Improving the Efficiency of PUF-Based Key Generation in FPGAs using Variation-Aware Placement

Shrikant Vyas, Naveen Kumar Dumpala, Russell Tessier, and Daniel E. Holcomb
Department of Electrical and Computer Engineering
University of Massachusetts Amherst
{svyas, ndumpala, tessier, dholcomb}@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 which use random manufacturing variations to produce secret keys that are unique to each device. The security of PUF-based keys comes at a high hardware cost. Due to the need for error correction to extract reliable keys from noisy PUFs, the total cost of an n-bit key far exceeds just the cost of producing n bits of PUF output. In this work, we propose variation-aware intra-FPGA PUF placement to reduce the area cost of PUF-based keys on FPGAs. We show that placing PUF instances according to the random variations of each chip instance reduces the bit error rate of the PUFs and consequently greatly reduces the overall cost of key generation. The proposed variation-aware placement approach is applicable to any PUF-based system implemented in reconfigurable logic. We demonstrate our approach on a Xilinx Zynq-7000 Programmable SoC using FPGA-specific PUFs with code-offset error correction based on BCH codes. We quantify the effectiveness of our approach by comparing the implementation costs of the same system when using the default approach of variation-agnostic placement and our proposed variation-aware placement. It is shown that our approach reduces the area required for PUF and error-correction circuitry by about 50% while achieving equivalent reliability.**

I. Introduction

FPGAs are used for an increasingly large number of applications which require security. Due to their volatile nature, SRAM-based FPGAs require security at multiple levels. Bitstream encryption is commonly used to protect the configuration bits which 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.

PUFs represent a per-device method of generating secret keys on-chip without reliance on non-volatile memory or battery-backed storage. PUF-based keys are uniquely tied to each device, and are not directly compromised by side-channel attacks that learn AES keys to decrypt bitstreams. These characteristics make PUFs well suited to key generation for FPGA-based applications. PUF-based keys are enabled in commercially available reconfigurable devices including MicroSemi IGLOO2 and Altera Stratix 10 FPGAs.

Although the logic needed to create PUFs in FPGAs is generally modest (e.g. a few lookup tables (LUTs)), the amount of circuitry needed to create repeatable and consistent keys from the PUF bits can be significant. In this paper we present a method that greatly reduces the hardware cost of PUF-based key generation by locating the best locations in reconfigurable logic to generate the most highly-reliable PUFs. The ideas proposed in this work are applicable to any PUF-based key implementation, and we focus our implementation on the LUT-based, FPGA-specific Anderson PUF [2]. The specific contributions of this work are as follows:

- We analyze the spatial randomness of unreliable PUF instances within chips and across chips, and based on our findings propose a novel system of per-device PUF configuration to improve PUF reliability and thereby reduce the implementation cost of the overall system.
- We demonstrate an area savings of about 50% for PUF and error correction circuitry over a default placement of the same logic for 56-, 128-, and 256-bit keys.
- We implement the Anderson PUF (previously targeted to a Virtex 5) on a more contemporary Xilinx architecture (Virtex 7) and quantify its uniqueness and reliability in the new device architecture.

II. 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 [17] 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 [1]. A comprehensive approach to protecting the use of intellectual property cores in FPGAs using public/private keys pairs is described in Kumar *et al.* [10]. Keys created by a PUF are used to initiate IP core operation and periodically authenticate its use [15]. In Hu *et al.* [8], 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.

A. 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 is given by Sedcole and Cheung [14]. Reconfigurability offers the opportunity for per-device logic placement as a mechanism to address process variations. Cheng *et al.* [4] use 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.* [3] optimizes placement 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.

B. Physical Unclonable Functions

Physical unclonable functions (PUFs) are circuits that leverage manufacturing variations to produce instance-specific output values. The output values produced by each PUF instance are persistent over time, but can be influenced by noise. PUF-based keys have been widely implemented on FPGAs. SRAM PUFs on FPGAs utilize the unique power-up state of SRAM cells [7] 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 been largely unsuccessful. Sander *et al.* [13] 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 [16] 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. The initial states of flip-flops in FPGAs depend on process variations [11], but are extremely biased and produce low quality outputs. Butterfly PUFs [10] use contention on cross-coupled latches to generate output bits, but later work questions the uniqueness of Butterfly PUF outputs [12].

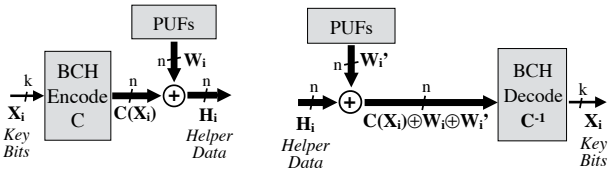
For this work we use an FPGA-specific PUF based on LUTs and hardened carry-chains found in Xilinx FPGAs [2]. Our implementation of this PUF is described in Section IV-A.

III. Error Correction for PUF-based Keys

Cryptographic keys must be repeatable over time. The outputs of PUFs are noisy and thus cannot be used directly as key bits. Fuzzy extractors [6], [9] are cryptographic primitives for deriving reliable key values from noisy biometric data, and are widely used with PUFs. When a key is first enrolled to (i.e. derived from) a PUF, the fuzzy extractor outputs helper data 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 derive the key. 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 inability to guess the PUF outputs which are never revealed in the clear.

A. Code-Offset Fuzzy Extractor using Helper Data

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. 1; k key bits are enrolled and generated using n PUF instances. A larger



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

Fig. 1: During key enrollment, key bits are encoded and offset by PUF outputs to produce helper data. The helper data is later used with the PUF to regenerate the key in the field. The generated key will match the enrolled key if the Hamming distance between the PUF outputs (W_i and W_i') is within the error correction capacity of the BCH code used.

key is generated by splitting the key into k-bit blocks and using n PUF instances to enroll and generate each block.

Key Enrollment: During key enrollment (Fig. 1a), the i^{th} key segment is chosen as a k-bit string X_i and encoded to an n-bit BCH codeword $C(X_i)$; X_i can be decoded from any n-bit string that is within Hamming distance t of codeword $C(X_i)$. The codeword is offset by XOR with n-bit PUF output W_i and the result is stored as helper data H_i.

Key Generation: During key generation (Fig. 1b), the helper data H_i is offset by a PUF output observation W_i' that may differ slightly from W_i used during enrollment; this produces the original codeword $C(X_i)$ corrupted by $W_i \oplus W_i'$. The corrupted codeword can be decoded to regenerate the enrolled value X_i as long as $C(X_i) \oplus W_i \oplus W_i'$ is within Hamming distance t of $C(X_i)$. Stated differently, the key bits X_i are generated correctly 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. Therefore, the BCH code must be chosen according to the reliability of the PUF outputs.

Security: The PUF outputs are never revealed in cleartext. As long as the adversary knows nothing about the PUF outputs, the public helper data H_i reveals no information about the codeword $C(X_i)$ or key bits X_i. If the PUF outputs are biased or predictable, then an additional key derivation function should be used to extract a full-entropy key from X_i.

B. Error Correction Cost versus Bit Error Rate

The overhead costs of error correction increase sharply with the bit error rate (BER) of the PUFs. Specifically, the costs that increase are the number of required PUF instances, and the complexity of the BCH decoder used to correct the errors.

For a given block size (n), there is a tradeoff between the number of information bits encoded (k), and the number of correctable errors (t). For example, in a 127-bit block, a code that corrects 5 errors carries 92 information bits, and a code that corrects 15 errors carries only 36 information bits. By choosing reliable PUFs, more key bits can be derived from each block without compromising reliability, and fewer PUFs are required overall.

The expected error rate of the key can be calculated as a function of the PUF BER and the BCH code parameters, for a given PUF BER, we minimize cost by choosing the most efficient BCH code parameters that will result in a key with an overall error rate not exceeding 1E-6. The value of 1E-6 as a reliability criterion is used for consistency with previous literature [5], [7].

IV. PUF Implementation and Characterization

A. Anderson PUF Implementation on a Virtex 7 Architecture

The Anderson PUF is an FPGA-specific PUF that generates chip-specific outputs by utilizing delay variations in the fixed carry chain logic of Xilinx FPGAs. Due to space limitations, we refer readers to the original paper on the Anderson PUF for background [2]. Anderson's original paper used Virtex 5 devices and showed that the carry chain across a vertical distance of six LUTs was optimal for performance [2]. Our experimentation with a Virtex 7 architecture showed that a vertical distance of five LUTs provides the best PUF performance.

B. PUF Reliability and Uniqueness

Reliability and uniqueness of the Anderson PUF on the Virtex 7 architecture is evaluated using Hamming distances between pairings of 128-bit PUFs across four chip instances. We implement the same 16 disjoint 128-bit PUFs on each chip, and record 1000 output trials from each. To mimic a noisy application that could be placed with the PUFs, each PUF is subjected to constant toggling of 5 toggle flip flops placed in the slices which combine to make each PUF bit. Within-class Hamming distances show reliability by comparing two randomly selected output trials from the same 128-bit PUF. The histogram of Fig. 2 comprises 10000 within-class comparisons, with random selection of PUFs and trials for each comparison. The mean within-class distance is 5.29. Uniqueness is demonstrated by between-class Hamming distances which are found to have a mean of 62.00 (Fig. 2).

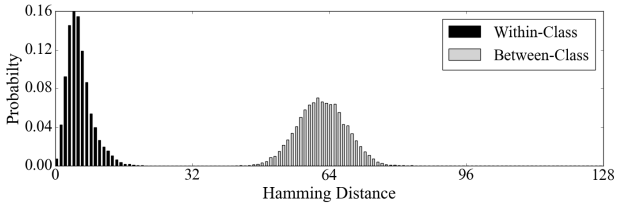


Fig. 2: Within-class and between-class Hamming distances show the reliability and uniqueness of the implemented PUF.

C. Device-Specific Location of Unreliable PUF Instances

Per-device variation-aware placement tries to avoid implementing PUFs on each chip at locations where they would be unreliable. A per-device placement approach is only necessary if the locations of unreliable PUF instances are unique to each chip, and in this subsection we offer empirical data showing this to be the case.

Fig 3 shows graphically the correlation of BER for a single pairing of chips; each point in this plot represents one of 2080 PUF instances, and its horizontal and vertical positions indicate its BER when instantiated on two different chips. If the BERs were correlated, the points would tend to cluster along the diagonal, but we do not observe this to occur. More formally, we analyze the correlation of per-location BERs among all pairs of chips using Pearson correlation coefficients. For two chips x and y, the Pearson coefficient $r_{x,y}$ is computed using Eq. 1, where B_i^x represents the BER of PUF location i on chip x, and $\bar{B^x}$ represents the mean BER on chip x; B_i^y and $\bar{B^y}$ have the same meaning for chip y. A coefficient close to 0 indicates that the BERs across chips are uncorrelated. The Pearson coefficients for all pairings of chips fall between -0.020 and 0.018, indicating that the locations of unreliable PUFs are largely unique to each chip.

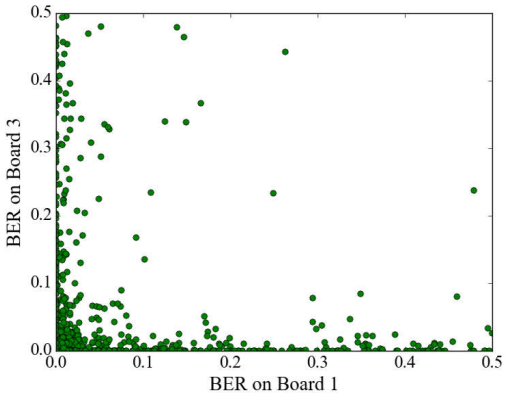


Fig. 3: Respective BERs of same-location PUFs on two different chips. The 2080 data points in the figure represent 2080 PUF locations instantiated on two different chips.

$$r_{x,y} = \frac{\sum_{i=1}^{2080} \left(B_i^x - \bar{B^x}\right)\left(B_i^y - \bar{B^y}\right)}{\sqrt{\sum_{i=1}^{2080} \left(B_i^x - \bar{B^x}\right)^2}\sqrt{\sum_{i=1}^{2080} \left(B_i^y - \bar{B^y}\right)^2}} \quad (1)$$

D. Spatial Autocorrelation of PUF Location BERs

It is important to consider whether unreliable PUFs 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. 4 shows, for a single chip, the reliability of 2080 PUF instances according to their locations. Informally, the lack of a clear pattern in this figure gives visual indication that the unreliable PUFs are likely to be random and chip-specific. To formalize the apparent lack of spatial correlation in Fig. 4, we use Moran's I as a metric to quantify the spatial autocorrelation in the BER of PUF instances. For any single chip instance, Moran's I is computed using Eq. 3, where B_i and $\bar{B}$ are the BER of PUF instance i and the mean BER of the chip 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. 2, where r_i and c_i are row and column indices of the i^{th} PUF location. Moran's I can take values between -1 and 1, where 1 indicates high spatial autocorrelation and 0 indicates no spatial autocorrelation. The I values for all

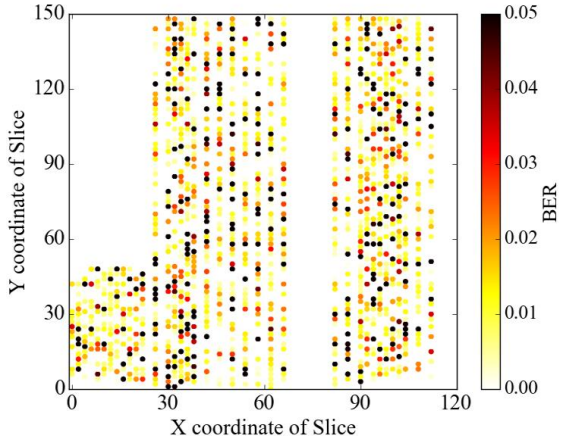


Fig. 4: Figure shows the BER of PUF instances placed at different locations on a chip. Unreliable instances are scattered and not concentrated in a particular area of the chip.

four chips fall between 0.013 and 0.017, indicating that the unreliable PUFs do not tend to be highly clustered. Because the locations of unreliable PUF instances are unique to each chip and spatially uncorrelated, it can be beneficial to perform per-device placement to use only the PUF locations that are highly reliable on that chip.

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

$$I = \frac{N}{\sum_i \sum_j w_{ij}} \frac{\sum_i \sum_j w_{ij}(B_i - \bar{B})(B_j - \bar{B})}{\sum_i (B_i - \bar{B})^2} \tag{3}$$

E. Cost Savings Through Per-Device Placement

Table I shows the number of LUTs saved by using per-device placement of PUFs for various key sizes. The use of selected PUF locations on each chip instance reduces BER to less than 0.01, which allows a BCH code with parameters $n = 127$, $k = 64$, $t = 10$ to be used while satisfying the overall key reliability criterion of failure rate less than 1E-6. When using the default approach of variation-agnostic PUF selection, the BER is 0.042 and a more expensive $n = 127$, $k = 29$, $t = 21$ BCH code must be used to achieve the same reliability. The per-device placement therefore reduces the complexity of the BCH decoder, and also reduces the total number of PUFs required because more key bits are extracted from each BCH codeword.

V. Conclusion

In this work we have proposed per-device variation-aware PUF placement for reducing area cost of PUF-based keys in reconfigurable computing systems. We have implemented three different sizes of PUF-based keys and have demonstrated that our approach can save between 49 and 55% of the area of PUFs and error correction. Future work will consider how per-device placement can be restricted to specific areas of the chip and used within an overall incremental compilation approach so that the majority of a design can be placed in a variation-agnostic fashion while still reaping the area savings of variation-aware per-device PUF placement.

	PUF	BCH	Reduction
Variation Agnostic 256-bit key	2569	2219	-
Variation Aware 256-bit key	1143	1249	49.39%
Variation Agnostic 128-bit key	1397	2160	-
Variation Aware 128-bit key	508	1292	50.04%
Variation Agnostic 56-bit key	889	2055	-
Variation Aware 56-bit key	254	1082	55.37%

TABLE I: Breakdown by function of the LUT counts used to implement 256, 128, and 56-bit key generation. Variation-agnostic denotes the default approach where instance-specific variation is not considered in PUF placement.

References

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

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

[3] 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*, pages 1809–1814, 2010.

[4] 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*, pages 1–6, Aug 2006.

[5] J. Delvaux, D. Gu, D. Schellekens, and I. Verbauwhede. Helper data algorithms for PUF-based key generation: Overview and analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(6):889, 2015.

[6] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM Journal on Computing*, 38(1):97–139, 2008.

[7] J. Guajardo, S. Kumar, G. Schrijen, and P. Tuyls. FPGA intrinsic PUFs and their use for IP protection. *Cryptographic Hardware and Embedded Systems*, pages 63–80, 2007.

[8] 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*, pages 1–6, 2014.

[9] A. Juels and M. Wattenberg. A fuzzy commitment scheme. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 28–36, 1999.

[10] S. S. Kumar, J. Guajardo, R. Maes, G. J. Schrijen, and P. Tuyls. The butterfly PUF protecting IP on every FPGA. *IEEE International Workshop on Hardware-Oriented Security and Trust*, pages 67–70, 2008.

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

[12] S. Morozov, A. Maiti, and P. Schaumont. An analysis of delay based PUF implementations on FPGA. In *Reconfigurable Computing: Architectures, Tools and Applications*, pages 382–387. Springer, 2010.

[13] 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*, pages 13–18, Dec 2010.

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

[15] E. Simpson and P. Schaumont. Offline hardware/software authentication for reconfigurable platforms. In *CHES*, volume 4249, pages 311–323. Springer, 2006.

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

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