LUT based PDL switch (b) 3-LUT based classic switch

Fig. 5: LUT based realization of APUF's ith switch.

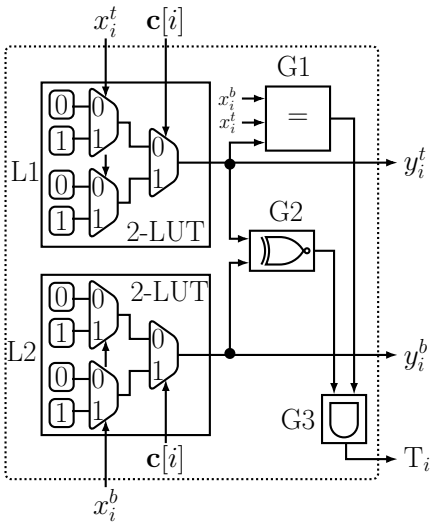
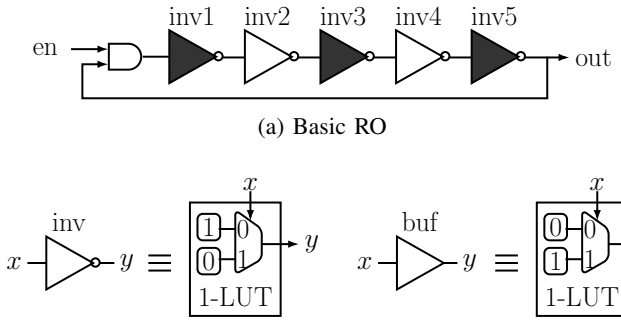


Fig. 6: Self-testable PDL switch using 2-LUTs.

'0'. This is achieved by the AND logic (G3). Note that the logic for each of the three functions G1, G2, G3 can be mapped to a single 4-LUT. Thus the self-testing APUF switch occupies just a singe LUT in Xilinx 7-series FPGAs.

TABLE II: Truth table of G1 in Fig. 6

Inputs x_i^t	Inputs x_i^b	Inputs y_i^t	Output T_i
0	0	0	1
1	1	1	1
x	0	1	0
x	1	0	0

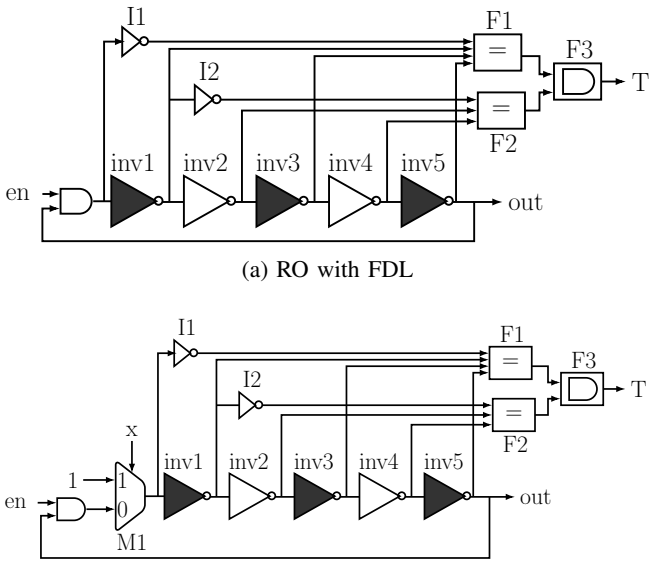


(a) Basic RO

(b) Inverter using 1-LUT

(c) Buffer using 1-LUT

Fig. 7: Realization of Ring oscillator using 1-LUT. Inverters with odd index are highlighted using black color. Functionality of F1 and F2 is similar as mention in Table II, and they output '1' if and only if all inputs are same.



(a) RO with FDL

(b) RO with FDL and an extra MUX

Fig. 8: Ring oscillator design with fault detection logic (FDL). Inverters with odd indices are highlighted in black.

IV. Countering Fault Attacks on ROPUF

A recently proposed fault attack on ROPUF [14] attempts to reduce the entropy of the generated key by converting one of a pair of ROs into a chain of even number of inverters by modifying the content of the LUT of the last inverter using a laser pulse injection. Consequently, the values of the counters corresponding to the fault free and faulty RO are significantly deviate from each other. In particular, the value of the counter corresponding to the fault free RO is always greater than that of the faulty RO, resulting in a biased distribution for the key generated by the ROPUF. We propose a countermeasure that detects the presence of this fault, and alerts the system using the corresponding key generated by the faulty ROPUF.

A feature of ROPUF, which is different from APUF, is that the functionality of the individual RO does not depend on the input challenge bits; the challenge is used only to select a pair of ROs. Unlike in APUF, the delay components (i.e. inverters) of an RO are connected in a loop, and it makes the modification detection difficult when the RO is being evaluated due to the presence of delays between the different stages. We first discuss a naïve fault detection scheme for ROPUF and point out the problems with this construction. We then modify it to develop a stronger countermeasure.

Figures 7b and 7c show the realization of an inverter and a buffer using 1-LUT. The following trivial facts can be observed from Figs. 7b and 7c:

1) The LUT content of inverter is complement of buffer LUT, and if one changes an inverter to a buffer (or a buffer to an inverter), then it can be easily detected by comparing their output while the same input is applied to both of them.
2) Outputs of two inverters are same when they have the same inputs.

In case of a RO, the input values of the inverters with odd indices (highlighted in black in Fig. 7a), are the same during each period of oscillation, and a similar property holds for inverters with even indices. Thus, a structural modification in a RO can be detected by checking the equality of the outputs of all the inverters with odd indices taken together, and symmetrically for all those with even indices. The RO design with these checkpoints is illustrated in Fig. 8a. The design is considered to operate free of any faults if the value of the T is '1' when output ('out') of RO changes its state (from $0 \rightarrow 1$ or $1 \rightarrow 0$) at the end of the oscillating period. Quite evidently, this circuit detects if any of the inverting outputs is converted to non-inverting (and vice versa), and hence adequately detects the fault attack in [14]. To bypass the countermeasure illustrated in Fig. 8a, the

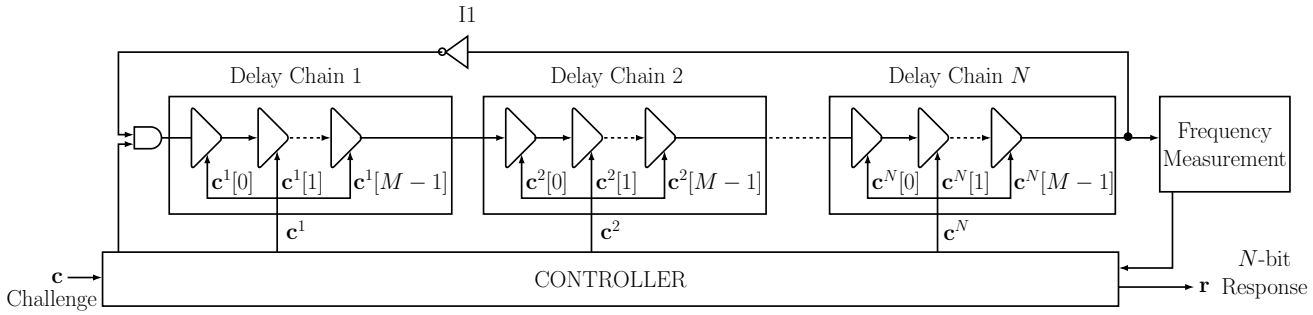


Fig. 9: Architectural overview of Loop PUF.

adversary must now inject an additional fault in one of the two detection logic entities F1 or F2, or in the final AND gate (F3). This makes the attack more difficult.

A problem in the above countermeasure technique is that the T signal needs to be monitored at the end of every oscillation, since the adversary may modify the circuit at any time before the RO evaluation is completed. This is expensive, especially in light of the fact that RO frequency for a 5-7 stage configuration is in MHz range. However, we note that it is logical to assume that while the adversary may change the circuit at any point during evaluation, with a very high probability she cannot revert the changes back before the evaluation is finished. Hence, it is sufficient to check the structural integrity of the RO before and after its evaluation. Moreover, in this case, the integrity check is on a RO *chain* (assuming no feedback path) with odd number of inverters, and *not* on a RO with feedback loop. This is fine since the feedback loop does not contain any logic component. It is however necessary to check the RO chain integrity for both 0 and 1 input to the first inverter (inv1) in Fig. 8a. This additional facility is incorporated in the countermeasure design using an additional MUX, as shown in Fig. 8b. If 'x=0' and 'en=0', then input of inverter 'inv1' is set to '0', whereas for 'x=1' and 'en=0/1', input of 'inv1' is fixed to '1'. These two assignments for signals 'en' and 'x' are required for fault detection, while for normal operation of RO following assignment is required:'x=0' and 'en=1'.

V. Countering Fault Attacks on Other Delay-PUFs

In this section, we explore possible fault attacks on two other varieties of delay PUFs: Loop PUF and Bistable Ring PUF. We also discuss countermeasure strategies to prevent such attacks.

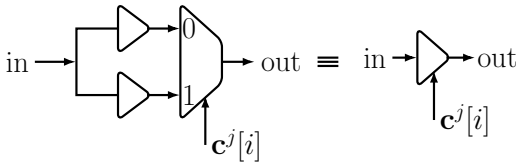


Fig. 10: Non-inverting delay element of Loop PUF with control bit $\mathbf{c}^j[i], i \in [0, M-1], j \in [1, N]$.

A. Loop PUF

The Loop PUF (LPUF) [21] is a type of delay PUF that comprises of a sequence of delay chains connected in a closed loop, as shown in Fig. 9. The input signal ('in' in Fig. 10) of each delay element of a delay chain follows one out of two possible paths to reach the output port ('out' in Fig. 10). The choice of path is decided by the control bit. The controller module of the LPUF instance generates N control words $(\mathbf{c}^1, \ldots, \mathbf{c}^N)$, each of M-bits, from an input challenge $\mathbf{c}$. For a given challenge, an N-bit response is generated using N different permutations of the control words $(\mathbf{c}^1, \ldots, \mathbf{c}^N)$. For more details, refer interested reader to [21].

We now discuss possible fault attack strategies on an FPGA implementation of a LPUF. Such attacks may be carried out using precise fault injection techniques, such as laser pulses, as in the attacks on the XOR APUF and the ROPUF presented in [14]. The objective of our proposed attacks is to modify the behavior of the Loop PUF in a manner that causes a authentication failure when this PUF instance is used for hardware authentication. One such attack strategy is for the adversary to modify the LUT of the inverter (I1) of LPUF (cf. Fig. 9) such that it becomes a non-inverting gate. This inhibits any oscillation in the input signal of the frequency measurement unit. An alternative fault injection strategy could be to modify a non-inverting delay element into an inverting delay element, provided that

the inverter I1 is unmodified. Thus in either attack, the fault injection stops the oscillation and renders the output of the PUF the same irrespective of the input challenge, leading to authentication failures. Injection of remnant faults on SRAM-based FPGA would allow the LPUF to revert back to its fault-free state after reconfiguration of the FPGA, but the service of the LPUF-based system would be disrupted for a certain time period before reconfiguration.

We next outline possible countermeasure strategies to prevent such attacks. As in the case of XOR APUF and ROPUF, an additional circuit is necessary to detect the any fault which results in a modified LPUF architecture. It can be observed from Fig. 9 that all non-inverting delay elements are homogeneous and have similar inputs during each oscillation period. Thus, a fault detection scheme similar to that in case of ROPUF in Fig. 8 to detect any modification in the subset of non-inverting elements. Modification in inverter I1 of LPUF can be detected by ensuring that there is no modification in non-inverting delay elements, and the output of the inverter is the complement of its input.

B. Bistable Ring PUF

A Bistable ring (BR) is a closed chain of an even number of inverters. The stable state of a BR is either '0' or '1', depending on the random process variations of inverters. The Bistable Ring PUF (BRPUF) exploits a BR to generate 1-bit response. To introduce the challenge-response mechanism, a compact representation of BRPUF was proposed in [22], where each stage of BRPUF comprises of a pair of NOR gates, a MUX and a DMUX. The main objective of using the DMUX is to prevent the output switching of both NOR gates for a given challenge bit. But in FPGA implementations, we can avoid the DMUX as we can realize the two NOR gates using a single LUT as shown in Fig. 12a. Evaluation of BRPUF for a given challenge begins by asserting 'reset' signal to '1', and it resets the output of all BRPUF's stages to '0'. This is the initial and unstable state of a BR. Then 'reset' is asserted to '0' and signal propagates through the loop to reach to a stable state.

Figure 12a makes it evident that a BRPUF instance is vulnerable to precise fault injections, such as laser shots discussed in [14]. If the attacker can modify the the inverting stage of BRPUF to a non-inverting stage as shown in Fig. 12b, then BR works as a RO and output will oscillate without reaching to stable state. Thus fault injection can make the BRPUF unreliable. We can employ the proposed fault detection scheme of

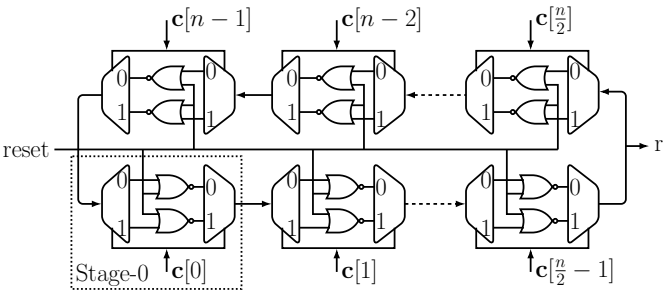
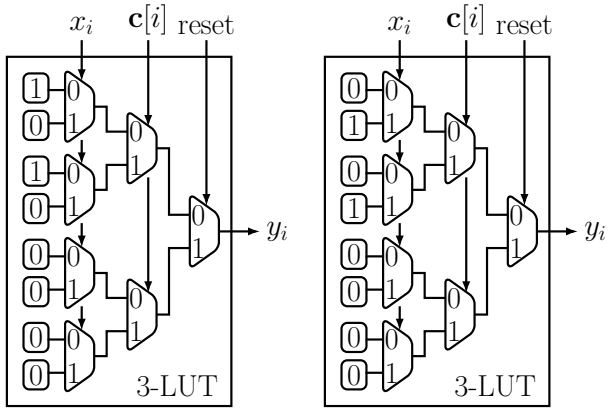


Fig. 11: Architectural overview of BRPUF with n-bit challenge $\mathbf{c}$ and 1-bit response r.



(a) Fault-free ith stage (b) Faulty ith stage

Fig. 12: A 3-LUT based realization of ith stage of a BRPUF.

ROPUF to detect any modification to the BRPUF loop as all the stage of BRPUF are homogeneous. In [14], authors also mentioned about the fault injection based modeling of XOR BRPUF. This attack can also be detected by employing a fault detection scheme proposed for APUF in Fig. 4d.

VI. Fault Recovery Schemes

We have thus far proposed a variety of fault detection mechanisms against fault attacks on both XOR APUF and RO PUF. However, besides fault detection, it is also important to recover the circuit to its original fault-free state. In this section, we explore possible efficient approaches for fault recovery with respect to APUF and XOR APUF circuits. Similar approaches may be developed for other delay based PUFs as well. Broadly, there are two main options for fault recovery:

1) **Rollback.** The objective here is to revert back to the original PUF instance with exactly the same timing behavior.

2) **Random-sliding.** The objective in this case is to replace the faulty PUF instance with a different PUF instance with different timing behavior.

We argue later that the second option is more advantageous than the first. Prior to that, we point out that the fault-assisted modeling attack on x-XOR APUF in [14] requires the adversary to restart the circuit x times to collect the CRPs of each of the x individual APUF instances. We take into account this assumption in our realizations of the *rollback* and *random-sliding* recovery schemes for APUFs and XOR APUFs.

1) Rollback: There are two ways to revert back to the original PUF instance after fault is detected in Xilinx FPGA: *partial reconfiguration* (PR) [30] and *dynamically configurable LUT*. In case of PR, we maintain bitstream of PR module (PRM) of the PUF design in some kinds of storage to be used later for partial reconfiguration without any external communication. In Xilinx FPGA, the internal configuration access port (ICAP) [31], [32] is used to allow user application to configure the application structure at run time. In our design we employ a *Fault Recovery Controller* (FRC) that is responsible for PUF reconfiguration upon detection of faults in the PUF design. The main drawback of this approach is that each time a fault is detected, the entire dynamic partition corresponding to PUF design needs to be re-configured, which is expensive.

Another way of rollback is the usage of configurable LUT (CFGLUT) in Xilinx FPGA. The CFGLUTs are widely used in the design of evolvable hardware (EH) [33], [34], as we can modify the content of LUT in a synchronous way to change the functional behavior of the circuit at run time. We use CFGLUT to design the switches of APUF, and it can help to recover modified switch due to fault. The Xilinx provides a HDL primitive CFGLUT5 [35, p. 106], for Virtex 5-7, Spartan 6, Kintex 7 and Artix 7 series of FPGAs, to instantiate CFGLUT in a design, and consumes a single 6-LUT in SLICEM (memory slice) [36, pp. 18-19] in FPGA. Figure 13 depicts the interface of CFGLUT5. This CFGLUT based rollback scheme for APUF is efficient than PR based scheme, since APUF switches are homogeneous and we can configure corresponding CFGLUT5s parallely with the same configuration bits in 32 clock cycles only.

2) Random-sliding: Figure 14a shows a realization of two-variable function (f) using 4-LUT, and only i0 and i1 inputs of 4-LUT are used for f and remaining inputs i2 and i3 are unused. For i0=0 and i1=0, only four SRAM cells are used out of 16 SRAM cells, as shown using a colored rectangle in Fig. 14a. It can be observed

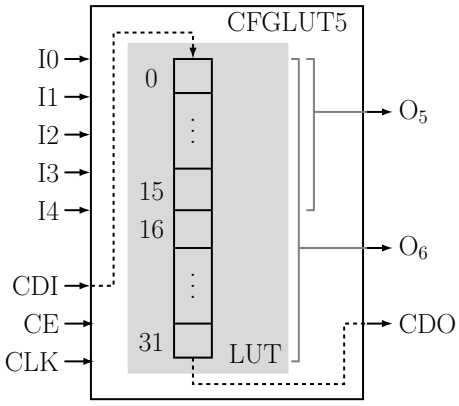
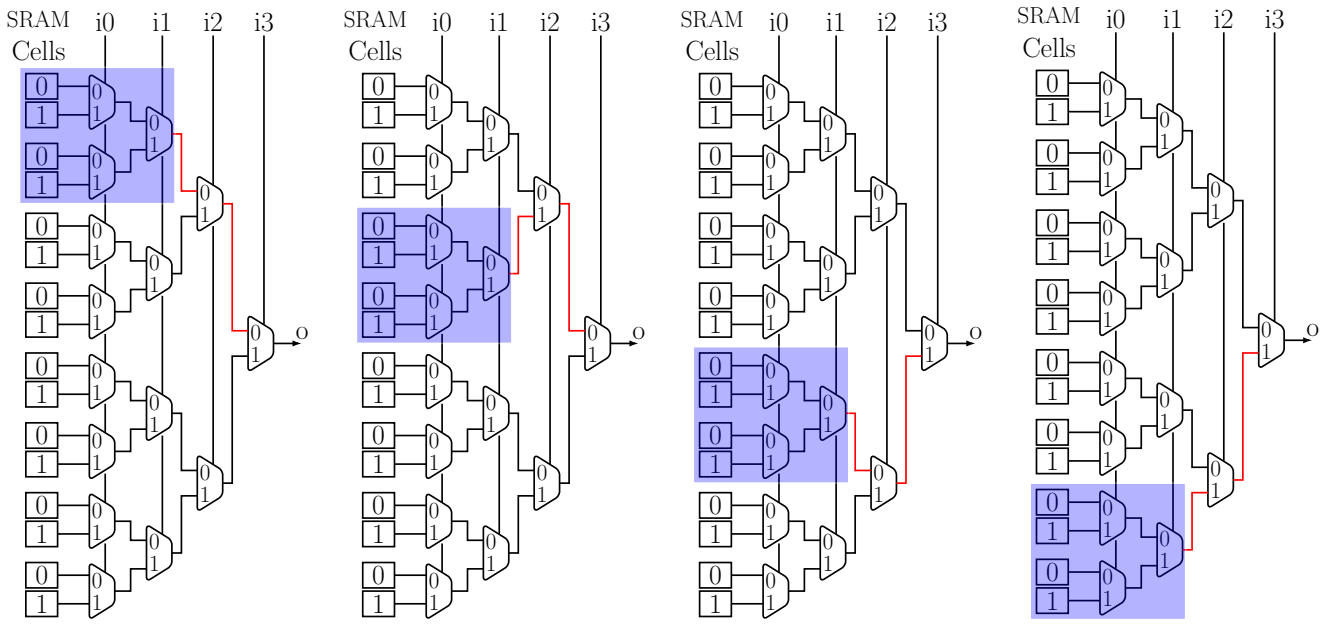


Fig. 13: CFGLUT5 HDL primitive is used as dynamically reconfigurable lookup table. Besides LUT inputs I0,...,I4 and outputs O_5 and O_6, it has a few reconfiguration interface with ports: *configuration data in* (CDI), *configuration data out* (CDO), *clock enable* (CE) and *configuration clock* (CLK). This primitive can be used to realize either a single 5-LUT with output O_6 or as two 4-LUTs using both the outputs O_5 and O_6.

from Fig. 14a that values of alternative blocks of four consecutive SRAM cells, starting from top, are the same, and this implies that even if one assigns some random values to unused inputs i2 and i3 of 4-LUT, LUT still works as the function f. Although there are no changes in the functional property (input-output mapping) of LUT due to random assignments to i2 and i3, the timing behavior of f can be different because input-to-output propagation delay in LUT are different for each input combination of LUT [25]. Figure 14 also depicts how different portions of LUT's SRAM cells are used to realize the function f for different assignments to i2 and i3 inputs of LUT. This feature can be used to make the function f fault-tolerant. For example, if we can detect that f is faulty for its present assignments to LUT inputs i2 and i3, we can employ a different assignments for i2 and i3 to exploit a different portion of LUT that seems to be non-faulty. We can try with all possible assignments for i2 and i3 until a fault-free realization of f is found. In the worst case, if the entire LUT is detected to be faulty, we need (partial)reconfiguration of the FPGA. We call this approach of fault-tolerant function realization as *Random Sliding*.

In Fig. 5a, we have shown the LUT based implementation of PDL switch that consists of two 2-LUTs, and thus if try to realize this PDL switch using 6-LUT of Xilinx Artix-7 FPGA, then we have four unused LUT inputs. If we provide the same assignments for unused inputs



(a) i2=0, i3=0 (b) i2=1, i3=0 (c) i2=0, i3=1 (d) i2= 1, i3=1

Fig. 14: A 2-variable Boolean function f(i0,i1) is implemented using 4-LUT. The circuit corresponding to f(i0,i1) shows four different timing behaviors for four different assignments for i2i3 $\in \{00, 10, 01, 11\}$.

of both the LUTs of PDL switch then we can have 2^4 switch configurations due the 16 possible assignments to four unused LUT inputs. Thus our random sliding notion can be used to make the PDL switch fault-tolerant. Subsequently, this random-slid PDL switch can be used to make the APUF design fault-tolerant in the presence of fault detection schemes as mentioned in Section III. It should be noted that one can assign different configuration bits for random sliding for each switch, but in our implementation we apply the same configuration bits to all unused inputs of LUTs employed in the design of PDL switches, and thus we have only 16 different configurations for a PDL APUF instance in FPGA.

This random-slid PDL APUF can be utilized to make the XOR APUF robust against fault-assisted modeling attack [14]. The XOR APUF attack in [14] needs to restart the PUF circuit x times to collect the CRPs of x APUF in x-XOR APUF. If the random-slid configuration bits are generated randomly every time circuit is restarted, x-XOR PUFs before and after the restart become different. It implies that even if the adversary can collect the CRPs by restarting the circuit x times, the collected CRPs of APUFs are not from the same XOR APUF, and thus, the modeling of XOR PUF by the modeling of individual APUFs becomes difficult. If we exploits this random-slid scheme with our fault detection schemes, the delay PUF designs can ensure the fault-tolerant property. The reader might think that how a verifier (in the authentication protocol) can authenticate the PUF in the presence of random sliding. The verifier maintains the database for all possible configurations of PUF and if the PUF is legitimate then PUF responses will match with the CRPs of any one the possible PUF configurations.

VII. FPGA Implementations and Overhead

To validate the correctness of the proposed countermeasures against fault attacks, we have implemented the original APUF and RO designs, along with their fault tolerant variants, on Xilinx Artix-7 FPGAs.

A. Implementation Details and Fault Simulation

Although Artix-7 FPGA has 6-LUT blocks, the CFGLUT5 HDL primitive allows realization of at most 5-input function. Implementing an APUF switch requires two 3-LUTs for classic switch and two 2-LUTs for PDL switch, which in turn means that there are two (or three) unused LUT inputs to be used for random sliding. So, CFGLUT5 primitive can be exploited to achieve both the dynamic configuration of switch as well as the random sliding.

As a fault recovery scheme, we used the combined approach of configurable LUT and random sliding. In the

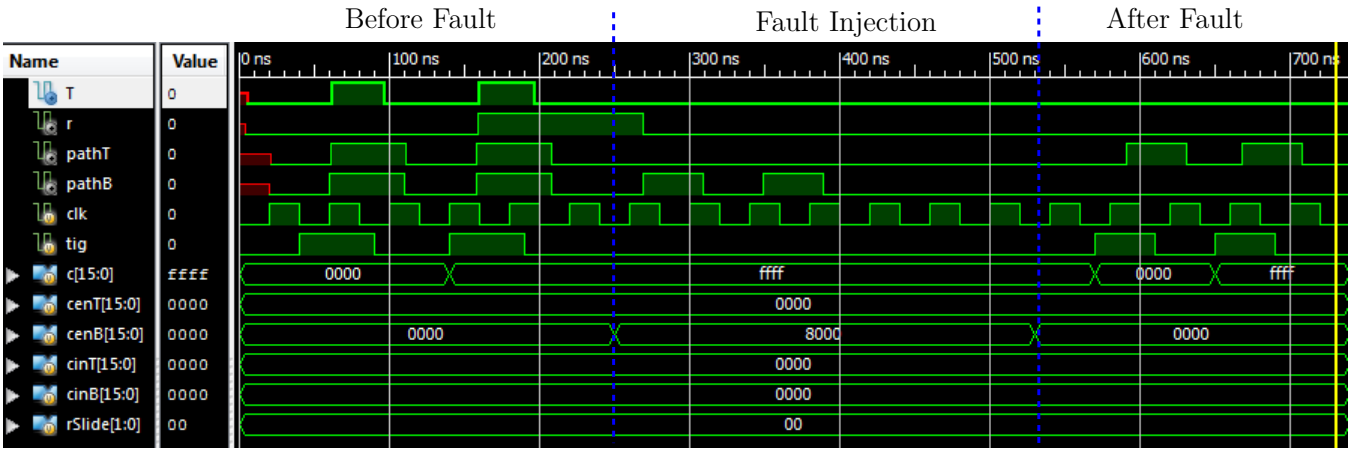


Fig. 15: Waveform of post place-and-route simulation 16-stage APUF (cf. Fig. 4a) with classic switches. It describes the behavior of APUF before and after the fault injection. This design is without self-testable switch. Here signal vector 'c' and signal 'r' is the challenge and corresponding response, respectively, and 'T' is the output of fault detection circuit in APUF.

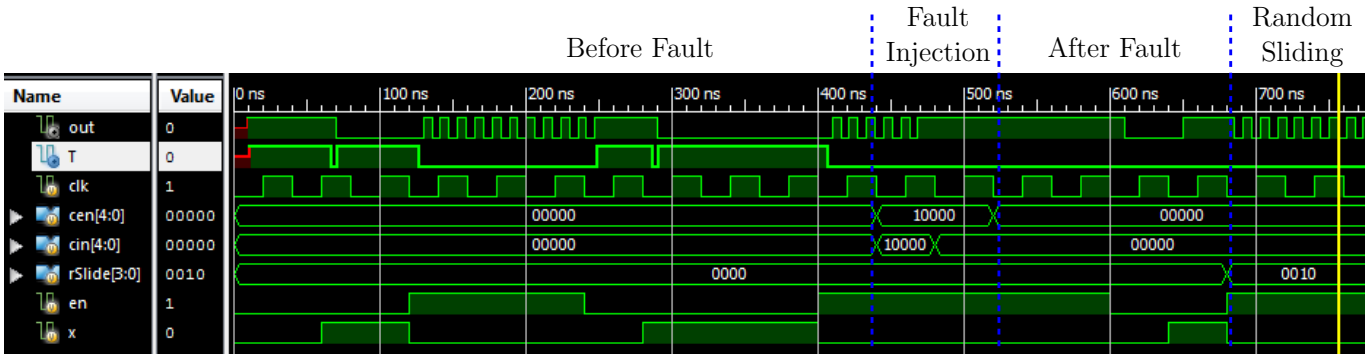


Fig. 16: Waveform of post place-and-route simulation of 5-stage RO with fault detection logic as shown in Fig. 8b. It describes the behavior of RO before and after the fault injection. It also shows how RO is recovered by using the random-sliding with 'rSlide' signal vector. 'T' is the output of fault detection circuit in RO.

presence of fault, first we apply random sliding to find a fault free random-slid instance, and if there are no such instance then we employ the dynamic reconfiguration of CFGLUT5 based switches. We have also followed a similar implementation approach for ROs in ROPUF. Each stage of a 5-stage RO is implemented using a CFGLUT5 instance.

In our experiment, we have simulated the fault injection process using the configurable LUT (CFGLUT5), that allows changes to the LUT's functionality at run time. In this case, we do not need any complex design for fault injections as our PUFs are implemented using CFGLUT5 primitives only. This fault simulation is more precise and controlled as we can reinitialize the LUT with arbitrary configuration bits. Number of clock cycles required for fault injection depends on the number of bits to be reconfigured and the positions of bits since reconfiguration happens serially.

B. Post Placed-and-Route Simulation Results

Figure 15 depicts the simulated behavior (post place-and-route) of APUF with classic switching stages before and after the fault injection. We need eight clock cycles (since there are eight SRAM cells as shown in Fig. 5b) to configure the bottom part of the last APUF switch, such that it outputs '0' regardless of input values. Signal vector 'cenB[15:0]' (for 16-stage APUF) in Fig. 15 is used to mention whether bottom part of switches to be reconfigured or not. 'cenB[i]=1' implies configuration to be performed for bottom part of ith switch. The vector 'cenB[15:0]' is used for fault injection. The

response of APUF is valid when 'tig=1' and 'T=1' hold. From Fig. 15 it can be observed that after fault injection, 'T' is not '1' even though trigger ('tig') is asserted to '1', and this an indication of fault in the APUF implementation. Note that signals 'pathT' and 'pathB' are top and bottom terminals of APUF's last switch. After fault injection, 'pathB' is fixed to '0'.

Figure 16 shows the simulated behavior (post place-and-route) of a 5-stage RO with fault detection logic (cf. Fig. 8b) before and after the fault injection. Each stage of RO is realized using a CFGLUT5 instance. As mentioned earlier, this design can be used to detect any fault in the RO before and after its evaluation. If 'T=1' happens at these checkpoints, then RO is considered as fault free and can be used for response generation; otherwise RO must be recovered from its faulty state. The vector 'cin[i]' in Fig. 16 is the configuration bit for the ith stage of RO, and 'cen[i]' is used as the configuration enable for CFGLUT5 of ith stage. We have injected faults in the last stage ($i = 4$) of the 5-stage RO to make it as a non-inverting buffer. It can be observed from the figure that after fault injection there is no oscillation at the output of RO even though 'en=1' and 'x=0'. To recover the RO from the faulty state, we have changed random-sliding configuration bits as 'rSlide[3:0]=0010', which achieves the desired fault recovery. One can also try dynamic reconfiguration of the corresponding CFGLUT5 instance.

C. Hardware Overhead

The resource overhead of our proposed countermeasures are summarized in Table III. In case of a single fault tolerant APUF, we need only one additional LUT to detect the fault. Since 3-XOR APUF consists of three fault tolerant APUFs, we need three LUTs to make the component APUFs fault tolerant, and an additional six LUTs to make the 3-XOR APUF fault tolerant, as shown in Fig. 4d. In case of 5-stage fault tolerant RO design in Fig. 8b, we need additional six LUTs to realize the logic of M1, I1, I2, F1, F2, F3, but we have mapped (I1, F1) in one LUT and (I2, F2) in another LUT; thus, it requires only four LUTs for the FDL logic of a single RO. Although we have used separate LUT for M1, but it can be mapped in the same LUT of AND gate (just before the first stage) of RO. Note that in Artix-7 FPGAs, each LUTs has dual output, and thus, further resource (number of LUTs) optimization is possible for FDL.

TABLE III: Summary of resource overhead of fault tolerant design in Artix-7 FPGA

Design	Original Design		FDL	
	LUT	FF	LUT	FF
APUF (Fig. 4a)	$2n$	1	1	0
3-XOR APUF (Fig. 4d)	$6n$ + 1	3	3+6	0
5-stage RO (Fig. 8b)	6	0	4	0

Note n implies number of stages in APUF. FDL means fault detection logic, and it is a pure combination circuit.

VIII. Conclusion

In this paper, we introduced novel and efficient fault tolerant implementations for delay PUFs that are robust against a variety of high precision fault attacks. We proposed fault tolerant implementations for Arbiter PUFs that use an additional fault detection logic (FDL) to detect the presence of malicious circuit modifications due to fault injections. We extended this technique to develop a fault tolerant implementation for an XOR APUF, with additional fortification for the final XOR gate to prevent an attacker from modeling the overall XOR APUF by collecting responses from the individual APUF instances. We also proposed a fault tolerant implementation for ROPUFs that thwarts attacks targeting one or more individual RO components to bias the output. Our proposed technique detects if any of the individual delay stages have been modified to prevent oscillations at the output of the RO. We next discussed possible fault attack scenarios for two more delay PUFs: the LOOP PUF and the Bistable Ring PUF, and proposed suitable circuit modifications to prevent such attacks. We introduced two fault recovery schemes - *Rollback* and *Random-sliding* and compared them in terms of their implementation overhead and efficiency. Finally, we validated the correctness of our proposed countermeasures by simulating precise fault injections on implementations of different delay PUFs on a Xilinx Artix-7 FPGA platform.

References

[1] D. Lim, "Extracting Secret Keys from Integrated Circuits," Master's thesis, MIT, USA, 2004.

[2] R. Maes, *Physically Unclonable Functions - Constructions, Properties and Applications*. Springer, 2013.

[3] A. Maiti, R. Nagesh, A. Reddy, and P. Schaumont, "Physical unclonable function and true random number generator: a compact and scalable implementation," in *Proc. of ACM Great Lakes Symposium on VLSI*, 2009, pp. 425–428.

[4] R. Maes, V. van der Leest, E. van der Sluis, and F. Willems, "Secure Key Generation from Biased PUFs," in *Proc. of 7th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, 2015, pp. 517–534.

[5] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Design Automation Conference*. New York, NY, USA: ACM Press, 2007, pp. 9–14.

[6] M. Majzoobi, M. Rostami, F. Koushanfar, D. S. Wallach, and S. Devadas, "Slender PUF Protocol: A Lightweight, Robust, and Secure Authentication by Substring Matching," in *Proc. of IEEE Symposium on Security and Privacy Workshops*, 2012, pp. 33–44.

[7] Ü. Koçabas, A. Peter, S. Katzenbeisser, and A. Sadeghi, "Converse PUF-Based Authentication," in *Trust and Trustworthy Computing - 5th International Conference, TRUST 2012, Vienna, Austria, June 13-15, 2012. Proceedings*, 2012, pp. 142–158.

[8] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling attacks on physical unclonable functions," in *Proc. of 17th ACM conference on Computer and communications security(CCS)*. New York, NY, USA: ACM, 2010, pp. 237–249.

[9] J. Delvaux and I. Verbauwhede, "Side Channel Modeling Attacks on 65nm Arbiter PUFs Exploiting CMOS Device Noise," in *IEEE 6th Int. Symposium on Hardware-Oriented Security and Trust* , 2013.

[10] ——, "Fault Injection Modeling Attacks on 65nm Arbiter and RO Sum PUFs via Environmental Changes," *IACR Cryptology ePrint Archive*, vol. 2013, p. 619, 2013.

[11] U. Rührmair, X. Xu, J. Sölter, A. Mahmoud, M. Majzoobi, F. Koushanfar, and W. P. Burleson, "Efficient Power and Timing Side Channels for Physical Unclonable Functions," in *Proc. of 16th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, 2014, pp. 476–492.

[12] F. Ganji, J. Krämer, J. Seifert, and S. Tajik, "Lattice Basis Reduction Attack against Physically Unclonable Functions," in *Proc. of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2015, pp. 1070–1080.

[13] S. Tajik, E. Dietz, S. Frohmann, J. Seifert, D. Nedospasov, C. Helfmeier, C. Boit, and H. Dittrich, "Physical Characterization of Arbiter PUFs," in *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, 2014, pp. 493–509.

[14] S. Tajik, H. Lohrke, F. Ganji, J. P. Seifert, and C. Boit, "Laser Fault Attack on Physically Unclonable Functions," in *12th Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 2015.

[15] D. Karaklajic, J. Schmidt, and I. Verbauwhede, "Hardware designer's guide to fault attacks," *IEEE Trans. VLSI Syst.*, vol. 21, no. 12, pp. 2295–2306, 2013.

[16] A. Maiti, V. Gunreddy, and P. Schaumont, "A Systematic Method to Evaluate and Compare the Performance of Physical Unclonable Functions," *IACR Cryptology ePrint Archive*, vol. 2011, p. 657, 2011.

[17] U. Rührmair, J. Sölter, F. Sehnke, X. Xu, A. Mahmoud, V. Stoyanova, G. Dror, J. Schmidhuber, W. Burleson, and S. Devadas, "PUF modeling attacks on simulated and silicon data," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 11, pp. 1876–1891, 2013.

[18] J. Tobisch and G. T. Becker, "On the Scaling of Machine Learning Attacks on PUFs with Application to Noise Bifurcation," in *Proc. of 11th International Workshop on Radio Frequency Identification: Security and Privacy Issues (RFIDsec)*, 2015, pp. 17–31.

[19] G. T. Becker, "The Gap Between Promise and Reality: On the Insecurity of XOR Arbiter PUFs," in *Proc. of 17th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, 2015.

[20] D. Merli, D. Schuster, F. Stumpf, and G. Sigl, "Semi-invasive EM Attack on FPGA RO PUFs and Countermeasures," in *Proceedings of the 6th Workshop on Embedded Systems Security (WESS 2011)*, 2011.

[21] Z. Cherif, J.-L. Danger, S. Guilley, and L. Bossuet, "An Easy-to-Design PUF Based on a Single Oscillator: The Loop PUF," in *Proc. of 15th Euromicro Conference on Digital System Design (DSD)*, 2012, pp. 156–162.

[22] Q. Chen, G. Csaba, P. Lugli, U. Schlichtmann, and U. R. ührmair, "The Bistable Ring PUF: A new architecture for strong Physical Unclonable Functions," in *Proc. of IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, june 2011, pp. 134 –141.

[23] T. Malkin, F. Standaert, and M. Yung, "A comparative cost/security analysis of fault attack countermeasures," in *Fault Diagnosis and Tolerance in Cryptography, Third International Workshop, FDTC 2006, Yokohama, Japan, October 10, 2006, Proceedings*, 2006, pp. 159–172.

[24] R. J. Anderson, *Security engineering - a guide to building dependable distributed systems (2. ed*. Wiley, 2008.

[25] M. Majzoobi, F. Koushanfar, and S. Devadas, "FPGA PUF using Programmable Delay Lines," in *IEEE International Workshop on Information Forensics and Security (WIFS)*, Dec 2010, pp. 1–6.

[26] R. Maes, P. Tuyls, and I. Verbauwhede, "Intrinsic PUFs from Flip-flops on Reconfigurable Devices," in *Proc. of 3rd Benelux Workshop on Information and System Security (WISSec)*, Eindhoven,NL, 2008, p. 17.

[27] J. Anderson, "A PUF design for secure FPGA-based embedded systems," in *15th Asia and South Pacific Design Automation Conference (ASP-DAC)*, jan. 2010, pp. 1 –6.

[28] S. Ghosh and K. Roy, "Parameter Variation Tolerance and Error Resiliency: New Design Paradigm for the Nanoscale Era," *Proceedings of the IEEE*, vol. 98, no. 10, pp. 1718–1751, 2010.

[29] D. P. Sahoo, P. H. Nguyen, R. S. Chakraborty, and D. Mukhopadhyay, "Architectural Bias: a Novel Statistical Metric to Evaluate Arbiter PUF Variants," *IACR Cryptology ePrint Archive*, vol. 2016, p. 57, 2016. [Online]. Available: http://eprint.iacr.org/2016/057

[30] Xilinx Partial Reconfiguration User Guide UG702 (v14.1) April 24, 2012. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ug702.pdf

[31] Xilinx 7 Series FPGAs Configuration User Guide UG470 (v1.10) June 24, 2015. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug470_7Series_Config.pdf

[32] V. Lai and O. Diessel, "ICAP-I: A reusable interface for the internal reconfiguration of Xilinx FPGAs," in *Proc. of International Conference on Field-Programmable Technology (FPT)*, Dec 2009, pp. 357–360.

[33] D. B. Bartolini, M. Carminati, F. Cancare, M. D. Santambrogio, and D. Sciuto, "HERA project's holistic evolutionary framework," in *Proc. of IEEE International Symposium on Parallel & Distributed Workshops*, 2013, pp. 231–238.

[34] F. Cancare, D. B. Bartolini, M. Carminati, D. Sciuto, and M. D. Santambrogio, "On the Evolution of Hardware Circuits via Reconfigurable Architectures," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 5, no. 4, p. 22, 2012.

[35] Xilinx 7 Series FPGA and Zynq-7000 All Programmable SoC

Libraries Guide for HDL Designs UG768 (v14.7) October 2, 2013. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/7series_hdl.pdf

[36] Xilinx 7 Series FPGAs Configurable Logic Block User Guide UG474 (v1.7) November 17, 2014. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf