Analysis of Ring Oscillator PUFs on 60nm FPGAs

Linus Feiten, Andreas Spilla, Matthias Sauer, Tobias Schubert, Bernd Becker

Albert-Ludwigs-University of Freiburg
Georges-Köhler-Allee 051
79110 Freiburg, Germany
{feiten|spilla|sauerm|schubert|becker}@informatik.uni-freiburg.de

Abstract—In hardware security and trusted computing it is often desired to uniquely and unambiguously identify a device among several others of the same brand. Physically unclonable functions (PUFs) take advantage of subtle variations in the devices’ production process to achieve this. A ring oscillator (RO) PUF exploits differing time delays of circuits to yield a unique response from each device.

The implementation of RO PUFs on FPGAs has been widely discussed but most experiments have been conducted on Xilinx FPGAs. In this paper we are reporting statistical results from an analysis spanning 20 equivalent Altera FPGAs. The presented results include the PUF quality’s dependency on different parameters like RO length and placement on the FPGA. We identify the optimal RO length of 16 Logic Elements (LE) and show some specific placement cases for which the otherwise very good PUF quality decreases drastically.

Keywords: Hardware Security, Trusted Computing, Physically Unclonable Function, Ring Oscillator, FPGA

I. INTRODUCTION

There are many scenarios in which computer devices need to be identified unambiguously. This is e.g. the case when the owner of a device uses it to get access to a resource, be it the physical opening of a door or the digital access to using a licensed software. Such scenarios imply that not only the device needs to be uniquely identifiable but furthermore that it is practically impossible to forge a second device with the same characteristics.

Physically unclonable functions (PUFs, e.g. [1]) are a promising technology to fulfil these demands. They make use of subtle physical differences in the devices’ fabric, which happen randomly during the fabrication process and cannot be manipulated, hence “unclonable”. Thus every device - albeit constructed identically - carries a unique fingerprint in its fabric left by the random process variations. The challenge of creating a PUF lies in eliciting these fingerprint characteristics in a way such that each device can always be identified (reliability) and none is ever mistaken for a different device (uniqueness). The subtleties of process variation effects make this difficult, because several sources of noise may predominate them.

Most PUFs realise a set of challenge response pairs (CRPs), i.e. each challenge of a CRP is the PUF input and the corresponding response is the respective PUF output. In order to make forging practically impossible, the number of CRPs should be so high that a counterfeiter cannot exhaustively enumerate and save all CRPs in a counterfeit device. An attack resilient PUF can only be forged if the physical fingerprint characteristics on which it operates are known. These, however, are embodied in the device’s fabric itself and every extraction attempt would significantly alter or even destroy them.

Ring oscillator (RO) PUFs use the delay of signals in integrated circuits to create unique and reliable responses. Their structure allows them to be easily implemented on FPGAs as shown in [2] and refined in several studies like [3], [4], [5], [6]. These experiments were all conducted using Xilinx FPGAs (with 90 nm technology). To our knowledge, only the authors of [7] and [8] have published their PUF examinations of Altera FPGAs (90 and 65 nm). Furthermore, most studies use rather small sets of test devices (< 16). An exception are [5] and [6], in which 125 FPGAs were tested. These studies, however, do not parametrise any structural characteristics of the circuit layout and merely examine one design, parametrising ambient temperature and supply voltage.

In this paper, we are presenting the findings of an analysis done with 20 Altera Cyclone IV FPGAs (60 nm) on DE0-Nano development boards. We have parametrised and tested the following properties: RO size, placement of the ROs on the chip and vicinity of the ROs to the controller logic. The detailed results are shown in Section IV. Before, we provide a brief introduction to RO PUFs and our set-up in Section II and a description of the metrics used to measure the PUF quality in Section III. Section V concludes the paper.

II. SET-UP

A. Ring Oscillator PUF

\[
\text{(a)}
\]

\[
\text{(b)}
\]

Figure 1. Ring oscillator: a) with an odd number of inverters. b) with one inverter and delaying drivers.
Figure 1 shows a Ring Oscillator (RO) circuit. It typically consists of an odd number of inverters arranged in a cycle (Figure 1a). The enable input activates the oscillation; i.e. the output of the RO switches between 0 and 1 with a frequency depending on the delays of the inverters. These delays and hence the ROs’ frequencies are determined by random process variations, which allows the utilisation of a PUF. Notice, that an RO can also be realised with just a single inverter and a number of delaying drivers (Figure 1b). ROs with a single inverter have a lower frequency but are just as prone to process variations as those with several inverters. In our designs we are using ROs with just one inverter as in [7].

A schematic view of an RO PUF, based on [2], is shown in Figure 2. Each of the ROs, \(1 \leq i \leq m\) is an RO as described above. Two ROs are determined by the PUF challenge and selected via multiplexers. The frequencies of the selected ROs \(f_A\) and \(f_B\) are counted individually by two counters for a fixed amount of time, before the counters’ results \(c_A\) and \(c_B\) are compared by a comparator. Depending on which counter stores the greater number, the comparator returns 0 or 1 as PUF’s response bit for this challenge.

If the RO frequencies are randomly different due to process variations on each device, the same challenge to different RO PUF devices will yield a randomly different response. This response is the device’s fingerprint.

A challenge can be considered not just one but a set of RO selections each of which yields another response bit. Then, the response is not just one bit but a string of bits. To uniquely identify a device from a larger device population a sufficiently big number of response bits is required. If the response is always the same for each device (reliability) but different from any other device (uniqueness), an ideal PUF is realised.

B. Altera Cyclone IV

Most analyses of RO PUFs published so far have used Xilinx FPGAs (e.g. [3], [4], [5], [6]) while Altera FPGAs have received relatively little attention. This is most likely due to the fact that the Xilinx design software allows for much easier definition of routing properties and for the copying of circuit parts while keeping the same routing. Achieving the same with Altera’s Quartus software requires bigger effort.

The authors of [7] give no details about how they implemented their RO PUF on Altera FPGAs. In [8] a different kind of delay based PUF is implemented on Altera FPGAs, but the authors only hint that “LogicLocks” and node location declarations had to be used. Due to the non-straight-forward nature of implementing an RO PUF with Quartus we are sharing some details here about how we achieved it.

The first thing to be aware of is the Altera FPGA architecture which is quite different from the Xilinx “configurable logic block and slices” layout. Figure 3 shows a screenshot from the Quartus Chip Planner viewing 3 out of a total 1395 Logic Array Blocks (LABs) on the floor plan of a Cyclone IV (EP4CE22F17C6N) FPGA, we used in our experiments. Each LAB consists of 16 Logic Elements (LEs) which are programmable through the lookup tables (LUTs) they contain. In our RO PUF design, an RO is realised by programming the first LE of an LAB as NAND-gate and, depending on the RO length, a number of successive LEs as delay elements (cf. Figure 1).

In the Chip Planner, portions of the FPGA can be defined as LogicLock regions which will not be relocated by the compiler. Each LogicLock region gets a name identifier by which it can be addressed in the VHDL files describing the ROs. Thus, we can make sure where an RO is placed. For the RO’s delay elements, the Quartus low-level primitive LCELL is used. LCELLs are not removed by the compiler’s optimisation, using them allows for the implementation of delay elements which are otherwise logically redundant. One LCELL is always implemented in one LAB. As LogicLock regions can only be defined as chunks of LABs, however, it is not possible to directly influence the routing between the LEs in each LAB. The routing done by the compiler connects the LEs in an arbitrary order, whereas we would like the topmost LE connected to the second topmost and so on. This has to be defined manually in the Chip Planner. What follows is a technical description of this process intended for researchers who want to implement their own RO PUFs with Quartus:

First, the “Netlist Type” of all RO LogicLock regions has to be set to “Post-Synthesis” in the Chip Planner’s “Design Partitions” tab. After a subsequent compilation, the ROs...
are placed in their respective LogicLock regions but the routing between their LEs has to be defined manually. As mentioned above, this has to be done for all (in our case 64) ROs because Quartus allows for no copying of such redefinitions. After confirming these changes in the Chip Planner’s “Change Manager” tab, Quartus’s “Back-annotate-feature” has to be applied. After another compilation, the “Back-annotate-feature” has to be removed again, the “Netlist Type” of all RO LogicLock regions set to “Post-Fit”, and “Fitter Preservation Level” set to “Placement”. This will prevent future compilations from changing the LCELL routing layout.

After yet another compilation, it is now possible in the Chip Planner to relocate the RO LogicLock regions anywhere on the FPGA while keeping their internal LE routing. Notice that the removal of the “Back-annotate-feature” also removes pin assignments, which have to be redefined in the Quartus Pin Planner. It is advised to copy these pin assignments from a previous copy of the project, where the assignments were still present.

C. Implementation environment

Figure 4 shows a schematic view of our overall set-up. The lower bar represents a development board and FPGA, the upper bar represents a PC on which a TCL script creates the challenges and stores the responses from the FPGA. These challenges and responses are communicated between the PC and the FPGA via JTAG and USB.

On the FPGA we have the 64 ROs as described in Section II-B but we also have the controller circuitry that is responsible for starting and stopping the ROs selected by a challenge, counting the oscillations, and returning the responses based on their comparisons.

Figure 5 shows our basic placement with the controller logic on the left and the ROs on the right. In Section IV-B and IV-C we are reporting our results of placing the ROs and controller logic in different locations on the FPGA.

III. Metrics

To measure the quality of our RO PUF implementation we used the metrics suggested in [5]. This section briefly describes them.

A. Uniqueness

The goal of a PUF is to unambiguously identify a device. It is therefore important that no two devices give the same responses. Otherwise, they are not distinguishable anymore and hence mistakenly identified.

The difference between the n-bit responses of two devices can be formalised as their hamming distance (HD). Let

\[ R_i = r_{i,1}r_{i,2} \ldots r_{i,n} \text{ and } R_j = r_{j,1}r_{j,2} \ldots r_{j,n} \]

be two response bit strings from device i and j. HD is then calculated as:

\[ HD(R_i, R_j) = \frac{1}{n} \sum_{i=1}^{n} (r_{i,t} \oplus r_{j,t}) \]

yielding a number between 1.0 (all bits are different) and 0.0 (all bits are the same).

For a population of m devices the average HD of all \( \binom{m}{2} \) possible comparisons is called inter distance of the population:

\[ HD_{\text{inter}} = \frac{1}{\binom{m}{2}} \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} HD(R_i, R_j) \]

An ideal HD_{\text{inter}} for large populations is 0.5 [9], because this is the case when all responses are uniformly distributed.

B. Bit-aliasing and response uniformity

An HD_{\text{inter}} below 0.5 means, that some response bits \( r_{i,t} \) are biased towards either 0 or 1 on all devices. Such bit-aliasing (BA) effects happen in the presence of static process variations affecting all devices in the same way; as opposed to random process variations affecting each device differently. The BA of the t‘th (out of n) response bit over a population of m devices is calculated as:

\[ BA(t) = \frac{1}{m} \sum_{i=1}^{m} r_{i,t} \]

A value of BA(t) = 1.0 means, bit t is 1 for all devices. The ideal value would be 0.5. Graphs like Figure 6, in which the BA values for all response bits are plotted, visualise if a mediocre HD_{\text{inter}} is due to all response bits being of mediocre quality (around 0.25 or 0.75) or if some response bits are very bad (close to 0.0 or 1.0) and others very good (close to 0.5).

Tightly coupled with bit-aliasing is the metric response uniformity (RU). Even if BA of each response bit is around 0.5, it could be that the bits are not independent among each other. If they are indeed independent, each response
string should consist of about equally many 0’s and 1’s. For one response \( R_i = r_{i,1}r_{i,2} \ldots r_{i,n} \) this is calculated as:

\[
RU(R_i) = \frac{1}{n} \sum_{t=1}^{n} r_{i,t}
\]

A value of \( RU(R_i) = 1.0 \) means all response bits of \( R_i \) are 1. The ideal value is 0.5. For a population of \( m \) devices, the average \( RU \) is calculated as:

\[
RU = \frac{1}{m} \sum_{i=1}^{m} RU(R_i)
\]

C. Reliability

It is important that the responses of each individual device are always the same, within an acceptable limit. Whether or not this is the case can be measured as the hamming distances between several responses to the same challenge for each device (intra HD as opposed to inter HD).

Let \( R_1, \ldots, R_k \) be \( k \) responses from the same device \( i \) to the same challenge at different times, \( R_i \) is selected as reference response and compared to the remaining \( k-1 \) responses, such that

\[
HD_{\text{intra}}(i) = \frac{1}{k-1} \sum_{t=2}^{k} HD(R_i, R_{i,t})
\]

reflects the reliability of device \( i \). The ideal value would be 0.00, but slightly larger values are also tolerable thanks to error correction schemes [10].

For a population of \( m \) devices the average \( HD_{\text{intra}} \) is calculated as:

\[
HD_{\text{intra}} = \frac{1}{m} \sum_{i=1}^{m} HD_{\text{intra}}(i)
\]

Notice that the selection of the reference response \( R_i \) is crucial for this metric: an “outlier” reference response leads to a larger \( HD_{\text{intra}}(i) \) than a reference response with small distance to the remaining responses.

IV. Results

As for now, our case study is focused on analysing the PUF properties on basis of 128 challenges. For these challenges we consider different RO lengths, RO placements, the relation between RO and controller placement, and RO distribution.

The challenges were selected as follows: The first 64 ROs to be compared are those with successive indexes \((1,2),(2,3),\ldots,(63,64),(64,1)\) and the last 64 pairs those with an index difference of \((1,5),(2,6),\ldots,(63,3),(64,4)\). We thus get 128 response bits per device.

A response bit is generated by passing the challenge (two indexes of ROs to be compared) from the PC to the FPGA where the controller logic starts the counters, stops them after 20 milliseconds, and passes the output of the comparator back to the PC.

To analyse \( HD_{\text{inter}}, BA(t) \) and \( RU \), only one run of response generation per device is required. For \( HD_{\text{intra}} \) we need more runs per device to see if the responses are stable. We chose a number of additional 99 runs (\( k = 100 \)), whose responses were compared with the (reference) response obtained from the first run of the respective device.

A. RO length

The length of an RO is determined by how many LEs are used as series-connected delaying elements to form the RO loop (cf. Figure 1). The RO shown in Figure 3 comprises all 16 LEs of an LAB. If an RO consists of more than 16 delaying elements, the LEs from more than one LAB have to be series-connected.

We analysed the impact of different RO lengths on the PUF quality. Table I shows the results. It can be seen that the design with ROs spanning all 16 LEs of an LAB has the best uniqueness (\( HD_{\text{inter}} \)) properties. The uniqueness goes down drastically when more than 16 LEs are used. This is most likely due to the fact that not only the relatively tight LAB-internal routing is used but the LAB-external routing between different LABs as well. Apparently there are static delay effects coming with the utilisation of LAB-external routing which predominate the subtle device-specific differences.

<table>
<thead>
<tr>
<th>RO length [used LEs]</th>
<th>( HD_{\text{inter}} )</th>
<th>( HD_{\text{intra}} )</th>
<th>( RU )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.4559</td>
<td>0.2593</td>
<td>0.6027</td>
</tr>
<tr>
<td>6</td>
<td>0.2674</td>
<td>0.0834</td>
<td>0.4945</td>
</tr>
<tr>
<td>9</td>
<td>0.2687</td>
<td>0.0864</td>
<td>0.5033</td>
</tr>
<tr>
<td>12</td>
<td>0.2256</td>
<td>0.0102</td>
<td>0.4960</td>
</tr>
<tr>
<td>15</td>
<td>0.3941</td>
<td>0.0091</td>
<td>0.4960</td>
</tr>
<tr>
<td>16</td>
<td>0.3625</td>
<td>0.0118</td>
<td>0.5042</td>
</tr>
<tr>
<td>18</td>
<td>0.0911</td>
<td>0.0032</td>
<td>0.5290</td>
</tr>
<tr>
<td>21</td>
<td>0.2317</td>
<td>0.0012</td>
<td>0.4878</td>
</tr>
<tr>
<td>24</td>
<td>0.1726</td>
<td>0.0045</td>
<td>0.4804</td>
</tr>
<tr>
<td>27</td>
<td>0.2515</td>
<td>0.0052</td>
<td>0.4938</td>
</tr>
<tr>
<td>30</td>
<td>0.2660</td>
<td>0.0082</td>
<td>0.4925</td>
</tr>
<tr>
<td>32</td>
<td>0.1787</td>
<td>0.0064</td>
<td>0.5085</td>
</tr>
</tbody>
</table>

Table I

Results for different RO lengths.

The same can be observed looking at bit-aliasing. In Figure 6, RO length 16 and 18 are compared by plotting the \( BA(t) \) values for all 128 response bits. The light line shows that a lot more bits are 1 (\( BA_t = 1.0 \)) or 0 (\( BA_t = 0.0 \)) for all 20 devices when the RO length is 18 LEs.

Notice that the response uniformity \( RU \) does not suffer from poor bit-aliasing. \( RU \) is for almost all designs close to the ideal value of 0.5. An exception is the design with the shortest RO length of only 3 LEs. Here we see a comparatively good uniqueness (\( HD_{\text{inter}} = 0.4559 \)) which comes at the cost of a very poor reliability (\( HD_{\text{intra}} = 0.2593 \)). Apparently the responses of 3-LE-long ROs are very random even on the same device. A possible explanation for this could be that these ROs have the highest frequency. It is possible that the RO frequency has exceeded the frequency capabilities of the PUF controller logic.
a threshold beyond which some undesired physical phenomena come into effect. Further investigation on this is needed.

Because of our findings in these experiments we used 16-LE-long ROs for the remaining tests.

B. RO placement

Figure 7 shows a view of the FPGA floor plan in which 7 different locations are marked to place the 64 ROs (all length 16) in a block of $10 \times 6 + 4$ LABs. The controller logic is located in the top left corner for all 7 locations. Table II shows the results.

C. RO and controller placement

The findings of our experiments presented in Section IV-B pose the question why location 5 did perform so poorly. One explanation might be that location 5 is influenced by the internal FPGA controller logic (black box in the picture) and that the routing between ROs and controller is thus different from the other locations. To examine this further, we located the controller logic right next to the ROs (again 64 ROs of length 16 in a block of $10 \times 6 + 4$) and tried the 7 different locations again as shown in Figure 9.

As Table III shows, we have a comparatively poor $HD_{\text{inter}}$ at location 5 as well. This suggests that the static variations are indeed within the LABs holding the ROs at location 5. That such bad spots exist on an otherwise RO PUF capable FPGA brand is a valuable discovery, because it shows that the RO logic may not be placed arbitrarily.

Comparing Table II and III it cannot be concluded that placing the controller logic far away from the ROs yields better results, because for none of the two tables all locations would be better then the other.

D. RO distribution

Figure 10 shows three different layouts to distribute the 64 ROs over the LABs: a) has them all next to each other in a rectangular block of $10 \times 6 + 4$ LABs, as already seen in the preceding sections, b) leaves one unused LAB between all ROs, and c1)/c2) order them in two columns of maximum lengths. In the latter 3 layouts the RO indexes do not go in

Interestingly, location 5 stands out with a comparatively very poor $HD_{\text{inter}}$ of just 0.2581. This is confirmed by analysing $BA(t)$ for all response bits: Figure 8 shows a comparison between location 1 and 5 and it can be clearly seen that many response bits are always 0 or always 1 for location 5. Apparently, locating the ROs in location 5 leads to massive static variation which predominates any random variation. This also explains the relatively good $HD_{\text{intra}}$ of 0.0061 for location 5.

Location 1 and 2 have the best $HD_{\text{inter}}$. $RU$ is nearly optimal for all locations.
Table III
RESULTS FOR DIFFERENT RO AND CONTROLLER LOCATIONS.

<table>
<thead>
<tr>
<th>Location</th>
<th>(HD_{\text{inter}})</th>
<th>(HD_{\text{intra}})</th>
<th>RO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.4594</td>
<td>0.0090</td>
<td>0.5015</td>
</tr>
<tr>
<td>2</td>
<td>0.4490</td>
<td>0.0091</td>
<td>0.5007</td>
</tr>
<tr>
<td>3</td>
<td>0.4239</td>
<td>0.0114</td>
<td>0.4972</td>
</tr>
<tr>
<td>4</td>
<td>0.3539</td>
<td>0.0674</td>
<td>0.5027</td>
</tr>
<tr>
<td>5</td>
<td>0.2586</td>
<td>0.0074</td>
<td>0.4902</td>
</tr>
<tr>
<td>6</td>
<td>0.3351</td>
<td>0.0106</td>
<td>0.4972</td>
</tr>
<tr>
<td>7</td>
<td>0.3249</td>
<td>0.0084</td>
<td>0.5070</td>
</tr>
</tbody>
</table>

Table IV
RESULTS FOR DIFFERENT RO DISTRIBUTIONS (CF. FIGURE 10).

<table>
<thead>
<tr>
<th>Structure</th>
<th>(HD_{\text{inter}})</th>
<th>(HD_{\text{intra}})</th>
<th>RO</th>
</tr>
</thead>
<tbody>
<tr>
<td>a)</td>
<td>0.4609</td>
<td>0.0102</td>
<td>0.5039</td>
</tr>
<tr>
<td>b)</td>
<td>0.4829</td>
<td>0.0104</td>
<td>0.4933</td>
</tr>
<tr>
<td>c1)</td>
<td>0.3429</td>
<td>0.1807</td>
<td>0.4910</td>
</tr>
<tr>
<td>c2)</td>
<td>0.3208</td>
<td>0.0058</td>
<td>0.4992</td>
</tr>
</tbody>
</table>

V. Conclusion

We have analysed different implementations of RO PUFs on 20 equivalent Altera FPGAs and reported statistical results. The major contributions of this paper are as follows. First we have shared detailed information about how to implement RO PUFs on Altera FPGAs; an effort only sparsely described in the literature so far. Furthermore, we have conducted several experiments parametrising placement and architecture aspects of the logic involved. We identified that using all 16 LEs of an LAB yields the best results. For specific cases we found an extraordinarily poor PUF performance, which should urge general caution about making quick statements on global PUF qualities.

In future works we will extend our experiments to more challenge-response pairs. It will be insightful to learn whether the poor performances we encountered for specific RO placements are connected to the way our challenges selected the ROs for comparison. Furthermore, we plan to investigate the effects of ambient temperature and on-chip activity on the PUF quality.

References