# Efficient PUF-Based Key Generation in FPGAs using Per-Device Configuration

Mohammad A Usmani*, Shahrzad Keshavarz*, Eric Matthews†,
Lesley Shannon†, Russell Tessier* and Daniel E Holcomb*,
*Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA, USA
†School of Engineering Science, Simon Fraser University, Burnaby, BC, Canada
Email: holcomb@engin.umass.edu

**Abstract**—Reconfigurable systems often require secret keys to encrypt and decrypt data. Applications requiring high security commonly generate keys based on physical unclonable functions (PUFs), circuits that use random manufacturing variations to produce secret keys that are unique to each device. Implementing PUFs on FPGAs is usually difficult because the designer has limited control over layout, and each PUF system requires a large area overhead to correct errors in the PUF response bits. In this work, we extend the state of the art for FPGA-based weak PUFs using a novel methodology of per-device configuration and a new PUF variant derived from the popular FPGA-specific Anderson PUF. The PUF is evaluated using Xilinx XC7Z020 programmable SoC chips from the Virtex-7 family on Zynq ZedBoard platforms. The design we propose has several advantages over existing work including the Anderson PUF on which it is based. Our design is tunable to minimize the response bias and can be implemented using the common SLICEL components on Xilinx FPGAs. Moreover, the proposed PUF design enables an efficient per-device configuration that reduces bit-error-rate by over 10× at room temperature, and improves response stability by over 2× across all temperatures. We demonstrate that the proposed per-device PUF configuration step leads to roughly 2× savings in area resources for PUFs and error correction as used in key generation.

## 1 Introduction

FPGAs are used for an increasingly large number of applications that require security. Due to their volatile nature, SRAM-based FPGAs require security at multiple levels. Bitstream encryption is often used to protect the configuration bits that define application implementation. Additionally, secure encrypt/decrypt cores are often implemented as part of a user's design to allow for the confidential processing of application data. These cores require secret keys that are often customized on a per-device basis. Securing these keys can be problematic. Common SRAM-based FPGAs do not have on-chip non-volatile or battery-backed key storage that a user can access, but if the keys are stored off-chip then they are susceptible to the many attacks that have been demonstrated against bitstream encryption [1].

PUFs represent a device-tied method of generating secret keys on-chip without reliance on secured non-volatile memory or battery-backed storage. PUF-based keys are uniquely tied to each device and are not directly compromised by attacks that are able to recover decrypted bitstreams. These characteristics make PUFs well-suited to key generation for FPGA-based applications. For example, PUF-based keys are currently available in reconfigurable devices including Microsemi IGLOO2 [2] and Intel Stratix 10 [3] FPGAs, but these keys are generated by specialized blocks and not from circuitry created from the user-accessible FPGA fabric.

A PUF's response should solely rely on the inherent process variation of its components. Therefore, most PUF designs rely on differential circuits in which two paths are designed with identical logic and matched routing, so that the difference only comes from process variation. This level of matching can be easily done in ASICs because the designer has the freedom to easily control the layout of the PUF. However, FPGA designers are limited in this freedom, and must work within the constraints of the unmovable look-up tables and routing tracks in the FPGA fabric. Based on the design, different sections of logic and routing resources in the FPGA can be used, leading to nonidentical paths. Therefore, different approaches must be considered for FPGA PUF design that consider the available resources and their abundancy. In this work, we build our implementation on the FPGA-specific Anderson PUF [4]. The specific contributions of this work are as follows:

- We implement a new PUF variant that overcomes limitations of the original Anderson's PUF design, and quantify its uniqueness and reliability on a more advanced Xilinx architecture (Virtex 7).
- We demonstrate tunable PUF properties to adapt its performance to produce unbiased responses. We use these properties to tune the PUF for our target architecture.
- We demonstrate a methodology for efficient per-device PUF configuration that reduces the PUF resources required for key generation by approximately 2× relative to the same PUF without per-device configuration.
- We evaluate the temperature stability of PUF responses and show experimentally that per-device PUF configuration reduces average bit error rates by more than 2× across a range of temperatures.

The remainder of this paper is structured as follows. Background and prior work related to our approach are presented in Section 2. Our evaluation of PUF performance impacted by FPGA process variation is reviewed in Section 3. Section 4 examines the per-device selection of PUFs. The impact of enhanced reliability on error correction is explored

- *The two first authors contributed equally to this work.*

*Manuscript received XXX; revised YYY.*

in Section 5 and Section 6 concludes the paper and offers directions for future work.

## 2 Background and Related Work

Many applications in communications, vision, networking, and consumer products require the data confidentiality offered by FPGA-based encryption. For example, Wu and Huang [5] document the use of an FPGA as an encryption engine for a wireless communication system. An FPGA-based version of the advanced encryption standard (AES) is used in conjunction with one or more disks to provide secure data storage [6]. A comprehensive approach to protecting the use of intellectual property cores in FPGAs using public/private keys pairs is described in Kumar *et al.* [7]. Keys created by a PUF are used to initiate IP core operation and periodically authenticate its use [8]. In Hu *et al.* [9], FPGA-based keys are used to validate the downloading of network router monitors implemented in FPGAs. The security of the download process is instrumental in maintaining proper network operation. These examples represent a small set of the diverse and growing set of applications which use FPGAs to provide confidentiality.

### 2.1 Process Variations in FPGAs

Process variations can impact the performance of all integrated circuits. Within the FPGA domain, a study of systematic and intrinsic sources of delay variability was performed by Sedcole and Cheung [10]. Reconfigurable hardware offers the opportunity for per-device logic placement as a mechanism to address process variations. Cheng *et al.* [11] used a simulation study to show possible performance improvements from a two-step process of extracting chip-specific delay information and then performing chip-specific placement. Similar work by Bsoul *et al.* [12] optimized placement on Xilinx Virtex II FPGAs in consideration of both process variation and aging. In contrast to these previous works on per-device logic placement, we will show in this work that per-device PUF placement has the potential for much greater gains on account of being highly sensitive to variations.

### 2.2 Physical Unclonable Functions

Physically unclonable functions (PUFs) are circuits that leverage manufacturing variations to produce instance-specific output values. Creating outputs that depend on process variations often requires delay matching or other types of symmetry that can be difficult to achieve in an FPGA. The output values produced by each PUF instance are persistent over time but can be influenced by noise. PUFs are sometimes classified as *strong* PUFs or *weak* PUFs depending on whether or not they accept input challenges.

Strong PUFs use process variations to map a large space of input challenges to corresponding output values. Strong PUFs are exemplified by the well-known Arbiter PUF [13] that maps input challenges to responses according to unique delay variations of each instance. The Arbiter PUF requires carefully matched wiring delays, and this has been achieved on Xilinx Virtex 5 FPGAs by selecting LUT input combinations to tune and match propagation delays [14]. Strong PUFs are subject to modeling attacks, in which an adversary uses a set of known input-output examples to train a model that predicts a PUF's outputs for previously-unseen inputs.

Unlike strong PUFs, weak PUFs do not have a large input space, and each PUF instance produces a single bit of variation-dependent output. Weak PUFs can, therefore, be considered as device-tied constants, and are not subject to modeling attacks. Weak PUFs have been widely implemented on FPGAs. The power-up state of SRAM can be used to save secret data [15] or as a unique chip identifier [16] but are no longer feasible on common FPGAs because SRAM blocks are now initialized to default states at power-up. Attempts to circumvent SRAM initialization have met with little success. Sander *et al.* [17] use JTAG to read out unique values from unused configuration memory regions of Virtex 5 FPGAs, but there is no evidence that these values result from process variation. Wild and Günesyu [18] manipulate power gating and partial reconfiguration to extract identifying values from block RAM on specific revisions of a Xilinx Zynq 7020 design, but they note that the same approach fails to work on later revisions of the same design. If uninitialized SRAM power-up state were accessible on an FPGA, then one can consider the possibility of using directed aging to enhance the reliability of the PUF at the circuit level [19], [20], [21]. The initial states of flip-flops in FPGAs depend on process variations [22], but are extremely biased and produce low quality outputs. Butterfly PUFs [7] use contention on cross-coupled latches to generate output bits, but later work questions the uniqueness of Butterfly PUF outputs [23]. Xu et al. [24] proposed a PUF design that performs autonomous majority voting without orchestration by a clock. Their proposed structure can also be implemented on FPGAs, though it is not optimized for this purpose in terms of resource usage.

Our work in this paper uses an FPGA-specific weak PUF that is derived from an existing design, the Anderson PUF, which uses LUTs and hardened carry chains found in Xilinx FPGAs [4].

### 2.3 Anderson PUF

The basic circuit of the Anderson PUF [4] is shown in Fig. 1. This PUF does not require the use of hard macros and is designed purely at the register-transfer level with the required constraints described in VHDL. The PUF uses two adjacent SLICEM blocks of a Xilinx FPGA. In SLICEM blocks, each LUT can be used as either a 6-input combinational logic function, or as a 16-bit shift register, and the Anderson PUF uses the shift register functionality. As shown in Fig. 1, the outputs of `Shift reg 1` and `Shift reg 2` are connected to the select lines of multiplexers that are separated by a chain of multiplexers that have their select inputs stuck at 1. In normal FPGA usage, the multiplexer chain is used as a carry chain. In the PUF setting, these intermediate multiplexers act as a delay line, such that the top multiplexer has its select input connected directly to `Shift reg 2` and has its 1-selected input connected to a delayed version of the output from `Shift reg 1`. The two shift registers are loaded with complemented alternating 0-1 values (e.g. `Shift reg 2` is loaded with `0x5555` and `Shift reg 1` is loaded with `0xAAAA`). Although the shift register outputs are never both 1 at the same time, due to the propagation delay of the carry chain,

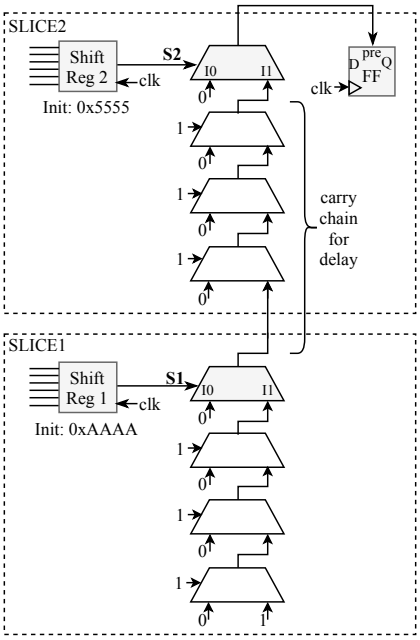
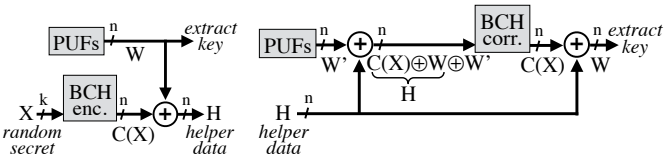



Fig. 1: Circuit diagram of Anderson PUF

there exists a window of time in which the top multiplexer has a 1-value on both its select signal and its 1-selected input. This creates a transient pulse on the multiplexer output with a pulse duration that is proportional to the propagation delay of the multiplexer carry chain. The multiplexer output that carries the pulse is attached to the asynchronous preset of a flip-flop. If the pulse has sufficient duration and amplitude when reaching the preset input, then the flip-flop value will be set to 1; otherwise, the flip-flop will remain at 0. Thus, the flip-flop value depends on the delay and attenuation of the multiplexer stages used as delay lines, which causes the circuit to act as a PUF. The outputs of Anderson's PUF have been shown in previous work to be unique to each PUF instance. Because this implementation of the Anderson PUF requires shift registers it can only be implemented in SLICEM cells of the FPGA chip.

### 2.4 Error Correction for PUF-based Keys

Cryptographic keys must be generated repeatably over time. The outputs of PUFs are noisy and thus cannot be used directly as key bits. Fuzzy extractors [25], [26] are cryptographic primitives designed for deriving reliable key values from noisy biometric data and are widely used with PUFs. When a PUF is first enrolled with a fuzzy extractor, a key is derived from the PUF and helper data is produced to facilitate generation of the same key at a later time. When the key is later generated in the field, the helper data and the PUF are used together to regenerate the enrolled PUF value so that the same key can again be derived from it. The generated key matches the enrolled key as long as the PUF values used at enrollment and generation are within some configurable Hamming distance of each other. The reliability of the key stems from the fuzzy extractor's use of error correcting codes and the security of the key relies on an adversary's limited ability to guess the PUF outputs, which are kept secret.




(a) one-time key enrollment (b) key generation in the field

Fig. 2: During key enrollment, a random secret is used to derive helper data from PUF values. The helper data is later used with the PUF to regenerate the initial PUF response, and from that to extract a key. The generated key will match the enrolled key if the Hamming distance between the PUF outputs ($W$ and $W'$) is within the error correction capacity of the BCH code used. Enrollment is performed off-line while key regeneration occurs on the FPGA at power-up.

We use a code-offset fuzzy extractor construction with BCH codes for error correction in this work. BCH codes are a family of error correcting codes where each code is described by a tuple $(n, k, t)$; parameter $n$ is the block size or equivalently the size in bits of each codeword, parameter $k$ is the number of information bits in each codeword and parameter $t$ is the number of correctable errors in each codeword. A simple code-offset construction using BCH codes is given in Fig. 2; in this example, a random $k$-bit secret value is used to enroll an $n$-bit PUF value. Multiple blocks can be enrolled in the same way, and when this is done, no values are reused across the different blocks.

**Key Enrollment:** During key enrollment (Fig. 2a), each $k$-bit secret $X$ is chosen at random and encoded to an $n$-bit BCH codeword $C(X)$; error correction ensures that $X$ can be decoded from any $n$-bit string that is within Hamming distance $t$ of codeword $C(X)$. The codeword is offset by XOR with $n$-bit PUF output $W$ and the result is stored as helper data $H$. Additionally, a key is extracted from PUF value $W$ using an entropy extractor such as a universal hash function [27] or a cryptographic hash function assumed to behave as a random oracle [28].

**Key Generation:** During key generation (Fig. 2b), the PUF produces value $W'$ that may differ slightly from the value $W$ used during enrollment. PUF output $W'$ is offset by helper data $H$ to produce a value that is the original codeword $C(X)$ offset by $W \oplus W'$. The corrupted codeword can be corrected to regenerate $C(X)$ as long as $C(X) \oplus W \oplus W'$ is within Hamming distance $t$ of $C(X)$. Stated differently, $C(X)$ can be corrected if the difference between the PUF values used at enrollment and generation does not exceed the maximum number of errors that are correctable by the BCH code. The corrected $C(X)$ is offset by the helper data $H$ to regenerate value $W$ that was produced by the PUF during enrollment. Then, the same key extraction procedure is used to generate the key from $W$.

**Security:** The security of error correction in PUF-based key generation is an active field of research, and we give here only a brief introduction to the topic. The PUF outputs and the key derived from them are never revealed in cleartext, but there are a number of factors that can diminish key entropy. Firstly, any bias or correlations in the PUF will reduce the

min-entropy of the source, and the key can of course never have more entropy than the source from which it is derived. Second, the BCH code leaks $n-k$ bits of information through the helper data. Finally, the extractor used to derive the key from value $W$ cannot extract all available entropy from $W$ into the key. We point interested readers to a number of works that consider in more depth the security details of key generation [29], [30], [28], [31].

### 2.5 Relationship to Previous Analysis

This manuscript builds on an earlier version of the work [32] that examined device-specific placement of the Anderson PUF. The work at hand provides a significant advancement over the previous work by developing a new PUF architecture and a new PUF selection approach. We also evaluate the performance of our new PUF across a range of temperatures. In this new study, nine identical XC7020 FPGAs are used to evaluate the PUFs, rather than the four devices used in the earlier work.

## 3 PUF Design and Characterization

In this section, we redesign the Anderson PUF to improve on its limitations. Our modifications allow for the creation of PUFs using lookup table (LUT) based SLICEL elements that cannot be used as shift registers. These elements are more plentiful in FPGAs than the SLICEM elements that the Anderson design uses. The modification provides the capability for fine-grained PUF tuning to adjust a PUF's Hamming weight, thereby optimizing its uniqueness.

### 3.1 PUF Implementation on a Virtex 7 Architecture

Families of Xilinx SRAM-based FPGAs since the Virtex 4 have two different types of logic clusters (SLICEs) that contain LUT-based logic elements. As described in Sec. 2.3, the LUTs in the SLICEM elements required in the Anderson PUF are implemented as shift registers to ensure synchronous switching of the multiplexer select lines at the top and bottom of a carry chain. This SLICEM restriction limits the placement of PUFs on the FPGA and consumes limited resources that may be required for other design purposes. In the Xilinx Zynq-7020 devices used for our work, two-thirds of all SLICEs are SLICEL elements (8,950 SLICEL vs. 4,350 SLICEM).

Our revised Anderson PUF implementation is shown in Fig. 3a. It consists of two Xilinx SLICEs that can be either SLICEMs or SLICELs, although we focus on SLICELs. In contrast to the Anderson PUF, our design replaces each SLICEM shift register output with a synchronous toggling signal created from a flip-flop (FF) and a LUT configured as an inverter.

The timing operation of the design is shown in Fig. 3b. Flip-flop `FF1` in `SLICE2` is initialized to a logic-1 and the corresponding flip-flop `FF1` in `SLICE1` is initialized to a logic-0. Since the LUTs are configured as inverters, during evaluation these flip flops will toggle their value at each rising clock edge. The race condition that occurs after every second rising clock edge determines the width of the pulse that is generated. The racing signals are as follows:

- On the clock edge, a falling transition propagates in `SLICE2` from signal `Q2` at the output of `FF1` through the inverter-configured LUT to a rising transition on signal `S2` at the select input of multiplexer `M1`. The rising transition arrives when the 1-selected input of the multiplexer holds a logic-1 value, and causes the output of `M1` to rise.
- On the same clock edge that triggered the above described sequence, in `SLICE1` a rising transition on signal `Q1` at the output of `FF1` propagates through the inverter-configured LUT to a falling transition on signal `S1` at the select input of multiplexer `M1`. A logic-0 then propagates upward through the carry chain. When this falling transition reaches the 1-selected input of `M1` in `SLICE2` it causes the output of the multiplexer to fall, terminating the pulse.

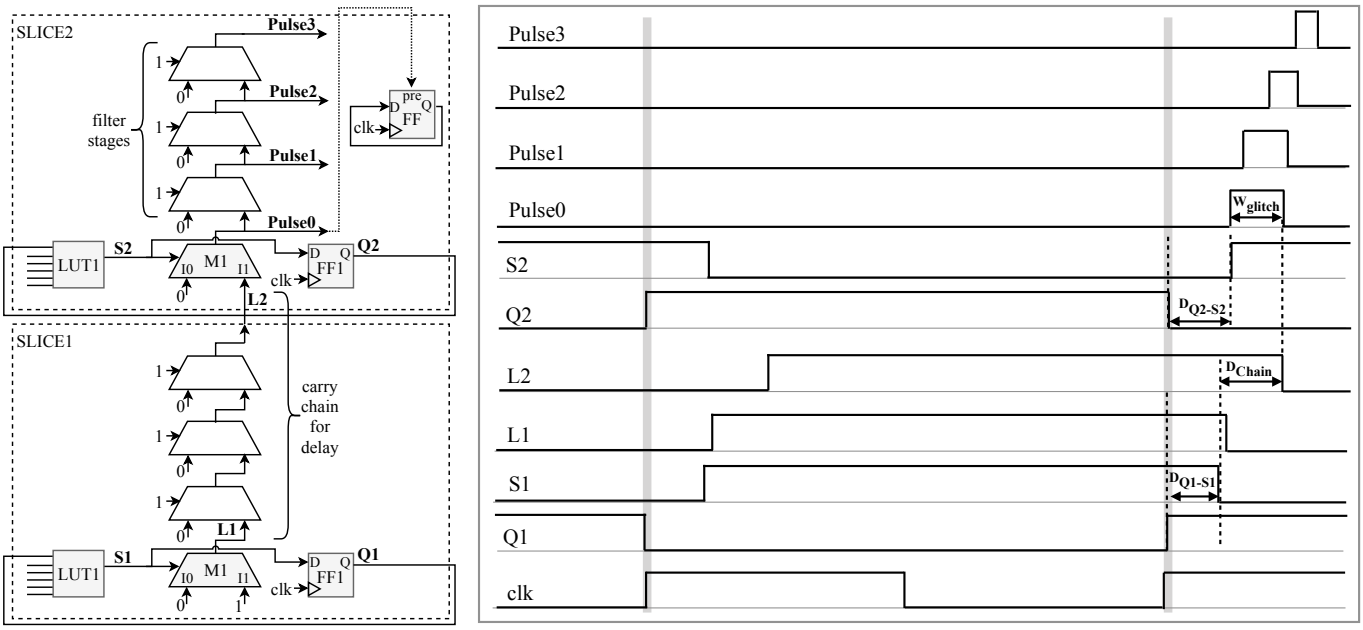
According to the race condition described above, the duration of the pulse on signal `Pulse0` can be described by Eq. 1. Here, $W_{glitch}$ is the duration of the pulse, $D_{chain}$ is the delay through the length of the carry chain from falling transition on `S1` to falling transition on `Pulse0`, $D_{Q1-S1}$ is the delay from `Q1` to signal `S1` in `SLICE1`, and $D_{Q2-S2}$ is the delay from `Q2` to signal `S2` in `SLICE2`.

$$W_{glitch} = D_{chain} + D_{Q1-S1} - D_{Q2-S2} \quad (1)$$

As shown in Fig. 3a, the pulse on signal `Pulse0` can be propagated through three additional multiplexers in `SLICE2` giving designers a choice of which pulse signal should be attached to the asynchronous preset signal of the flip-flop that will capture the PUF response. The different pulse signals can be viewed as different signal taps of the generated pulse. In the figure, `Pulse0` is the signal attached to the flip-flop, but in the next subsection we show how the additional multiplexers can act as filters that attenuate the pulse. The selection of different taps for the pulse signal allows for the tuning of the Hamming weight of the PUF responses. Note that the preset signal is attached to all flip-flops within a slice. Consequently, `FF1` in `SLICE2` may also be preset by the generated pulse during a PUF evaluation. This can provide a second path through which the PUF output can be set high, but does not harm the PUF evaluation.

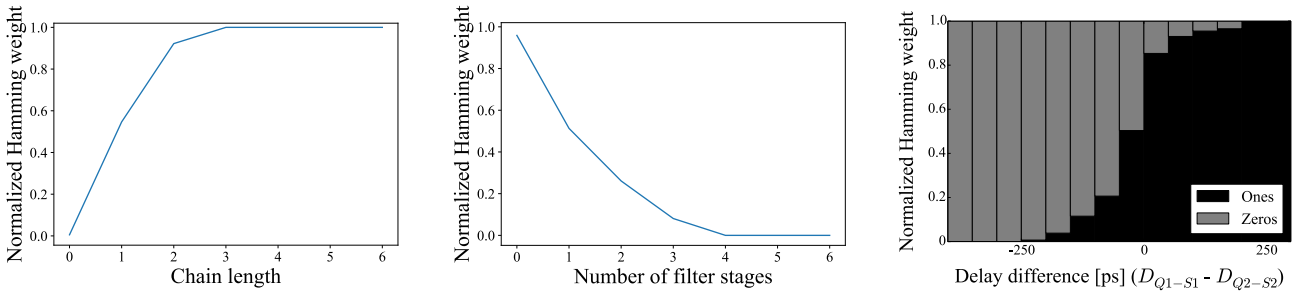
### 3.2 PUF Hamming weight tuning

A PUF design in which the responses of all instances are biased toward the same value will tend to be less unique than one in which the responses are unbiased. An average fractional Hamming weight close to 0.5 (50%) is an indicator that PUF responses include a balanced mixture of 0 and 1 bits and are not systematically biased. In the Anderson PUF and our variation thereof, the average Hamming weight of the PUF response bits depends on the expected width of the glitch that arrives at the asynchronous preset input of the capturing flip-flop. A short glitch is less prone to setting the flip-flop value high, while a long glitch increases the chance that the flip-flop will be triggered to store a response value of 1. Ideally, the average pulse width will be on the cusp of the two response values, such that the response of each PUF instance will be caused by its process variations and their effect on the pulses of that PUF instance. We explore three different knobs that can be used to control the average Hamming weight of the PUF response bits. The first of these knobs has been proposed by Anderson in his original work




(a) Implementation (b) Timing

Fig. 3: a) Enhanced implementation of the Anderson PUF [4] on a Virtex 7 architecture. The two LUTs are separated vertically by three multiplexer stages. Each LUT feeds a flip-flop to generate a toggle signal for the select line of the attached multiplexer. b) The timing waveforms govern the operation of glitch generation and glitch filtering.




(a) Effect of carry chain length (b) Effect of filter stages (c) Effect of route skew

Fig. 4: Figure shows three different design changes that are used as tuning knobs to increase or decrease the width of the generated glitch and thereby increase or decrease the response bias; a) elongating the carry chain widens the glitch; b) adding multiplexer filters attenuates and narrows the glitch; c) setting the lower feedback path to have a longer delay widens the glitch. Results were generated using a Xilinx XC7Z020 device.

implementing this style of PUF [4] in SLICEM blocks; the second and third knobs are novel and proposed for the first time in this work.

**Tuning by varying the chain length:** Changing the number of delay stages in the design can be used to adjust the term $D_{chain}$ in Eq. 1, which describes the propagation time for a signal to propagate through the carry chain. As implied by Eq. 1, if the delay values $D_{Q1-S1}$ and $D_{Q2-S2}$ are held constant, changing this term has a direct impact on the glitch width, and therefore can be used to tune the Hamming weight of a design. The effect of changing the length is shown in Fig. 4a. A longer carry chain increases the response Hamming weight. Note that, when two or fewer multiplexers are used in the carry chain, the entire PUF can fit within one slice; when between 3 and 6 multiplexers are used for the carry chain, the PUF requires two slices. For carry chains not having length 3, we are unable to route the PUF in a way that holds $D_{Q1-S1}$ and $D_{Q2-S2}$ constant. Therefore, to study the impact of the carry chain delay in these cases, we use SLICEM cells in the shift-register configuration from the original Anderson's PUF. A carry chain of length 3 allows the same relative multiplexer, lookup table, and flip-flops of two different slices to be used at the two ends of the carry chain, making it easier to match the routing and propagation delays of $D_{Q1-S1}$ and $D_{Q2-S2}$ in the adjacent SLICEL cells.

**Tuning by addition of filter stages:** The second tuning parameter is the number of filtering stages used between the pulse generation and the preset input of the flip-flop that generates the PUF response. As mentioned in the previous subsection, a generated pulse can be propagated through additional stages to create a number of different signal taps that can be attached to the flip-flop preset to capture the

response. To optimize the PUF response Hamming weight, the extra stages at the top of the selected carry chain can be used to attenuate the pulse and tune the expected width of the signal that will arrive at the preset input of the flip-flop. Figure 4b shows the effect of adding extra multiplexer stages as filters when the carry chain is comprised of three multiplexers, as shown in Fig. 3a. Note that only three filters can be fit within a two slice design, so the variants using more than three filters require three slices. From the result, we can see that adding one filter stage when the carry chain contains three multiplexers gives a Hamming weight close to the ideal value of 50%. Regardless of the chain length, it was found that adding more filter stages reduces the Hamming weight.

**Tuning by adjusting path delays:** The third parameter used to tune the Hamming weight of the PUF response involves varying the delay differences of the FF-LUT paths that route signals from flip-flops back to the carry chain. These FF-LUT delays are denoted as $D_{Q1-S1}$ and $D_{Q2-S2}$ in Eq. 1 and are annotated in Fig. 3b. The two delays affect the width of the produced glitch (shown as $W_{glitch}$). Any increase in delay $D_{Q1-S1}$ relative to $D_{Q2-S2}$ will delay the end of the pulse and thereby increase the expected width of the pulse. Because wide pulses are more likely to preset the PUF response flip-flop, increasing the delay difference between $D_{Q1-S1}$ and $D_{Q2-S2}$ will increase the Hamming weight of the PUF responses. Similarly, decreasing the delay difference by making $D_{Q1-S1}$ smaller than $D_{Q2-S2}$ will cause shorter pulses to be generated and a lower Hamming weight of the PUF responses. The Hamming weight impact of varying the delay difference is shown in Fig. 4c.

To examine how path delays change the Hamming weight of responses we begin with a PUF configuration that uses a carry chain of length 2 and no filter stages on the generated pulse. The following steps are performed:

- For each of the two feedback paths used in the PUF, we apply different minimum delay constraint values in steps of 10ps, and for each constraint let the Xilinx Vivado routing tool find feedback paths of appropriate lengths. This provides a catalog of different-length paths for $D_{Q1-S1}$ and $D_{Q2-S2}$.
- Repeatedly, we randomly choose feedback paths from the cataloged set of paths, implement the PUFs using those two paths, and record both the delay difference between the paths and the Hamming weight of the PUF responses.
- The PUF configurations are then binned according to their delay difference, and the average Hamming weight for each bin is calculated and plotted in Fig. 4c. The plot shows the expected result, in which making $D_{Q1-S1}$ longer than $D_{Q2-S2}$ will lead to wider pulses being generated (see Eq. 1), and correspondingly increase the Hamming weight of the responses.

For the specific case of a length 2 carry chain with no filter stages, it is found that delay differences between -25ps and -75ps produce a fractional Hamming weight closest to 0.5. This means that a fractional Hamming weight closest to 0.5 is obtained when routes are chosen such that the delay term $D_{Q1-S1}$ is 25ps to 75ps less than $D_{Q2-S2}$. If it is necessary to implement the PUF in a single slice, this technique could be used to obtain desirable PUF statistics without requiring a long multiple-slice carry chain.

After evaluating the three tuning knobs, we settle on one specific tuning that will be used throughout the remainder of the paper. Our design uses a multiplexer carry chain of length three with one filter stage and feedback paths with equal delay. Aside from the observation that this design has good PUF statistics, we choose this design for its simplicity, as the two feedback paths with matched delays utilize identical paths in different slices. All the results presented hereafter in the paper will be with respect to this tuning.

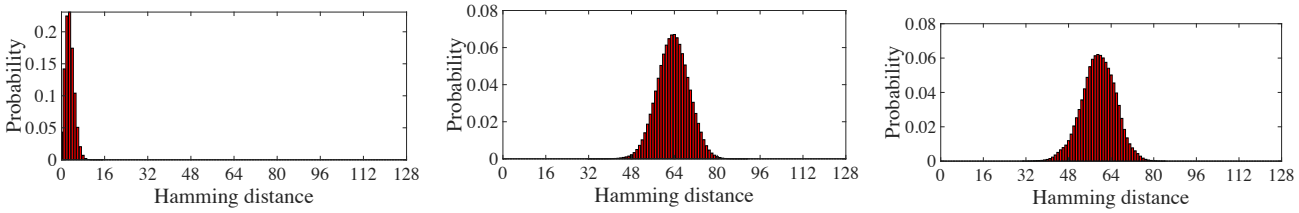
### 3.3 PUF Reliability and Uniqueness

Reliability and uniqueness are the most important properties of a weak PUF. Reliability refers to the repeatability of PUF outputs over time, which lessens the burden of error correction. Uniqueness is a measure of how different the output values are across PUF instances. We quantify these two properties for the Virtex 7 architecture by analyzing the Hamming distances between pairings of 128-bit PUFs across nine XC7Z020 chip instances. In total, we implement the same 28 disjoint 128-bit PUFs on each chip and record 250 output trials from each.

For reliability, we consider the distribution of intra-class Hamming distances between outputs of two different trials from the same 128-bit PUF on the same chip. The histogram in Fig. 5a shows the distribution of Hamming distances for 7,843,500 comparisons, representing all combinations of the 250 trials that are collected for each of the 28 different 128-bit PUFs on each of the nine chips ($28 \times 9 \times \binom{250}{2}$ comparisons). The mean intra-class distance is 3.06 bits (2.39%). A bit error occurs when a PUF output bit takes different values at enrollment and key generation. Therefore the average fractional intra-class Hamming distance also represents the average bit error rate (BER) of the PUF.

We consider two different variants of inter-class Hamming distance in our analysis. The first is the Hamming distance between outputs from two 128-bit PUFs that are on different chips or different locations on the same chip. Over all combinations of PUFs and trials ($\binom{28\times 9}{2} \times 250^2$ = 1,976,625,000 total comparisons), the mean distance is found to be 63.47 bits (49.59%) (Fig. 5b). The second set of inter-class distances are similar to the first except confined to only compare PUF pairings that occupy the same locations on different chips; this case is interesting because it will show a reduced Hamming distance if PUF output values are significantly influenced by deterministic bias instead of device-specific process variations. Comparing across all trials and chips for each of the 28 different PUF locations ($28 \times \binom{9}{2} \times 250^2$ = 63,000,000 total comparisons), the mean distance is found to be 62.39 bits (48.74%) (Fig. 5c).

In addition to the above analysis of Hamming distances between 128-bit PUFs, we also evaluate the standard PUF metrics of uniqueness [33], [34] and bit-aliasing [34]. Both of these metrics are computed using the "golden" response of each PUF instance, which is the response bit value produced by that PUF on a specific chip in a majority of trials. We use $r_{i,l}$ to denote the golden response on the $i$-th chip of the $l$-th PUF bit. The uniqueness value of $n$ single-bit PUFs in a population of $k$ chips is calculated according to Eq. 2. The ideal value for uniqueness metric is 50% [34], and the




(a) Intra-class Hamming Distance (b) Inter-class, all locations (c) Inter-class, same location

Fig. 5: Histograms of intra-class and inter-class Hamming distances of 128-bit PUFs. Intra-class distances compare two measurements from the same 128-bit PUF instance. Inter-class distances (in b) compare two 128-bit PUFs from different chips or different locations on the same chip, and (in c) compare 128-bit PUFs from the same locations on different chips. All measurements were made at room temperature of approximately $24°\mathrm{C}$ and at the nominal supply voltage.

TABLE 1: Reliability and uniqueness of PUF implementations on Zynq-7000 FPGAs.

| Design | Intra-class HD (%) | Uniqueness (%) |
|---|---|---|
| Anderson PUF [4] | 3.60 | 48.00 |
| Butterfly PUF [7] | 3.80 | 43.16 |
| RO PUF [36] | 0.48 | 46.15 |
| Proposed scheme | 2.37 | 46.25 |

uniqueness value for our PUF is found to be 46.25%. Bit-aliasing describes the propensity of each bit to take the same value on different chips. Bit-aliasing of the $l$-th bit is calculated according to Eq. 3. The ideal value for the average bit-aliasing is 50% [34]. The average bit-aliasing in the proposed design is found to be 45.88%.

$$\text{Uniqueness} = \frac{2}{k(k-1)} \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \frac{\sum_{l=1}^{n} r_{i,l} \oplus r_{j,l}}{n} \times 100\% \tag{2}$$

$$(\text{Bit-aliasing})_l = \frac{1}{k} \sum_{i=1}^{k} r_{i,l} \times 100\% \tag{3}$$

Table 1 compares the reliability (average intra-class Hamming distance) and uniqueness of our proposed PUF to other designs implemented on the Xilinx Zynq-7020 FPGAs as reported in [35]. Our PUF is found to be comparable to the other small PUF designs. Ring oscillator PUFs are known to be reliable but expensive since multiple oscillators are needed for generating each response bit.

### 3.4 Spatial Autocorrelation of PUF Location BERs

It is important to consider whether the PUF BERs are correlated spatially within each chip, as spatial correlation could imply a common cause for unreliability, instead of random per-device variations. The heatmap of Fig. 6 shows the average BER of the PUF at each location, averaged over all chips. Informally, the lack of a clear pattern in this figure gives some visual indication that the unreliable PUFs are likely to be random and not highly clustered.

To formalize the apparent lack of spatial correlation in Fig. 6, we use Moran's $I$ [37] as a metric to quantify the spatial autocorrelation in the average BER of PUF instances. Moran's $I$ is computed using Eq. 5, where $B_i$ and $\bar{B}$ are the average BER of PUFs at the $i$-th location and the average

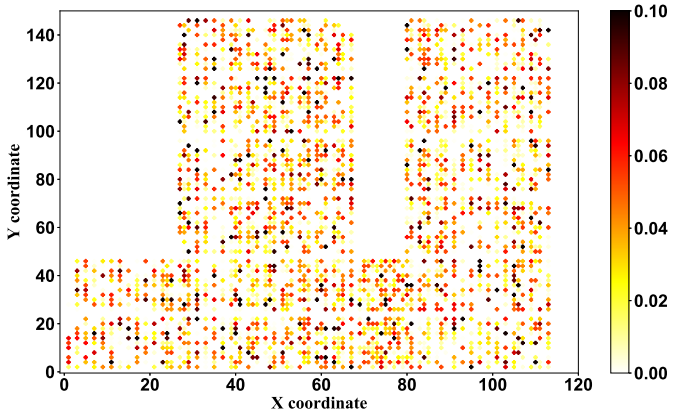



Fig. 6: Figure shows the average BER of PUF instances placed at different locations on the chips. Unreliable instances are not concentrated in a particular area of the chip.

BER across all locations respectively. Computing Moran's $I$ requires a spatial weight $w_{ij}$ to indicate which PUF locations should be considered local to each other. For PUF locations $i$ and $j$, we compute the weight $w_{ij}$ as shown in Eq. 4, where $r_i$ and $c_i$ are row and column indices of the $i^{th}$ PUF location, and $i \neq j$ enforces that each PUF is not considered to be its own neighbor. Restating this, the weight is set to 1 if the Euclidean distance between the row and column indices of two different PUF locations is less than 10. Moran's $I$ can take values between -1 and 1, where -1 and 1 indicate high spatial autocorrelation, and values close to 0 indicates less spatial autocorrelation.

To evaluate whether the BER values are spatially correlated, we consider the value of Moran's I against a null hypothesis of no spatial autocorrelation using a two-tailed test. All analysis is performed using the publicly available spatial analysis software PySAL (release 1.14.4) [38]. Under the null hypothesis (no spatial correlation), for our positions and weights, $I$ has an expected value of -2.50E-4 and variance of 5.78E-6. The value of $I$ in our dataset (BER values from Fig. 6 with weights according to Eq. 4) is 2.74E-3, which corresponds to a p-value of 0.21. This p-value means that spatially random data would on 21% of trials be expected to produce a value of $I$ that is more extreme that ours, and thus we cannot reject the null hypothesis of no spatial correlation. Stated more plainly, this analysis indicates that our BER

values do not have a significant spatial correlation.

$$w_{ij} = \begin{cases} 1 \text{ if } i \neq j, \sqrt{(r_i - r_j)^2 + (c_i - c_j)^2} < 10 \\ 0 \quad \text{otherwise} \end{cases} \quad (4)$$

$$I = \frac{n}{\sum_i \sum_j w_{ij}} \frac{\sum_i \sum_j w_{ij}(B_i - \bar{B})(B_j - \bar{B})}{\sum_i (B_i - \bar{B})^2} \quad (5)$$

## 4 Per-Device PUF Configurations

As mentioned in Section 3.2, our chosen PUF design uses for a delay path a carry chain with 3 multiplexers and 1 stage of glitch filters to achieve an average Hamming weight close to 50% (see Fig. 4b). There are multiple PUF configuration options for placing the design in a pair of SLICEs. Fig. 7 shows the two possible configurations that we consider. In one configuration (Fig. 7a), the PUF's carry chain extends from the bottom-most multiplexer in one SLICE to the bottom-most multiplexer in the next; in the second configuration (Fig. 7b), the entire PUF is shifted up by one multiplexer. It is possible to shift the entire design up by one more multiplexer to obtain a third PUF configuration from the two slices, but we consider in this work only the two configurations shown. We have implemented each of the two PUF configurations of Fig. 7 on nine different chips. The average intra-class Hamming distance is 2.37% for configuration 1 and 2.39% for configuration 2, while the uniqueness value (based on Eq. 2) is 46.25% for configuration 1 and 48.74% for configuration 2. This implies that two SLICEs can be configured in two different ways as a PUF, and in either configuration will have similarly desirable PUF statistics.

The average reliabilities of 2.37% and 2.39% are aggregated over a population of PUF instances with heterogeneous BERs. Within the population, there are many PUF instances that are extremely reliable, and some that are very unreliable. At a single location with two slices, we will show it is often the case that one PUF configuration will be unreliable and the other will be reliable. Because of this, it is possible to create a highly reliable PUF by choosing the more reliable configuration in each of the PUF locations, but the choice must be made uniquely for each chip. The only scenario in which per-device configuration will not be beneficial is if both configurations for a given PUF bit are equally unreliable. Note that per-device configuration does not impact uniqueness. Our implementation results show that the uniqueness value for PUFs after per-device configuration is 48.53%, which is comparable to either configuration 1 or configuration 2.

Implementing the per-device configuration yields a intra-class distance of 0.22%. The improved reliability after per-device configuration is also apparent in Fig. 8. The figure shows that after per-device configuration, a smaller fraction of PUFs produce a 1-response close to 50 percent of the time (fewer unreliable responses), and that larger fractions of PUFs produce a 1-response 0 or 100 percent of the time (more reliable responses).

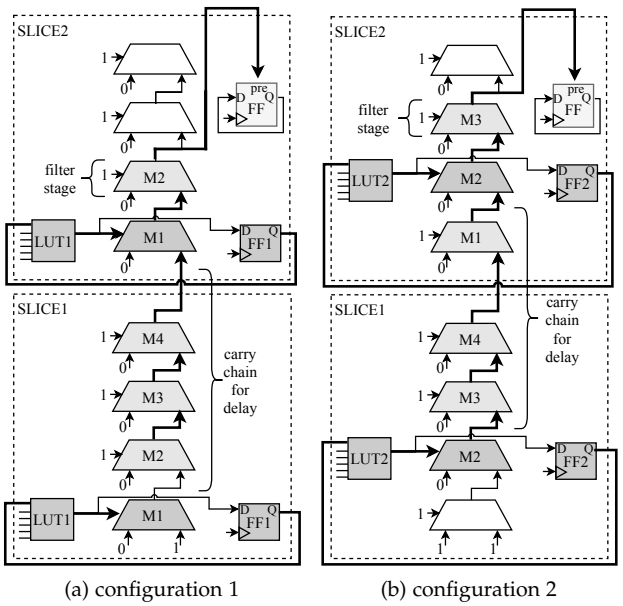



Fig. 7: Two possible PUF configurations using the same slices.

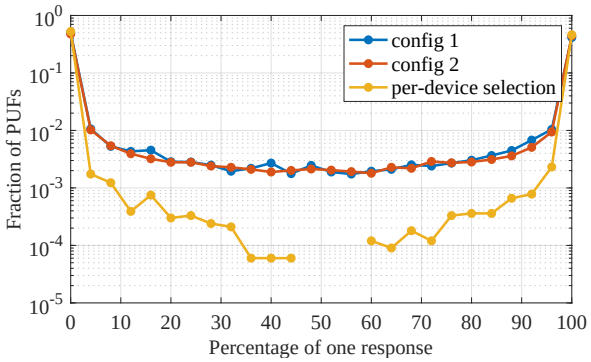
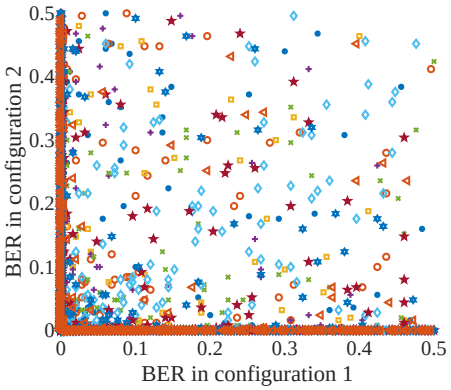



Fig. 8: Figure shows the fraction of PUFs that have each different probability of a producing a 1-response. PUFs that produce 1-response on close to 0 or 100 percent of trials are highly reliable, and cases falling between the two extremes are less reliable. Data generated from nine chips. Broken line indicates points where the value is zero.

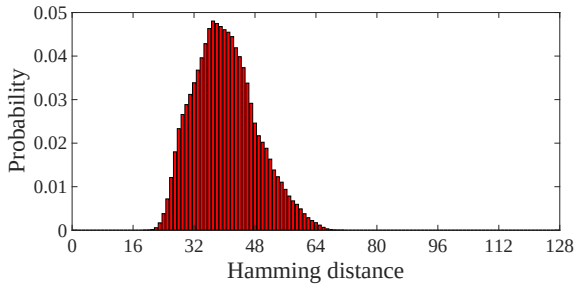
### 4.1 Comparing between-configuration PUF responses

Either of the configurations shown in Fig. 7 can be implemented at each PUF location on a chip, and we exploit this flexibility to choose the more reliable PUF configuration at each location. Noting that the two configurations share in common much of the carry chain, it is reasonable to wonder whether the two configurations would tend to have similar reliabilities, which would eliminate the benefits of per-device configuration. Our findings in Fig. 9a show the BERs of the two different configurations at one location do not tend to be correlated; if one configuration happens to have a high-BER PUF, it is not the case that the other configuration would also have a high-BER PUF. Choosing the more reliable configuration therefore can lower BER.

In fact, despite the two configurations sharing a number of delay stages, the responses of the two configurations are relatively unique to each other, as shown in Fig. 9b. The




(a) BER of the two configurations at each location. Different markers are used for each of the nine chips.




(b) Hamming distance between the two configurations.

Fig. 9: When the same logic slices are configured in two different ways (see Fig. 7) on the same chips: (a) their BERs are uncorrelated; and (b) the Hamming distance between 128-bit responses from each configuration is 40.23-bits (fractional Hamming distance of 31.43%).

Hamming distance between 128-bit responses in this case is 40.23-bits (fractional Hamming distance of 31.43%), which is less than ideal, but more than ten times higher than the intra-class distances. The between-configuration Hamming distance indicates that the two configurations can produce glitches with different propensities for setting their flip flops to 1, which explains why the two configurations can likewise have different reliabilities.

### 4.2 Design flow for per-device PUF configuration

Per-device configuration will only be attractive if the amount of work performed to customize the design for each device is minimal. The scheme we propose ensures that global design costs are only paid once, and that customizing the design for each chip is relatively inexpensive. Our CAD flow is a two step process, in which a global design is created that contains PUFs, and this design is then customized with a chip-specific partial bitstream that customizes PUF configurations without going through place-and-route. The procedure uses the steps described below.

**One-time Design Procedure:** Two configuration-agnostic steps are performed at design time.

- When performing place-and-route of the overall design, each PUF is represented by a placeholder design that uses the union of configuration 1 and configuration 2 resources. This produces the bitstream for the uncustomized global design. The PUF placeholders ensure that the overall design preserves all resources that will be needed for either PUF configuration. Each placeholder will later be replaced by a PUF in one configuration or the other. Regardless of which PUF configuration is used, the response bit is read from the same flip-flop.
- Once the PUF locations are chosen during place-and-route, two bitstreams are created to characterize the PUF configurations at each location. The first characterization bitstream instantiates configuration-1 PUFs at each PUF location, along with testbench logic to collect 100 responses from each PUF instance and communicate back the results. A similar bitstream is created for configuration-2 PUFs.

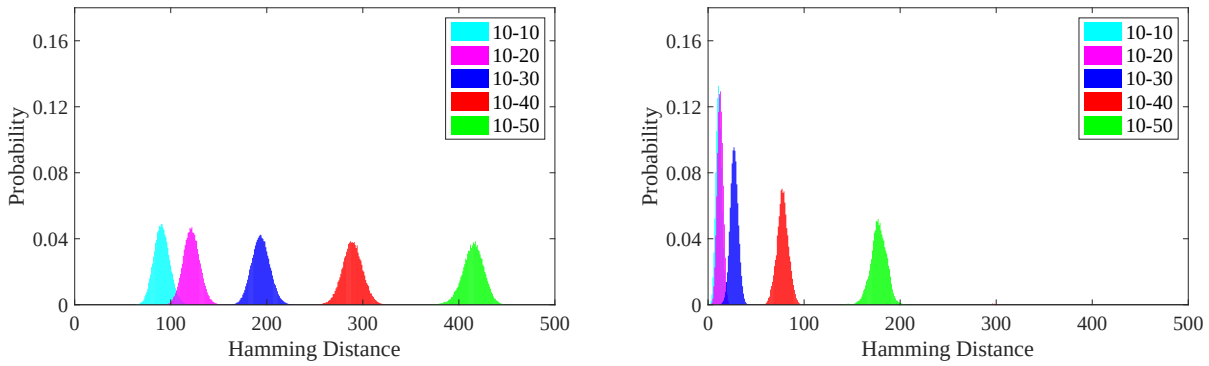
**Per-device PUF Configuration:** To deploy the PUF-containing design with per-device PUF configuration, the following steps are performed for each chip instance.

- Apply the characterization bitstream for configuration-1 PUFs and collect results. Repeat characterization using the configuration-2 bitstream. For each PUF location, decide whether configuration 1 or configuration 2 has a lower BER.
- Customize the global design bitstream for the chip by replacing the PUF placeholders with each location's chosen PUF configuration.

### 4.3 Temperature stability of PUF response

Although per-device configuration of PUFs does not explicitly consider temperature, we confirm in this section that configuring PUFs to be reliable at one temperature will implicitly lead to improved across-temperature reliability. Thermal data is collected in a TestEquity 115A Temperature Chamber. The soak time for each temperature is 10 minutes, and the on-chip temperature sensor is used to verify the temperature after the soak time. After the soak, 250 trials are collected from each chip and the process is repeated for each temperature. The $10°C$ trials are considered as the enrolled responses and trials from other temperatures are compared against these enrolled responses to obtain across temperature intra-class Hamming distances. Fig. 10a shows how the Hamming distances change from $10°C$ to $50°C$ in steps of $10°C$. Table 2 shows the average across-temperature BER for each configuration. In absence of per-device PUF configuration, when comparing a $50°C$ response to a $10°C$ enrolled response, the average BER is 11.24%.

The across-temperature reliability of the PUF design is improved by selecting at a single temperature the best configurations for each chip. In this case, we select the best PUF configuration based on their responses at $10°C$. The selected configuration for each PUF is then implemented and evaluated at all the temperatures from $10°C$ to $50°C$. The reliability data after selection of the best configurations is tabulated in the final row of Table 2 and plotted in Fig. 10b. Even though the choice of which PUF configuration to use is made based on responses at a single temperature, these PUFs




(a) Intra-class, nominal configuration

(b) Intra-class, per-device configuration

Fig. 10: Effect of temperature on reliability of 3,584-bit PUF response: (a) reliability of arbitrarily chosen PUF configurations; (b) reliability after per-device PUF configuration where configurations are chosen according to $10°\mathrm{C}$ responses.

TABLE 2: Average bit-error-rate of PUF configurations at different temperatures. Responses from each temperature are compared against $10°\mathrm{C}$ enrolled responses. Even in the worst case of comparing $10°\mathrm{C}$ to $50°\mathrm{C}$ responses, the per-device configuration reduces average bit error rate by over $2\times$ compared to nominal configurations.

| Temp(°C) | 10 | 20 | 30 | 40 | 50 |
|---|---|---|---|---|---|
| Normal config. | 2.47% | 3.32% | 5.26% | 7.86% | 11.24% |
| Per-device config. | 0.33% | 0.37% | 0.76% | 2.15% | 4.83% |

are found to be more reliable across a range of temperatures, with an average BER of 4.83% in the worst case, compared to 11.24% without per-device configuration.

## 5 Error Correction Impact of Reliability

Reliable key generation using PUFs, as described in Sec. 2.4, requires error correction. The error correction parameters are chosen according to the reliability of the PUF bits, and the cost of the error correction therefore depends on the reliability of the PUF design. In this section we show that the improved reliability from per-device PUF configuration enables simpler error correction and reduces the area cost of the PUFs and error correction in key generation to approximately half of their nominal cost in the Xilinx XC7020 FPGAs.

To evaluate PUF reliability and determine the error correcting code strength that will be needed, we use the standard heterogeneous reliability model for PUFs from Roel Maes [39]. Because our application of the model directly follows Maes' work, we refer readers to his paper [39] for complete details and do not reproduce his formulation here. In the heterogeneous model, which accounts for non-uniform bit error rates of PUFs, reliability targets are specified using both a key generation error rate and a percentage of chips in which that error rate must be satisfied. Consistent with previous work on PUFs, the reliability criterion that we enforce is for 99% of chips to generate 128-bit keys with failure rates of less than $10^{-6}$. Our design must therefore choose a BCH code that will meet that criterion. The first step in using the heterogeneous error rate model is to fit its 2-parameter model to empirical PUF data using least squares fitting. We fit the model to data from nine chips with 3,584 PUF instances per chip and 250 responses collected from each. Figure 11 shows agreement between the model and the empirical data on the 1-probability distribution of PUF cells, indicating that the model provides an accurate fit.

Once the analytical model is fit to empirical data, it is used to judge which BCH codes will meet the specified reliability target. By testing all BCH codes, we can select the lowest-cost code that satisfies our reliability criterion for the per-device configuration and for the standard configurations. In the case of PUFs taken from the two nominal configurations, the code required to meet the key reliability criterion described above is $BCH(255, 91, 25)$, meaning that 91 reliable bits can be extracted from 255 PUFs with the capability to correct 25 errors. Because each block produces only 91 bits, two code blocks are needed. The 128-bit key would be extracted from the 182 reliable bits. After performing the per-device configuration to improve PUF reliability, the same reliability criterion can be satisfied with $BCH(255, 199, 7)$. The full distribution of key reliability for these two cases is shown in Fig. 12. In this figure, the x-axis values represent key generation failure rates and the corresponding y-axis values show the proportion of chip instances that will exceed each failure rate.

The reliability of the per-device configuration saves resources in two ways, as shown in Table 3. Firstly, because the more reliable PUFs require only a single code block instead of two, the PUF resources are halved. Secondly, the more robust $BCH(255, 91, 25)$ decoder is itself significantly more expensive than the $BCH(255, 199, 7)$. Note that, for the $BCH(255, 91, 25)$ decoder, a single decoder circuit is reused to decode each block, so the cost difference between the two decoders is due to the complexity of the circuit and not the number of blocks that must be decoded. The resource savings given in Table 3 are computed by comparing the requirements for the two cases. The total savings are about 55% for LUTs and 41% for flip-flops.

## 6 Conclusion

In this work, we have proposed a new PUF on Xilinx FPGAs that is based on the Anderson PUF design [4], and proposed a methodology for per-device configuration of PUFs. Our

TABLE 3: Resource requirements for PUF instances and error correction in key generation. The scheme using per-device configuration generates a larger number of PUF-derived secret bits (199 vs. 182) while using 50% fewer PUFs, and in error correction using 57% fewer LUTs and 36% fewer flip-flops.

| Mode | BCH code (n,k,t) | Error Correction | | PUF | | Total | | Savings% | |
|---|---|---|---|---|---|---|---|---|---|
| | | LUTs | Flip-Flops | LUTs | Flip-Flops | LUTs | Flip-Flops | LUTs | Flip-Flops |
| Nominal configurations | 255,91,25 | 3914 | 3000 | 1020 | 1530 | 4934 | 4530 | 55% | 41% |
| Per-device configuration | 255,199,7 | 1689 | 1924 | 510 | 765 | 2199 | 2689 | | |

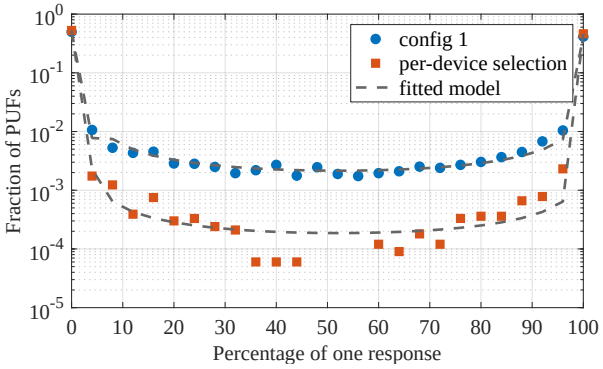



Fig. 11: The distribution of 1 response across 3,584 PUF instances on each of nine different chips. The markers show empirical data and the dashed lines shows the fitted model for the relative distributions.

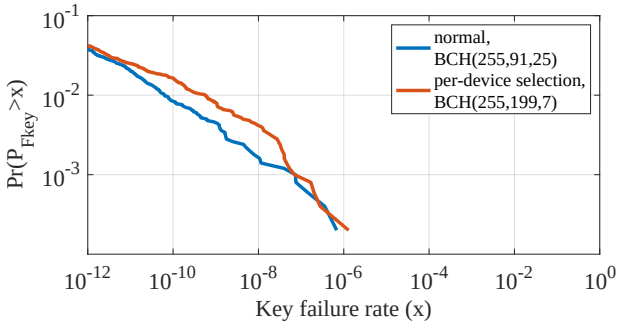



Fig. 12: Key failure rate after applying error correcting codes for nominal and per-device configuration. Different codes are applied for each case to meet the reliability criterion.

new PUF design uses the more plentiful SLICELs instead of the conventional design that uses the rarer SLICEMs. The response bias of the PUF design is shown to be tunable using carry chain length, mismatch in delay of feedback paths, and glitch filtering stages. The per-device configuration allows for one of two logically equivalent PUFs to be selected at each PUF location, and this improves the response reliability. Improving response reliability reduces the burden of error correction, and reduces the overall cost of PUFs and error correction by 55% in LUTs and 41% in flip-flops compared to a design that doesn't use per-device configuration. Furthermore, the per-device placement is shown to improve the stability of the PUF with respect to temperature by over 2× even when the two responses are taken at 10°C and 50°C respectively. The PUF design, and the methodology that we suggest, can be used to produce reliable LUT-based PUF keys on FPGAs, and can find broad applicability in reconfigurable systems going forward.

**Acknowledgement**: This work has been supported in part by a grant from the National Science Foundation (NSF).

## REFERENCES

[1] A. Moradi, D. Oswald, C. Paar, and P. Swierczynski, "Side-Channel Attacks on the Bitstream Encryption Mechanism of Altera Stratix II - Facilitating Black-Box Analysis using Software Reverse-Engineering," in *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Feb. 2013, pp. 91–100.

[2] "Microsemi Corp, Introduction to Implementing Design Security with Microsemi SmartFusion2 and IGLOO2 FPGAs," November 2013.

[3] "Intel Corp., Stratix 10 Device Data Sheet," January 2018.

[4] J. H. Anderson, "A PUF design for secure FPGA-based embedded systems," in *Proceedings of the 2010 Asia and South Pacific Design Automation Conference*, 2010, pp. 1–6.

[5] J. Wu and R. Huang, "A FPGA-based wireless security system," in *Proceedings of the International Conference on Multimedia Information Networking and Security*, 2011, pp. 512–515.

[6] "Helion Corp., Product Brief: AES-XTS Cores for FPGA," January 2016.

[7] S. S. Kumar, J. Guajardo, R. Maes, G. J. Schrijen, and P. Tuyls, "The butterfly PUF protecting IP on every FPGA," 2008, pp. 67–70.

[8] E. Simpson and P. Schaumont, "Offline hardware/software authentication for reconfigurable platforms," in *Cryptographic Hardware and Embedded Systems*, 2006, pp. 311–323.

[9] K. Hu, T. Wolf, T. Teixeira, and R. Tessier, "System-level security for network processors with hardware monitors," in *Proceedings of IEEE/ACM Design Automation Conference*, 2014, pp. 1–6.

[10] P. Sedcole and P. Y. Cheung, "Within-die delay variability in 90nm FPGAs and beyond," in *International Conference on Field Programmable Technology*, 2006, pp. 97–104.

[11] L. Cheng, J. Xiong, L. He, and M. Hutton, "FPGA performance optimization via chipwise placement considering process variations," in *International Conference on Field Programmable Logic and Applications*, Aug 2006, pp. 1–6.

[12] A. A. Bsoul, N. Manjikian, and L. Shang, "Reliability-and process variation-aware placement for FPGAs," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2010, pp. 1809–1814.

[13] B. Gassend, D. Clarke, and M. Van Dijk, "Silicon physical random functions," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2002, pp. 148–160.

[14] M. Majzoobi, F. Koushanfar, and S. Devadas, "FPGA PUF using programmable delay lines," in *International Workshop on Information Forensics and Security*, Dec 2010, pp. 1–6.

[15] S. Keshavarz and D. Holcomb, "Threshold-based obfuscated keys with quantifiable security against invasive readout," in *Proceedings of the 36th International Conference on Computer-Aided Design*, 2017, pp. 57–64.

[16] J. Guajardo, S. Kumar, G. Schrijen, and P. Tuyls, "FPGA intrinsic PUFs and their use for IP protection," 2007, pp. 63–80.

[17] O. Sander, B. Glas, L. Braun, K. Mller-Glaser, and J. Becker, "Intrinsic identification of Xilinx Virtex-5 FPGA devices using uninitialized parts of configuration memory space," in *International Conference on Reconfigurable Computing and FPGAs*, Dec 2010, pp. 13–18.

[18] A. Wild and T. Guneysu, "Enabling SRAM-PUFs on Xilinx FPGAs," in *International Conference on Field Programmable Logic and Applications*, Sept 2014, pp. 1–4.

[19] D. E. Holcomb, W. P. Burleson, and K. Fu, "Power-up SRAM state as an identifying fingerprint and source of true random numbers," vol. 58, no. 9, Sept 2009, pp. 1198–1210.

[20] R. Maes and V. van der Leest, "Countering the effects of silicon aging on SRAM PUFs," in *IEEE International Symposium on Hardware-Oriented Security and Trust*, May 2014, pp. 148–153.

[21] A. Roelke and M. R. Stan, "Controlling the reliability of SRAM PUFs with directed NBTI aging and recovery," 2018, pp. 1–11.

[22] R. Maes, P. Tuyls, and I. Verbauwhede, "Intrinsic PUFs from flip-flops on reconfigurable devices," in *3rd Benelux Workshop on Information and System Security*, vol. 17, 2008.

[23] S. Morozov, A. Maiti, and P. Schaumont, "An analysis of delay based PUF implementations on FPGA," in *Reconfigurable Computing: Architectures, Tools and Applications*, 2010, pp. 382–387.

[24] X. Xu, S. Keshavarz, D. J. Forte, M. M. Tehranipoor, and D. E. Holcomb, "Bimodal oscillation as a mechanism for autonomous majority voting in PUFs," 2018, pp. 1–12.

[25] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," vol. 38, no. 1, 2008, pp. 97–139.

[26] A. Juels and M. Wattenberg, "A fuzzy commitment scheme," in *Proceedings of the ACM Conference on Computer and Communications Security*, 1999, pp. 28–36.

[27] J. L. Carter and M. N. Wegman, "Universal classes of hash functions," in *Proceedings of the ninth annual ACM symposium on Theory of computing*, 1977, pp. 106–112.

[28] J. Delvaux, D. Gu, I. Verbauwhede, M. Hiller, and M.-D. M. Yu, "Efficient fuzzy extraction of PUF-induced secrets: Theory and applications," in *International Conference on Cryptographic Hardware and Embedded Systems*, 2016, pp. 412–431.

[29] R. Maes, V. van der Leest, E. van der Sluis, and F. Willems, "Secure key generation from biased PUFs," in *Cryptographic Hardware and Embedded Systems*, 2015, pp. 517–534.

[30] M. Hiller and A. G. Önalan, "Hiding secrecy leakage in leaky helper data," in *Cryptographic Hardware and Embedded Systems*, 2017, pp. 601–619.

[31] P. Koeberl, J. Li, A. Rajan, and W. Wu, "Entropy loss in PUF-based key generation schemes: The repetition code pitfall," in *IEEE International Symposium on Hardware-Oriented Security and Trust*. IEEE, 2014, pp. 44–49.

[32] S. Vyas, N. Dumpala, R. Tessier, and D. Holcomb, "Improving the efficiency of PUF-based key generation in FPGAs using variation-aware placement," in *International Conference on Field Programmable Logic and Applications*, Sep. 2016, pp. 1–4.

[33] Y. Hori, T. Yoshida, T. Katashita, and A. Satoh, "Quantitative and statistical performance evaluation of arbiter physical unclonable functions on FPGAs," in *IEEE International Conference on Reconfigurable Computing and FPGAs*, 2010, pp. 298–303.

[34] A. Maiti, V. Gunreddy, and P. Schaumont, "A systematic method to evaluate and compare the performance of physical unclonable functions," in *Embedded systems design with FPGAs*, 2013, pp. 245–267.

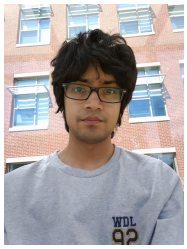
[35] J.-L. Zhang, Q. Wu, Y.-P. Ding, Y.-Q. Lv, Q. Zhou, Z.-H. Xia, X.-M. Sun, and X.-W. Wang, "Techniques for design and implementation of an FPGA-specific physical unclonable function," in *Journal of Computer Science and Technology*, vol. 31, no. 1, 2016, pp. 124–136.

[36] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *ACM/IEEE Design Automation Conference*, June 2007, pp. 9–14.

[37] P. A. P. Moran, "Notes on continuous stochastic phenomena," vol. 37, no. 1/2. [Oxford University Press, Biometrika Trust], 1950, pp. 17–23.

[38] S. J. Rey and L. Anselin, "Pysal: A python library of spatial analytical methods," in *Handbook of Applied Spatial Analysis: Software Tools, Methods and Applications*, 2010, pp. 175–193.

[39] R. Maes, "An accurate probabilistic reliability model for silicon PUFs," in *Proceedings of the 15th International Conference on Cryptographic Hardware and Embedded Systems*, 2013, pp. 73–89.

**Mohammad A. Usmani** Received his B.S. degree in Electronics and Communication engineering from Aligarh Muslim University, UP, India and his M.S. degree in Electrical and Computer engineering from the University of Massachusetts Amherst, Amherst, MA, USA. He is currently working as a Silicon Design Engineer at Netronome Systems. His research interests include Hardware security and VLSI design.

**Shahrzad Keshavarz** is pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering at University of Massachusetts, Amherst. She received her M.S. from University of Tehran, Iran in computer engineering, in 2014. Her current research interests includes hardware security, digital systems testing and testable design and verification.

**Eric Matthews** received his BASc and MASc degrees from the School of Engineering Science at Simon Fraser University in 2009 and 2012, respectively. He is currently pursuing his Ph.D. at Simon Fraser University with a research focus on soft-processor architectures for FPGAs.

**Lesley Shannon** is an Associate Professor Chair for the Computer Engineering Option in the School of Engineering Science at Simon Fraser University. She is also the NSERC Chair for Women in Science and Engineering for the BC/Yukon Region. She received her Bachelors in Electrical and Computer Engineering in 1999 from the University of New Brunswick and her Masters and PhD in Computer Engineering from the University of Toronto in 2001 and 2006, respectively. Dr. Shannons research focuses on computing system design, including reconfigurable computing and heterogeneous computing architectures, application-specific architectures and reconfigurable digital microfluidics.

**Russell Tessier** (M00-SM07) received the B.S. degree in computer and systems engineering from Rensselaer Polytechnic Institute, Troy, NY, USA, in 1989, and the S.M. and Ph.D. degrees in electrical engineering from the Massachusetts Institute of Technology, Cambridge, MA, USA, in 1992 and 1999, respectively. He is currently Professor of Electrical and Computer Engineering with the University of Massachusetts, Amherst, MA. His current research interests include computer architecture and FPGAs.

**Daniel Holcomb** (M07) received the B.S. and M.S. degrees in electrical and computer engineering from the University of Massachusetts Amherst, Amherst, MA, USA, and the Ph.D. degree in electrical engineering and computer sciences from the University of California at Berkeley, Berkeley, CA, USA. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, University of Massachusetts Amherst. His research interests are in hardware security and embedded systems.