Improving the Quality of Ring Oscillator PUFs on FPGAs

Dominik Merli
Fraunhofer Institute for Secure Information Technology
Munich, Germany
merli@sit.fraunhofer.de

Frederic Stumpf
Fraunhofer Institute for Secure Information Technology
Munich, Germany
stumpf@sit.fraunhofer.de

Claudia Eckert
Dpt. of Computer Science
Technische Universität München
Munich, Germany
eckertc@in.tum.de

ABSTRACT

Physical Unclonable Functions (PUFs) based on Ring Oscillators (ROs) are a promising primitive for FPGA security. However, the quality of their implementation depends on several design parameters. In this paper, we show that ring oscillator frequencies strongly depend on surrounding logic. Based on these findings, we propose a strategy for improving the quality of RO PUF designs by placing and comparing ROs in a chain-like structure. We also show that an increased RO runtime and RO disabling has a clear positive effect on the quality of a RO PUF. We implemented a RO PUF key generation system on an FPGA using our design strategy. Our results clearly indicate that our proposed design strategy can significantly improve the quality of a RO PUF implementation.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Security, Measurement, Experimentation

Keywords

Physical Unclonable Function (PUF), FPGA, Ring Oscillator, Fuzzy Extractor

1. INTRODUCTION

Field Programmable Gate Arrays (FPGAs) gain increasingly in importance as highly flexible alternative to Application Specific Integrated Circuits (ASICs). Their reconfiguration property enables fast prototyping and updates for hardware devices even after market launch. These advantages are already exploited in, e.g., automotive applications [16].

However, FPGAs are more vulnerable to intellectual property theft than ASICs. Bitstreams stored in external flash memory may be read out and reverse engineered to an analysable netlist [15]. This means that not only implementation details of algorithms can be disclosed but also stored cryptographic keys can be extracted.

One way to prevent reverse engineering attacks, is to encrypt the bitstream [5] before saving it to the flash memory. A decryption engine, included in a high-end FPGA, then decodes the bitstream while loading it into its internal memory. Since side-channel attacks against hardware implementations [6] improve continuously, it might be possible to successfully attack these decryption engines and break their protection. Also, attacks on ASICs and FPGAs using a Focused Ion Beam (FIB) to analyse and manipulate the physical construction of integrated circuits are already feasible today [7]. Hence, FPGA implementations are exposed to a variety of security threats.

A promising approach towards holistic embedded security arises through the application of Physical Unclonable Functions (PUFs). These physical structures enable the exploitation of unavoidable variations in manufacturing processes, e.g., lithographic processes. Signal properties, such as the propagation delay on metal lines of a microchip, depend on these variations. Circuits measuring and comparing these properties can extract PUF responses. These bit strings are noisy by nature and not uniformly distributed, but sufficiently unique to identify a silicon device. A PUF can be seen as a function mapping a response to a challenge input in an unpredictable way. In contrast to challenge-response authentication, where a PUF has to generate a big amount of responses to given challenges, i.e., Challenge-Response Pairs (CRPs), for the purpose of secure key generation, only a small number of challenges or even only a single one is applied.

PUFs based on Ring Oscillators (ROs) currently seem to be the most reasonable PUF construction to securely identify FPGAs. Their quality has been evaluated in several analyses recently [12, 14] and enjoys great interest within the scientific community.

However, in order to use PUFs for the generation of cryptographic keys, an additional module, a fuzzy extractor, is necessary. It consists of an error-correction stage to cancel noise related errors and an privacy amplification stage to provide a uniformly distributed key in the end.

WESS 2010, Scottsdale, Arizona, USA

In this contribution, we show that the quality of RO PUF implementations depends on several design parameters. We demonstrate that ring oscillator frequencies strongly depend on the logic implemented close to them. Based on this results, we define a RO pair comparison strategy which considers RO pair placement and selection, RO runtime and the enable/disable function of ROs in order to improve the quality of RO PUF responses. We implemented RO PUFs with configurable runtime and selection strategy to verify our methods. Our results show clear improvements in quality of RO PUF responses.

The remainder of this paper is organised as follows. Section 2 gives a brief overview of previous work related to our contribution. Reasons for a trend towards ring oscillator based PUFs are given in Section 3. In Section 4, we show that RO frequencies depend on their spatial location and the logic surrounding them. Our comparison strategy to improve the quality of PUF responses is explained in Section 5. An implementation analysis of our comparison strategy is found in Section 6, as well as a discussion about the results of an FPGA implementation of a RO PUF key extraction system. Our contribution is completed by a conclusion in Section 7.

2. RELATED WORK

In 2002, Gassend et al. introduced the first PUFs based on silicon structures [2]. Their implementation on an FPGA included eight self-oscillating loops consisting of 32 buffers and one inverter each. To measure the delay differences of each loop, their oscillations were counted during a period of typically 2^{20} clock cycles at 50 MHz (approx. 0.02 s). The results were clear: the measurement involves a certain level of noise, but the information extracted from present delay variations allows to uniquely identify a silicon device. Further, it was stated that environmental changes, e.g., temperature increase from 25 to 50 °C and supply voltage variation between 2.5 V and 2.7 V cause a higher level of noise in an FPGA's PUF response.

An arbiter-based PUF model was analysed by Lim et al. in 2005 [10]. Besides determining the quality of their proposed symmetric delay structure, their focus was set on authentication by utilizing CRPs depending on the physical structure. It was also shown that non-linear functions, e.g., feed forward arbiters, have to be introduced to destroy the linear dependence between challenge and response in order to render modelling attacks considerably harder.

The RO PUF was introduced by Suh and Devadas in 2007 [18]. Five inverters and one AND-gate, to enable the circulation, form a single ring oscillator. In the cited document, an implementation of 1024 oscillators is described, where all oscillator outputs are connected to inputs of two multiplexers to select a pair of oscillators and compare the counter values, representing the oscillator frequencies. To enhance the reliability of compared pairs, a 1-out-of-8 scheme is applied, which chooses the best pair of oscillators out of eight pairs available. It requires seven times more bits to generate a response, but also limits the probability of different responses from the same chip to 0.48% even for worst-case environmental changes. Maiti and Schaumont pursued a similar, but more optimized approach in [13] by introducing configurable ring oscillators. During the enrolment phase, the optimal configuration for largest distances of compared oscillator frequencies is determined and saved (3 bits per oscillator). They also suggest to compare adjacent RO pairs by controlled oscillator placement, which we will extend to a chaining strategy for optimized RO comparison.

In [3], PUFs based on SRAM memories were proposed to provide an FPGA intrinsic security primitive. The great advantage is that no FPGA resources are occupied by this PUF construction, but unfortunately most FPGA devices automatically reset all SRAM blocks after power-up. A similar approach was described in [11], where random start-up values of flip-flops are exploited. However, the extraction of these start-up values has to be done by the configuration controller or by a specific circuit, which has to be reset externally after every start-up, which makes it impractical.

Another idea, the Butterfly PUF, was presented by Kumar et al. in 2008 [9]. Here, bits are generated from two cross-coupled latches, which leave their meta-stable state after releasing the excite signal and tend to output a high or low voltage level depending on manufacturing variations of involved latches and wires.

In [14], Morozov et al. analysed the influence of FPGA internal routing for proposed FPGA PUF types and conclude that Arbiter PUFs and Butterfly PUFs are not as appropriate for stable identification as RO PUFs.

Fuzzy extractor implementations on FPGAs have been proposed by Bösch et al. in 2008 [1]. They utilise concatenated error-correcting codes and a family of universal hash functions (Toeplitz hash) to generate uniformly distributed and noise-free cryptographic keys.

3. FPGA PUF CONSTRUCTIONS

SRAM PUFs offer a good basis for key generation, but as they are not realisable on every FPGA, they are not suitable as a general FPGA PUF primitive. Flip-Flop PUFs require manual bitstream modifications and a special read-out technique, which makes them impractical. During our research, we investigated Arbiter PUFs, Butterfly PUFs and Ring Oscillator PUFs, but focused on the latter for the following reason. FPGA routing renders symmetric Arbiter PUF and Butterfly PUF design very hard as explained in Section 3.2.

One should notice, that the quality and possibilities of an FPGA PUF implementation always depend on the tools provided by FPGA manufacturers and the internal low-level design of their devices. During the following paragraphs, we concentrate on tools and structures of a Xilinx Spartan-3E device. However, it would be an important step to evaluate advantages and disadvantages of PUF design on other FPGA platforms, e.g., Altera devices, with different low-level concepts and design tools.

3.1 Ring Oscillator PUFs

RO PUFs are based on manufacturing variations of silicon devices. The frequency of ring oscillators, built of an odd number of inverters, depends on these manufacturing variations. The general structure proposed by Suh and Devadas [18] is shown in Figure 1. It consists of several ROs from which two are selected by a challenge signal using a multiplexer and are each connected to a counter. During the enable signal is high, the counters increment with the oscillators' frequencies, representing their frequencies when disabling the oscillators again. A final relative comparison of the counter values generates a logical 0 or 1 for this RO pair, depending on which oscillator had the higher frequency.

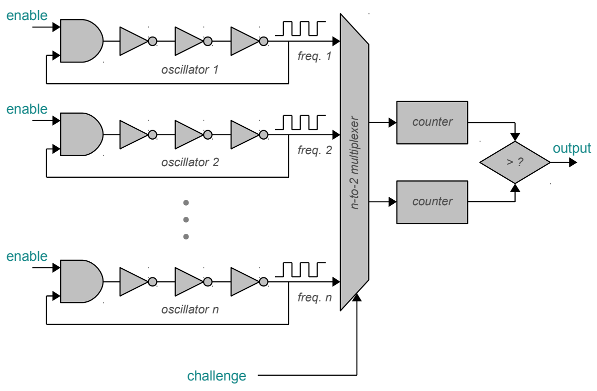

Figure 1: RO PUF structure

3.2 Hard Macro Design

The key to solid PUF design is the usage of lowest-level FPGA engineering techniques. The Xilinx software FPGA Editor enables the creation of hard macros, which are manually placed, routed and configured designs, which can be instantiated multiple times in an FPGA design. This feature can be exploited to design an exactly defined ring oscillator and instantiate it several times. Figure 2 shows two nearly identical three-inverter ring oscillators, implemented in a single Configurable Logic Block (CLB), the basic division of a Xilinx FPGA.

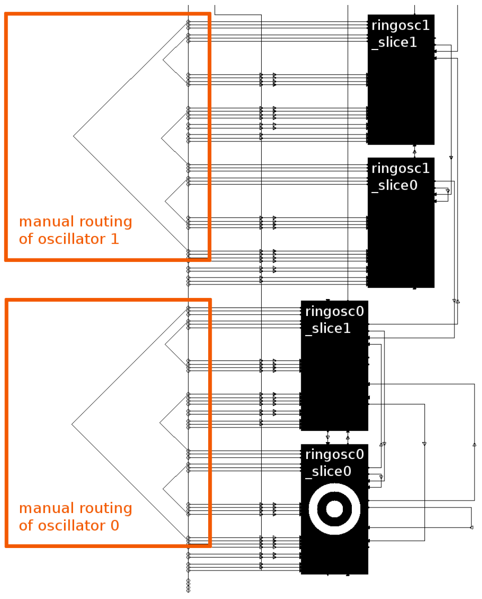

Figure 2: Hard macro with local routing for two three-inverter ring oscillators in one CLB

To ensure identical routing for all RO instances, only local routing (within a CLB) is allowed, which can be achieved with three-inverter ROs in Spartan-3E devices. Arbiter PUF and Butterfly PUF architectures are not satisfied by instantiation of identically layout. They require symmetric routing on local and non-local wires, which is very hard to achieve, as shown in [14]. Therefore, RO PUF are currently the best choice for general FPGA PUF designs.

4. SPATIAL FREQUENCY DEPENDENCE

In [13] Maiti and Schaumont showed that ring oscillator frequencies depend on their location on an FPGA die, e.g., frequencies of ROs at the edges of an FPGA are slower than central ROs. We show that logic placed around ROs has an influence on their frequencies and also causes a significant spatial dependence.

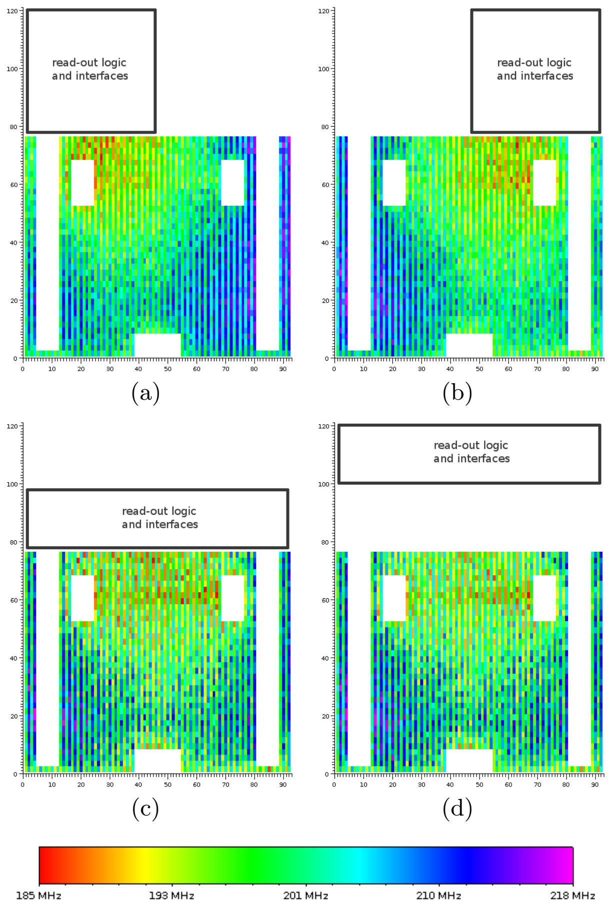

Figure 3: Spatial frequency distribution of 2712 ROs on a Spartan-3E, influenced by surrounding logic

In order to analyse the spatial dependence of RO frequencies, we instantiated an array of 2712 ring oscillators with the hard macro shown in Figure 2 to cover more than half of a Spartan-3E die. Additionally, a RS232 interface, counter and comparison logic was implemented to allow measurements on all RO frequencies. To analyse the internal environmental changes within an FPGA depending on surrounding logic, we restricted us to four example cases where the placement of all interface and measurement logic, a disturbance source, was predefined. In the first case, it was limited to the left corner of the empty floorplan space as depicted in Figure 3(a), in the second one, only the right corner was allowed, see Figure 3(b). The third case was the area directly at the border of the RO array as shown in Figure 3(c) and the fourth one was as far as possible away from the array of oscillators as depicted in Figure 3(d). Of course, other cases based on other logic can be defined for future analyses. However, we believe that these cases are representative and allow drawing conclusions about spatial dependence of RO frequencies.

In Figure 3, the frequencies of all implemented oscillators are depicted by different colours at the floorplan position of the corresponding RO. The presented results show that the position of interfaces and read-out logic significantly alters the intra-die conditions.

The first characteristics one observes, are the circular areas of lower frequencies around the position of the read-out and measurement logic. Our analysis results show that the communication and read-out logic 'radiates' a condition to decrease frequencies, for each case depicted in Figure 3. One possible effect could be a local temperature rise which originates from flowing currents in the comparison logic and drops with increasing distance to it. Our measurements do not allow conclusions about which effect, e.g., temperature, exactly causes frequency deviations. However, they clearly indicate that frequencies of ROs depend on their spatial location and on surrounding logic.

Second, a stripe pattern is visible over the whole area. This depicts the fact, that the ring oscillator implementations of Figure 2, although nearly identical, still show a noticeable mean frequency difference of approximately 10 MHz. Therefore, only exactly identically routed oscillators may be compared to avoid predetermined comparison results.

When looking closer, the locations at the bottom borders of Figures 3(a)-3(d) again show slower frequencies instead of higher although they have the largest distance to the measurement logic. This might result from higher energy densities at the die borders because of nearby output buffer locations or local heat accumulation.

Comparing Figures 3(c) and 3(d), one will find that moving the logic farther away only shows little improvement.

Since the effects influence not only the edge of our oscillator array, the idea of placing dummy cells around it, as proposed in [17], is not applicable here. A comparison strategy is needed to overcome this disturbances.

5. RO COMPARISON STRATEGY

In previous publications, only little focus was put on the circuitry for creating a PUF response from comparing oscillators. We identified three parameters, which influence the quality and resource usage of the overall system and are therefore critical for area, time or power constrained devices. First, we present a place and map method to avoid foreseeable comparison results of oscillator pairs because of their location. It allows to extract $n-1$ bits from n oscillators while comparing only neighbouring ROs. Second, we show that an increasing RO comparison runtime has a positive effect on measurement errors of oscillator frequencies. Third, advantages and disadvantages for a mechanism to only enable the actually measured ring oscillators are balanced. However, other parameters, such as placement and design of frequency counters and RO output multiplexers may influence the quality of a RO PUF, but are not considered in our strategies.

5.1 RO Pairs

In [13] the authors extracted $n-1$ bits from n ROs. In order to avoid correlation of response bits, i.e., no comparison result can be foreseen when knowing all other comparison results, we use this approach as basis for our strategy. As shown in [13], the quality of RO PUFs increases with controlled RO placement. We extend this idea to a chain-like placement and comparison strategy.

Based on our observations in Section 4, we conclude that only completely identical ROs, with identical hard macros, are allowed to be used for comparison in order to avoid foreseeable comparison results. However, it might be necessary to use two RO types as shown in the hard macro of Figure 2, because of design specific area constraints. In this case, two PUF arrays of different types have to be considered. Only RO pairs of the same type are to be compared to generate the overall PUF response. For example, if a response of 128 bits is necessary, it can be achieved by 129 ROs of type 0, but if this is not possible for any reasons, a combined solution of 65 type 0 and 65 type 1 ROs is imaginable. One should notice that m types of ROs in a PUF design require $n+m$ oscillators to extract n bits.

In order to create a RO PUF challenge, a list of RO pairs to be compared has to be chosen carefully. Closely located pairs are exposed to similar spatial conditions. Therefore, we suggest to compare only oscillators which are direct neighbours. To achieve this while still enabling a simple counter based selection mechanism, we propose a chain-like approach in the following paragraphs.

We label all available n ring oscillators with 0 to $n-1$. To simplify and optimize oscillator comparison, a counter runs from 0 to $n-2$ and always selects two consecutive oscillator outputs with a multiplexer. Step by step, these pairs are compared and $n-1$ bits are extracted.

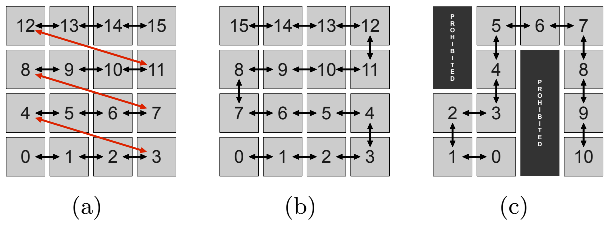

Figure 4: Spatial RO mapping strategies

With our results of Section 4 in mind, we also propose a spatial chain mapping of ROs. Since ROs are often organised in arrays as shown in Figures 4(a) and 4(b) or even in unshaped structures as depicted in Figure 4(c), mapping the numbered oscillators to the physical layer of an FPGA demands a strict pattern. Figure 4(a) shows that wrong physical mapping can cause long distance comparisons. Based on our results of Section 4, we believe that these long distance comparisons might produce predetermined results because of different dependence on local conditions. Chain-like mapping as shown in Figures 4(b) and 4(c) achieves comparison pairs which are exposed to the same spatial intra-die conditions and therefore minimizes the risk of location predefined comparison results. Many FPGA design tools allow controlled placement of hard macro ROs and therefore enable a low footprint RO PUF challenge circuitry. This guarantees an optimized selection of RO pairs to compare with a single counter.

5.2 RO Runtime

In order to enable exploitation of frequency variations of ROs to extract unique bit strings, one has to precisely measure those frequencies. For RO PUFs, frequency measurement is performed by incrementing counters using a ring oscillator's output during a specified time. Higher frequencies lead to higher counter values.

We show that the runtime, during which a RO pair is compared, has significant influences on the present measurement error and therefore on the quality of a RO PUF. A first requirement is, that runtime and corresponding counter sizes are matched with the estimated maximal RO frequency, in order to avoid counter overflows. Further, the runtime specification has to be balanced between reliable RO discrimination and fast response extraction. Especially the latter might be critical for embedded cryptographic applications that are limited in time.

If f_{clk} is a system's clock frequency and n_{clkcyc} is the number of clock cycles during which ROs are enabled, the resulting maximum measurement error $e_{measure,max}$ can be calculated as follows:

$$e_{measure,max} = \pm \frac{f_{clk}}{2 \cdot n_{clkcyc}} \qquad (1)$$

Assume a circuit with a very short read-out time, which enables two counters only for a single cycle of a 50 MHz clock. Measurements of this circuit resulting in counter values of $1, 2, 3, ..., x$ would represent frequencies of $50, 100, 150, ..., x \cdot 50$ MHz respectively. This means, all frequencies between 25 and 75 MHz will be mapped to 50 MHz and so forth, resulting in a maximum measurement error of $e_{measure,max} =$ 25 MHz. For an array of hard macro ROs, assuming a RO frequency range from 175 to 225 MHz with a mean value of 200 MHz, an error of 25 MHz would be unacceptable.

Therefore, the measurement time for RO comparisons should always be chosen in respect to a carefully defined maximum measurement error. For instance, if a maximum error of 0.1 MHz is required on a 50 MHz clocked system, the runtime for each RO pair has to be at least $n_{clkcyc} = \frac{50\,\text{MHz}}{2 \cdot 0.1\,\text{MHz}} =$ 250 cycles. This defines the lower bound on RO comparison runtime.

One should also notice that enable signal propagation delay deviations can cause comparison errors. By increasing measurement time, these delays lose their significance.

Upper bounds on RO runtime are always determined by the application using RO PUFs, since the evaluation time of a large number of RO pairs might be time-critical. For example, an evaluation of 1001 ROs (1000 RO pairs), based on the parameters calculated in the previous example, would last at least $1000 \cdot 250 \cdot 20 \cdot 10^{-9}\,\text{s} = 5 \cdot 10^{-3}\,\text{s}$. The only way to accelerate evaluation, would be to measure and compare several RO pairs in parallel, which again causes a larger hardware footprint.

5.3 RO Disabling

In Section 4, we showed that RO frequencies strongly depend on surrounding logic and the resulting local environmental changes. Therefore, we believe that oscillators also depend on neighbouring ROs. As a result, we suppose that a RO enabling/disabling strategy can improve quality.

A trivial enable/disable strategy is activating all oscillators with a single enable signal every time RO pairs are compared as depicted in Figure 5(a). However, an array of running ROs might cause changes in local temperatures and electromagnetic emission. These parameters might again alter the intra-die environment and increase the present noise level. This results in decreased measurement precision.

We only enable currently compared ROs and keep all others disabled as shown in Figure 5(b). The advantages of implementing such an enabling module are power consumption reduction for battery powered devices and less noise on RO pair comparisons. A disadvantage, shown in the following implementation results, is the additional, not negligible, need of hardware resources to build this selection logic.

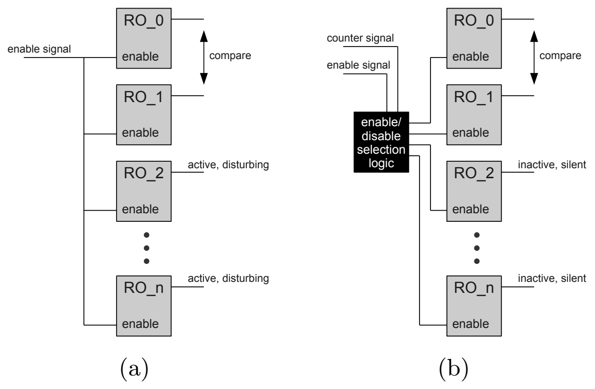

Figure 5: RO enable/disable strategies

6. IMPLEMENTATION RESULTS

We implemented a group of RO PUFs on a Xilinx Spartan-3E FPGA, to demonstrate the positive effects of our proposed strategies on the quality of RO PUFs. Further, we implemented a key extraction system based on our RO PUFs and a fuzzy extractor architecture as an example application.

6.1 RO Quality

To monitor the quality of our RO PUFs, we used two criteria, reliability and uniqueness, defined in [12]. However, for reliability, Maiti and Schaumont only consider Hamming distances to a reference response, although every response could be a reference response. For uniqueness, only one response per device is used, but it is not stated how to select or calculate this response. In order to obtain complete results, we extend the proposed definitions to consider all distances between all responses.

6.1.1 Reliability

Reliability characterizes intra-device variations of PUF responses R. To measure it, each of m PUF devices is evaluated x times. Depending on the test purpose, environmental conditions are kept stable or altered accordingly. Then, the average of the percentage Hamming distance $\frac{d(R_{j,u},R_{j,v})}{n} \times 100\%$ between each of the measured n-bit responses $R_{j,u}$ and $R_{j,v}$ of each device j is calculated, where u and v run from 1 to x to address every recorded response. Since reliability is a positive measure, we define a maximum of 100% as the most desirable result and subtract the calculated average from it. Reliability is then defined as follows:

$$R_{PUF} = 100\% - \frac{1}{m \times \sum_{i=1}^{x-1} i} \times \sum_{j=1}^{m} \sum_{u=1}^{x-1} \sum_{v=u+1}^{x} \frac{d(R_{j,u}, R_{j,v})}{n} \times 100\% \quad (2)$$

Reliability is an average property, but when specifying error-correcting methods for a PUF system, also the maximum response deviation is significant. It is defined as the maximum percentage Hamming distance between two intra-device responses out of all devices.

6.1.2 Uniqueness

Uniqueness characterizes inter-device variations of PUF responses R. Its calculation can be done with the same x responses of each of the m PUF devices as described for reliability. This estimate is defined by the average of the percentage Hamming distance $\frac{d(R_{i,u},R_{j,v})}{n} \times 100\%$ between all n-bit responses $R_{i,u}$ and $R_{j,v}$ of two different devices i and j out of all m devices. u and v run from 1 to x to address every recorded response of each device. Uniqueness is then defined by the following equation, with an optimal value of 50%:

$$U_{PUF} = \frac{1}{x^2 \times \sum_{k=1}^{m-1} k} \sum_{i=1}^{m-1} \sum_{u=1}^{x} \sum_{j=i+1}^{m} \sum_{v=1}^{x} \frac{d(R_{i,u}, R_{j,v})}{n} \times 100\% \quad (3)$$

Uniqueness values below 50% indicate correlation between PUF responses and therefore lower PUF quality.

6.2 PUF Test Array: 129 ROs, 128 RO Pairs

We implemented five instances of 129 ROs in non-overlapping areas in each of two Spartan-3E1200 devices. These ten PUF devices represent only a small test group, but they already allow to show improvements with our strategies. The available 128 RO pairs enable the extraction of 128-bit responses.

First, the 129 oscillators were placed in random order within the given area of each PUF 'device'. This allows us to compare our chain placement approach with random placement.

Second, all oscillators were physically placed by our proposed chain-like system as shown in Figure 4(b). Hence, the challenge, a list of RO pairs, is created by a counter and only direct neighbouring oscillators are compared.

Our configurable VHDL design allowed to synthesize and test all PUFs with specified RO runtimes and different enable/disable strategies. Runtime was increased from 10 cycles at 50 MHz to 10240 cycles. For three chosen runtimes, the feature of RO enable selection was activated. Environmental conditions were not altered intentionally, because the main focus of this test was to show the effects of our proposed strategy, even under normal conditions.

Table 1: Quality improvement by chain-like placement (RO runtime: 640 clock cycles)

Placement	Reliability	Maximum response deviation	Uniqueness
random	99.20%	4.69%	43.40%
chain	98.28%	6.25%	48.51%

The comparison of random and chain-like placement is shown in Table 1. Uniqueness improves by approx. 5% with the use of our chaining strategy. Using random placement, where long distance comparisons are likely to happen, causes a loss in uniqueness and provides more foreseeable comparison results. Reliability and maximum response deviation seem to be better in the random case, but this is only because more RO comparison results stay at a fixed, placement dependent value.

Table 2 shows the results for our runtime and enable/disable strategies. Tests with the smallest runtime configuration of 10 cycles showed an unacceptable uniqueness value of 30.42%. Uniqueness improves while increasing measurement time until a value of 640 cycles, which equals a measurement error of ±0.04 MHz. For longer runtimes, no improvements in uniqueness are observed.

Curiously, the enable/disable selection logic does only have a slightly positive effect on reliability, but also a slightly negative effect on uniqueness. On the other hand, the maximum response deviation is lowered by almost 1% every time the selection module is activated. However, the large resources required for the module make it only interesting for reliability critical applications.

Regarding runtime, between 40 and 10240 cycles, an improvement of approximately 5% in maximum response deviation is observable. This confirms the effectiveness of increased measurement time in order to obtain more reliable PUFs with a higher quality.

Hardware resources for larger comparison counters and re-

Table 2: RO PUF quality results (*includes an RS232 interface: 93 slices, 75 registers and 124 LUTs)

Runtime (50 MHz cycles)	Maximum measurement error	Enable signal	Counter size	Reliability	Maximum response deviation	Uniqueness	Read-out time (for 128 bits, at 50 MHz)	Spartan-3E (slices, registers, LUTs)*
10	± 2.50 MHz	all	6-bit	98.39%	7.81%	30.42%	≈ 0.03 ms	499, 232, 847
40	± 0.63 MHz	all	8-bit	97.85%	10.16%	45.55%	≈ 0.10 ms	502, 238, 858
40	± 0.63 MHz	select	8-bit	97.94%	9.36%	45.45%	≈ 0.10 ms	608, 238, 1067
160	± 0.16 MHz	all	10-bit	97.45%	8.59%	48.02%	≈ 0.41 ms	508, 245, 866
640	± 0.04 MHz	all	12-bit	98.28%	6.25%	48.51%	≈ 1.64 ms	512, 251, 874
640	± 0.04 MHz	select	12-bit	98.96%	5.47%	47.05%	≈ 1.64 ms	617, 250, 1082
2560	± 9.77 kHz	all	14-bit	98.65%	5.47%	48.65%	≈ 6.55 ms	516, 257, 882
10240	± 2.44 kHz	all	16-bit	99.16%	5.47%	47.82%	≈ 26.21 ms	521, 264, 890
10240	± 2.44 kHz	select	16-bit	99.19%	4.69%	46.38%	≈ 26.21 ms	624, 262, 1101

lated logic are negligible compared to the enable logic module, but the listed read-out times might become critical for some applications. Time optimized read-out strategies, other than parallel read-out with a higher hardware footprint, have to be investigated in future.

6.3 128-bit RO PUF Key Extraction System

In order to demonstrate an example FPGA application based on RO PUFs, we implemented a key extraction system. Therefore, we also show how many RO pairs are needed to generate a 128-bit key reliably.

RO PUFs can be used to generate cryptographic keys from physical structures without actually storing the key. We implemented a 128-bit RO PUF key extraction system to analyse its applicability and its resource requirements.

Noisy, non-uniformly distributed sources, such as RO PUF responses, have to be processed by a fuzzy extractor to obtain stable, uniformly distributed keys. These modules consist of two stages, an error-correcting, noise eliminating stage and a redistributing hashing stage. The first step is usually based on error-correcting code constructions, which are capable of detecting and correcting a defined number of bit errors. The corrected responses are then transformed by a universal hash function to achieve a uniform distribution of keys.

In Figure 6, our hardware architecture modules are depicted, which were needed to implement the helper data protocol of [1]. First, the protocol requires an enrolment phase where a reference PUF response is read and corresponding helper data is generated. Afterwards, the created key can be reconstructed any time by applying the saved helper data to the PUF system.

To the best of our knowledge, no statistical entropy analyses of RO PUFs have been carried out yet. In [9], an entropy for Butterfly PUFs is assumed for the calculation of required PUF cells. Since our aim is the demonstration of an example key extraction application, we also assume an arbitrarily chosen entropy of 0.95. Beginning with the requirement of generating a 128-bit key, we find $\frac{128}{0.95} \approx 135$ bits needed to be hashed. As described in [8] and [1], we utilize a hardware design of a Toeplitz hash function to achieve a uniform distribution.

In [1], Bösch et al. proposed hardware architectures for Reed-Muller and Golay code decoders on FPGAs. The authors state that a combination of these codes and repetition codes are very area efficient solutions. An obvious disadvantage is, that they require approx. double the number of source bits than BCH code designs, because their error-correction properties are not as strong as those of BCH codes.

In order to save source bits, which again require hardware resources in case of a RO PUF, we compared several versions of BCH codes and concatenations of BCH and repetition codes. We calculated the residual error probability as defined in [1], which should be less than the Failure In Time (FIT) rate of the used FPGA. A value below 10^{-6} is a good pessimistic requirement. At the same time, the second parameter - required source bits - should be kept as small as possible. Based on our results above, we assume a bit error rate $p_b = 0.08$ for a reliable RO PUF construction. Our code comparison results are listed in Table 3. The BCH(255,37,45) code shows a low residual error probability while requiring an acceptable number of 1020 source bits.

Table 3: Error probabilities and source bits of possible error-correcting codes (for 135 bits)

Error-correcting code	Error probability	Source bits
BCH(127,15,27)	$0.75 \cdot 10^{-6}$	1143
BCH(127,22,23)	$0.66 \cdot 10^{-4}$	889
BCH(255,37,45)	$0.18 \cdot 10^{-6}$	1020
BCH(255,71,29)	$0.22 \cdot 10^{-1}$	510
BCH(511,76,85)	$0.57 \cdot 10^{-10}$	1022
BCH(511,139,54)	$0.16 \cdot 10^{-1}$	511
BCH(1023,143,126)	$0.75 \cdot 10^{-6}$	1023
REP(3,1,1) + BCH(255,71,29)	$0.11 \cdot 10^{-14}$	1530
REP(5,1,1) + BCH(255,139,15)	0.31	1275

In [4], a tool for generating BCH codecs in VHDL language was designed. We used it to generate a BCH(255,37,45) encoder and decoder, which represent our error correction module.

We implemented 1021 ROs with an optimized read-out circuitry to obtain a high quality PUF. Further, our designed fuzzy extractor system consists of a universal hash function,

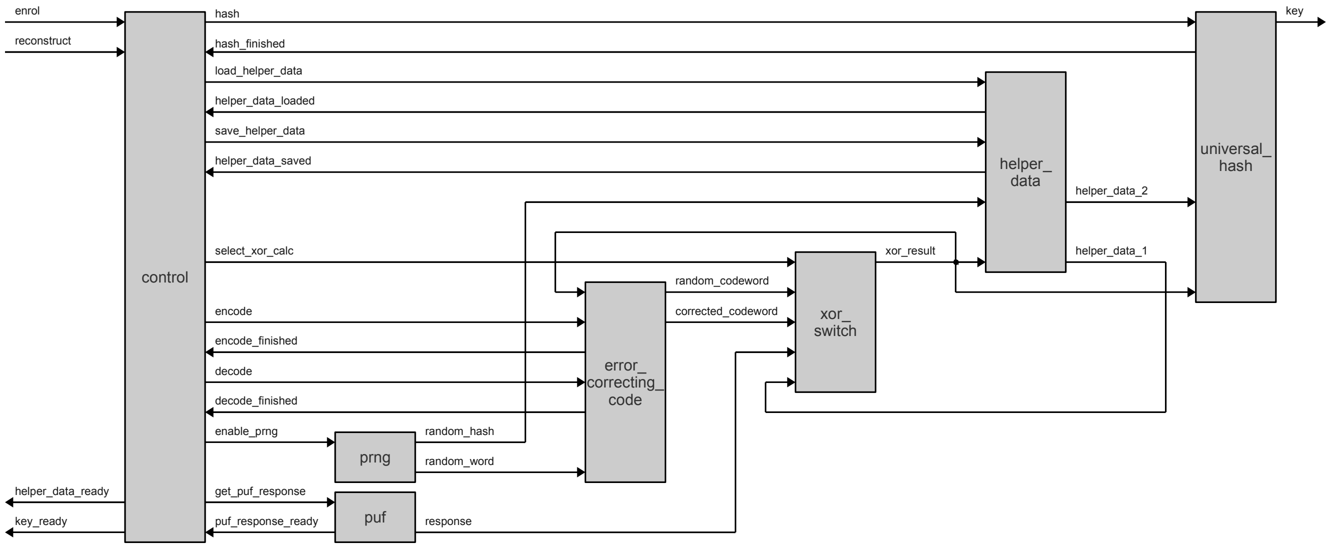

Figure 6: Key extraction system modules

an error-correcting stage and a key extraction algorithm similar to the one explained in [1]. For communication purposes, an RS232 interface was added to our architecture. Xilinx ISE synthesis results for a Spartan-3E1200 FPGA led to the following utilization: 6622 registers (38%), 13490 LUTs (77%), 8429 slices (97%). The fraction of the error-correcting module is estimated to be approx. 42% of the required slices, while the RO array occupies approx. 10%. The hash implementation requires approx. 4% of this design and the wide XOR module approx. 9%. A disadvantage of this design is that the helper data is also saved within FPGA resources and occupies approx. 10%. The RS232 interface, which has access to the PUF response, helper data and generated key, uses another approx. 3%. Approx. 22% consist of control logic, a pseudo random number generator and module interfaces.

The size of this construction, admittedly not fully optimized yet, is unacceptable for low-cost FPGAs, since it does not leave much resources for the actual application. The error-correcting code module occupies a significant fraction, therefore reasonable trade-off between required source bits and area efficient error-correcting code implementations could improve the resource usage of key extraction systems. Further, a even higher quality of PUF responses could lower the number of required source bits. An optimized control system and a trade-off between serial and parallel processing could reduce the amount of required FPGA resources, also. However, there is a strong need to investigate PUF systems with a significantly smaller footprint to enhance the applicability of PUFs in small and middle size FPGAs.

7. CONCLUSION

We have shown that RO frequencies strongly depend on their spatial location on an FPGA because of surrounding logic altering intra-die conditions. Based on this fact, we proposed a chain-like mapping strategy for controlled physical placement of oscillators. We were able to demonstrate RO PUF quality improvements by adjusting RO runtime and usage of an enable/disable logic module. Our contribution is completed by an implementation of a RO PUF key extraction system.

8. REFERENCES

[1] C. Bösch, J. Guajardo, A.-R. Sadeghi, J. Shokrollahi, and P. Tuyls. Efficient helper data key extractor on fpgas. In *CHES '08: Proceedings of the 10th International Workshop on Cryptographic Hardware and Embedded Systems*, pages 181–197, Berlin, Heidelberg, 2008. Springer-Verlag.

[2] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas. Silicon physical random functions. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 148–160, New York, NY, USA, 2002. ACM.

[3] J. Guajardo, S. S. Kumar, G. J. Schrijen, and P. Tuyls. Fpga intrinsic pufs and their use for ip protection. In P. Paillier and I. Verbauwhede, editors, *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 63–80. Springer, 2007.

[4] E. Jamro. The design of a vhdl based synthesis tool for bch codecs. Master's thesis, School of Engineering, The University of Huddersfield, Sep 1997.

[5] T. Kean. Secure configuration of field programmable gate arrays. In *FPL '01: Proceedings of the 11th International Conference on Field-Programmable Logic and Applications*, pages 142–151, London, UK, 2001. Springer-Verlag.

[6] P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[7] O. Kömmerling and M. G. Kuhn. Design principles for tamper-resistant smartcard processors. In *WOST'99:*

Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology, pages 2–2, Berkeley, CA, USA, 1999. USENIX Association.

[8] H. Krawczyk. Lfsr-based hashing and authentication. In *CRYPTO '94: Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology*, pages 129–139, London, UK, 1994. Springer-Verlag.

[9] S. S. Kumar, J. Guajardo, R. Maes, G. J. Schrijen, and P. Tuyls. Extended abstract: The butterfly puf protecting ip on every fpga. *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on Hardware-Oriented Security and Trust (HOST)*, pages 67–70, June 2008.

[10] D. Lim, J. W. Lee, B. Gassend, G. E. Suh, M. van Dijk, and S. Devadas. Extracting secret keys from integrated circuits. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 13(10):1200–1205, December 2005.

[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 (WISSec 2008)*, page 17, Eindhoven,NL, 2008.

[12] A. Maiti, J. Casarona, L. McHale, and P. Schaumont. A large scal characterization of ro-puf. In *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 66–71, 2010.

[13] A. Maiti and P. Schaumont. Improving the quality of a physical unclonable function using configurable ring oscillators. In *19th International Conference on Field Programmable Logic and Applications (FPL), 2009. FPL '09.*, 2009.

[14] S. Morozov, A. Maiti, and P. Schaumont. A comparative analysis of delay based puf implementations on fpga. Cryptology ePrint Archive, Report 2009/629, 2009. http://eprint.iacr.org/.

[15] J.-B. Note and E. Rannaud. From the bitstream to the netlist. In *FPGA '08: Proceedings of the 16th international ACM/SIGDA symposium on Field programmable gate arrays*, pages 264–264, New York, NY, USA, 2008. ACM.

[16] J. Saad, A. Baghdadi, and F. Bodereau. Fpga-based radar signal processing for automotive driver assistance system. In *RSP '09: Proceedings of the 2009 IEEE/IFIP International Symposium on Rapid System Prototyping*, pages 196–199, Washington, DC, USA, 2009. IEEE Computer Society.

[17] Y. Su, J. Holleman, and B. P. Otis. A digital 1.6 pj/bit chip identification circuit using process variations. *IEEE JOURNAL OF SOLID-STATE CIRCUITS*, 43(1):69–77, Jan 2008.

[18] G. E. Suh and S. Devadas. Physical unclonable functions for device authentication and secret key generation. *Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE*, pages 9–14, 2007.