Securing FPGA SoC Configurations Independent of Their Manufacturers

Nisha Jacob*, Jakob Wittmann*, Johann Heyszl*, Robert Hesselbarth*,
Florian Wilde†, Michael Pehl†, Georg Sigl*†, Kai Fischer‡
*Fraunhofer Institute for Applied and Integrated Security (AISEC), Munich, Germany
†Technische Universität München, Munich, Germany
‡Siemens AG, Corporate Technology, Munich, Germany
Email: nisha.jacob@aisec.fraunhofer.de

***Abstract*—System-on-Chips which include FPGAs are important platforms for critical applications since they provide significant software performance through multi-core CPUs as well as high versatility through integrated FPGAs. Those integrated FPGAs allow to update the programmable hardware functionality, e.g. to include new communication interfaces or to update cryptographic accelerators during the life-time of devices. Updating software as well as hardware configuration is required for critical applications such as e.g. industrial control devices or vehicles with long life-times. Such updates must be authenticated and possibly encrypted. One way to achieve this is to rely on static FPGA manufacturer-provided cryptography and respective master keys. However, in this contribution, we show how to retrofit Xilinx Zynq FPGAs with an alternative cryptographic accelerator and how to establish device-individual keys using Physical Unclonable Function (PUF) technology. These two key aspects reduce the required trust in manufacturer-provided security features while increasing the security by binding configurations to a specific device.**

***Index Terms*—SoC, FPGA, Zynq, PUF, cryptographic engine, partial reconfiguration, secure boot**

I. Introduction

Powerful System-on-Chips (SoCs) which include configurable hardware in the form of field programmable gate arrays (FPGAs) on the same chip are increasingly popular platforms for embedded systems because they offer high performance and high flexibility in software as well as hardware. Embedded systems, which are deployed in the field for a long life-time, require updates from a functional as well as from a security perspective. These updates need to be authentic and partly encrypted to avoid manipulations. For FPGA devices in particular and also for more general SoCs, a typical method to achieve this is to use chip-manufacturer-provided master-keys or individualized keys in manufacturer-provided key storage as well as built-in cryptographic engines. The encryption and authentication of FPGA configurations as well as secure boot processes of SoCs are classic examples. There have been security issues, however, with manufacturer-provided features. For example, encryption engines included in FPGAs for the decryption and authentication of configurations have been shown to be vulnerable to side-channel analysis, where attackers use measurements of power consumption during cryptographic operations to extract and misuse keys. Successful attacks have been reported for FPGA manufacturers of FPGA SoC platforms. First reports hit devices such as the Xilinx Virtex-II Pro [13] where full triple DES keys for bitstream encryption have been extracted. Current devices such as the Xilinx 7 series have also been targeted successfully [15]. The same issues have been reported for Altera, the second important manufacturer of FPGA-SoCs, where Stratix II and Stratix III devices have been broken in a similar way [14], [21]. Finally, also Microsemi's ProASIC3 devices have been broken [20], [19]. These weaknesses can be encountered by not solely relying on manufacturer-provided security features for core operations such as the authentication and decryption of configurations.

We present a practical realization of a system design that allows to retrofit traditional FPGAs and contemporary FPGA SoCs with alternative cryptographic accelerators for the authentication and decryption of hardware configurations as well as software images to support secure boot processes. Since manufacturer-provided secure key storage is typically not accessible from such custom designed hardware, and any form of master key is ill-advised, we also implemented a hardware-configured physical unclonable function (PUF) which is used as device-individual credential store. An expample of a PUF-based key provision token is described by Fischer et al. [4]. PUFs generate unique-per-device keys by exploiting smallest manufacturing variations in integrated circuits even though the corresponding design is stable. These keys are generated only on request as soon as the design is configured into the FPGA and are, thus, not hard-coded in the bitstream or binary. In summary, this system design allows for more flexibility through the integration of custom cryptographic accelerators and storing keys in PUF modules to bind sensitive intellectual property (IP) in the software or hardware configuration to a specific device. Another reason for such an architecture is a possibility for crypto-agility, e.g. to employ post-quantum-secure cryptography in future. The general idea behind this has also been outlined in a white paper by Xilinx [17]. However, no working implementation of this scheme was described. As our contribution, we describe an implementation of this system design on the popular Xilinx Zynq-7000 FPGA SoC which can be transferred to other FPGAs or FPGA SoCs from different manufacturers. As a PUF, we implement a Twisted Bistable Ring PUF (TBR-PUF). A PUF module always requires error

978-1-5386-4034-0/17/$31.00 ©2017 IEEE

correction of the output extracted from the variations, and binding of concrete keys to those corrected outputs. The respective functionality is called fuzzy extractor (FE). We implemented two variants, one based on Differential Sequence Coding and Convolutional Code (DSC+CC) and one based on Repetition and BCH Code (Rep+BCH). Keys are used by our custom AES-GCM engine. All the custom hardware cores are controlled by lightweight software configuration routines to provide a novel implementation of two important use-cases: (1), the secure boot of software, and (2), the secure update of hardware configuration using partial reconfiguration. As an additional contribution, we test the device-dependency of the PUF-derived keys and the robustness of the keys in conjunction with partial reconfiguration.

II. System Design Overview

Figure 1 provides an overview of the system design when it is applied to a typical FPGA SoC. The figure shows the FPGA SoC which includes an FPGA part and a multi-core CPU. The parts which are added for this security-enhanced design are depicted as shaded boxes. The main modules for the security enhancement are the PUF, FE, AES and the partial reconfiguration (PR) controller, all of which are integrated in the FPGA fabric of the FPGA SoC. The PUF (Sec. III-B), is used with the subsequent FE (Sec. III-C) as a secure key storage module.

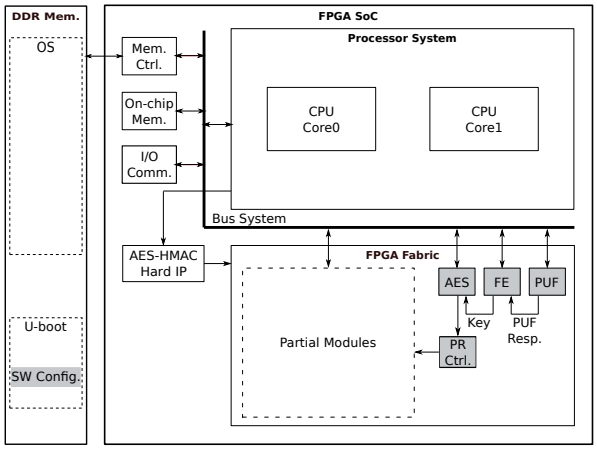

Fig. 1: System design overview

The PUF generates a device-unique but noisy bit sequence. The FE module uniquely binds a key to the device-specific bit sequence, i.e. to a certain FPGA, during enrollment, corrects errors in the sequence and reproduces the key later on demand. This reconstructed key is directly used by cryptographic core and is not accessible by any other IP core and software in the system. The cryptographic core, AES-GCM in this case, uses this key to decrypt and authenticate data, such as software images and hardware configurations (bitstreams), to be loaded on the device. The decrypted and authenticated software images or bitstreams can then be either transfered to the external DDR memory or the PR controller respectively. The PR controller provides access to the internal configuration port which can be used to partially reconfigure parts of the FPGA. All the modules on the FPGA are configured and controlled by integrating simple software routines in U-boot, which is a standard bootloader commonly used to load operating systems (OSs) of embedded systems.

Manufacturers usually provide a secure key storage, and one specific cryptographic core. A weakness in this core could potentially expose the entire system. Using our proposed design, both hardware and software, can be authenticated and encrypted without having to rely on possibly insecure security features provided by manufacturers. The subsequent sections show how two important processes, (1), the secure boot of software, and (2), the secure update of hardware, can be performed on this design.

A. Secure Boot of Software

Secure boot is crucial to modern embedded systems with critical applications. In this process, all code is verified for integrity and authenticity using cryptographic means before being loaded and configured or executed. In case the code contains IP relevant data, it could be encrypted in addition.

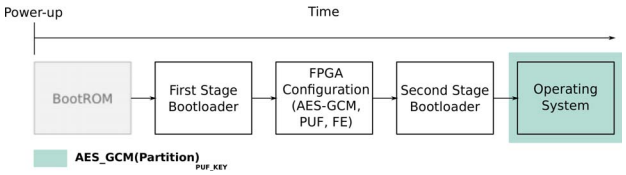

Fig. 2: Modified secure boot sequence

Figure 2 depicts a regular boot sequence on FPGA SoCs with modifications in order to support the enhancements. The boot process begins after power-up and is divided into the following five stages:

1) BootROM - non-accessible internal hardwired code
2) First stage bootloader (FSBL)- User code used to initialise the system and load subsequent partitions i.e, bitstream and second stage boatloader
3) FPGA configuration - *This is where the FPGA is configured with the previously listed modules*
4) Second stage bootloader - User code e.g., U-boot, typically used to load the operating system
5) Operating system (OS)

Starting from the FSBL, all subsequent stages could be encrypted and authenticated using solely manufacturer provided features. However, the goal here is to bind the software to the device using the AES-GCM core and PUF-based key storage instead of using manufacturer-provided features. Hence, a key is reconstructed by the PUF and FE after step 4 on request of U-boot. This key is used to authenticate and decrypt the software images of the OS to be run. Only after the authenticity of the image has been successfully verified, the OS is allowed to boot.

B. Secure Update of Hardware

Nowadays, FPGAs can be re-configured in parts while other parts remain unchanged. This allows to update features during

run-time. The described architecture supports secure partial reconfiguration of hardware modules in the FPGA.

As a base, the PUF, FE, AES and PR controller modules are configured. Although these modules contain no confidential information, their authenticity must be checked using manufacturer provide schemes like RSA, to guarantee no unauthorized modifications were made to the bitstream e.g., modifying the PUF design to output the key after reproduction. All possibly sensitive modules are loaded securely at a later boot stage with the help of these modules and respective process. To support this, the FPGA is divided into a static, and partially reconfigurable region. The static region, consisting of the PUF, FE, AES-GCM and PR controller, is loaded directly after the FSBL similar to the boot process in Section II-A. Sensitive or confidential modules are part of the reconfigurable region, for which a separate configuration bitstream is usually generated. This configuration bitstream is protected using the device-specific key with the new cryptographic engine.

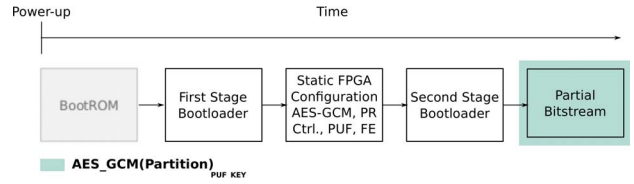

Fig. 3: Secure update of FPGA using partial reconfiguration

Figure 3 depicts the boot process as before and indicates, how this secure reconfiguration of FPGA parts is done after the second stage bootloader. Hence, after U-boot is loaded, the bitstreams containing the sensitive modules are decrypted and authenticated using the new modules. The decrypted data is passed to the PR controller to configure the partial regions of the FPGA.

C. Security Discussion

The goal of this design is to improve the security of systems by using new custom and self-controlled modules for the most important parts of a secure boot and update process, the PUF-based key storage, cryptographic engine, and software control routines, to replace manufacturer-provided modules.

As a helpful property in the proposed design, those added modules can be kept public because they do not contain confidential data, in contrary to manufacturer-provided ones, where the source code can usually not be accessed. This allows for a public review and a high confidence that no hidden functionality is contained. As an important improvement, the FPGA configuration and software routine (U-boot) which realize those parts, thus, do not need to be encrypted using the manufacturer-provided encryption module with its respective manufacturer controlled keys. This is a significant difference. All other images and FPGA configurations can be encrypted using the new modules.

However, there would still be a threat of malicious modification to the FPGA configuration in order to extract a PUF-generated key on an existing interface. Hence, the integrity and authenticity of those parts loaded in step 3 need to be ensured using manufacturer-provided means. This can be achieved using e.g. asymmetric signatures using a public key on the device. For this part, the proposed architecture still relies on manufacturer features. However, the symmetric cryptographic encryption engine, which has been the target of published attacks in the past, is successfully replaced which increases the security level significantly.

III. Implementation

This section provides a detailed description of the implementations of the previously described modules.

A. Software Routines

All the hardware building blocks are configured and managed over the AXI interface by software drivers that are patched to a standard U-boot. For key enrollment, the software drivers first configure the FE and PUF by loading a random number and initial PUF challenge. Subsequently, the PUF and FE are triggered to begin the enrollment process. Upon completion, the drivers read the helper data generated by the FE and store it in the non-volatile memory (NVM) as a text file. The helper data does not contain information which allows access to the device-individual key. Thus no additional protection mechanism for the NVM is necessary.

Key reproduction is triggered by the drivers after loading the helper data to the FE. Once the key reproduction process is done, the encrypted data is passed, block by block, to the AES-GCM core for verification. U-boot only continues the boot process if the verification succeeded. If the verification fails, U-boot aborts the boot process.

B. Physical Unclonable Function

The PUF module is used to generate a device unique bit sequence, which is further processed by the FE module (Sec. III-C), to uniquely bind a key to a device and to reconstruct the key on demand.

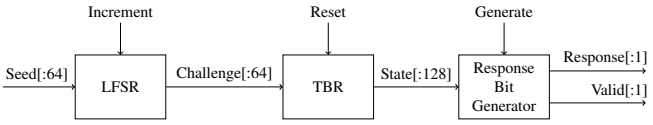

Fig. 4: PUF module building blocks for generating a response bit sequence for the fuzzy extractor module

Figure 4 shows the three building blocks of the PUF module and some of their inputs in order to illustrate the basic data and control flow. It consists of a 64-bit linear feedback shift register (LFSR), a 64-bit twisted bistable ring (TBR) and a reponse bit generator. The LFSR is used to generate a sequence of challenges to configure the TBR continuously. The TBR is a circuit specifically designed in such a way that the behavior of its state is very sensitive to the applied challenge and the manufacturing variations, which make each instance of the circuit uncontrollably unique. In order to generate the device unique bit sequence the TBR is evaluated for a number of

challenges provided by the LFSR. For each evaluation the response bit generator takes the 128-bit state of the TBR and generates a response bit and a valid bit. The valid bit together with multiple evaluations of the PUF for the same challenge is used to generate reliability information for a response bit, which is used by the FE.

The LFSR uses the polynomial $x^{64}+x^{41}+x^{17}+x^{7}$. It can be initialized with a 64-bit random challenge as its seed input. By incrementing the LFSR a new 64-bit challenge is generated based on the seed challenge or last active challenge.

The TBR is a Bistable Ring (BR), i.e. an inverter ring with an even number of inverting stages. A challenge is used to reorder the stages of the TBR, which is what "twisted" refers to in the name twisted bistable ring (TBR). This is the most important difference to the Bistable Ring PUF (BR PUF) as introduced by Chen et al. [2], where the stage instances are not reordered but are simply selected from a set of at least two redundant instances. Schuster et al. showed that reordering the stages in the TBR improves response bit statistics and resilience against attacks [18].

In order to generate a response bit for a given challenge the TBR is evaluated. For such an evaluation the challenge is applied to the TBR circuit while it is being held in reset. The reset puts the TBR into an unstable reset state in which all outputs of the inverting stages are forced to ground. By design TBRs have two stable states and the TBR state starts developing towards one of the stable states, when reset is released. After a predetermined evaluation time the response bit generator examines the state of the TBR. It determines whether the TBR is in one of it's two stable states. It is able to distinguish between the two stable states and represents the information, which state it is, in the response bit. The valid bit represents, if the TBR has stabilized or not. This method of generating response bits is functionally equivalent to the one in the experimental setup of Chen et al. [2] but could easily be replaced by the more recently developed methods by Hesselbarth et al. [5], which provide improved reliability and speed.

C. Fuzzy Extractor

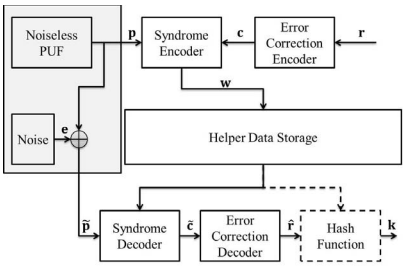

Fig. 5: Sketch of a model for a typical FC scheme with enrollment (upper part) and reconstruction (lower part). Noiseless and noisy PUF are modeled by the gray box.

Strong Error Correction (EC) is required to get a stable key with error probability of 10^{-6} down to 10^{-9} from a noisy PUF response with an error probability of 15% to 25% per bit. This is achieved by a FE in two phases: During *enrollment* (upper part of Fig. 5) a random number is encoded to a codeword $\mathbf{c}$. A syndrome encoder – or Helper Data Algorithm (HDA) – maps $\mathbf{c}$ and a noiseless PUF-response $\mathbf{p}$ to helper data $\mathbf{w}$. The HDA can be designed so that $\mathbf{w}$ does not leak information about the later extracted key $\mathbf{k}$ [16]. In this work, Fuzzy Commitment (FC) [9] is used for this purpose. Note that for good PUFs, $\mathbf{p}$ is different for another instance of the same PUF. I.e., $\mathbf{w}$ cannot be used to reproduce the key on another device, making the binding of $\mathbf{k}$ to the FPGA unique.

During *reconstruction* (lower part of Fig. 5), a syndrome decoder – or HDA – maps $\mathbf{w}$ and a noisy PUF-response $\tilde{\mathbf{p}}$ to a noisy codeword $\tilde{\mathbf{c}}$. $\tilde{\mathbf{c}}$ is decoded to derive the secret value which – for FC – is $\hat{\mathbf{r}}$ and should fulfill $\hat{\mathbf{r}} = \mathbf{r}$. If the entropy per bit in $\hat{\mathbf{r}}$ is low, e.g. due to low entropy in the PUF-response or to helper data leakage [3], $\hat{\mathbf{r}}$ can be compressed by a hash function to a shorter key $\mathbf{k}$. The hash function can also be used to prevent helper data manipulation attacks [3].

To allow for different HDAs and EC approaches, a modular design of the FE was developed in this work. Submodules for enrollment and reproduction were designed with unified interfaces for PUF data, helper data, random number, key, and control signals. This allows to switch between approaches during runtime and to easily add new approaches. The overall module is designed as an AXI slave with additional interfaces to the PUF and AES-GCM core.

During enrollment, the FE module is provided with a random number via AXI. It takes responses and reliability information from the PUF and generates helper data in a 2 KiB memory, where it can be fetched by software after the FE signaled completion.

For reproduction, the FE module is provided with the helper data. It obtains responses and reliability information from the PUF and upon success outputs the key on the designated key interface of the AES-GCM core so that it is not readable by potentially untrusted software. Note, that for enrollment and reproduction the same EC approach, HDA and challenges to the PUF must be used to get the same key.

This work contains the following two approaches, which are designed to produce a 128-bit key with error rate 10^{-6} from a PUF-response with a bit error probability of 15%:

1) DSC and Convolutional Code (DSC+CC) [7], [8], [6]: Differential Sequence Coding (DSC) is a pointer based syndrome coding scheme. During enrollment, it selects sufficiently reliable bits of the PUF-response. Pointers are stored to these bits and are encoded by Run-Length Encoding which results in a very low helper data size. An additional inversion bit is used to map the selected PUF-response bits to the codeword $\mathbf{c}$ of the subsequent code. During reproduction, the pointers are used to find the same bits in the PUF-response. Bit flips are applied to receive $\tilde{\mathbf{c}}$. $\tilde{\mathbf{c}}$ is corrected in this work using a Viterbi code with seven states.

Since helper data manipulation attacks are possible for DSC, a SPONGENT hash-function – shown to be very efficient for

this task [10] – is used for protection. It hashes $\mathbf{w}$ and its output is XORed onto $\hat{\mathbf{r}}$ to produce $\mathbf{k}$.

2) Repetition and BCH Code (Rep+BCH) [11]: The EC in this case is a combination of a $(n, k, t) = (127, 64, 10)$ BCH code as the outer code and a $(n, k) = (7, 1)$ repetition code (RC) as inner code. n is the code length, k is the dimension, t is the error correction capability. I.e., two BCH codewords or 254 RC codewords are required for a 128-bit key. Helper data are computed as $\mathbf{w} = \mathbf{c} \oplus \mathbf{p}$.

D. AES-GCM

The Galois Counter Mode (GCM) is an authenticated encryption algorithm designed to provide data confidentiality and authenticity. GCM is the preferred mode of operation for applications like high speed communication which require high-throughput and low-latency at low-overhead. This is because e.g. in case of hardware implementations, the decryption and authentication can be done in parallel, thus making them very efficient.

For this work, we implemented the NIST standard block cipher AES-128 in GCM mode for the decryption and authentication of binaries. The AES-GCM module processes blocks of 128 bits of data at a time which is sent to it over the AXI interface. Additionally, the module also takes as input a 128-bit nonce as initialization vector and optionally also unencrypted additional authentication data of arbitrary length. The 128-bit key for the AES is reconstructed using the PUF and FE and directly transferred to it. The authentication is done over the entire image and the tag is validated at the end of the image. If the tag validation fails, the software routine aborts the boot process and resets the device.

E. Partial Reconfiguration Controller

This controller module receives the decrypted partial bitstreams from the AES-GCM core and then passes it to the Xilinx internal configuration access port (ICAP) interface. The ICAP is a proprietary interface from Xilinx typically used for configuration readback and reconfiguration of the FPGA. The ICAP has direct access to the configuration memory through the configuration registers. The ICAP processes 32 bits of data per clock cycle and, thus, one block of decrypted data is processed over four clock cycles by the ICAP.

IV. Analysis of Implementation

We successfully implemented and tested the design and use-cases on the Zedboard which is a development board for the Xilinx Zynq-7000 FPGA SoC. For the secure update of hardware, a simple LED driver module was implemented as a partial module and its configurations were securely updated using the AES-GCM and PUF-based key storage. Similarly, for the secure boot of software, the `Linux-xlnx` OS was booted following its decryption and authentication using the AES-GCM and PUF-based key storage.

All the building blocks, PUF, FE, AES-GCM and PR controller, were implemented as AXI-lite IP cores with a set of AXI registers for configuration. Table I provides a summary of the post-synthesis hardware utilization of all the building blocks on the Zynq-7000. The drivers for key reproduction, decryption and authentication of software images and partial bitstreams were integrated in the `U-boot-xlnx` v2016.3. As all the IP cores were built as standard AXI IP cores, instead of proprietary interfaces, they can be easily ported to other platforms. Likewise as the software configurations were integrated in U-boot, which is extensively supported by all major vendors, porting of the complete design to other platforms can be done with a low effort.

TABLE I: Hardware utilization of IP cores

IP Core	LUTs	FF	BRAM
TBR-PUF	1048	985	-
FE	1559	1653	2.5
AES-GCM	3458	2419	-
PR Controller	98	26	-

A. Analysis of Fuzzy Extractor

For DSC+CC, a 271-bit codeword is required for the Viterbi decoder. DSC encodes every bit by at least 3 bits. Thus, pointers and flip bits require at least 813 bits of helper data. The amount of helper data depends on the reliability of the PUF-responses and is not fixed. But in our experiments, it was always below 1000 bits. In comparison, the used state-of-the-art implementation of Rep+BCH requires a fixed number of 1778 bits of helper data. Since less helper data corresponds to less non-volatile memory or less communication, this shows the efficiency of DSC+CC compared to state-of-the-art, cf. [8].

While for the state-of-the-art implementation of Rep+BCH the only known attack is a side channel attack [12], for DSC+CC – which is still under research – the only known attack is a helper data manipulation attack [7], which is circumvented by using a hash function. However, Becker et al., [1] has shown in a different setting that a pointer based approach together with a challenge-response PUF can enable machine learning attacks. Thus, the used combination of DSC+CC with a TBR PUF has to be taken with care.

B. Device Dependence of Reproduced Keys

This section shows, that the PUF is able to bind keys uniquely to a device, i.e., if the helper data from a device is used to reproduce the key on the same device it must be stable. However, if the helper data is used to reproduce a key on another identical device, the key must differ. The tests were carried out on two identical Zedboard Rev. D with the same software and hardware configurations running at room temperature without a heat sink. Ten sets each consisting of randomly chosen key and PUF challenges were used.

In a first step, the sets of key and PUF challenge are used on both boards to generate the corresponding helper data (enrolment phase). Helper data is generated for each board using the TBR-PUF with Rep+BCH as well as with DSC+CC. This resulted in 200 sets of key, challenge, and helper data for each of the two boards. In a second step, the stability of each of key was evaluated. Each helper data was taken to reproduce the

key over five successive iterations on the same device where the helper data was generated and using the same challenge. Since no single bit flip occurred in any of the keys, all keys are considered stable.

Finally, the helper data sets of the boards were exchanged. Although, all of the 400 helper data sets were used in combination with the correct challenge, each of the resulting keys differed from the correct key. This shows that the keys are bound to a specific hardware by the PUF.

C. Robustness of Key against Partial Reconfiguration

As mentioned in Section II-B, one application of the proposed design is to protect the partial bitstreams which are used to update the FPGA. Here, it is critical that the PUF behavior does not change when parts of the FPGA are reconfigured. For this, a hardware design with five partial reconfigurable modules is used. Figure 6 shows a simplified floorplan of the design used in the tests with the partially reconfigurable regions highlighted in grey and the PUF highlighted in red.

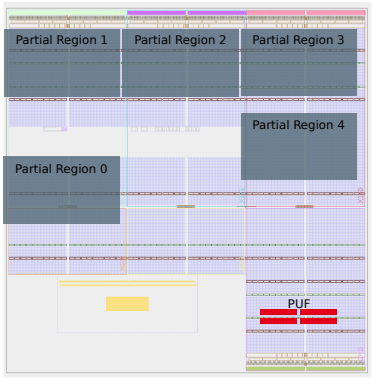

Fig. 6: Floorplan for PUF key stability test after partial reconfiguration

Five keys and five challenges are randomly chosen for the test which were then used to generate helper data on both boards using the TBR-PUF with Rep+BCH. The stability of the keys was tested as it is explained in Sec. IV-B. Next, all five reconfigurable modules were reconfigured and the keys were reproduced again over five successive iterations. Our test results show, that stable and correct keys were reproduced both before and after partial configuration.

V. Conclusion

In this work, we implemented a scheme to secure FPGA SoC configurations by retrofitting a standard FPGA SoC with a custom cryptographic core and a PUF-based secure key storage to bind the configuration and software to a specific device. It is no longer necessary to rely on manufacturer-provided (possibly insecure) encryption, although it may be still necessary to sign the static FPGA configuration by manufacturer-provided features. A future extension of this work is to study whether effects of bitstream modifications on the PUF responses can be used to secure the static FPGA configuration, which might allow to remove the need to trust manufacturer-provided features all together.

References

[1] G. T. Becker, A. Wild, and T. Güneysu. Security analysis of index-based syndrome coding for PUF-based key generation. In *Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on*, pages 20–25. IEEE, 2015.

[2] Q. Chen, G. Csaba, P. Lugli, U. Schlichtmann, and U. Rührmair. The bistable ring PUF: A new architecture for strong physical unclonable functions. In *IEEE Int. Symposium on Hardware-Oriented Security and Trust*, June 2011.

[3] 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):889–902, 2015.

[4] K. Fischer, A. Mucha, E. Hess, and F. Riess. PUF based lightweight hardware trust anchor for secure embedded systems. In *International Conference on Security and Management (SAM 2016)*, July 2016.

[5] R. Hesselbarth and G. Sigl. Fast and reliable PUF response evaluation from unsettled bistable rings. In *2016 Euromicro Conference on Digital System Design (DSD)*, pages 82–90, Aug 2016.

[6] M. Hiller, L. R. Lima, and G. Sigl. Seesaw: An area-optimized FPGA viterbi decoder for PUFs. In *Digital System Design (DSD), 2014 17th Euromicro Conference on*, pages 387–393. IEEE, 2014.

[7] M. Hiller, M. Weiner, L. Rodrigues Lima, M. Birkner, and G. Sigl. Breaking through fixed PUF block limitations with differential sequence coding and convolutional codes. In *Proceedings of the 3rd international workshop on Trustworthy embedded devices*, pages 43–54. ACM, 2013.

[8] M. Hiller, M.-D. Yu, and G. Sigl. Cherry-picking reliable PUF bits with differential sequence coding. *IEEE Transactions on Information Forensics and Security*, 11(9):2065–2076, 2016.

[9] A. Juels and M. Wattenberg. A fuzzy commitment scheme. In *Proceedings of the 6th ACM conference on Computer and communications security*, pages 28–36. ACM, 1999.

[10] B. Jungk, L. R. Lima, and M. Hiller. A systematic study of lightweight hash functions on FPGAs. In *ReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on*, pages 1–6. IEEE, 2014.

[11] R. Maes, A. Van Herrewege, and I. Verbauwhede. PUFKY: A fully functional PUF-based cryptographic key generator. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 302–319. Springer, 2012.

[12] D. Merli, F. Stumpf, and G. Sigl. Protecting PUF error correction by codeword masking. *IACR Cryptology ePrint Archive*, 2013:334, 2013.

[13] A. Moradi, A. Barenghi, T. Kasper, and C. Paar. On the vulnerability of FPGA bitstream encryption against power analysis attacks - extracting keys from Xilinx Virtex-II FPGAs. *IACR Cryptology ePrint Archive*, 2011:390, 2011.

[14] 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 *The 2013 ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA '13, February 11-13, 2013*, pages 91–100, 2013.

[15] A. Moradi and T. Schneider. Improved side-channel analysis attacks on Xilinx bitstream encryption of 5, 6, and 7 series. *IACR Cryptology ePrint Archive*, 2016:249, 2016.

[16] M. Pehl, M. Hiller, and G. Sigl. *Secret Key Generation for Physical Unclonable Functions*, chapter 13. Cambridge University Press, 2017.

[17] E. Peterson. *Leveraging Asymmetric Authentication to Enhance Security-Critical Applications Using Zynq-7000 All Programmable SoCs*. Xilinx, October 2015.

[18] D. Schuster and R. Hesselbarth. Evaluation of bistable ring PUFs using single layer neural networks. In *Trust and Trustworthy Computing*, volume 8564 of *Lecture Notes in Computer Science*. 2014.

[19] S. Skorobogatov and C. Woods. Breakthrough silicon scanning discovers backdoor in military chip. In *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, pages 23–40, 2012.

[20] S. Skorobogatov and C. Woods. In the blink of an eye: There goes your AES key. *IACR Cryptology ePrint Archive*, 2012:296, 2012.

[21] P. Swierczynski, A. Moradi, D. Oswald, and C. Paar. Physical security evaluation of the bitstream encryption mechanism of Altera Stratix II and Stratix III FPGAs. *TRETS*, 7(4):34:1–34:23, 2015.