STRUCTURE-AWARE DESIGN OF SECURITY PRIMITIVES ON RECONFIGURABLE HARDWARE

Dissertation

zur Erlangung des Grades eines Doktor-Ingenieurs
der Fakultät für Elektrotechnik und Informationstechnik
an der Ruhr-Universität Bochum

by Alexander Wild
Bochum, November 2016
Abstract

A Field Programmable Gate Array (FPGA) is a programmable hardware device, which is employed in a wide range of commercial products, e.g., in satellite receivers, secured Universal Serial Bus (USB) flash drives but also in infrastructures such as network backbones. The popularity of this device family is based on its flexibility and programmability combined with the advantages of hardware (with respect to high performance, low energy consumption, and security through integration). Hence, the hardware structure of FPGAs provides an excellent platform for fast and efficient applications with minimal development cost. Nonetheless, FPGA manufacturers provide only minimal support to secure their devices against various attack scenarios.

One attack scenario each physically accessible hardware device has to deal with is physical attacks. This attack class is currently one of the major topics in the area of hardware security. The current FPGA generations are not inherently equipped with countermeasures against this attack class.

Furthermore, the majority of FPGAs available on the market are Static Random Access Memory (SRAM)-based and do not provide any secure non-volatile memory included in the chip package to the users. Hence, a persistent device internal key storage is not feasible in the conventional way, and an alternative key management is required. Physical Unclonable Functions (PUFs) seem to provide a promising solution for this problem by generating a device-specific secret key during the initialization phase or the run time of the FPGA.

This dissertation addresses with the two mentioned problems by focusing on the low-level aspects and structures of FPGAs. More precisely, the device structure itself is used to (i) increase the resistance against physical attacks, and (ii) propose and analyze PUF constructions which are implementable on FPGAs without further investigations of additional hardware.

Keywords.

Field Programmable Gate Array (FPGA), Cryptology, Countermeasure, Hardware, Implementation

Eine Art der Angriffsklassen, gegen die sich jede physikalisch zugängliche Hardware schützen muss, sind physikalische Angriffe. Diese Angriffsklasse ist momentan eines der größten Themen im Bereich der Hardware-Sicherheit. Die aktuellen FPGA-Generationen sind inhärent nicht mit Gegenmaßnahmen gegen diese Angriffsklasse ausgestattet.

Des Weiteren basiert die Mehrheit der auf dem Markt erhältlichen FPGAs auf Static Random Access Memory (SRAM)-Zellen und stellen dadurch keinen sicheren, nicht flüchtigen Speicher im Chipgehäuse für Benutzer bereit. Dadurch ist eine konventionelle Umsetzung von persistenter, internen Schlüsselspeicherung nicht möglich und ein alternatives Schlüsselmanagement erforderlich. Physical Unclonable Functions (PUFs) scheinen eine vielversprechende Lösung für dieses Problem zu bieten, indem ein gerätespezifischer, geheimer Schlüssel während der Initialisierungsphase oder zur Laufzeit des FPGAs generiert wird.

Diese Dissertation behandelt die beiden erwähnten Probleme und konzentriert sich dabei auf die Aspekte und Strukturen der untersten Ebene eines FPGAs. Genauer gesagt wird die Gerätestruktur selbst verwendet, um (i) die Widerstandsfähigkeit gegenüber physikalischen Angriffen zu erhöhen und (ii) um PUF-Konstrukte zu realisieren, welche ohne die Verwendung von zusätzlicher Hardware auf FPGAs umsetzbar sind.

**Deutscher Titel**

Strukturorientiertes Design von Sicherheitsprimitiven auf rekonfigurierbarer Hardware

**Schlagworte.**

Field Programmable Gate Array (FPGA), Kryptologie, Gegenmaßnahme, Hardware, Implementierung
# Table of Contents

Imprint ........................................ iii
Abstract ....................................... v
Kurzfassung .................................. vii

## I Preliminaries 1

### 1 Introduction 3
  1.1 Motivation .................................. 3
  1.2 Summary of Research Contribution and Outline .......... 4

## II Background 7

### 2 Background 7
  2.1 Field Programmable Gate Array (FPGA) .......... 7
  2.1.1 Device Structure .......................... 7
  2.1.2 Xilinx Tool Flow .......................... 9
  2.1.3 Xilinx Design Language .................... 10
  2.1.4 Bitstream Composition .................... 11
  2.2 Passive Implementation Attacks .................. 12
  2.2.1 Side-Channel Analysis .................... 12
  2.2.2 Countermeasures .......................... 13
  2.2.3 Evaluation Methods ........................ 16
  2.2.4 Previous Work ........................... 20
  2.3 Physical Unclonable Functions (PUF) ........... 27
  2.3.1 Weak and Strong PUFs .................... 28
  2.3.2 Evaluation Metric ........................ 28
  2.3.3 Properties .............................. 29
  2.3.4 Previous Work ........................... 30

## III Power-Equalization as Side-Channel Countermeasure 39

### 3 Evaluating the Duplication Concept 41
  3.1 Introduction .................................. 41
  3.2 Performing Circuit Duplication .................. 43
  3.2.1 Distinct Circuit Duplication ............... 43
  3.2.2 Interleaved Circuit Duplication ............. 43
  3.3 Data-Dependent Time of Evaluation ................ 44
# Table of Contents

3.4 Practical Evaluation ................................................. 46
  3.4.1 Case Study .................................................. 46
  3.4.2 Side-Channel Analysis .................................... 47
3.5 Conclusion ......................................................... 49

4 Hiding First-Order Leakage: Glitch-Free Duplication 51
  4.1 Introduction .................................................. 51
  4.2 Concept ....................................................... 52
  4.3 Automated Circuit Transformation .......................... 56
  4.4 Resource Overhead ........................................... 58
  4.5 Practical Evaluation ......................................... 59
    4.5.1 Side-Channel Analysis ................................. 61
    4.5.2 Discussions about Remaining Leakage ............. 64
  4.6 Conclusion ................................................... 65

5 Hiding Higher-Order Leakages 67
  5.1 Introduction .................................................. 67
  5.2 Background .................................................... 69
    5.2.1 Threshold Implementation ............................ 69
    5.2.2 GliFreD* ............................................... 69
  5.3 Case Study .................................................... 70
    5.3.1 KATAN-32 ............................................... 70
    5.3.2 PRESENT ............................................... 72
    5.3.3 Implementation ....................................... 75
  5.4 Practical Evaluation ......................................... 77
  5.5 Conclusion/Discussion ....................................... 80

III PUFs on Reconfigurable Hardware 85

6 Recovering Initial SRAM Values 87
  6.1 Introduction .................................................. 87
  6.2 Bypass BRAM Reset .......................................... 88
  6.3 Evaluation .................................................... 89
    6.3.1 Memory Decay and Flip Direction .................. 89
    6.3.2 Unpredictability of SRAM Cells .................... 91
    6.3.3 Reliability ........................................... 92
    6.3.4 Uniqueness ........................................... 93
  6.4 Conclusion ................................................... 93

7 Implementation of Ring-Oscillator PUFs 95
  7.1 Introduction .................................................. 95
  7.2 Case Study .................................................... 96
  7.3 Implementation Issues ...................................... 97
    7.3.1 Structural Bias ....................................... 97
# Table of Contents

7.3.2 Malicious IP .................................................. 98  
7.3.3 Counter Failures .............................................. 99  
7.4 Efficient Implementation Techniques .............................. 100  
7.5 Evaluation .......................................................... 102  
7.6 Conclusion .......................................................... 104  

8 Large-Scale Analysis of Oscillation-Based PUFs ............... 107  
8.1 Introduction ...................................................... 107  
8.2 Setup .............................................................. 108  
8.3 Response Generation ............................................ 108  
8.4 Implementation and Analysis .................................. 109  
8.4.1 Ring Oscillator PUF ........................................... 109  
8.4.2 Loop PUF ...................................................... 110  
8.4.3 TERO PUF ..................................................... 111  
8.5 Comparison of the PUF Constructs .............................. 111  
8.5.1 Reliability ..................................................... 112  
8.5.2 Uniqueness ..................................................... 114  
8.5.3 Analysis of the Loop PUF .................................. 115  
8.5.4 Other Response Generation Schemes ....................... 117  
8.6 Conclusion .......................................................... 118  

IV Conclusion ........................................................ 119  

9 Conclusion and Future Work .................................... 121  
9.1 Conclusion ........................................................ 121  
9.2 Future Work ...................................................... 122  
9.2.1 Power-Equalization ........................................... 122  
9.2.2 PUFs .......................................................... 123  

V Appendix ............................................................ 125  

10 The KATAN AND/XOR Module ................................ 127  

11 PRESENT Sbox ..................................................... 129  

12 Evaluation Results of PRESENT-2nd by 1 Billion Traces ...... 133  

Bibliography .......................................................... 135  

List of Abbreviations ............................................... 153  

List of Figures ....................................................... 157
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Tables</td>
<td>161</td>
</tr>
<tr>
<td>About the Author</td>
<td>163</td>
</tr>
<tr>
<td>Publications and Academic Activities</td>
<td>165</td>
</tr>
</tbody>
</table>
Part I

Preliminaries
Chapter 1

Introduction

This chapter briefly introduces the research motivation of this thesis. In addition, an outline and the scientific contribution of this thesis are given.

Contents of this Chapter

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Motivation</td>
<td>3</td>
</tr>
<tr>
<td>1.2</td>
<td>Summary of Research Contribution and Outline</td>
<td>4</td>
</tr>
</tbody>
</table>

1.1 Motivation

The modern society and business life has their fundamentals in modern communication, transport, and automation techniques. Nowadays, almost every electronic device we touch contains an embedded processing unit like a microcontroller (μC), an Application Specific Integrated Circuit (ASIC), or a Field Programmable Gate Array (FPGA). In the past years, the number of embedded devices (most of them have been connected to the internet) was growing constantly and will continue to grow in the future [Ins16]. The usage and interconnection of those devices offer great benefits and advantages with respect to productivity, simplification, and support but also hold great risks since the majority of devices collect and hold private and hence sensitive data which need to be protected.

To offer embedded devices or components as a developer, several aspects and properties with respect to the product use case have to be taken into account. The main aspects embedded devices have to address are performance, low energy, flexibility, and low-cost. While microcontrollers offer great flexibility and low development and acquisition cost, the performance is often a drawback. ASICs, on the other side, are fast in executing a defined task but are restricted to this given task without any modification opportunities and show high development cost for low quantities. FPGAs are often noted as a link between μCs and ASICs as they provide a flexible and low cost platform for embedded devices. Hence, FPGAs can be found in a variety of modern embedded systems, e.g., USB flash drives, satellite receivers, (autonomous) cars, mobile devices, and robotics [FPG15].

State-of-the-art FPGAs are highly integrated circuits [Xil16b] and are capable to hold large and complex designs. The market-leading Xilinx and Altera [Sou13] primarily provide SRAM-based FPGAs, but also alternative concepts like flash-based FPGAs are provided as well by Xilinx or Microsemi.
Chapter 1. Introduction

The various application areas of FPGAs require to address a variety of issues and risks, e.g., Intellectual Property (IP) protection, countermeasures against physical attacks, key management or device authentication. Yet, little support is given by the manufacturers (i.e., basic IP protection mechanisms on netlist level or encryption of the configuration bitstream) to deal with these threats. It is worth to mention that the bitstream encryption of most device families from the market leaders are vulnerable against Side-Channel Analysis (SCA) [MBKP11, MKP12, MOPS13]. Hence, the hardware developers are responsible for the system security.

This thesis aims to provide solutions for the open security related problems and tries to support the work of hardware designers. To be independent of any algorithm or design the focus of this thesis is to use the given hardware structure to form countermeasures against SCA and provide analysis and solutions of PUF classes for secret key management.

1.2 Summary of Research Contribution and Outline

Most of the work in this thesis has been published in peer-reviewed conference proceedings [uBG16, Mu15, uMG15, uG14, uB17] or journals [uMG17] and form the basis of the chapters in this work. The structure of the published papers has been modified for a better readability. Further projects that are not considered in this thesis are [EuM17, BuG15, EDu +14].

The first part of the thesis (see Part I) introduces the topic of this thesis and gives a brief background to the structure, tooling, and low-level information of Xilinx-based FPGAs (see Chapter 2.1). Additionally, the concept of SCA is introduced and categorized into the field of cryptanalysis. The most common analysis methods and concepts for countermeasures are also part of the background chapter (see Chapter 2.2). Since this thesis is also dealing with PUFs, a general definition and classification of PUFs, evaluation methods, and metrics are given as well (see Chapter 2.3).

The research contribution of this thesis is twofolded: it is categorized into power-equalization schemes (see Part II) and PUFs (see Part III) on FPGAs. The last part of the thesis concludes and summarizes the findings of each chapter and gives an outlook into related open research topics.

Power-Equalization Schemes on FPGAs

Power-equalization is a hiding concept to counter SCA that was originally proposed for ASICs. Several proposes have been published to transfer this concept to FPGAs (see Chapter 2.2.4). Nonetheless, all of the previously proposed schemes suffer from at least one of the known major pitfalls. This part of the thesis provides further analysis and concepts to properly instantiate this concept on FPGAs. We first analyze the duplication concept proposed in [YS07] which has been considered by the community to be the best way to balance a Dual-Rail Precharge (DRP) circuit. We will show in Chapter 3 that this concept indeed balances coupled wires on the logical level and reduces the leakage of a design, but the concept is still flawed for the evaluated logic styles, Wave Dynamic Differential Logic (WDDL), Dual-Rail Precharge Logic without Early Evaluation (DPL-noEE), and Asynchronous Wave Dynamic Differential Logic (AWDDL), resulting in a data-dependent time of evaluation. Chapter 4 proposes a logic style which overcomes the three major problems of DRP schemes and analyzes it with respect
to the achievable leakage reduction and induced resource overhead. Besides power-equalization, the most popular and best understood countermeasure against SCA is masking. Since masking the leakage of higher statistical orders is a hard problem, we implemented and analyzed a hybrid concept of power-equalization and masking. We evaluated the concept in terms of the achievable resistance against SCA on higher statistical orders and the induced resource overhead (see Chapter 5). We further contributed to the field of power-equalization by providing tools and techniques that have been developed to support hardware designers to implement and instantiate the power-equalization concepts on commercial Xilinx FPGAs (see Section 3.2 and Section 4.3). Please note that the author of this thesis has cooperated with Amir Moradi during the practical evaluation of the power-equalization part and hence does not solely contribute to Section 3.4.2, 4.5.1, and 5.4.

**PUFs on FPGAs**

Almost similar to the power-equalization approaches, the majority of PUF constructions that have been published so far suffer from the predefined and resource-restricted structure of FPGAs. We first summarize the most promising concepts and PUF variants for FPGAs that have been published in the recent years (see Chapter 2.3.4). One of the most promising PUF class is the SRAM PUF. Unfortunately, the power-up reset on modern Xilinx FPGA makes this approach not suitable and do not allow the instantiation of this class. To bring this PUF class back to Xilinx FPGAs, we propose a method which allows the SRAM cells to recover their initial state. The recovered SRAM values have been analyzed with respect to their uniqueness and reliability (see Chapter 6). Another promising PUF class available on FPGAs is the Ring Oscillator (RO) PUF. This concept has been used in several works, but often it is not instantiated efficiently and not fully evaluated. We analyze this concept based on a case study with respect to outer influences, biases induced by the FPGA structure and also reveal a problem with the counter (which is part of the RO PUF class) that has not been noted in the literature so far (see Chapter 7). However, most of the so far published PUFs were implemented and evaluated on different (and often outdated) FPGA families with different metrics. Missing implementation details in many papers further hamper a fair analysis as small details such as the exact routing can have significant impact on the PUF performance. To overcome these problems we provide a fair comparison of some of the most promising Weak PUFs for FPGAs, the classic RO PUF, the Loop PUF and the TERO PUF (see Chapter 8). Please note that the author of this thesis has cooperated with Georg T. Becker during the practical evaluation of the PUF part and hence does not solely contribute to Section 7.5 and 8.5.
Chapter 2
Background

This chapter briefly introduces the low-level aspects of Xilinx FPGAs while the hardware description focuses on device components used in the research. Additionally, the principles of passive implementation attacks are described which include general countermeasure approaches and evaluation methods. Furthermore, Physical Unclonable Functions (PUFs) are introduced, defined, and classified.

Contents of this Chapter

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Field Programmable Gate Array (FPGA)</td>
<td>7</td>
</tr>
<tr>
<td>2.2</td>
<td>Passive Implementation Attacks</td>
<td>12</td>
</tr>
<tr>
<td>2.3</td>
<td>Physical Unclonable Functions (PUF)</td>
<td>27</td>
</tr>
</tbody>
</table>

2.1 Field Programmable Gate Array (FPGA)

FPGAs are programmable hardware devices that are often noted as a link between ASICs and μCs. On the one hand, FPGAs are able to form almost arbitrary logic circuits, show low energy consumption, and high performance like ASICs but are, on the other hand, reprogrammable and flexible like μCs. Due to the fusion of both worlds and the technology advantages, the popularity of FPGAs is growing in a wide spectrum of general industrial and security related applications [FPG15].

2.1.1 Device Structure

In general, FPGAs are reconfigurable integrated circuits that consist of a grid of logic and memory cells. These cells are hardwired to routing components which can be configured to provide a logical connection between the cells. The majority of FPGAs are SRAM-based and need to be configured after power-up [Xil16a]. The configuration of the device and hence its functionality is defined in a binary file called bitstream, which mostly needs to be loaded into the FPGA from an external source. There exist various FPGAs from different vendors (e.g., Xilinx, Altera, Lattice, or Microsemi) which follow the same principles, but its low-level structure differs, even between device families of the same vendor. Since this thesis addresses Xilinx FPGAs and, as noted, the low-level structure also varies between device families, the following low-level description is for the sake of simplicity restricted to Xilinx Spartan-6 devices [Xil10].
An Xilinx Spartan-6 FPGA is organized in a grid structure composed of Configurable Logic Blocks (CLBs) and dedicated hardware components which are hardwired to routing elements called Switch Matrices (SMs) [Xil10].

CLBs can be further subdivided into logical components called slices. There are three different slice types in a Spartan-6: SliceX, SliceM, and SliceL. Each CLB consists of two slices while one slice is always a SliceX and the other is either a SliceM or a SliceL. A SliceX is made of four Look-Up Tables (LUTs) to implement Boolean logic and eight Flip Flops (FFs) to store intermediate values. Four of these FFs can alternatively be configured as latch. A SliceL and SliceM extend the basic functionality of a SliceX with a carry-chain and the ability to use the LUTs as distributed memory elements or shift registers.

The LUTs of a Spartan-6 can be either configured as a single 6-to-1 LUT or as two 5-to-1 LUTs with shared inputs. The literature assumes that a LUT consists of a multiplexer tree with two output pins called $O_5$ and $O_6$ [Xil13c, MM12]. Due to the internal structure of a LUT, both outputs do not share any internal resources and are hence capable of calculating independent outputs based on shared input signals. Figure 2.1 shows the logical structure of a Spartan-6 LUT [Xil13c].

Beside CLBs, the Spartan-6 is also equipped with dedicated hardware components to, e.g., increase the internal memory with Block RAMs (BRAMs), accelerate arithmetic operations with Digital Signal Processors (DSPs), or to provide access to peripheral components via Input/Output Blocks (IOBs) or high speed transceivers.

Signal routing in an FPGA is performed by routing elements called Switch Matrices (SMs) which are interconnected to form a mesh structure. Each CLB is hard wired to an SM. By configuring the SMs, the CLBs can be logically interconnected, and signal values can be transmitted between CLBs. Such a logical connection, which in general involves multiple SMs, is called a route or a net. The internal structure of the SMs used by the Spartan-6 family is not publicly available, but in general, such switches can be realized with tri-state buffers, pass-through transistors, or a mixture of both techniques [KTR07].
The SMs in a Spartan-6 are interconnected in a structured way. Directly neighboring SMs are connected by *single wires*. SMs that are two or four positions apart are connected via *double* and *quad wires* respectively. In comparison to single wires, double and quad wires also connect SMs that are placed diagonal. Figure 2.2 depicts the single, double, and quad connections with respect to a reference SM. A switch matrix can also be used to bounce signals from the same CLB. Wires that remains inside a CLB are called by the literature *local wires* [MSE10].

![Visualization of the hardwired SMs interconnection structure of the Xilinx Spartan-6 device family.](image)

The SMs are commonly used to route data signals. Clock signals, which are required in synchronous design circuits, are handled differently. The FPGA provides a special network called clock tree to guarantee a minimal skew of clock signals. Commonly, an FPGA provides multiple clock trees to handle several clock signals which raises the ability to run parts of a design with different frequencies. The exact number of available clock trees is device family dependent [Xil15].

### 2.1.2 Xilinx Tool Flow

An FPGA is, as previously noted, a hardware device programmable after manufacturing. The configuration code is usually written in a Hardware Description Language (HDL), e.g., VHDL, Verilog or System-C, which is translated by a tool flow into a hardware readable file called bitstream. Currently, Xilinx provides two flows: *Integrated Synthesis Environment (ISE) Design*
Suite for their legacy devices and Vivado for their latest device generations. Independent of the flow, some major processing steps are done to translate an HDL-based design into a bitstream. Please note that the processing steps are not mandatorily distinct and hence sometimes merged by the tool flow.

(1) **Synthesis:** The synthesis is the first step in the flow that directly works with the HDL sources. During this step the HDL design is translated into specific design building blocks or generic macros like multiplexer, Random Access Memory (RAM) components, or adders which can be easier processed by further steps. The synthesis step often contains a low-level optimization process which checks for shared resources or technology-based optimizations that can be done at this point in time. The output of the synthesis step is hence a low-level description of the design, which is known in the literature as a netlist. Note that the output of the following steps, up to the bitstream generation, is also called a netlist while the content of these netlists changes with each step.

(2) **Translate:** The translate step further processes the design building blocks or generic macros and translates them into Xilinx specific primitives. In general, these Xilinx primitives are technology and device family independent.

(3) **Map:** The mapping process adapts the given netlist to device-specific components like IOBs, LUTs, BRAMs, DSPs, etc. Optionally, the components are placed on the FPGA in this step as well.

(4) **Place & Route:** The netlist given to the place & route process contains just device specific primitives. This processing step tries to ideally place and interconnect these primitives on a defined FPGA. The term "ideally" is in this context a user constraint, whether the process should optimize the design with focus on resource consumption (area), on minimal routing delay (performance) or both. The netlist generated by the place & route step holds the structure of the design and how it will be instantiated on the FPGA.

(5) **Bitstream Generation:** The last processing step translates the netlist into an FPGA readable bit format called bitstream.

### 2.1.3 Xilinx Design Language

A Xilinx netlist is stored in a proprietary file format. For low-level access the Xilinx ISE flow provides a tool to convert the proprietary netlist file format into (and vice versa) a human-readable format called Xilinx Design Language (XDL). This tool offers the ability to access and manipulate a fully-routed circuit on the netlist level. Note that the XDL structure may change between FPGA families caused by technological progress in FPGA structure.

The XDL file format is organized in instances and nets. An instance is an instantiated component on the device, e.g., slice, a BRAM, or a DSP. The configuration of an instance is given by its attributes. In case of a slice, these attributes contain the LUT content, FF initial values, multiplexer configuration, and so on. The physical location of an instance is defined in the instance header. Components of Xilinx FPGAs are organized in a grid and can be identified by their X and Y coordinates.
2.1. Field Programmable Gate Array (FPGA)

The routing of the signals is organized in a quite similar manner. The nets are routed via switch boxes that are able to interconnect different wires and are also identifiable via their X and Y coordinates. A switch matrix configuration is called a Programmable Interconnect Point (PIP). Hence, all PIPs used to route a net are written to the configuration part of that net. The reader interested in detailed information of the XDL file format is referred to [BKT11].

2.1.4 Bitstream Composition

In this section we give a brief introduction into the composition of the Xilinx bitstream. The work done in Chapter 6 requires to manipulate the bitstream composition and is only applicable on the 28 nm generation (7 series) of Xilinx FPGA. Hence, for the remainder of this section we explicitly focus on the 7 series, i.e., the Xilinx Zynq device family. The following explanation is certainly likewise transferable since the bitstream layout just slightly differs, e.g., in word or frame length between Xilinx product families.

The Xilinx Zynq bitstream is a machine-code organized in 32-bit words. Each word is either an opcode or an operand (or part of an operand). The opcodes define an instruction which is executed, e.g., a read or write operation to a specific register. The bitstream opcodes are organized in two different structures: type 1 header and type 2 header. The opcode structures and the two header types are given in Table 2.1 and 2.2.

<table>
<thead>
<tr>
<th>Table 2.1: The type 1 header format.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Header Type</strong></td>
</tr>
<tr>
<td>[31 : 29]</td>
</tr>
<tr>
<td>001</td>
</tr>
</tbody>
</table>

Note: “R” marks unused bits which are reserved for future use.

<table>
<thead>
<tr>
<th>Table 2.2: The type 2 header format.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Header Type</strong></td>
</tr>
<tr>
<td>[31 : 29]</td>
</tr>
<tr>
<td>010</td>
</tr>
</tbody>
</table>

Note: “R” marks unused bits which are reserved for future use.

The opcode of an instruction can be either read [01], write [10], or No Operation (NOP) [00]. The register address defines on which register the instruction operates, like Frame Address Register (FAR) [00001], Frame Data Register, Input Register (FDRI) [00010], Cyclic Redundancy Check (CRC) [00000], or others. The word count defines the number of 32-bit operand words following the header. The maximum word count of a type 1 header can be $2^{11}$. To increase the word count, a type 2 header can follow a type 1 header. With an attached type 2 header at most $2^{27}$ operand words can follow [Xil13a].

The most important registers are the FDRI and the FAR registers. The FDRI register contains the device component configuration which is split into frames. A Zynq (XC7Z020)
frame consists of 101 × 32-bit words. Each frame of the FDRI is identified by a frame address. The frame address defines which part of the FPGA is configured by the corresponding frame and stored in the FAR. So the combination of FAR and FDRI registers hold all information about the component type, the component configuration, and the component location. In a full and uncompressed bitstream the FAR is set once before the FDRI is written. During the FDRI writing process, the frame address (i.e., the FAR) is incremented automatically. Hence, the order of the frames written to the FDRI is implicitly given and any removal or reordering of frames results in a misconfiguration of the FPGA. In case of a compressed or partial bitstream the FAR register can be set individually which allows a partial or arbitrary configuration order of the FPGA.

### 2.2 Passive Implementation Attacks

In general, cryptanalysis or attacks either target the mathematical foundations or a device specific implementation of a cryptographic construction. Hence, attacks are categorized into mathematical or implementation attacks. The goal of an attack varies and is strongly based on the underlying cryptographic construction, e.g., to find collisions in hash functions [WY05], to recover encrypted messages in cryptographic protocols [Ble98], or to reveal the secret key of a cipher [SMY09]. Devices holding an implementation of a cryptographic construction are further called cryptographic devices.

Implementation attacks are classified by two properties: their influence on the device during run time and the required device connection. Passive attacks only observe the device while active attacks manipulate the device during run time to trigger an abnormal behavior. The connection to the device defines whether the attack is categorized as an invasive, semi-invasive, or non-invasive attack.

**Invasive attacks** are in general the most powerful and expensive attacks. Invasive attacks depackage a device and try to extract data, e.g., from an internal data bus via probing, or inject respectively manipulate data. **Semi-invasive attacks** also depackage a chip but do not remove the depassivation to establish an electrical connection to the device’s internal structure in order to extract or inject data, e.g., via laser to change the functionality of the device. The third category is the **non-invasive attacks** which do not modify the chip in a non-reversible way and just use directly accessible interfaces. Non-invasive attacks deal with a device as it is and do not leave any attack evidence.

Passive non-invasive attacks are also known as Side-Channel Analysis (SCA) which observes a directly accessible channel like time, current consumption or Electromagnetic (EM) radiation and extracts secret information from the cryptographic device via analysis of the data given by these side-channels. Mostly, SCA is applied to attack ciphers or cryptographic constructions that make use of a secret value. The goal of the SCA is then to reveal the mentioned secret value via the observed channel.

#### 2.2.1 Side-Channel Analysis

The most common side-channels of an FPGA are time, power consumption, and EM radiation. Using the time as a side-channel to attack symmetric ciphers is often pointless since most symmetric constructions are time-constant. Hence, the power consumption and the EM radiation
2.2. Passive Implementation Attacks

Power analysis attacks target cryptographic devices containing secret keys in order to extract these keys by exploiting the fact that the dynamic current consumption of a device depends on the data it processes and the operation it performs. Although in most attack scenarios the current consumption is measured, the literature often refers to the term power analysis. This terminology is adapted in this thesis as well to avoid inconsistencies with the literature.

The dynamic power consumption of Complementary Metal Oxide Semiconductor (CMOS)-based devices occurs during a signal transition: it is a combination of the charging current and the short-circuit current. The charging current is the current required to load the output capacitance of the switching CMOS cells. This capacitance heavily depends on the technology the circuit is built with, the wire length and the number of subsequent cells. The short-circuit current occurs during a transition when both transistors of the pull-up and the pull-down networks conducting simultaneously.

In conclusion, the dynamic power consumption (and the EM radiation) is determined by the operation the logic cells implement and the processed data. This link is used to extract secret intermediate values and hence the secret key from a CMOS-based device. The direct dependency of the power consumption from a secret value is often noted as leakage and has been studied extensively in the recent years. Various attacks like Simple Power Analysis (SPA), Differential Power Analysis (DPA) [KJJ99] (respectively Simple Electromagnetic Analysis (SEMA) and Differential Electromagnetic Analysis (DEMA) [GMO01]), Correlation Power Analysis (CPA) [BCO04] (respectively Correlation Electromagnetic Analysis (CEMA) [DCYZ09]), or template attacks [CRR02] have been published which proves the risk that naturally comes with the mentioned leakage.

2.2.2 Countermeasures

As already noted in the previous section, power analysis attacks work due to the correlation between the power consumption, the processed data, and performed operation. Hence, the goal of a countermeasure is to break these dependencies by building cryptographic devices which consume random or equal amount of power, independent of the processed data or performed operation. To realize the decorrelation, three major concepts are known which either work on a transistor/gate level (hiding), an algorithmic level (masking), or perform a periodic update of the secret key (rekeying). Implementing these concepts properly in practice is a non-trivial task.

Masking

Masking is probably the most investigated and best understood protection mechanism against side-channel attacks [CJRR99, DDF14, RP10]. It takes the algorithmic level into account and randomizes the intermediate values processed by the cryptographic device. Since masking schemes are intended to work on the algorithmic level [AG01, BGK04, RP10, CJRR99], power characteristics of the underlying device do not have to be changed. Nonetheless, some research moved masking schemes to the gate level [ISW03, FG05, LMW14].

Masking conceals an intermediate value $v$ with a uniform random chosen mask $m$ by $v_m = v \star m$. The operator $\star$ is typically chosen as Boolean XOR $\oplus$, modular addition $+$, or modular multiplication $\times$, based on the underlying masking scheme. To reconstruct $v$, $v_m$ and
Chapter 2. Background

$m$ are required so that $v$ is represented by the shares $(v_m, m)$. By knowing only one of the two shares no information about $v$ can be revealed. This concept can be extended to make use of $d$ shares to build schemes that protect against higher-order attacks.

Ideally, the random masks are added to the plaintext or key and just removed from the ciphertext after the encryption has been performed. The randomly chosen masks randomize the intermediate values and hence randomize the power consumption of the device.

Assuming that the leakages of the shares are independent of each other, a successful key-recovery attack needs to observe – at least – the $d$th-order statistical moment of the leakage distributions, where the corresponding complexity increases exponentially with $d$.

**Pitfalls and Limitations** The development and implementation of a proper masking scheme is a non-trivial task. Hence, several schemes are prone by major pitfalls or limited in practice.

The masks and hence the shares have to be independent and uniformly random values at every point in time. In case of dependencies between the shares or non-uniformity of a share, the power consumption of the scheme would not be uniformly random, and the statistical dependencies can probably be used to apply an SCA.

Further, glitches violate the assumption of independent share associated leakages. Glitches are faulty gate output values that occur for a short time period caused by different propagation delays of the gate input signals. These glitches are data dependent and can flaw a masking scheme, e.g., a Boolean masking with two shares which desired to provide first-order resistance [MME10, MPO05].

The resistance of a masking scheme against higher-order SCA depends on the number of shares used to represent the intermediate values. In general, increasing the number of shares also increases the resistance order of the scheme while the exact dependencies between the investigated shares and the achieved resistance is scheme dependent. By increasing the number of shares, the area requirements of the integrated circuit also increase which may result in impractical implementations. On the one hand, a $d$th-order SCA-resistant masking scheme is still vulnerable by $d + 1$th-order attacks. On the other hand, the number or required leakage traces to perform a successful attack on a vulnerable order rises exponentially with the number of investigated shares [SVO+10].

**Hiding**

Hiding methods can be applied on a transistor, gate, or application level and process the same intermediate values as an unprotected design version. To perform the decorrelation between the processed data and the power consumption, hiding either tries to randomize the power consumption or to equalize it. To randomize the power consumption of a cryptographic device two techniques can be applied: instruction shuffling [VMKS12] or the usage of dummy operations [CK09]. Both randomizes the performed instructions and hence the power consumption in the time dimension which significantly increases the number of traces required to perform a successful attack. To equalize the power consumption of a cryptographic device, special logic styles were developed. The goal of these logic styles, known as DRP logic styles, is to equalize the power consumption, independent of the processed data or performed operation. It turned out that a perfect power equalization is hard to achieve in practice and hence DRP logic styles are just capable of reducing the Signal to Noise Ratio (SNR).
The principles of DRP logic styles base on the concepts of Dual-Rail and Precharge logic styles and hence form, like the name depicts, a combined approach. Dual-Rail logic styles typically encode each signal in a differential way so that a single signal is represented by the signal itself and an inverted (complementary) signal. For instance, the single wire $a$ is encoded into a differential tuple $(a, \overline{a})$. Mostly, Dual-Rail logic styles do not define gates for signal inversion since a simple line swap performs this operation.

Circuits built on Precharge logic styles alternate between a precharge and an evaluation phase. During the precharge phase each signal in the circuit is set to a constant precharge value which is either 0 or 1. During the evaluation phase the signals hold the data that shall be processed by the circuit.

DRP logic styles use the differential encoding and alternate between the precharge and evaluation phase. Valid values of a signal for the evaluation phase are hence $(0, 1)$ or $(1, 0)$ while the precharge values can be $(0, 0)$ or $(1, 1)$. Due to the differential encoding, exactly one gate output ($a$ or $\overline{a}$) changes its state at a phase transition to either $0 \rightarrow 1$ or $1 \rightarrow 0$. In other words, independent of the processed data, exactly one signal transition per logic cell is performed at a phase transition. This can be scaled up to a full circuit so that a constant number of signal transitions is performed independent of the processed data or performed operation. Indeed, this allows to equalize the power consumption, since (noted in Section 2.2.1) the dynamic power consumption of CMOS-based circuits highly depends on the circuits’ signal transitions.

To classify the proposed DRP concepts, several properties can be taken into account, e.g., the layer on which they are applied, the phase control, the induction of randomness, or circuit instantiation strategies.

**Layer:** The concept of DRP schemes is independent of the algorithmic level and can be applied on the gate level as well as on the transistor level. Schemes that are applied on transistor level defines special gates and are often not compatible with standard CMOS libraries. Schemes applied on gate level make use of standard cells and could potentially be applied on FPGAs. These schemes are hence of more interest to this thesis.

**Phase Trigger:** The two alternating phases (the precharge and the evaluation phase) of DRP schemes are a vital design element. To trigger the phase transitions, the data signals themselves or a synchronous or asynchronous control signal is usually used. The trigger is a time sensitive element since an early or late gate transition usually leads to glitches or the Early Propagation (EP) effect.

**Masking:** As known by the literature, DRP schemes are just capable of hardening a circuit against SCA. Based on this aspect, some schemes make use of random values to perform masking on gate level.

**Duplication:** The wire balancing on FPGAs is one of the major challenges. Following the duplication concept given in [YS07] turned out to be a promising approach for placing the true and false network on an FPGA. Hence, some schemes, explicitly designed for FPGAs, make use of this concept.

**Pitfalls of Power-Equalization Schemes** The development of a DRP logic style for FPGAs turned out to be a non-trivial task since most of the proposed schemes suffer from at least
one of the following pitfalls which impedes the data decorrelation or at least an adequate SNR reduction.

**Glitches:** Glitches are faulty gate output values that occur for a short time period caused by different propagation delays of gate input signals. These glitches are data dependent and can flaw a power-equalization scheme.

**Early Propagation:** The gates of some logic styles evaluate (and precharge) based on the input signal arrival. The EP occurs in logic styles which define gates that change their state before all input signals are available. This results in a data-dependent time of evaluation or precharge and hence leakage in the power consumption.

**Imbalanced Routing:** Wires of different lengths and thus capacitive loads differently contribute to the power consumption at a phase transition. Therefore, the coupled wires of the differential encoding in a DRP scheme have to be balanced to minimize the corresponding data-dependent leakage.

### 2.2.3 Evaluation Methods

Evaluating a cryptographic device with respect to its side-channel vulnerability is a complex task. There are plenty of tests and evaluation methods available which all have advantages and disadvantages. This section introduces the power analysis methods used in this thesis and classifies them with respect to their purpose into leakage assessment, quantification, and exploitation methodologies. Note that the side-channel analysis done in this work only deals with a univariate setting and hence handles each sample point independently. For simplicity, the formulas given in this section are for a single sample point and have to be repeated for further sample points to cover the time interval intended for analysis.

**Higher-Order Leakage and Trace Preprocessing**

In general, side-channel attacks or evaluation methods compare or correlate the leakage distributions at a defined statistical moment. The formulas given in this chapter are for evaluating traces on the first order. To perform a univariate higher-order analysis, the traces have to be preprocessed. Note that the formulas of the following steps are just processing the sample points of a single point in time and have to be repeated to cover the time interval under test.

Based on which statistical order the analysis should be performed, the traces have to be preprocessed in different ways. Let \( l \) be a single data point of a trace, \( d \) the statistical order, \( \mu \) the mean value of the data point, and \( \delta^2 \) the variance of the data point [BGN+14b, SMG15, Mor12, MS14].

\[
\begin{align*}
l'_1 &= l \\
l'_2 &= (l - \mu)^2 \\
l'_d &= \left(\frac{l - \mu}{\delta^2}\right)^d
\end{align*}
\]

\( d = 1 \)

\( d = 2 \)

\( d \geq 3 \)
2.2. Passive Implementation Attacks

Leakage Assessment

A leakage assessment methodology, also known as t-test, was introduced in [GJJR11] and is appealing due to its simplicity, performance and the independence of any underlying attack model or architecture. Nonetheless, the t-test gives just evidence about present leakage and does not quantify it or even evaluate the hardness of an actual key-recovery attack.

As noted, a SCA exploits the information about sensitive intermediate values present in power or EM traces. In other words, the computation of any sensitive intermediate value that influences the traces in a statistical significant way could lead to potential vulnerabilities. Hence, the t-test defines a null hypothesis as follows: if two sets of power traces \( L_0 = \{l_{L_0,0}, \ldots, l_{L_0,N_0}\} \) and \( L_1 = \{l_{L_1,0}, \ldots, l_{L_1,N_1}\} \) which differ in a sensitive intermediate value are not distinguishable via their statistical orders, then the sensitive intermediate does not influence the power traces in a statistical significant way and hence does not leak in the traces. To check if the hypothesis holds for two populations, Student’s (two-tailed) t-test is computed. To analyze the power traces with the t-test for a higher-order leakage, the previously mentioned preprocessing steps are applied in advance.

Based on the preprocessed traces, the t-value is calculated by

\[
t = \frac{\mu_{L_0} - \mu_{L_1}}{\sqrt{\frac{\delta^2_{L_0}}{|L_0|} + \frac{\delta^2_{L_1}}{|L_1|}}}
\]

while \( \mu_{L_0} \) (respectively \( \mu_{L_1} \)) be the sample mean and \( \delta^2_{L_0} \) (respectively \( \delta^2_{L_1} \)) be the sample variance of \( L_0 \) (respectively \( L_1 \)). The calculation has to be repeated for all sample points independently to generate the t-trace. If the generated t-trace exceeds a given threshold \( |c| \) at any point in time the null hypothesis is rejected by a given confidence and the power traces contain enough information to potentially extract the secret intermediate value. The confidence level depends on the chosen threshold and the number of power traces, e.g., for \( |c| = 4.5 \) and about 500 traces per population a confidence level of 99.999% is given.

Further, different distinguishers are introduced for the population generation. The most popular ones are the fixed vs. random t-test (also known as non-specific t-test), semi-fixed vs. random and the specific t-test. For the fixed vs. random t-test, the recorded traces either base on a fixed or random plaintext and are grouped via the processed plaintexts. The traces of the semi-fixed vs. random t-test either base on a plaintext from a predefined set or a random plaintext and are also grouped via the processed plaintext. The predefined set consists of plaintexts which fulfill a certain condition, like a partially equal intermediate state. This test avoids peaks in the t-trace caused by the input or output processing of the cryptographic device. The specific t-test separates the recorded traces (which base on random plaintexts) by a sensitive intermediate value, which can be done, e.g., bit, nibble, or byte-wise.

Leakage Quantification

In some cases the leakage of a side-channel is not fully removed by a countermeasure, e.g., by hiding schemes. In such cases it may be of interest to determine the remaining leakage level of the traces to evaluate the benefit of a single countermeasure or to compare it with others. Hence, in this section two evaluation methods are introduced that quantify the leakage of a trace set.
Chapter 2. Background

Signal to Noise Ratio  SNR \[MOP07\] calculates the dependency between the power traces and the processed values by calculating the ratio between the signal and the noise level and hence quantifies the signal level (which is the exploitable part) in the power traces. SNR can also be used to analyze recorded power traces on higher statistical orders by applying the trace preprocessing described at the beginning of this chapter.

\[
SNR = \frac{\delta^2_{signal}}{\delta^2_{noise}}
\]

First, it has to be defined which value defines the signal in the traces. As an example a 128 bit plaintext is split into 16 bytes, while each byte defines the signal and can be evaluated independently. More precisely, let us denote \(N \) traces by \(l_i \in \{1, \ldots, N\}\).

The sets \(L^j_{x \in \{0,1\}^8}\) are formed as \(\{i|p^j_i = x\}\), where \(1 \leq j \leq 16\) identifies the targeted plaintext byte. The mean trace \(\mu^j_x\) and variance trace \(\delta^2_{x^j}\) are estimated by

\[
\mu^j_x = \frac{\sum_{l_i \in L^j_x} l_i}{|L^j_x|}, \quad \delta^2_{x^j} = \sum_{l_i \in L^j_x} \frac{(l_i - \mu^j_x)^2}{|L^j_x|}.
\]

Finally, for each plaintext byte \(1 \leq j \leq 16\), \(\delta^2_{signal}\) is estimated by

\[
\delta^2_{signal} = \sum_{x \in \{0,1\}^8} \frac{(\mu^j_x - \bar{\mu})^2}{256}, \quad \bar{\mu} = \sum_{x \in \{0,1\}^8} \frac{\mu^j_x}{256}.
\]

Correspondingly \(\delta^2_{noise}\) is estimated by

\[
\delta^2_{noise} = \sum_{x \in \{0,1\}^8} \frac{\delta^2_{x^j}}{256}.
\]

Perceived Information  Another leakage quantification method is the Perceived Information (PI), an Information-Theoretic (IT) metric \[SMY09\] that shows the dependencies between two random variables and hence the mutual shared information or entropy. The IT metric follows the standard definition of entropy (Shannon entropy) as the information content of a random variable.

\[
H[X] = - \sum_{\forall x \in X} \Pr[x] \log_2 \Pr[x]
\]

The Mutual Information (MI) of two random variables is defined as

\[
\]

The IT analysis examines the amount of exploitable information by estimating the mutual information between the processed data \(x \in X\) and the associated leakages \(l \in L\). The results of the analysis can be directly translated to the success rate of a univariate template attack \[RSV+11\].
2.2. Passive Implementation Attacks

\[ PI(X; L) = H[X] - \sum_{x \in X} \Pr[x] \sum_{l \in L} \Pr[l|x] \log_2 \Pr_{\text{Leakage}}[x|l] \]

\[ \Pr_{\text{Leakage}}[x|l] = \frac{\Pr[l|x] \Pr[x]}{\sum_{x' \in X} \Pr[l|x'] \Pr[x']} \]

For the IT analysis, the probability distribution of the leakage, i.e., \( \Pr_{\text{Leakage}} \) is unknown and has to be estimated for each data value \( x \). The usage of Gaussian templates to model the leakage Probability Density Function (PDF) was published in [DSV14] and first noted in [RSV+11] with the notion of PI. Hence, the leakage distribution is estimated to be Gaussian; the means and variances of the distribution are enough to perform a first-order analysis. The leakage probability for a given data value \( \Pr_{\text{Data}} \) is generated from a different and in general smaller trace set, e.g., by a simple histogram.

**Leakage Exploitation**

This section deals with key recovery attacks that are used in this work. Even from designers’ or evaluators’ point of view, it may be of interest to know the resistance of a device against actual key recovery attacks. Both described attacks, CPA and Moments-Correlating Profiled DPA (MCP-DPA), use a divide and conquer approach which splits the secret key into subkeys and attack them individually. For simplicity, the attacks and hence the given formulas describe the attack of a single key byte. To run a full key recovery, the given steps have to be repeated for the remaining subkeys.

**Correlation Power Analysis**

One of the first proposed key recovery attacks is the CPA [BCO04]. To perform a CPA attack, a set of power traces \( L = \{l_1, \ldots, l_N\} \), which base on the same secret key and random plaintexts, has to be available. Each recorded trace \( l_i \) bases on the processing of a secret intermediate value \( v_{k,i} \). This intermediate value depends on the subkey \( k \) and is going to be attacked. Typically, the output value of a non-linear cipher element is chosen. To perform the attack, a hypothesis \( \hat{k} \) for the attacked subkey is made and the power consumption of the key-dependent intermediate value is modeled with \( H_\hat{k} = \{h_{\hat{k},0}, \ldots, h_{\hat{k},N}\} \) for \( h_{\hat{k},i} = f(v_{k,i}) \) while \( f \) describes the chosen power model, which is typically the Hamming Weight (HW) or Hamming Distance (HD). To detect the dependencies between the hypothetical and measured power consumption, the Pearson correlation coefficient \( \rho_{L,H_\hat{k}} \) is calculated by

\[
\rho_{L,H_\hat{k}} = \frac{\text{cov}(L, H_\hat{k})}{\sqrt{\delta_L \delta_{H_\hat{k}}}},
\]

\[
\mu_L = \frac{\sum_{i=0}^{\mid L \mid-1} l_i}{\mid L \mid},
\]

\[
\mu_{H_\hat{k}} = \frac{\sum_{i=0}^{\mid H_\hat{k} \mid-1} h_{\hat{k},i}}{\mid H_\hat{k} \mid},
\]

\[
\mu_L = \frac{\sum_{i=0}^{\mid L \mid-1} (l_i - \mu_L)(h_{\hat{k},i} - \mu_{H_\hat{k}})}{\mid L \mid - 1},
\]

\[
\delta_L^2 = \frac{\sum_{i=0}^{\mid L \mid-1} (l_i - \mu_L)^2}{\mid L \mid},
\]

\[
\delta_{H_\hat{k}}^2 = \frac{\sum_{i=0}^{\mid H_\hat{k} \mid-1} (h_{\hat{k},i} - \mu_{H_\hat{k}})^2}{\mid H_\hat{k} \mid}.
\]
This is repeated for each key hypothesis while the hypothesis with the maximum correlation coefficient is probably the correct one.

**Moments-Correlating Profiled DPA**  In contrast to CPA, the MCP-DPA [MS14] relaxes the necessity of a hypothetical leakage model. MCP-DPA also calculates the Pearson correlation coefficient, but the correlation is calculated between the power traces and a statistical moment. The attack is performed in two phases: the profiling and the attack phase. For the profiling phase a set of traces \( LP = \{l_{P0}, \ldots, l_{PN_P}\} \) is required. Similar to CPA a key dependent intermediate value \( v_k \) is attacked. The profiling traces are grouped by this intermediate value, and the statistical moments are calculated. The formulas for the first moment are given by

\[
M = \{\mu_0, \ldots, \mu_{256}\}, \quad \mu_x = \frac{\sum_{i \in LP_x} l_i}{|LP_x|}, \quad \text{LP}_z \in \{0,1\}^s = \{i | v_{k,i} = x\}.
\]

Note that for the profiling phase of MCP-DPA the secret key has to be known to group the profiling traces correctly. For the attack phase a second set of traces \( LA = \{l_{A0}, \ldots, l_{AN_A}\} \) is required, while both sets can be of different size. The vector \( V \) holds the secret intermediate value processed during the trace recording so that \( V = \{v_{k,0}, \ldots, v_{k,N_A}\} \). This vector is unknown to an attacker. Equivalently to a CPA attack, a key hypothesis \( \hat{k} \) for each possible subkey is generated and the key-dependent intermediate value is calculated for each recorded power trace \( V_{\hat{k}} = \{v_{\hat{k},0}, \ldots, v_{\hat{k},N_A}\} \). The hypothesis of the intermediate value defines a new vector \( M_{\hat{k}} = \{\mu_{v_{\hat{k},0}}, \ldots, \mu_{v_{\hat{k},N_A}}\} \) consisting of the mean values of the hypothetical processed intermediate values. At least, the Pearson correlation coefficient \( \rho_{LA,M_{\hat{k}}} \) is calculated by:

\[
\rho_{LA,M_{\hat{k}}} = \frac{\text{cov}(LA,M_{\hat{k}})}{\sqrt{\delta^2_{LA}\delta^2_{M_{\hat{k}}}}}, \quad \text{cov}(LA,M_{\hat{k}}) = \frac{\sum_{i=0}^{[LA]-1} (l_{ai} - \mu_{LA})(\mu_{v_{k,i}} - \mu_{M})}{|LA|},
\]

\[
\mu_{LA} = \frac{\sum_{i=0}^{[LA]-1} l_{ai}}{|LA|}, \quad \mu_{M} = \frac{\sum_{i=0}^{[M]-1} \mu_{i}}{|M|},
\]

\[
\delta^2_{LA} = \frac{\sum_{i=0}^{[LA]-1} (l_{ai} - \mu_{LA})^2}{|LA|}, \quad \delta^2_{M} = \frac{\sum_{i=0}^{[M]-1} (\mu_{i} - \mu_{M})^2}{|M|}.
\]

The attack phase is repeated for each possible key hypothesis while the hypothesis with the maximum correlation coefficient is probably the correct key.

**2.2.4 Previous Work**

Despite of the wealth of related published research, all previously reported DRP schemes suffer from at least one of the aforementioned problems (glitches, EP, or wire imbalances). This section briefly reviews the most promising DRP schemes available for FPGAs and compares them with respect to the resource overhead they induce.
2.2. Passive Implementation Attacks

Dual-Rail Precharge (DRP) logic styles consist of gates with two outputs, $O_t$ and $O_f$. In the precharge phase both outputs show the same value ($O_t = 0, O_f = 0$ or $O_t = 1, O_f = 1$), while in the evaluation phase only one output changes its state so that exactly one signal transition per phase transition is guaranteed. $O_t$ presents the true value while $O_f$ the false pendant. The signals driven by the true outputs form a network we address as true network (respectively false network made by the false outputs). Some DRP schemes allow the presence of negative gates while others do not. If negative gates, e.g., NAND or NOR gates, are not provided by the logic style, the corresponding non-negative gates (respectively AND or OR) are instantiated with swapped output signals. Clearly, in those logic styles an inverter gate is realized by swapping the dual rails. Hence, such logic styles form a circuit with interconnected true and false networks. Further, we define a subgroup of DRP schemes called duplication-schemes that follow the duplication concept of [YS07]. Duplication schemes which instantiate two independent circuits while the circuits, respectively, form one of the (true or false) networks.

The purpose of a DRP logic styles is to increase the security level with respect to the resistance against SCA attacks. Applying a DRP logic style expectedly increases the resource utilization of a design. The efficiency of a logic style can hence be defined as the ratio of achieved resistance level per deployed resource. The logic styles presented in this chapter have been published over the last 10 years while different devices, setups, designs under test, and metrics have been used for their evaluation. A fair comparison of their security level is hard to achieve without reimplementing and evaluating all schemes under equal conditions. Hence, this chapter compares the occupied resource overhead of former logic styles in a theoretical way, using the Canright AES Sbox [Can05] with buffered input and output signals as case study.

To estimate the resource overhead of a logic style, we first estimate the resource consumption of a single logic gate and further scale the resource utilization to a full Canright AES Sbox, implemented on a Xilinx Spartan-6. Note that the resource estimation in this chapter bases on post-map values which means that resources occupied by the place & route process (e.g., LUTs configured as pass-through to achieve minimal routing delays) are neglected.

**Wave Dynamic Differential Logic**

WDDL [TV04] is one of the most common DRP styles initially designed for ASICs. WDDL allows only AND and OR gates while an XOR gate is constructed by two AND and one OR gate. As stated, modern Xilinx FPGAs are equipped with 6-to-2 LUTs. This gives the advantage to realize each WDDL gate by one LUT. A building block of WDDL – with respect to 6-to-2 LUTs – can be seen in Figure 2.3. The transitions between the precharge and evaluation phase of a WDDL circuit are controlled and triggered by its input signals.

As stated in [SS06] and [BGF+10], a WDDL gate evaluates its output at different points in time depending on the input signals. For example, $O_f$ of a WDDL AND gate is derived from two signals of the false network. Regardless of other gate inputs, $O_f$ evaluates as soon as one of its inputs from the false network rises to logic high. This phenomenon is known as EP effect and causes a data-dependent power consumption and hence side-channel leakage.

**Resources Utilization** Each WDDL gate can be implemented in a single 6-to-2 LUT. The resource consumption of a Canright AES Sbox in WDDL is hard to estimate. Hence we took the numbers from our AES implementation given in Section 3.4 which makes use of the Canright
Chapter 2. Background

Sbox. Our Sbox implementation requires 286 LUTs and 32 FFs to buffer the Sbox Input/Output (I/O).

![Figure 2.3: WDDL AND Gate.](image)

**Dual-Rail Precharge Logic without Early Evaluation**

DPL-noEE [BGF+10] is unofficially the successor of WDDL and addresses the EP present in WDDL. As shown in Figure 2.4, $O_f$ of a DPL-noEE AND gate depends on all four input signals, so that the gate only evaluates with the availability of all signals. Due to the consideration of the true and false signals in each part of the gate — contrary to WDDL — a DPL-noEE XOR gate can be natively constructed by a single LUT. This results in reduced LUT utilization, particularly in designs with a high number of XOR gates.

The authors of [MI14] noted in their work that DPL-noEE prevents the early propagation just at the beginning of the evaluation phase. It has been shown that the output of DPL-noEE gates falls back into precharge as soon as one of the input signals changes; hence the propagation in the precharge phase is faster. Although such an issue is not data dependent, it has been shown in [MI14] that the amount of power consumption (respectively amount of side-channel leakage) at the precharge phase is higher than that of the evaluation phase.

**Resources Utilization** Similar to WDDL each DPL-noEE gate can be implemented in one 6-to-2 LUT. Compared to WDDL, not only non-linear gates are allowed, but also XOR gates can be implemented [BGF+10]. Again, we refer to our implementation in Section 3.4. The AES Sbox instantiated in DPL-noEE utilizes 122 LUTs and 32 FFs.

![Figure 2.4: DPL-noEE AND Gate.](image)
2.2. Passive Implementation Attacks

Asynchronous Wave Dynamic Differential Logic

AWDDL emulates a Reset-Set-Latch inside each gate by looping the gate output to its input. Hence, an AWDDL gate does not change its state until all input signals are in the evaluation or precharge phase. Figure 2.5 shows the internal structure of an AWDDL AND gate. In comparison to WDDL and DPL-noEE, one AWDDL gate should be realized by two 6-to-1 LUTs. In fact, a true 6-to-2 LUT would suffice to make both outputs of a gate. However, as shown in Figure 2.1(b) the 6-to-2 LUT available in Xilinx FPGAs are made of two multiplexed 5-to-1 LUTs [Xil13c].

The necessity of employing two 6-to-1 LUTs per AWDDL gate as well as the gate loop paths increases the resource utilization and routing complexity compared to DPL-noEE. The authors of [MI14] have proposed to place both LUTs of each gate into the same slice to minimize the delay difference between the true and false outputs. Further, a customized router has been developed which tries to find equivalent routes for coupled true and false signals. The authors reported that the router improves the result but does not fully avoid the leakage associated to the signal delay differences.

Resources Utilization The implementation properties of AWDDL are similar to DPL-noEE with one exception: one gate requires 2 LUTs [MI14]. This doubles the LUT utilization compared to DPL-noEE and results in 244 LUTs and 32 FFs.

Figure 2.5: AWDDL AND Gate.

Triple-Rail

The authors of [LMT+09] employed an asynchronous approach by making use of the so-called Triple-Rail approach and evaluated their implementation on a Spartan-3 FPGA. In this approach an asynchronous control signal is connected to the gates which is considered as being the latest arriving signal. This control signal fires the gate in order to prevent early propagation.
and glitches. Figure 2.6 shows a compact Triple-Rail AND gate as presented in \([LMT^{+}09]\). No routing strategy is considered in this work so that different routing delays will show up between the original and the dual circuit.

**Resources Utilization** A Triple-Rail gate processes 2 input signals and can be implemented, as noted in \([LMT^{+}09]\), by 6 4-to-1 LUTs. To be more fair, we assume that adapting the technique to more recent devices using 6-to-2 LUTs, the number of LUTs per gate can be reduced to 3. This ends up in a resource consumption of 366 LUTs and 32 FFs for an AES Canright Sbox that follows the Triple-Rail concept.

![Triple-Rail Gate](image)

**Balanced Cell-Based Dual-Rail Logic**

In \([NBD^{+}10]\) a dual-rail logic style called Balanced Cell-based Dual-rail Logic (BCDL) that employs a rendezvous box to connect all gate inputs with a global precharge signal was introduced. One of these rendezvous boxes is placed in front of each gate (see Figure 2.7) to fire the gate evaluation as soon as all dual-rail input signals become stable. BCDL prevents early propagation but is not aware of different wire delays which end up with different route capacitances between the true and false networks of the dual-rail circuit – finally leading to information leakage.

**Resources Utilization** The authors of BCDL note that every (at most) 2-input gate can be implemented in BCDL by utilizing 2 LUTs \([NBD^{+}10]\). Each gate that can be realized with 1 or 2 input signals is either a representation of an inverter, AND, OR, or XOR gate (with/without inverted input/output values). Based on our observation, a BCDL-based circuit consumes the same number of LUTs and FFs as its AWDDL equivalent design.

**Double Wave Dynamic Differential Logic**

The seminal work \([YS07]\) introduces Double Wave Dynamic Differential Logic (DWDDL) which duplicates a fully placed-and-routed WDDL circuit resulting in a massive resource utilization. An example gate is given in Figure 2.8. Beside the duplication, the gate functionality of each gate in the duplicated circuit is inverted. Since the duplication process keeps the routing information of the original WDDL circuit, two true and two false networks are formed which are pairwise balanced and eliminates or more precisely minimize the capacitance differences between coupled
2.2. Passive Implementation Attacks

signals. However, we will show in Chapter 3 that this scheme shows a data-dependent time of evaluation and hence exhibits leakage even for perfect balanced wires.

Resources Utilization  One WDDL gate can be implemented in a single 6-to-2 LUT and deals with two data signals. DWDDL performs a further duplication so that 2 LUTs per gate are required [YS07]. This logic style is restricted to non-linear gates which impedes a proper resource estimation. Therefore, we refer to the resource utilization of the AES Sbox in our implementation (see Section 3.4) which is also synthesized for Spartan-6. Our Sbox implementation requires 572 LUTs and 64 FFs.

Partial Separated Dynamic Differential Logic

Partial Separated Dynamic Differential Logic (SDDL) [KV10] also follows the duplication concept and inserts asynchronous latches to precharge the dual-rail circuit. Further, the precharge signal is forwarded from FFs placed next to the gate. As noted by the authors, the logic style is vulnerable due to upcoming glitches. Figure 2.9 depicts the concept of an SDDL gate.
Chapter 2. Background

Resources Utilization Each SDDL gate consumes 2 LUTs, 2 latches, and 2 FFs while the full capability of the LUT can be used [KV10]. Hence, SDDL doubles the LUT utilization. To keep the same design structure, the input and output signals of the Sbox are still buffered, which adds additional 32 FFs to the design. In total, 108 LUTs, 108 Latches, and 140 FFs are required for the SDDL-based AES Canright Sbox implementation.

![SDDL Gate](image)

Figure 2.9: SDDL Gate.

Precharge Absorbed Dual-Rail with Precharge Logic

The authors of [HdlTR11] mainly focused on the well-known early propagation effect by connecting control signals to every LUT of a fully combinatorial circuit with duplicated routing, as given in Figure 2.10. However, the circuit starts to glitch right after the first LUT stage if no register is present between the subsequent LUTs to stop the signal propagation.

![PA-DPL Gate](image)

Figure 2.10: PA-DPL Gate.

In a more recent work [HOdlTR12], the authors applied duplication while minimizing the area overhead. Their underlying strategy is to duplicate the original circuit to realize the dual one, but – in contrast to all other similar works – these two subcircuits are placed in an interleaved way but are not connected to each other. This strategy results in routing conflicts which are iteratively solved by an algorithm the authors provide.

Resources Utilization Similar to Glitch-Free Duplication (GliFreD) (presented in Chapter 4), PA-DPL connects two control signals to each LUT so that the LUT utilization of PA-DPL is equivalent to GliFreD. Under the condition that the input and output signals of the Sbox are buffered, the implementation utilizes 126 LUTs and 32 FFs.
2.3 Physical Unclonable Functions (PUF)

Summary

Like previously noted, each of the so far published DRP logic styles suffers from at least one of the well known issues: early propagation, glitches, or different wire capacitance. The flaws of each logic style and the resources utilization is summarized in Table 2.3. Note that the numbers given in Table 2.3 are post-synthesis numbers.

To get a single comparable number of the implementations, we decided to compare the induced overhead of each logic styles on slice level. On Spartan-6 devices each slice is equipped with 4 LUTs and 8 FFs (while only 4 of them can be used as latch). The slice utilization numbers for each logic style are given in Table 2.3 and are the minimum number of slices required to place the Sbox with the estimated LUT and FF resources. The minimum number of slices is estimated by $\text{Slice} = \max\left(\lceil \frac{\text{LUT}}{4} \rceil, \lceil \frac{\text{FF}}{8} \rceil, \lceil \frac{\text{Latch}}{4} \rceil \right)$, whereby the routing difficulties are ignored.

Table 2.3: Estimated post-synthesis resources and properties of the most common logic style available for FPGAs.

<table>
<thead>
<tr>
<th>Logic Style</th>
<th>Native Gates</th>
<th>Resources/Gate</th>
<th>Addressed Flaws</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>LUT FF</td>
<td>LUT FF Slice Unprotected</td>
</tr>
<tr>
<td>None</td>
<td>6-to-1</td>
<td>1 -</td>
<td>54 16 14 1</td>
</tr>
<tr>
<td>WDDL [TV04]</td>
<td>AND, OR</td>
<td>1 -</td>
<td>286 32 72 5.14</td>
</tr>
<tr>
<td>DPL-noEE [BGF+10]</td>
<td>AND, OR XOR</td>
<td>1 -</td>
<td>122 32 31 2.21</td>
</tr>
<tr>
<td>AWDDL [MI14]</td>
<td>AND, OR, XOR</td>
<td>2 -</td>
<td>244 32 61 4.36</td>
</tr>
<tr>
<td>Triple-Rail [LMT+09]</td>
<td>AND, OR XOR</td>
<td>3 -</td>
<td>366 32 92 6.57</td>
</tr>
<tr>
<td>BCDL [NBD+10]</td>
<td>AND, OR, XOR</td>
<td>2 -</td>
<td>244 32 61 4.36</td>
</tr>
<tr>
<td>DWDDL [YS07]</td>
<td>AND, OR</td>
<td>2 -</td>
<td>572 64 143 10.21</td>
</tr>
<tr>
<td>Partial SDDL [KV10]</td>
<td>6-to-1</td>
<td>2 4</td>
<td>126 32 32 2.29</td>
</tr>
<tr>
<td>PA-DPL [HdlTR11]</td>
<td>4-to-1</td>
<td>2 -</td>
<td>126 32 32 2.29</td>
</tr>
</tbody>
</table>

✓: Solved
(✓): Partially solved

2.3 Physical Unclonable Functions (PUF)

A Physical Unclonable Function (PUF) is a function defined by the physical conditions of an object the PUF is instantiated on. The object specific physical conditions are induced during the object manufacturing process. Unclonable is in this context defined as physically unclonable, which means that it should be extremely hard to produce two objects which come up with the same physical conditions. Even with an extremely controlled manufacturing process no identical objects should be producible due to uncontrollable random effects.
Chapter 2. Background

PUFs provide an instance specific response \( r \in \{0, 1\}^* \) for a given challenge \( c \in \{0, 1\}^* \). Based on the construction, evaluation and the underlying object a PUF is instantiated on, PUFs are classified into electronic, non-electronic, silicon, intrinsic, and non-intrinsic PUFs.

(Non-)electronic PUF: Non-electronic PUFs are all PUF classes whose behavior is not determined by an electrical component, e.g., optical PUFs which use random reflections or scattering characteristics of an optical medium. Electrical PUFs are PUF constructions which use variations of electronic characteristics, e.g., capacitance or resistance of electronic components.

Silicon PUF: Silicon PUFs are a subclass of electronic PUFs and are defined as PUF constructions that are embedded in a silicon chip. In the field of hardware security and secure key management, these PUF constructions are of major interest.

(Non-)intrinsic PUF: Intrinsic PUFs are defined as PUF construction which are able to evaluate internally with an embedded measurement setup and derive its random instance behavior implicitly during the manufacturing process.

With respect to the motivation of this thesis, the focus of this work is clearly on intrinsic silicon PUF constructions that can be instantiated on already fabricated FPGAs. An overview of proposed FPGA-based intrinsic PUF constructions is given in Section 2.3.4.

2.3.1 Weak and Strong PUFs

PUF constructions are beside their underlying medium and evaluation methods classified by their challenge and response behavior. The literature distinguishes between weak and strong PUFs. A PUF construction is called a strong PUF if an instance of the PUF can be hand out to an adversary for a long time period in order to collect as much challenge and response pairs as possible. After that time period, it should be possible with a high probability to find challenges to which the adversary does not know the response. This requires a PUF construction which at least has a large challenge space to hinder an adversary to query all possible challenges. Additionally, the challenge and response pairs the adversary has collected should not leak enough information about the embedded randomness to model the PUF instance. All PUF constructions that do not fit to the definition of a strong PUF are thus called weak PUF. PUF constructions with a single challenge are also noted in the literature as Physically Obfuscated Key (POK).

2.3.2 Evaluation Metric

To clearly define the properties and the quality of PUF constructions, a unified metric needs to be used. In general, two distance metrics are used: the intra-distance and the inter-distance.

Intra-Distance: The intra-distance is defined as the distance between two PUF responses, given by the same PUF instance, for the same challenge, under the same environmental conditions. The distance can be calculated by any given distance metric while the literature mostly uses the HD.
2.3. Physical Unclonable Functions (PUF)

**Inter-Distance:** The inter-distance is defined as the distance between two PUF responses, given by different PUF instances, for the same challenge, under the same environmental conditions. Similar to the intra-distance, any distance metric can be used, while the HD is quite common in the literature.

PUFs are constructions that make use of the physical properties of their underlying object and evaluates them. These physical object properties are drastically influenced by the conditions the environment provides. For intrinsic PUFs, the major aspects that affect the response behavior are the temperature and the provided voltage. Even small changes in at least one of these parameters can drastically change the PUF response behavior. Hence, to evaluate the quality of a PUF construction, the quality related tests have to be repeated under different environmental conditions. Note that equalizing the conditions for the PUF instance is challenging in practice, e.g., during run time an electronic device is going to change its temperature based on its workload (caused by the PUF instance or control logic) and hence, the device-specific current consumption. The voltage provided to a device differs as well, e.g., caused by manufacturer differences of the Printed Circuit Board (PCB) or voltage regulators. Hence, performing the evaluation under exactly the same conditions for each PUF instance is hard in practice.

### 2.3.3 Properties

The quality of a PUF construction bases on different properties the construction inherently involves. These properties can strongly vary between different PUF constructions and even between the same constructions instantiated on different physical objects, i.e., different device types. This section briefly introduces the main PUF characteristics which in total determine the quality of a PUF construction.

**Reliability:** One of the most important quality characteristic of PUF constructions is the reliability. The reliability (or often noted as reproducibility or stability) is defined as the statistical deviation a PUF instantiation shows in its responses given by the same challenge. The deviation is often estimated by calculating the mean and the variance of the intra-distance for a set of challenges and instances. Ideally, both statistical moments are as small as possible. Since the response behavior of the PUF instances are influenced by environmental conditions, the reliability tests have to be performed under the same conditions for each PUF instance and repeated under several environmental conditions to get meaningful results.

**Uniqueness:** The uniqueness is beside the reliability another important characteristic and is defined as the statistical deviation a PUF construction shows in its responses given by different challenges. Commonly, this deviation is estimated by calculating the mean and the variance of the inter-distance for a large set of challenges and instances. Ideally, the responses of the PUF construction differs in 50% of their bits.

**Unpredictability:** Another important property a PUF construction has to come with is the unpredictability of responses. Even though an adversary has knowledge about a big subset of challenge and response pairs, the adversary should not be able to determine the response for a given challenge which is not in the known subset.
Chapter 2. Background

**Unclonability:** The unclonability property is inherently given by the PUF definition. To recap, a PUF construction is unclonable if it is hard to produce two physical objects with the same physical properties (which are evaluated by the PUF construction) even with an extremely controlled manufacturing process.

The literature defines further properties, which partially extend the given definitions, e.g., identifiability, mathematical unclonability, or one-wayness. These properties are often used in mathematical proofs and hence not further discussed. More information about PUF properties is given in [Mae13].

2.3.4 Previous Work

The PUF community has grown rapidly in the last years and proposed several PUF constructions as well as post-processing techniques. Nonetheless, just a few of those PUF classes are able to be instantiated properly on FPGAs. This chapter introduces recent and most promising PUF constructions proposed to be implementable on FPGAs.

**SRAM PUF**

SRAM is a common technology for volatile memory that loses its content shortly after power-down. An SRAM cell achieves the memory effect by two cross-coupled inverters (see Figure 2.11). The inverter circuit has two stable states (bistable) which represent the bit value stored in the cell. The initial state after power-up is determined by the transition speed and strength of the inverters’ transistors. In theory, the transistors of both inverters are equally built and balanced so that the initial state is random at each power-up. But in practice SRAM cells show a bias due to process variations. The unpredictable initial state of biased SRAM cells is hence used as a PUF response.

![Figure 2.11: The SRAM cell on logic level.](image)

The SRAM PUF, introduced by Guajadro et al. [GKST07], shows promising results, but some modern FPGAs are equipped with an automatic triggered power-up reset and delete the initial SRAM values. Hence, further PUFs constructions have been introduced which work on a similar principle of cross-coupled memory elements [SHO07, MTV08, KGM+08, SvdSvdL12, BNCF14].
2.3. Physical Unclonable Functions (PUF)

The SRAM PUF is one of the most popular and well studied PUF constructions. The concept has been analyzed on different hardware components like FPGAs [GKST07], commercial SRAM components, or microcontrollers [HBF07, HBF09] but also on different process technologies [SKA+11, SvdL12] with respect to their reliability, uniqueness and aging.

Even though the SRAM PUF provides promising results, a lot of effort has been done to improve the results, e.g., on transistor level [VN09], by optimizing the voltage ramp-up time [ChvdL+13, CHK+15], or using anti-aging strategies [MvdL14] to increase the reliability of the PUF class.

Despite of the noted improvements, some of the SRAM cells are well balanced so that the initial value of those cells is unstable and hence drastically reduces the reliability of this PUF class. A straightforward approach to increase the reliability is to mask the memory block and just make use of stable cells [HB10]. Another approaches to deal with unstable cells are, e.g., to make use of repetition coding [BHP11] and evaluate each cell multiple times or to use controlled cell aging [GK14].

As noted, a general use case of weak PUFs is the device internal secret key generation. Hence, dealing with the physical security of SRAM memory is another aspect that has to be considered. Motivated by the security aspect of SRAM PUFs, research results in the field of invasive attacks [NSHB13], semi-invasive attacks [SSAQ02], and non-invasive attacks [OSW13, ZOW+16] to readout the SRAM cell content had been published. The work of Helfmeier et al. [HBNS13] is going one step further and produces a physical clone of an SRAM PUF.

Butterfly PUF

One of the proposed PUF constructions working on the principle of cross-coupled memory elements is the Butterfly PUF [KGM+08] shown in Figure 2.12. The Butterfly PUF uses cross-coupled latches connected to an excite signal. The excite signal presets one and resets the other latch when turned high. While the excite signal is set to high, the latches show different signal levels on their input and output ports. Releasing the excite signal turns the loop, formed by the latches, into a metastable state. Based on driving strength of the latches as well as the capacitance of the wires, the Butterfly PUF resolves the metastable state. The stable state is hence defined as PUF response.

![Figure 2.12: The Butterfly PUF.](image-url)
This PUF construction was proposed and evaluated on Xilinx Virtex-5 FPGAs. Morozov et al. revisited this construction on Xilinx Spartan-3E devices [MMS10] and contradicted the good results of this construction noted in [KGM+08]. As previously noted, the state, in which the PUF settles, is highly influenced by the wire capacitance and requires a very symmetric structure to produce unbiased results. Even though the Butterfly PUF has been introduced to be a PUF class for FPGAs, Morozov et al. have shown in [MMS10] that the required routing symmetry is hard to achieve with the limited routing structure of FPGAs. It was noted, that the delay difference and hence the wire capacitance induced by the static variation are about an order of magnitude higher than the variation induced by the manufacturing process. A technique called Programmable Delay Line (PDL) which has been originally proposed in the context of Arbiter PUFs may help to minimize the static variance and hence the routing bias. A more detailed description of PDL is given later on in this section. Beside the structural bias of the wires connecting the latches, the excite signal needs to release the two latches at the same time to ensure an unbiased starting situation. This further requires a symmetric routing of the control signal and is another aspect that has to be considered during the implementation process of this PUF class.

**Ring Oscillator PUF**

A RO is an asynchronous device element realized by an AND gate and an odd number of inverters connected to a cycle. The cycle can be enabled via a control signal connected to the AND gate. The oscillation frequency of an enabled RO depends on the routing capacitances as well as the transistor transition speeds. Therefore, the RO frequency depends on the design, e.g., the number of inverter stages but also on process variations. Hence, ROs with identical layouts will not show the same frequency once they are fabricated. This process variation dependency is used in the RO PUF to generate a PUF response.

In the classical approach introduced in [SD07] and shown in Figure 2.13 the frequency of two ROs is indirectly measured by keeping both ROs oscillating for a specified time interval $T$ and counting the number of rising edges. Subsequently, both counter values $counter = \frac{T}{f_{RO}}$ are compared to generate a single response bit, based on the algebraic sign of the counter difference. The classical RO PUF approach extracts only a single bit from two ROs. Creating a response based on this approach consumes much area and hence some research has been done to decrease the area utilization, increase the number of response bits extracted from the ROs, and improve the properties of the concept.

Maiti and Schaumont proposed in [MS11] an FPGA friendly RO PUF concept which increased the reliability of the concept by investigating more area. For each inverter stage of the RO a

![Figure 2.13: Architecture of a classical RO PUF.](image-url)
set of inverters is instantiated while only one of these inverters is selectively part of the RO loop. During an analysis phase, the frequency differences of two ROs are determined for all configurations while the configuration for both RO are kept synchronized to avoid the induction of layout specific variances into the responses. The configuration which generates the highest frequency difference is then stored in the device and used to produce the PUF responses after the device’s roll out. Due to the usage of frequencies with a higher difference, the noise level on the ROs can be higher before the sign bit of the frequency difference changes. Hence, the concept is more robust against noise and shows an increased reliability. The architecture proposed in [HGK13] benefits from the same idea but makes use of LUT internal delays to tweak the RO frequencies.

The work of Yin and Qu [YQ10] increases the number of extractable bits from the PUF by taking more ROs into account during the response generation. Instead of comparing two ROs, the PUF response is gathered from sorting a set of ROs. Hence, with this concept the authors claim to extract $\log_2(n!)$ response bits from $n$ ROs.

Since the RO PUF is, besides the SRAM PUF, one of the most analyzed PUF concepts, the resistance against physical attacks has been analyzed as well. For instance, the authors of [MSSS11, MHH+13] successfully extracted the RO frequencies via EM analysis from a Spartan-3A FPGA and hence the PUF response. But also active attacks had been applied to change the RO behavior [BBA+12]. Nonetheless, further aspects like aging [RFFT14], locking phenomenon [BBFV10], influence of surrounding logic [EB11], or temperature-aware designs [Qu09] had been addressed as well.

**TERO PUF**

The Transient Effect Ring Oscillator (TERO) PUF proposed in [BNCF14] makes use of two (or more) cross-coupled inverters. Additionally, a control signal, two AND gates, and a counter are added to the construction as shown in Figure 2.14. The control signal is set for some time and turns the loop formed by the inverters and the AND gates into a metastable state. Releasing the control signal let the loop oscillate for a short time period before a stable state is settled. The oscillations are counted by an asynchronous counter connected to one of the inverter outputs and base on the frequency and the oscillation time of the loop. The origin work about TERO PUF [BNCF14] uses only reliable and unique bits of the counter value as PUF response. Instead of a binary encoding, the counter values can be encoded in Gray code which has a positive effect on the reliability of the PUF class. The consecutive values encoded in Gray code differ in only one bit position which means that a low variance in the counter values has a lower impact on their distance. Since the TERO PUF concept proposed in [BNCF14] makes partial use of the counter values as response, the Gray encoding minimizes the distance between two slightly varying counter values and hence improves the reliability. As an alternative to the proposed response generation, the counter values of multiple TERO instances can be combined to form the PUF responses, e.g., as used in the RO PUFs concept.

One fact that makes this PUF class a promising concept for FPGAs, is that a loop formed by the TERO does not need to be symmetrically instantiated. In case of a perfectly balanced loop, the TERO would never settle into a stable state and the response would only rely on the frequency of the loop. In case of a very unbalanced architecture, the loop would just oscillate for a short time and the uniqueness given by the counter value is expected to be low. Hence,
a somewhat balanced design seems to be optimal for this PUF class, which is an achievable condition on FPGAs.

The two connected parameters that a designer of this PUF class needs to care about are the acquisition time and the number of inverters per loop. Some of the TERO loops settle pretty quick while others oscillate much longer. In case of a short acquisition time, some of the loops do not reach their final state and the entropy extracted from the loops is just given by the loop frequency which drastically reduces the uniqueness [MBC16]. In case of a long acquisition time, most of the loops have been settled, but the loops are more influenced by noise which reduces the reliability [MBC16]. In this context, it has been shown in [CBM16] that with increasing number of inverters per loop the settle time increases as well.

The proof of concept implementation of a TERO PUF has been done on Altera Cyclone II FPGAs without taking environmental conditions into account. In [MBC16] the concept has been further characterized under environmental conditions on Altera Cyclone II and Xilinx Spartan-6 devices, and [CBM16] extended and characterized the concept on an ASIC with promising results.

**Loop PUF**

The Loop PUF [CDGB12] was originally proposed as a Strong PUF and forms a loop of $n$ delay stages (and if necessary an additional inverter), which is oscillating similar to the RO PUF (see Figure 2.15(b)). Each delay stage contains $m$ controllable delay elements. A delay element, as given in Figure 2.15(a), selects between two paths the signal has to use. Each delay stage $i$ is hence controlled by an $m$-bit challenge word $C^m_i$. For response generation, the frequencies of the loop are measured for a word-wise rotating challenge $C^m_1, \ldots, C^m_n$. The response bits are defined as the sign bits of frequency differences. The Loop PUF uses the same structures for different challenges, which makes it vulnerable to, e.g., machine learning algorithms [RSS*10, GTS15]. In Chapter 8 we use the Loop PUF as a Weak PUF construction by ensuring that no entropy is shared among different responses at the cost of a greatly reduced challenge space of one challenge per delay stage.

The authors of [CDGB12] claim that the Loop PUF is hardened against environmental influences since all delay elements are equally affected by the noise. The results presented in Chapter 8 reject this claim and show that the Loop PUF is highly affected by a noise source placed on the FPGA. The structure of the Loop PUF is quite similar to the RO PUF while, in general, the acquisition time of the Loop PUF is higher compared to a RO PUF instance. This is, on the one side, related to the large cycles instantiated by the Loop PUF and, on the
2.3. Physical Unclonable Functions (PUF)

other side, related to the fact that the same cycle needs to be executed at least twice for a response generation. One positive aspect of the Loop PUF is its simple structure which is easy to be instantiated on FPGAs. The cycle itself does not need to be constrained and just the delay stages require to have the same layout. For a more detailed comparison of RO, Loop, and TERO PUFs see Chapter 8.

Bistable Ring PUF

The Bistable Ring PUF \[\text{CCL}^{+11, \text{CCL}^{+12}}\] forms a cycle of an even number of inverter stages. The loop formed by the inverters has two stable states. During power-up the ring stabilizes into one of those states. Since FPGAs are configured after they are powered and the initial state of the loop is determined by the configuration process, the concept is modified to be implementable on FPGAs as shown in Figure 2.16. The inverters in the loop are replaced by NOR gates and a global reset signal is attached. A high signal at reset moves the loop into a metastable state which is resolved by the loop after reset is set to low. The final settled state of the loop is defined as the response bit.

Figure 2.16 shows a simplified version of the Bistable Ring PUF, as the concept was proposed as a Strong PUF by multiplexing between two NOR gates per stage to create the challenge space. Nevertheless, it had been shown in [YTST14, SH14, GTFS16] that a strong dependency between the challenge and response bits exist which can be extracted by machine learning approaches and used to model the PUF.

Furthermore, the Bistable Ring requires to have a symmetric and balanced layout to produce unbiased responses. This requirement is hard to achieve on FPGAs due to the limited placement and routing capabilities. Hence, even the weak PUF construction shows significant weaknesses, as given in [CCL^{+11}].

To circumvent the bias in the response bits and the dependencies between the challenge and response space, the authors of [BGK04] proposed to combine multiple Bistable Rings by XORing
their output. Indeed, this improves the uniqueness and hardens the PUF class against machine learning attacks, but, on the downside, it decreases the reliability and also increases the area utilization of the PUF construction.

**Arbiter PUF**

The classical Arbiter PUF proposed in [LLG+04] consists of two signals racing against each other to an arbiter. Along the path, both signals have to pass several multiplexer stages with two inputs and two outputs. A challenge bit connected to a multiplexer stage defines which input is connected to which output and hence defines if a wire swap is performed or not. Based on the first arriving signal, the arbiter evaluates to 0 or 1 which is also used as response value of the PUF. The full construction can be seen in Figure 2.17.

![Figure 2.17: Architecture of an Arbiter PUF.](image)

The Arbiter PUF was proposed as a Strong PUF while it turned out in [LLG+04] that the responses leak enough information to successfully run machine learning attacks to model the internal signal delays which are the entropy source of the PUF construction. To harden the concept against machine learning attacks, the XOR Arbiter PUF has been introduced in [SD07]. The XOR Arbiter PUF combines (i.e., XOR) the output of multiple Arbiter PUFs to hide the direct delay dependencies in the response bits. Nonetheless, Rührmair et al. have shown in [RSS+10] that this concept does not fully overcome the problem, and XOR Arbiter PUFs are still vulnerable against machine learning attacks. Another disadvantage of the XOR Arbiter PUF concept is that the reliability diminishes exponentially with the number of involved Arbiter PUFs. Beside machine learning attacks applied to this PUF class also physical attacks had been successfully applied [TDF+14, TDF+17].

Implementing the Arbiter PUF concept properly on FPGAs is also hard to achieve since all signal delays and multiplexer structures have to be balanced to avoid structural dependencies in the response bits. Especially, the multiplexer stages and the routes to the arbiter are hard to balance with the limited routing resources FPGAs provide. Hence, Morozov et al. proposed an arbiter-like PUF structure which bases on the concept of Programmable Delay Lines (PDLs) [MKD10]. The arbiter-like construction is depicted in Figure 2.18(b) and forms two independent paths passing multiple PDL before entering an arbiter. A PDL is a LUT configured as a pass-through LUT and simply forwards the input signal connected to the first multiplexer stage to the LUT output. The inputs of the remaining multiplexer stages are set to a constant value and hence determine the signal path through the LUT. As shown in Fig-
Figure 2.18(a), the internal paths differ by their propagation delays due to the LUT-internal wire length, the usage of different multiplexer stages, and hence different transistors. Both properties, the structural difference and the manufacturer differences embedded in the transistors and wire capacitances are used by the arbiter-like PUF concept proposed in [MKD10].

Similar to the classical Arbiter PUF approach, two signals race against each other to an defined arbiter. On the way to the arbiter, both signals pass multiple PDL stages but are not swapped like in the classical Arbiter PUF concept. The purpose of PDLs is split into two parts. The first part of PDL stages is connected to challenge signals that keep for both signals the same LUT internal configuration and add the entropy given during the manufacturing process to the signals. The second part of PDL stages is used to compensate structural biases in the signal paths, e.g., the different routing used to connect the competing signals with the arbiter. Hence, the different path lengths of the multiplexer tree inside a LUT of the second PDL part are used to add configurable signals delays in order to remove the structural bias from the competitive signals.

(a) Structure of a 3-to-1 LUT and PDL concept.

(b) Architecture of an arbiter-like PUF construction.

Figure 2.18: Arbiter-like PUF based on PDLs.
Part II

Power-Equalization as Side-Channel Countermeasure
Chapter 3
Evaluating the Duplication Concept

Wire balancing is due to the limited routing resources FPGAs provide one of the most challenging tasks in DRP design. Hence, the duplication concept proposed by Yu and Schaumont [YS07] provides a promising solution. Nonetheless, we show in this chapter that the duplication process actually provides a solution for wire balancing but, on the other hand, leads to a data-dependent time of evaluation when applied to some logic styles.

Contents of this Chapter

3.1 Introduction ........................................ 41
3.2 Performing Circuit Duplication ....................... 43
3.3 Data-Dependent Time of Evaluation .................. 44
3.4 Practical Evaluation ................................... 46
3.5 Conclusion ........................................... 49

3.1 Introduction

SCA is one of the major challenges for secure-hardware designers. The most popular techniques to harden a design are hiding, masking, and rekeying. The goal of power-equalization schemes, which are a part of the hiding category, is to equalize the power consumption of a cryptographic circuit independent of the processed data. In hardware devices such schemes often follow the DRP logic concept like Sense Amplified Based Logic (SABL) [TAV02], WDDL [TV04], Dual-Rail Random Switching Logic (DRSL) [CZ06], Masked Dual-Rail Pre-charge Logic (MDPL) [PM05], and improved Masked Dual-Rail Pre-charge Logic (iMDPL) [PKZM07]. Most of the dual-rail schemes have been developed to be implemented in ASICs. The fixed architecture as well as limited wire routings on FPGAs do not allow a straightforward porting of those schemes. During the last years, much effort was put in bringing the DRP concept to FPGAs. Bhasin et al. introduced in their work [BGF+10] an improved version of WDDL called DPL-noEE. Since it has been shown that WDDL suffers from the early propagation effect [SS06], DPL-noEE connects the signals of the true and false networks to the same gate and evaluates the gate output with the arrival of the last incoming signal. According to the authors’ report, compared to WDDL the leakage is approximately halved by means of DPL-noEE. It has been pointed out in [MI14] that DPL-noEE just solves the early propagation effect in the evaluation phase of the circuit but not in the precharge phase. The authors introduced a logic style called AWDDL
Chapter 3. Evaluating the Duplication Concept

that solved this problem. An AWDDL gate switches to precharge state when the last input signal turns to precharge. Additionally, a customized router was developed that tries to find routes with minimal delay differences for the true and false networks by moving the routing process into a Satisfiability (SAT) solvable problem. It is also noted by the authors that the router could minimize the leakage of the circuit but did not completely remove it due to the nonexistence of perfectly-identical dual-rail routes.

To address routing imbalances, Yu and Schaumont had formerly proposed to duplicate a fully-placed-and-routed WDDL circuit [YS07] (known as DWDDL). As a result, two equivalently routed WDDL circuits with swapped true and false networks (dualized) were placed on the same device. With investment of doubled resources the leakage of a WDDL circuit was drastically reduced.

Contribution

Power-equalization schemes place a circuit \( C \) and the dual of the circuit \( \overline{C} \) on the same device. Ideally, the total power consumption of both circuits cancel out the data dependency in the power traces. This idea implies that the logic gates of \( C \) and \( \overline{C} \) have to be synchronized and switched at the same point in time. This synchronization can be achieved by a global or asynchronous control signal connected to the gates like in [NBD+10, HdlTR11, LMT+09] or with the arrival of the input signals at the gate like in [TV04, BGF+10, MI14]. To use the input signals as a trigger for the evaluation, the signals of the dualized circuit \( \overline{C} \) must show exactly the same delays as their pendant signals in \( C \). Beside the synchronization aspect, the signal delays are strongly correlated with the capacity of the used wires so that even in a control signal synchronized circuit the signal delays of both circuits shall be equivalent to minimize the data-dependent side-channel leakage. The task of placing two circuits with the same signal delays is hard to achieve in FPGAs due to the static routing structure. Some logic styles, e.g., the most popular one is WDDL, require the interconnection of both \( C \) and \( \overline{C} \) circuits. In such cases the task to place and route the logic gates in such a way that the routing delays of coupled signals are equivalent is challenging. In [MI14] this task was addressed with a custom router based on a SAT solver.

As stated, in DWDDL [YS07] the fully-routed-and-interconnected original circuit \( (C, \overline{C}) \) is additionally placed with inverted logic on the same device \( (\overline{C'}, C') \). The cloning process was performed in a way that the routing information is transferred to the cloned circuit. According to the report of the Xilinx design tools, the signal delays of the original circuits \( (C, \overline{C}) \) and the cloned circuits \( (\overline{C'}, C') \) are equivalent. So this method turned out to be the best way to implement two circuits of equivalent routing delays without any routing restrictions or any custom routers. The drawback of the technique is clearly the high resource overhead.

In this chapter we thoroughly investigate the duplication scheme of [YS07]. Since the early propagation issue of WDDL makes it still vulnerable to the state-of-the-art attacks, we consider its successors DPL-noEE and AWDDL as well to examine the benefit of the duplication. We show that even in case of AWDDL, where early propagation at both phases is avoided, applying the duplication does not prevent data-dependent time of evaluation. By means of a Spartan-6 evaluation platform (SAKURA-G [sak]) we provide practical evidences to our findings that a DWDDL circuit and the equivalent ones realized by DPL-noEE as well as AWDDL still have leakage.
3.2 Performing Circuit Duplication

3.2.1 Distinct Circuit Duplication

As our target platform was a Spartan-6 FPGA, we developed a tool to realize a duplication procedure for that device family. The tool clones the structure of a given (original) circuit (in Xilinx Spartan-6 netlist format) and changes the LUTs content of the second (duplicated) circuit. It is indeed based on the same concept as the one introduced in [YS07]. Below a detailed description of the developed tool is given.

The tool uses the XDL netlist format noted in Section 2.1.3 and was written with the help of RapidSmith [LPL+12] which is a Java-based Application Programming Interface (API) to read, write, and edit XDL files. To recap, a circuit is defined in XDL by instances and nets. An instance holds the component type and its configuration in defined attributes. Additionally, the placement information of the instance is stored in the instance header by X and Y coordinates. Nets hold the routing information of signals that logically interconnect instances. To define the route of a net, the PIPs of passed switch matrices are stored in the body of the net. The PIPs are also equipped with X and Y coordinates to provide a unique identification.

Our tool duplicates a given (original) circuit or subcircuit by placing a copy of the circuit at a vertically shifted location of the FPGA. This is done by adding the instances and nets of the circuit with modified Y coordinates to the XDL file. Additionally, the attributes of the instances are modified in a way that the duplicated circuit forms the inverted network of the DRP scheme.

The copy process of the nets ensures that the routing structure of both circuits are equivalent. As an exception, the circuit I/O signals have to be handled differently. These nets cannot and are not needed to be duplicated. It is assumed that the I/O signals (related to, e.g., plaintext and ciphertext) do not leak any information. To route the I/O signals of the duplicated circuit, the standard Xilinx ISE routing tool cannot be used. In some cases it changes the already-routed nets, which may destroy the symmetry between the original and duplicated circuits. To route the I/O nets, the FPGA Editor, which is also part of the Xilinx ISE Design Suite, offers the possibility to only route specified signals. It is an adequate scenario for the low number of I/O signals.

To guarantee that the physical area, at which the duplicated circuit should be placed, is unused by the remaining design, the Xilinx ISE synthesizer can be parametrized with the prohibit constraint to avoid placing any instance in that specified area. Another constraint we used is area_group to keep the original circuit in a specified area as well. We also made sure that no routing resources of the prohibited area was used by other logics.

3.2.2 Interleaved Circuit Duplication

The previously described (in Section 3.2.1) duplication process generates two circuits which are placed in two distinct regions of the FPGA. Since the combined power consumption of both circuits performs the equalization, each circuit by its own is still vulnerable. In case of a power analysis attack an adversary is just able to measure the conjunct power consumption. An adversary performing an EM analysis is maybe able to measure the EM radiation of one circuit. Hence, a simple countermeasure to overcome this problem was used in [VK11, HOdlTR12] by interleaving but not interconnecting both circuits.
Chapter 3. Evaluating the Duplication Concept

Following this concept, it may happen that the interleaved duplication process face routing conflicts. In general, two strategies exist to overcome the routing problem. Either the routing conflicts are resolved after the duplication process or the routing conflicts are avoided by restricting the available routing resources of the original circuit. The resource restriction strategy uses the full power of the synthesizer which adopts the circuit to the available resources in context of minimal routing delays. The conflict resolving after duplication can result in solutions where conflicted nets are rerouted in an inefficient way which drastically reduces the clock frequency of the design.

The Xilinx ISE Design Suite tool flow does not allow explicit restrictions on routing resources. Hence, as a work-around, blocker macros are defined which consume the routing resources that are not allowed to be used by the original circuit. This forces the synthesizer to place and route the original circuit into a predefined shape. Before the duplication process is triggered, the blocker macros are removed from the netlist to free the resources for the circuit copy. Note that circuits with high routing complexity may not be able to be routed due to the drastically reduced resources. These blocker macros can be easily generated with the help of Go Ahead [BKT12], a free to use framework for partial reconfiguration.

### 3.3 Data-Dependent Time of Evaluation

This section takes on with the idea proposed in [YS07] to duplicate a dual-rail circuit. Figure 3.1 shows an overview of an exemplary circuit and its duplicated counterpart. Due to the copied structure the signal routings and corresponding delays $t_1...t_4$ are transferred from the original to the duplicated circuit and just the logic gates are replaced. Attended by the gate replacement, the true and false networks are swapped in the duplicated circuit. Thus, the true network of the original circuit and the false network of the duplicated circuit (respectively the false network of the original circuit and the true network of the duplicated circuit) are equivalently routed that hence results in an overall design with balanced true and false networks to minimize the leakage caused by different wire capacitances.

![Figure 3.1: Example of a fully routed circuit and its duplication.](image)

As stated before (see Section 2.2.4 and 2.2.4), DPL-noEE and AWDDL avoid the early propagation issue in contrast to WDDL. Along the same lines, the term *data-dependent time of evaluation* is defined as the cases when the gate evaluates its output at different time instances.
3.3. Data-Dependent Time of Evaluation

depending on its input values [MOP07]. The duplication concept proposed in [YS07] DWDDL aims at mainly avoiding such a data-dependent time of evaluation caused by difference in routing of dual-rail signals. In the following we show that such a scheme cannot avoid data-dependent time of evaluation even if the underlying logic style prevents the early propagation.

In order to explain the concept we focus on the example given in Figure 3.1. We just consider the start of the evaluation phase of an AND gate in both original and duplicated circuit (\(O_t\) and \(O_f\) in Figure 3.1). Further, due to the early propagation issue of WDDL, we suppose that the gates in these circuits are realized by DPL-noEE or AWDDL (there is no difference in this example since both avoid early evaluation).

As marked in Figure 3.1, we denote the delay of the input signals of the considered AND gate as \(t_1\) to \(t_4\), which stays the same for the corresponding OR gate in the duplicated circuit. We should highlight that no customized router is employed to route the signals in the original circuit. Indeed, the idea of duplication [YS07] is to avoid such a necessity. Therefore, the signal delays \(t_1\) to \(t_4\) can have any arbitrary value, and there is no guarantee to to keep them the same or even approximately the same.

![Figure 3.1](attachment:image.png)

(a) Under the condition \(t_1 > t_2 > t_3 > t_4\).

![Figure 3.2](attachment:image2.png)

(b) Under the condition \(t_1 > t_3 > t_2 > t_4\).

Figure 3.2: Timing diagrams of the circuit of Figure 3.1.

We first suppose that \(t_1 > t_2 > t_3 > t_4\) and draw the timing diagram of the output signals (\(O_t\) and \(O_f\)) of both original and duplicated circuits for all four possible input values, e.g., 11, 10, 01, and 00. The corresponding timing diagram is shown by Figure 3.2(a), where the black waveform belongs to the AND gate of original circuit (Figure 3.1(a)) and the red waveform to its complementary OR gate of the duplicated circuit (Figure 3.1(b)). Under such a condition the AND gate evaluates when the last input signal arrives. In case of 11 and 10 the output is evaluated at \(t_1\) (when \(A_t\) arrives). For other input cases 01 and 00 \(t_2\) define the evaluation time (when \(A_f\) arrives). The duplicated OR gate shows a complementary behavior for the time of evaluation. That is, independent of the input value one gate always evaluates at \(t_1\) and the other one at \(t_2\). Hence, the overall power consumption is then ideally independent of the given input value. It should be noted that exchanging the values of \(t_1\) by \(t_2\) and/or \(t_3\) by \(t_4\) as well as \((t_1, t_2)\) by \((t_3, t_4)\) (i.e., providing another condition, e.g., \(t_3 > t_4 > t_2 > t_1\)) has no effect on the shown balanced behavior.
Chapter 3. Evaluating the Duplication Concept

Figure 3.2(b) gives the waveforms under a different condition \( t_1 > t_3 > t_2 > t_4 \). For the given condition the gates evaluate the outputs at \( t_1, t_2, \) or \( t_3 \). In case of 11 and 00, \( t_1 \) and \( t_2 \) define the evaluation time, while for 01 and 10 the evaluation time depends on \( t_1 \) and \( t_3 \). In other words, one gate either in the original circuit or in the duplicated one evaluates at \( t_1 \), but the other gate evaluates at \( t_2 \) or at \( t_3 \) depending on the input value. This clearly shows a data-dependent time of evaluation that should lead to a leakage exploitable by an attack. Again, we should note that exchanging \( t_1 \) by \( t_3 \) and/or \( t_2 \) by \( t_4 \) as well as \((t_1, t_3)\) by \((t_2, t_4)\) does not show any difference on the presented data-dependent time of evaluation.

Remarks:
In case of DWDDL the situation is much worse. That is, for every condition (except \( t_1 = t_3, t_2 = t_4 \)) there is a data-dependent time of evaluation due to the early propagation of WDDL. The same issue holds true at the start of the precharge phase for all three considered logic styles. As a result, since in the certain conditions the duplication can be beneficial, we expect the duplication scheme to reduce but not fully avoid the leakage.

3.4 Practical Evaluation

3.4.1 Case Study

We implemented three AES-128 encryption cores using WDDL, DPL-noEE, and AWDDL gates. Following the scenario explained in Section 3.3, each of these cores is duplicated to realize the DWDDL, Double Dual-Rail Precharge Logic without Early Evaluation (DDPL-noEE), and Double Asynchronous Wave Dynamic Differential Logic (DAWDDL) AES cores. Hence, in total we analyze six full AES encryption cores. In general, for a fair comparison the best would be to keep the placement and routing of all these cores the same. However, WDDL has no defined native XOR gate, so a DPL-noEE circuit cannot be converted to a corresponding WDDL one. Also, AWDDL requires feedback loops as well as two LUTs per gate, hence converting an AWDDL circuit to a corresponding DPL-noEE one would be not fair with respect to resource utilization. Therefore, an identical placement and routing for these three logic styles is not possible.

We implemented a round-based AES core with a composite field Sbox of [Can05] (see Figure 3.3). Due to the known issue of register cells in dual-rail logics [MEPP09], we followed the master-slave fashion for the registers, i.e., two register stages in each rail. Therefore, every cipher round operates in two clock cycles: one for the precharge and the other one for the evaluation. Indeed, in an interleaved fashion one of the round register stages holds the AES state and the other one the precharge value.

As we do not target any template attacks and ignore the leakage solely associated to the process on the key, the key expansion unit is implemented normally without using any secure logic style. As shown in Figure 3.3, the plaintext and round key (the output of the key expansion) are converted to a dual-rail precharge form and stored in dual-rail master-slave registers. For the sake of simplicity, we have not shown the key expansion and the control unit, which we kept equal regardless of the used logic style. The area which is marked (by a red dashed line) in Figure 3.3 is the only part of the circuit that is implemented by WDDL, DPL-noEE, or AWDDL. Further, only this area is duplicated to realize the DWDDL, DDPL-noEE, and
3.4. Practical Evaluation

![AES Core Diagram]

Figure 3.3: Architecture of the full AES encryption core with key expansion and control logic.

DAWDDL circuits. The duplication process is performed by the tool proposed in Section 3.2. The resource consumption of each core can be seen in Table 3.1.

Table 3.1: Resource overview of the implemented AES cores.

<table>
<thead>
<tr>
<th>Logic Style</th>
<th>Utilized Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Slice</td>
</tr>
<tr>
<td>None</td>
<td>364</td>
</tr>
<tr>
<td>Control Logic</td>
<td>8</td>
</tr>
<tr>
<td>Key Expansion</td>
<td>97</td>
</tr>
<tr>
<td>WDDL</td>
<td>3,214</td>
</tr>
<tr>
<td>DWDDL</td>
<td>6,428</td>
</tr>
<tr>
<td>DPL-noEE</td>
<td>3,712</td>
</tr>
<tr>
<td>DDPL-noEE</td>
<td>7,424</td>
</tr>
<tr>
<td>AWDDL</td>
<td>3,724</td>
</tr>
<tr>
<td>DAWDDL</td>
<td>7,448</td>
</tr>
<tr>
<td>Control Logic</td>
<td>15</td>
</tr>
<tr>
<td>Key Expansion</td>
<td>83</td>
</tr>
</tbody>
</table>

3.4.2 Side-Channel Analysis

For the practical evaluations we used SAKURA-G [sak] as a side-channel evaluation board which is equipped with a Xilinx Spartan-6 FPGA. The power traces have been collected by monitoring the voltage drop over a 1Ω resistor at Vdd path. The target FPGA operated at a frequency of 3 MHz; power traces have been obtained by means of a digital oscilloscope at a sampling rate of 1 GS/s. We also made use of the amplifier embedded on the SAKURA-G board and limited the acquisition bandwidth to 20 MHz to achieve high quality signals. Note that the practical evaluation in this section was done by Amir Moradi.
As evaluation metric we used the leakage assessment methodology (t-test) presented in Section 2.2.3. [GJJR11] In a non-specific t-test (known also as fix vs. random test) a fix input (here plaintext) is selected, and during the measurements the fix or a random input is given to the target device\(^1\). The traces are categorized into two groups based on the associated fix or random input. Then, the Student’s (two-tailed) t-test is computed based on the mean and variance of each group of the traces (at each sample point independently). The outcome gives a level of confidence to reject a hypothesis as the traces of these two groups are drawn from the same population. If so (i.e., \(|t| > 4.5\)), it can be confidently concluded that a first-order leakage can be exploited from the device under evaluation.

Such a fix vs. random test is useful particularly to evaluate masked implementations. For instance, the same scheme has been used to examine the leakage of a higher-order attack resistant implementation of KATAN block cipher in [BGN+14b]. However, in case of our designs where no masking is involved we cannot easily apply such a test. That is because the plaintext, which is not masked, is sent during the communication (between the FPGAs of the SAKURA-G) and processed by the target FPGA. Therefore, regardless of the protection that the AES core provides, the leakage associated to the plaintext is observable at every sample point of the traces\(^2\).

As a solution, a semi fix vs. random test can be performed. In such a scenario, a set of plaintexts are selected, all of which lead to the partially same intermediate value. We have

\(^1\)Note that such a selection should also be randomized.
\(^2\)One reason is also related to the static leakage [Mor14].
precomputed 1024 plaintexts in such a way that the first 64 bits of the cipher state at the start of the round 5 of the AES encryption are 0 (a similar scheme has been introduced in [GJJR11]). During the measurements a random plaintext or one of the precomputed plaintexts (again, randomly) is taken. The rest of the evaluation stays the same as in a fix vs. random test.

A sample trace of the WDDL design is shown at the top of Figure 3.4. The first high peaks are related to the conversion and synchronization of the plaintext and the roundkey to the dual-rail precharge form (see Figure 3.3). The traces of the other designs – particularly the duplicated ones – look the same but with higher peak-to-peak value. For each of the six designs we collected 1 000 000 traces following the scenario explained above. It means that approximately 500 000 traces are with random plaintext, and the rest is with a randomly-chosen plaintext amongst the 1024 precomputed ones. The results of the tests on all six designs are shown by Figure 3.4.

It can be seen that none of the designs is able to avoid the leakage, and the tests strongly confirm the existence of a leakage. However, as we expected – stated in Section 3.3 – the duplicated designs can reduce the leakage, but due to the flaw (i.e., data-dependent time of evaluation for certain conditions) it cannot be prevented. It is noteworthy that the evaluation scheme which we performed is a "leakage assessment methodology" on one of the middle rounds of the cipher. Although we have not performed any key-recovery attack and have not provided any information about the simplicity/hardness of such an attack, the result of the presented tests indicate the failure of the underlying design methodologies to prevent the leakages.

3.5 Conclusion

Regardless of its significant area overhead, duplicating a dual-rail precharge circuit (DWDDL) was considered as the only scheme that can be used for power-equalization on FPGAs. In this chapter we have shown that this scheme is not flawless. By an extensive evaluation we found situations where the time of evaluation (of the gates in a double dual-rail precharge circuit) still depends on the input values. Such a data-dependent time of evaluation is caused by the difference between the signal delays of the gate inputs that cannot be avoided. Our theory is supported by practical analysis that we conducted on a Xilinx Spartan-6 FPGA. Although DPL-noEE and AWDDL avoid the well-known early propagation issue of WDDL, we have shown that still none of them can be considered as a potential solution to prevent the side-channel leakage.
Chapter 4

Hiding First-Order Leakage: Glitch-Free Duplication

This chapter introduces the first DRP logic style, especially designed for FPGAs, which addresses the three major flaws of power-equalization schemes noted in Section 2.2.2. The proposed DRP scheme, called GliFreD is analyzed and evaluated on its remaining leakage and the hardness to exploit this leakage. Further, a technique is introduced to automatically transfer almost arbitrary circuits into their GliFreD-based pendant.

Contents of this Chapter

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>51</td>
</tr>
<tr>
<td>4.2</td>
<td>Concept</td>
<td>52</td>
</tr>
<tr>
<td>4.3</td>
<td>Automated Circuit Transformation</td>
<td>56</td>
</tr>
<tr>
<td>4.4</td>
<td>Resource Overhead</td>
<td>58</td>
</tr>
<tr>
<td>4.5</td>
<td>Practical Evaluation</td>
<td>59</td>
</tr>
<tr>
<td>4.6</td>
<td>Conclusion</td>
<td>65</td>
</tr>
</tbody>
</table>

4.1 Introduction

Due to predefined structures and restrictions in routing, the techniques of popular DRP schemes as DRSL [CZ06], SABL [TAV02], WDDL [TV04], MDPL [PM05], or iMDPL [PKZM07] cannot be easily applied to FPGAs. An FPGA contains similar blocks formed by a couple of slices with (nearly) equal inter- and intraconnections. Hence, re-instantiating a part of a circuit at another location on the FPGA and converting it to its dual function seems to be a viable option. Therefore, most of the efforts to equalize the power consumption on FPGAs have been put in the direction of duplication. Previous works investigated this concept [BGF+10, HdlTR11, HOdITR12, KV10, LMT+09, MI14, NBD+10, SNG+09, YS07], but all proposed schemes still show some vulnerabilities against certain power analysis attacks.

It is well known that DPA-resistant logic styles do not solely suffice to completely prevent SCA attacks although they drastically reduce the signal-to-noise ratio. We will show in Chapter 5 that a combination of a proper first-order secure masking scheme, e.g., threshold implementations [NRS11] and GliFreD, has the potential to build highly-secure circuits. Such designs are provably secure against first-order attacks (due to the underlying proper masking scheme) and
in practice secure against higher-order attacks (due to the SNR lowered by the hiding scheme, i.e., GliFreD).

In the recent years several works have been published that tried to define logic styles which properly follow the DRP concept on FPGAs. The most recent and promising DRP logic styles are summarized in Chapter 2.2.4.

GliFreD takes two different schemes into account: first, the work of [HdlTR11], where each LUT is enabled by at least one global signal, and second, the scheme presented in [MM12] which considers a register at the output of each LUT. Recall that the work by [HdlTR11] follows the duplication scenario with the issue that glitches arise for large circuits. The concept of [MM12] avoids propagation of glitches but does not include techniques for duplication. Note that the description of GliFreD as well as the proof of concept implementation are based on a Xilinx Spartan-6 FPGA. This does not necessary conclude that the concept is restricted to that device family, and we assume that the general approach can be transferred to other device families and other FPGA vendors as well. The concept, needs to be slightly adopted if 6-to-1 LUTs and/or Phase Locked Loops (PLLs) are not available.

**Contribution**

This chapter provides a scheme that rules out weaknesses of previous schemes to provide an SCA-resistant implementation of cryptographic circuits on FPGAs. Our logic style, denoted as GliFreD, avoids (1) glitches in combinatorial circuits, (2) forms a pipeline architecture, and (3) efficiently instantiates the duplication concept. Side-channel analysis of the converted circuits implemented on the SAKURA-G platform [sak] indicates the success of GliFreD to significantly mitigate the success of DPA attacks. We further elaborate on the limitations of the duplication concept and provide reasons for the leakages that cannot be completely avoided. Apparently, the efficiency of duplication is crucially affected by process variations that cannot be completely avoided without device-specific back annotation.

**4.2 Concept**

In this section we explain our proposed technique denoted as Glitch-Free Duplication (GliFreD).

GliFreD uses primarily two components: Look-Up Tables (LUTs) and Flip Flops (FFs). Right after each LUT at least one FF has to be placed so that a transition at a LUT output does not directly propagate into other LUTs. An exemplary instantiation is depicted in Figure 4.1(a). Note that Figure 4.1(b) shows the corresponding waveforms and transitions between the precharge and evaluation of the given example.

GliFreD makes use of three global control signals: active1, active2 and CLK. One of the active signals as well as CLK are connected to each LUT to control the precharge and evaluation phases. active1 and active2 are phase-shifted to each other by 180 degree and run at half of the CLK frequency. active1 and active2 are also phase-shifted from CLK in order to arrive at the LUT slightly before CLK (see Figure 4.1(b)). The minimal phase-shift between CLK and active is technology dependent. It is based on the LUT internal propagation delay, the delay from the LUT output to the FF as well as the minimal hold time required by the FF. Unfortunately, these values are not publicly available and have to be obtained experimentally.
The depth of the LUT\(^1\) defines which \texttt{active} signal has to be connected to the LUT. More precisely, LUTs with an odd depth are connected to \texttt{active_1} and LUTs with an even depth are connected to \texttt{active_2}. As given in Section 2.1.1, we assume that each LUT consists of a multiplexer tree, and therefore the control signals should be connected to the first and second multiplexer stages (inputs \texttt{I}_0 \texttt{I}_1) to avoid LUT internal glitches. The control signals are fixed with the \texttt{LOCK_PINS} constraint to keep them connected at \texttt{I}_0 and \texttt{I}_1. \texttt{CLK} is the clock signal of the circuit and is also connected to each FF. LUTs are configured in a way that data signals connected to \texttt{I}_2-I_X are only processed if \texttt{active=HI} and \texttt{CLK=LO}. Otherwise, the LUT output is set to \texttt{zero}, i.e., \(O = \texttt{active} \cdot \texttt{CLK} \cdot f(I_2, \ldots, I_X)\).

\(^1\)The depth of a LUT input signal is defined as the number of FF that it has to pass to reach the LUT.
We should highlight that our construction (Figure 4.1(a)) avoids any glitches at the LUTs' output. The inputs of each LUT are provided by an FF that becomes stable right after the positive edge of \( \text{CLK} \). Since the connected \texttt{active} signal enables the output of the LUT only after a negative edge of \( \text{CLK} \) (when the LUT inputs are stable), the LUT output toggles at most once at start of the LUT evaluation phase (see Figure 4.1(b)). The same holds at the start of the LUT precharge phase as the LUT becomes disabled at the positive edge of \( \text{CLK} \) shortly before the connected FFs go to the precharge phase.

Note that in this step all input and output signals are computed in a single-rail fashion. After duplication (see Section 3.2), the LUTs in the dual circuit are configured to work complementary to the inverted incoming data signals, i.e., \( O = \text{active} \cdot \text{CLK} \cdot \overline{f(I_2, \ldots, I_X)} \). Clearly, \( \text{CLK} \) and \texttt{active} – as control signals – are not dualized, and both original and dual circuits receive the same control signals.

\( \text{CLK} \) is connected to FFs and routed by tools via a special routing network called clock tree which provides low-capacitance and low-skew interconnections [Xil15]. To connect \( \text{CLK} \) to the LUT input \( I_0 \), the clock tree must be expanded. This is done by connecting the clock tree to the closest switch box adjacent to the target LUT to route \( \text{CLK} \) to the LUT input. We assume that the delay difference between \( \text{CLK} \) signal connected to the LUT and FF is small and negligible. \( \text{CLK} \) triggers the FFs before the falling edge of \texttt{active}. Without connecting \( \text{CLK} \) to the LUTs, glitches would appear right before the start of the LUT precharge phase. Apparently, such a connection shortens the LUT evaluation phase and also avoids glitches at the LUT output.

Both \texttt{active}_1 and \texttt{active}_2 signals are generated with one of the available PLL modules and hence moved into the clock tree as well. Due to a slight phase-shift between \texttt{active} and \( \text{CLK} \) the LUTs' evaluation phase is short. For a glitch-free data processing, the LUT inputs have to be stable. In general, the path delays from FF to LUT are much larger than the delays from LUT to the FF, since each FF is located right after the LUT in the same slice. In other words, the short evaluation phase is reasoned in the FPGA structure and avoids glitches at higher clock frequencies.

Generally, DRP schemes follow the predicate that the number of toggling gates in the entire circuit (i.e., original and dual) is always constant at each transition between precharge and evaluation phases. For LUTs, this is guaranteed by adding a dual circuit. For memory elements it is noted in [MEPP09] that they also have to alternate between the precharge and evaluation state. This is guaranteed by alternately connecting \texttt{active}_1 and \texttt{active}_2 to the LUTs. This ensures alternating between the precharge and evaluation for the FFs, and the number of transitions is kept constant as well after dualization.

As mentioned in Section 2.2.2, an important issue of the DRP schemes is imbalanced routing. In order to equalize routing delays of complementary signals. In GliFreD, the single-rail circuit is placed and routed in a restricted area, then its dual is added to the design by duplication, i.e., by copying the dual of all the components and corresponding routings. Such a restriction is to make sure there is room available for the dual circuit. Due to the vertically-identical architecture of the FPGAs, the strategy which we followed for duplication was to put the dual circuit below the original one. In other words, two identical and vertically-adjacent parts of the FPGA were chosen. The placement of the original circuit is limited to the upper part, and after placement and routing its dual was copied to the lower part. Note that only the LUT configurations need to be dualized as explained above. To increase the usability of GliFreD we
developed some tools (see Section 3.2 and Section 4.3) which support the circuit transformation process.

We now summarize the properties and limitations of GliFreD:

■ Some dedicated modules available on FPGAs, e.g., multiplexer and carry chains, cannot be used due to their fixed behavior. To generate a dual circuit, it is necessary to invert the behavior of all elements. Hence, any such module must be converted to its LUT-equivalent.

■ The concept of GliFreD makes use of a synchronous driven LUT/FF combination. A similar structure is given by the BRAMs, available in most FPGA device families. Hence, it seems viable to extend GliFreD to BRAM, e.g., by adapting the concept proposed in [BDGH15].

■ Two control signals have to be connected to each LUT that results in routing and resource overhead due to a reduction from 6-to-1 LUTs to 4-to-1 LUTs.

■ The FF utilization of a design is increased due to the required placement of at least one FF between two connected LUTs. It is noteworthy that in each slice of the Xilinx FPGAs there exists a FF right after each LUT, and in non-GliFreD designs with large combinatorial circuits such FFs are left unused. In recent Xilinx 7 series, even two FFs per LUT exist in each slice [Xil14] that can be used to add further FFs. Hence, the FF-utilization overhead of GliFreD is limited to the extra FFs required to form the pipeline architecture (described in Section 5.3.3).

■ The number of clock cycles required for the operation of the circuit depends on the LUT depth of the circuit.

■ A high clock frequency can be easily achieved due to a minimal number of LUTs between FFs (i.e., depth $\delta \leq 1$).

■ With additional FFs a circuit can be easily transformed into a fully-pipelined circuit.

■ Not only fully-unrolled designs can be implemented by GliFreD. Indeed, regardless of existence of a loop in the design it can be transformed into a GliFreD circuit (described in Section 4.3).

■ DRP schemes in general form two circuits which either work independent or in combination with each other. The two circuits work complementary to each other and hence the input-, intermediate-, and output values are inverted to each other. This data redundancy can be used to detect at least single bit faults [BDF+09].

■ Beside the natural resistance of DRP schemes against faults, GliFreD forms two distinct circuits which can run with different input values. Loading the circuits with different values destroys the DRP concept and hence the side-channel resistance but doubles the throughput. Therefore, GliFreD allows a dynamic switching between a secure and high throughput mode without reconfiguration of the FPGA.
### 4.3 Automated Circuit Transformation

This section introduces a tool which is able to automatically translate almost arbitrary circuits into a Glitch-Free duplication-based design. Glitch-Free duplication enforces each LUT to be connected to two control signals. Therefore, only 4 of 6 available input pins of each LUT (in modern Xilinx FPGAs) can be used as data pins. We therefore should restrict synthesis to map a circuit into 4-to-1 only LUTs\(^2\). As a simple solution, the circuit can be synthesized for a Spartan-3 that natively uses only 4-to-1 LUTs. Next, a HDL representation is re-generated from the resulting netlist using the ISE Design Suite. By means of a simple script, all 4-to-1 LUT macros (in the recreated HDL file) are transformed into 6-to-1 LUT macros and adopted to consider two (clk and active) control signals at \(I_0\) and \(I_1\). Finally, the control signals are connected to each LUT, and FFs are added to build a pipelined circuit.

The complexity to form the pipelined architecture from a circuit depends on the presence of loops as well as the designer’s need of a fully-pipelined architecture to hold multiple states in the circuit. An example of a loop-free circuit is given in Figure 4.2(a). In this scenario FFs can simply be placed between the LUTs, which increases the latency of the circuit. In some cases further FFs need to be placed, so that the depth of entire inputs of each LUT is even or odd. Figure 4.2(b) shows its Glitch-Free duplication representation without full pipelining.

In case of a loop-free circuit and a desire to form a fully-pipelined architecture, the LUT depth is important. First, FFs are naively placed after each LUT. Second, extra FFs are required to hold multiple states in the circuit. The placement of these FFs is design dependent, and is based on the LUT depth. The depth of a LUT input signal is defined by the number of FFs that it passes to reach the LUT. For a fully-pipelined architecture, all input signals of a LUT must have the same depth; hence, the signals with lower depth have to be further delayed by means of FFs. Figure 4.2(c) shows the fully-pipelined Glitch-Free representation of Figure 4.2(a).

Adding FFs to a circuit with loops may change its functionality. Hence, in order to form a pipelined representation of a circuit with loops, placing additional FFs without changing the circuit behavior is required. It should be noted that this cannot be done without increasing the circuit’s latency. The transformation of an almost arbitrary circuit with loops into its pipelined representation can be performed as follows.

In general, the Glitch-Free structure fits to the definition of a systolic circuit, which is a unit-delay circuit (i.e., propagation delay of all functional element are the same) in which at least one register is placed along every interconnection between two functional elements. In [LS91] it was shown, how to form a systolic circuit from an arbitrary synchronous circuit by combining a technique called c-slow and retiming. To apply the transformation, the circuit has to be modeled as a finite, edge-weighted, directed multigraph \(G\) as \((V, E, w)\). The vertices \(V\) of the graph \(G\) model the LUT of the circuit, while the propagation delay of each vertex \(v \in V\) is neglected in the model. The directed edges \(E\) of the model interconnect the vertices and hence the output of one LUT to the input of another LUT. The edge-weight \(w(e)\) of each edge \(e \in E\) represents the number of registers placed on the given edge \(e\). A virtual vertex \(v_k\) is created to which all primary inputs and outputs of the circuit are connected; it in fact simulates the

\[^2\]This synthesis step has a high impact on the latency and resources consumption of the resulting Glitch-Free duplication circuit. Based on the chosen synthesis strategy, the circuit slightly changes in size while the circuit depth can strongly vary. Predicting a synthesis strategy in advance, that leads to a resource-efficient Glitch-Free duplication design, is a non-trivial task since it is unclear at this point how many FFs are required to form the pipeline. To go for high throughput, synthesis strategies which result in a small circuit depth are amongst obvious choices.
4.3. Automated Circuit Transformation

![Circuit Diagrams](a) Loop-free example circuit. (b) Corresponding GliFreD circuit without full pipelining. (c) Corresponding GliFreD circuit.

Figure 4.2: Transformation process of a symmetric circuit into an systolic circuit without loop.

interfaces of the circuit. With $v_h$, the modeled circuit forms a finite graph. For a well-designed circuit, each edge-weight $w(e)$ is always a non-negative integer. Additionally, any directed loop in $G$ has at least one edge with positive weight $w(e) > 0$. Otherwise, the loop may oscillate, or race conditions may occur. In Figure 4.3(a) and Figure 4.3(b) an exemplary circuit and its corresponding graph is given.

It has been shown in [LS91] that if $G - 1$ has no negative cycle, there exist a valid retiming $r$ that forms a systolic circuit from the synchronous circuit $G$. While $G - 1$ forms the graph $\langle V, E, w' \rangle$ with $w'(e) = w(e) - 1$ for each $e \in E$.

The retiming specifies a transformation of the original circuit in which registers are added and removed in order to minimize the propagation delay without further affecting the structure of the circuit. Hence, a new graph $G_r : \langle V, E, w_r \rangle$ is formed from $G$. Each edge-weight in $G_r$ is defined as $w_r(e) = w(e) + r(v) - r(u)$ while $e$ is the edge from vertex $u$ to vertex $v$.

In general, most of the synchronous circuits do not hold the condition of non-negative cycles in $G - 1$. Hence, c-slow technique multiplies each edge-weight with a positive integer $c$ so that the graph $cG : \langle V, E, w^* \rangle$ is constructed, where $w^*(e \in E) = c \cdot w(e)$. The data flow in $cG$ is slowed down by a factor of $c$, but it performs the same computations as $G$. $cG$ is called the c-slow representation of $G$. With c-slow, additional FFs are placed in the circuit which may allow the retiming process to form the systolic structure. In other words, $c$ has to be increased until $cG - 1$ does not contain any negative cycle. Further, the GliFreD concept (similar to all DRP logic styles) requires to process the precharge and evaluation phases alternately. To guarantee this in a loop, each loop has to hold an even number of delay stages. Hence, $c$ multiplies the already available delay stages; limiting $c$ to an even value guarantees the availability of an even number of delay stages. Figure 4.3(c) shows the c-slow representation of the example graph with $c = 4$. The detection of the cycles with negative weight and the calculation of the retiming $r$ can be done by the Bellman-Ford algorithm [BG02] which calculates the shortest path from each vertex to the virtual vertex $v_h$, and aborts if a negative cycle exists. As a side note, the combined process of c-slow and retiming expects a circuit with at least one register stage. Otherwise, c-slow would not add additional delay elements, and the process would never terminate. For plain combinatorial circuits, at least one register stage at the output of the circuit has to be placed to run the process properly. Further, to follow the GliFreD concept the depth of all inputs of each LUT must be even or odd. Because the retiming step moves delay stages along the circuit, this condition limits the transformation process to circuits with a single delay stage per loop. With the given technique, almost arbitrary circuits that include
(a) Example circuit with loop. (b) Corresponding graph $G$ from circuit. (c) $c$-slow representation of $G$ with non-negative cycles in $cG - 1$.

Figure 4.3: Transformation process of a symmetric circuit into an systolic circuit with loop.

cycles can be transformed into the GliFreD structure. Figure 4.3(d) and Figure 4.3(e) show the $c$-slowed and retimed graph and respectively the transformed circuit.

4.4 Resource Overhead

In the following the theoretical resource overhead is estimated that is required to transform an arbitrary unrolled design based on LUTs and FFs into a GliFreD circuit. The overhead of the LUT utilization is design- and synthesizer-dependent and can be determined by the following formula:

$$u_{\text{LUT}}/i_{\text{LUT}} = 2\left(i_{\text{LUT}}(1 + p_{\text{LUT}}) + m_{\text{LUT}}\right)/i_{\text{LUT}},$$

where $u_{\text{LUT}}$ represents the number of LUTs used in the GliFreD circuit, $i_{\text{LUT}}$ the number of 6-to-1 LUTs of the original design, $m_{\text{LUT}}$ the number of elements e.g., multiplexers to be replaced by LUTs, and $p_{\text{LUT}}$ refers to the resource utilization penalty that is inferred by using 4-to-1 LUTs instead of 6-to-1 LUTs. $p_{\text{LUT}}$ depends on the underlying circuit as well as the synthesizer, but it can be estimated to be at most $2^6/2^4 = 4$ if all 6 inputs of all 6-to-1 LUTs are used in the original design.

The overhead of the FF utilization is also design-dependent and can be given by $u_{\text{FF}}$ as the number of FFs used in the GliFreD design and $i_{\text{FF}}$ as the number of FFs already available in the original design by

$$u_{\text{FF}}/i_{\text{FF}} = \left(u_{\text{LUT}} + e_{\text{FF}}\right)/i_{\text{FF}}.$$
When transforming an unrolled design into a GliFreD circuit without full pipelining, the number of additional FFs mostly depends on the number of used LUTs and is approximately two times the number of single-rail LUTs (so, $\epsilon_{FF} = 0$). For a fully-pipelined design, this number is higher as additional FFs must be added to form the pipeline in all circuit stages. In such a case, $\epsilon_{FF}$ refers to the number of extra FFs which are required to balance all the inputs of each LUT. This can be seen in Table 4.1 that compares the resource utilization and run time properties of the GliFreD AES Sbox and an unmodified Canright design [Can05] with buffered I/O. All values are post-place-and-route results, generated with Xilinx ISE 14.7.

<table>
<thead>
<tr>
<th>Logic Style</th>
<th>LUT</th>
<th>FF</th>
<th>Clock (#clock)</th>
<th>Freq. (MHz)</th>
<th>Throughput (MBit/s)</th>
<th>Latency (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>54</td>
<td>16</td>
<td>1</td>
<td>110</td>
<td>110</td>
<td>9.09</td>
</tr>
<tr>
<td>GliFreD</td>
<td>176</td>
<td>262</td>
<td>10</td>
<td>557</td>
<td>55.7</td>
<td>17.95</td>
</tr>
<tr>
<td>GliFreD (fully-pipelined)</td>
<td>276</td>
<td>588</td>
<td>10</td>
<td>596</td>
<td>298</td>
<td>16.78</td>
</tr>
</tbody>
</table>

Based on Table 4.1, the LUT overhead induced by GliFreD is 2.26 and 4.11 for the non-pipeline and fully-pipelined versions respectively. This is mainly caused by the dual circuit, the use of 4-to-1 instead of 6-to-1 LUTs and the use of 50 (GliFreD) respectively 150 (GliFreD fully-pipelined) pass-through LUTs to achieve minimal routing delays. With the pipelined structure the FF utilization drastically increases; here we state an overhead of 35.75 for a fully-pipelined design. The not-pipelined AES Sbox requires 262 FFs which results in an overhead factor of 15.38. The pipelined architecture of GliFreD results in circuits with a minimal LUT depth $\delta \leq 1$ and hence in designs with a high clock frequency. As given in Table 4.1 the fully-pipelined GliFreD construction can operate at a frequency of 596 MHz and hence almost triple the throughput of the Sbox. On the other hand, GliFreD increases the latency of the circuit.

Comparing GliFreD with former logic styles, it can be seen in Table 2.3 that all logic styles add an overhead factor to the case study which ranges between 1.21 and 9.21. GliFreD is in this context one of the resource-friendliest concepts with an overhead factor of 1.36 while the fully-pipelined version is still in the mid-range with an overhead of 4.29. Note that the previously mentioned pass-through LUTs are induced by the place & route process to achieve minimal routing delays. The number of induced pass-through LUTs strongly depends on the underlying design and used place & route strategy which makes it hard to estimate. For a fair comparison we neglected the pass-through LUT for each logic style and hence give estimated post-synthesis numbers in Table 2.3.

4.5 Practical Evaluation

To evaluate the side-channel resistance of our proposed scheme we transformed a round-based AES implementation into a GliFreD circuit while the Sboxes were implemented based on the Canright AES SBbox [Can05]. The resulting GliFreD design forms a fully-pipelined circuit that is able to hold 7 plaintexts. The architecture of the design is given in Figure 4.4. It needs $7 \times 2$ clock cycles per round. Hence, after $14 \times 10$ clock cycles 7 ciphertexts can be sequentially obtained from the circuit.
In order to better examine the impact of each component in our construction, we created five slightly different architectures of our constructed Glitch-Free circuits. All modifications have been applied to the final and fully-functional Glitch-Free design netlist, and kept the routing of the original design untouched.

**Profile 1:** The Glitch-Free design as described above while the phase-shift of active signals is fixed to 10%.

**Profile 2:** LUT configurations have been changed so that CLK signal has no impact on the LUT behavior.

**Profile 3:** LUT configurations have been modified so that active signal is don’t care.

**Profile 4:** The dual part of the Glitch-Free circuit has been removed.

**Profile 5:** The LUTs have been re-configured so that CLK and active signals have no impact on the LUT behavior; all FFs re-configured as constantly transparent latches, and the dual part of the circuit has been removed.

We consider Profile 1 as the reference and compare it with the others to determine the effect of each feature that is part of our proposed scheme. With Profile 2 and 3 we evaluate the necessity of routing CLK and active signals to each LUT. Profile 4 provides an insight on the efficiency of the dual-rail concept. Profile 5 – with the same placement and routing – emulates an unprotected design, i.e., a single-rail design without an FF between the LUTs, and the LUTs are not controlled either by CLK or active signals. It is noteworthy that in Profile 5 we did not modify the state machine in order to keep the same placement and routing for all profiles.
4.5. Practical Evaluation

Therefore, since Profile 5 does not form a pipeline, its output after 140 clock cycles does not match with that of other profiles. However, its first encryption round is compatible with other profiles.

4.5.1 Side-Channel Analysis

For practical SCA evaluations we used the SAKURA-G [sak] side-channel evaluation platform based on a Xilinx Spartan-6 FPGA. Power traces have been collected by monitoring the voltage drop over a 1 Ω resistor at the Vdd path. The target FPGA operates at a frequency of 3 MHz; power traces have been obtained by means of a digital LeCroy WavePro 715Zi oscilloscope at a sampling rate of 500 MS/s. Due to the small scale of the power consumption signal, we made use of the amplifier embedded on the SAKURA-G board to better acquire low-noise signals. Note that the results presented in this section are generated in cooperation with Amir Moradi.

For each of the aforementioned design profiles we collected $n$ traces while the plaintext bytes $p_i^{j \in \{1,...,16\}} \in \{0,1\}^8$ with $i \in 1 \ldots n$ were randomly chosen from a uniform distribution. We measured different number of traces $n$ for different profiles. $n = 10000000$ traces have been measured for Profile 1 and Profile 2, while we collected $n = 100000$ traces for the other profiles. Indeed, the leakage could be observed by low number of traces ($n = 100000$) for the last 3 profiles, hence we did not collect more traces. Figure 4.5 shows a sample trace of each profile that also gives a comparative impression about the amount of their power consumption.
The power traces cover full encryption of the given plaintexts while for all the measurements
the key is kept constant. Since our developed GliFreD AES encryption forms a 7-stage pipeline
design: 7 different plaintexts can be consecutively given to the encryption module. However,
pipeline designs add noise to the measurements if one particular pipeline stage of the design
is targeted [SÖP04]. Hence, for each power trace we consecutively gave the same randomly-
selected plaintext 7 times to the encryption module to fill all pipeline stages with the same
values, and hence reduce the switching noise and ease detecting the leakage.

To fairly compare the design profiles, we considered the following four evaluation metrics.

- Signal to Noise Ratio (SNR) [MOP07] to compare the amount of dependency of the power
traces to the processed values.
- Information-Theoretic (IT) metric [SMY09] that examines the amount of exploitable in-
formation using mutual information between processed data and the associated leakages.
The results of an IT analysis can be directly translated to the success rate of a univariate
template attack [SVO+10].
- Correlation Power Analysis (CPA) [BCO04] with common HW model (Sbox output) to
observe the success of common key-recover attacks.
- Moments-Correlating Profiled DPA (MCP-DPA) [MS14] that provides insights on the data
complexity (i.e., the number of required traces) of a successful attack without considering
any particular hypothetical model. It is indeed useful when CPA with common power
models does not succeed.

In our analyses each plaintext byte is considered separately, and by \( \frac{\text{var(signal)}}{\text{var(noise)}} \) at each sample point we estimate the SNR. Because we target first-order analysis, \( \text{var(signal)} \) is estimated by the variance of the mean traces obtained by categorizing the traces based on
the targeted plaintext byte.

Figure 4.6 shows 16 SNR traces (one trace for each plaintext byte) for each profile. It can
be clearly seen that Profile 1 has the least SNR. The necessity of each design principle of
GliFreD, i.e., having active signal (Profile 3), dual-rail fashion (Profile 4), and prevention of
glitches (Profile 5), can also be determined by comparing the corresponding figures. Notably,
the SNR is reduced by a factor of around 100 when all features of GliFreD are enabled (Profile
1). Further, the effect of having CLK signal (Profile 2) is minimal; therefore, we performed
other evaluations (given below) to examine its necessity.

For the IT analysis, the probability distribution of the leakages should be estimated. We
estimated the PDFs by Gaussian distributions where the means and variances at each sample
point were estimated by formulas given in Section 2.2.2. Similar to [MSQ07] we evaluated the
amount of available information based on the conditional entropy and by means of the integral
over the leakages. We, in fact, measured the amount of perceived information [DSV14] as (i)
we assume a Gaussian distribution for the leakages and, (ii) we follow a leakage model based
on the plaintext byte value. In this experiment, we targeted only the first plaintext byte. The
obtained perceived information diagrams for all the design profiles are shown in Figure 4.7.
Similar to the results of the SNR, the mutual information is reduced by a factor of more than
100, and the effect of CLK is minimal.
For CPA with HW of the Sbox output, we also focused on only the first key byte. The results of the attack on all profiles are depicted in Figure 4.8. Note that the correlation trace for the correct key is plotted in black. Since GliFreD (and any power-equalizing scheme) can only mitigate the leakage, a CPA attack – as expected – is possible on a GliFreD design (Profile 1) with correlation $4.2 \times 10^{-3}$ (turns into $1,600,000$ required traces [MOP07]). However, the correlation value is significantly reduced compared to Profile 4 ($0.1/0.0042 : \approx 24$) and Profile 5 ($0.04/0.0042 : \approx 9.5$). This can be directly mapped to the number of required traces, i.e., around 570 times and 90 times more traces compared to Profile 4 and Profile 5 respectively.

By CPA and HW of the Sbox output we cannot differentiate Profile 1 and Profile 2 (Figure 4.8). However, it cannot be concluded that controlling the LUTs by CLK signal has no advantage, because the success of CPA depends on the suitability of the considered hypothetical model. Therefore, we also considered MCP-DPA in profiling settings, which avoids the need of any hypothetical model. In the first-order profiling MCP-DPA, the mean traces $\mu_j$ based on particular plaintext byte $j$ are used as a power model to conduct a CPA based on the same plaintext byte $j$ using another set of traces. Following its underlying profiling settings, for each design profile, we used the first $n/2$ traces for the profiling (i.e., to estimate the mean traces) and the second $n/2$ traces to conduct the attack. The result of the attacks shown in Figure 4.9 are also based on the first plaintext byte $j = 1$. In this experiment we can now detect the difference between Profile 1 and Profile 2, i.e., the effect of controlling the LUTs by CLK signal.
4.5.2 Discussions about Remaining Leakage

GliFreD and many other similar schemes, e.g., [HdlTR11, YS07], are founded on the assumption that duplicated routes have exactly the same signal delay (and capacitance) on an FPGA. Tools like the Xilinx FPGA Editor confirm this assumption in theory. Since all values reported by FPGA Editor are worst-case simulation times, the actual signal propagation delay of routes can differ in practice due to process variations and other influences (e.g., temperature, supply voltage, or aging).

Variation in capacitances and transistor strength of components are well known by the PUF community where such variations are used to generate device-specific fingerprints (e.g., [GCvDD02, TSS+06, MTV08, KGM+08]). Especially the results of delay-based PUFs (which make use of the process variation of drivers and wires) support our assumption that the process variation is amongst the reasons for the imperfect signal balancing.

The complementary input signals and configuration of two coupled LUTs are another point. As noted in [MKD10], the signal propagation delay of a LUT is affected by its input signals. It was shown that the propagation delay of a signal connected to $I_0$ while $I_1I_2I_3I_4I_5$ are kept constant $HI$ or constant $LO$, differ at max by 9 ps on a Xilinx Virtex-5 FPGA. Hence, two LUTs with complementary configuration and inverted input signals (as utilized in GliFreD) show different propagation delays which also results in an imperfect signal balancing.
4.6 Conclusion

In this chapter, the DRP scheme called GliFreD is proposed to diminish leakages of critical information processed by cryptographic operations. Our scheme is based on reducing the side-channel information leakage of FPGA-based implementations by avoiding the glitches, preventing early propagations, and mitigating the imbalanced routings.

By several practical side-channel analysis evaluations, we have expressed the necessity of each of the design specifications of GliFreD. We further successfully showed that vertical copies of routing structures are sub-optimal to obtain virtually identical routes since minimal leakage is still present. We assume that the remaining leakage resides in different LUT internal propagation delays and manufacturing differences. Hence, an ideal and perfect signal balancing is hard (seems practically impossible) to achieve.

In short, GliFreD (and any other power-equalizing technique) can reduce – rather than ideally avoid – the exploitable side-channel leakages. Therefore, such schemes are usually combined with a proper masking scheme, which can provably protect against 1st-order attacks, and at the same time higher-order attacks become practically infeasible due to the employed power-equalizing technique. As an example, this is examined in Chapter 5, where a former variant of GliFreD has been merged with a provably 1st-order secure threshold implementation.

Figure 4.8: CPA curves targeting the first key byte.
Figure 4.9: Moment-Correlating DPA results targeting the first key byte.
Chapter 5
Hiding Higher-Order Leakages

State-of-the-art masking schemes provide SCA resistance up to a defined order \( d \), but still show leakages at an even higher statistical moment \( d+1 \). By increasing the resistance level of a masking scheme, the induced resource overhead also increases. It has been shown in [PRB09] that performing higher-order attacks on noisy traces becomes extremely hard. Since hiding schemes, i.e., DRP schemes, reduce the signal level in the leakage traces, an equivalent effect as of increasing the noise is reached. In conclusion, the combination of a proper first-order masking and power-equalization scheme provides a solid basement to form an alternative to higher-order masking. This chapter evaluates the countermeasure combination and provides results with respect to the achievable security level and investigated resources.

Contents of this Chapter

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Introduction</td>
<td>67</td>
</tr>
<tr>
<td>5.2</td>
<td>Background</td>
<td>69</td>
</tr>
<tr>
<td>5.3</td>
<td>Case Study</td>
<td>70</td>
</tr>
<tr>
<td>5.4</td>
<td>Practical Evaluation</td>
<td>77</td>
</tr>
<tr>
<td>5.5</td>
<td>Conclusion/Discussion</td>
<td>80</td>
</tr>
</tbody>
</table>

5.1 Introduction

The technique presented in Section 4 has been designed particularly for Xilinx FPGAs and aims at avoiding early propagation, preventing the glitches, and relaxing the necessity of a dual-rail routing tool. It seems that GliFreD can satisfy its goals toward equalizing the power consumption, but an ideally-equal situation still cannot be achieved, e.g., due to the process variation, violating the balance between the cloned routes.

On the other hand, probably the most investigated and the best understood protection against side-channel attacks is masking [CJRR99, DDF14, RP10]. The underlying principle of masking is to represent any sensitive variable in the implementation by \( d \) shares in such a way that the computations are performed only on these shares. Assuming that the leakage of the shares are independent of each other, a successful key-recovery attack needs to observe – at least – the \( d \)th-order statistical moment of the leakage distributions, where the corresponding complexity increases exponentially with \( d \).
However, the independence of leakages associated to the shares is an assumption which is usually violated in hardware applications. As an example, the masked AES Sbox designs [CB08, OMPR05], where the glitches are ignored, failed in practice to satisfy the desired security level, i.e., first-order resistance [MPO05, MME10]. Instead, based on Boolean masking and multiparty computation, Threshold Implementation (TI) [NRS09, NRS11] can ensure first-order resistance in the presence of glitches. Indeed, not only its underlying principles are sound and realistic, but also practical investigations confirmed its effectiveness [BGN +14a, MPL +11]. Trivially, higher-order attacks are feasible on TI designs [BGN +14a, Mor12], which motivated the work presented in [BGN +14b] where the concept of higher-order TI that extends its definitions to any order is demonstrated. Regardless of its significant overhead (e.g., requiring at least $d = 5$ for a second-order security) the note given in [Rep15] and later practically confirmed in [SMG15] made clear that the definitions of the higher-order TI stand valid only in univariate scenarios.

**Contribution**

Indeed, it is known to the community that hiding techniques (in particular power-equalizing approaches) are not solely capable to prevent key-recovery attacks. It is always suggested that such techniques should be combined with other countermeasures, but the benefit of such a combination has never truly been examined for a hardware platform. More precisely, exploiting higher-order leakages becomes extremely hard in practice when the leakage traces are sufficiently noisy [PRB09]. Along the same lines, power-equalization schemes are also expected to reduce the signal (versus the noise) and have the same effect. To the best of our knowledge, the only work which tried to proceed toward this goal is [MM12], where a flawed masking scheme [CB08] has been implemented in a glitch-free setting. No particular attention has been payed on equalizing the power and hence not on a concrete hiding technique.

The contribution of this chapter is to examine the benefit of combining two sound hardware-based countermeasures. More precisely, we aim at considering a provably (first-order) secure masking scheme (TI) and realize it under the principles of a proper power-equalizing technique (GliFreD). We pursue an investigation of our combined construction compared with:

- the same masking design (first-order TI) without employing any hiding technique, and
- the second-order TI of the same design excluding any power-equalization scheme.

Such comparisons with respect to the data complexity of leakage detection as well as time and area overheads of the designs allow us to have an overview on the tradeoff between the gains and overheads of different countermeasures as well as their combination.

Since the design overheads are application specific, we consider two design methodologies: first, a fully serialized architecture for lightweight applications with KATAN-32 cipher and, second, a parallelized architecture for high-speed applications with PRESENT cipher. Amongst the achievements of this chapter – including a second-order TI of PRESENT – we can refer to the designs we developed with a combination of GliFreD and the first-order TI (of both KATAN-32 and PRESENT) which showed to be secure by up to 1 billion power traces measured from a Spartan-6 FPGA platform.
Threshold Implementation (TI) is a masking scheme that bases on multi-party computation and secret sharing. It was introduced by Nikova et al. [NRS09, NRS11] and provides provable univariate first-order security in the presence of glitches. This scheme was extended in [BGN+14b, BGN+14a] to provide higher-order security. Let us denote an intermediate value of a cipher by \( x \) made of \( s \) single-bit signals \( \langle x_1, \ldots, x_s \rangle \). The underlying concept of TI is to use Boolean masking to represent \( x \) in a shared form \( \langle x_1, \ldots, x_n \rangle \), where \( x = \bigoplus x_i \) and each \( x_i \) similarly denotes a vector of \( s \) single-bit signals \( \langle x_{i1}, \ldots, x_{is} \rangle \). A linear function \( l(\cdot) \) can be trivially applied over the shares of \( x \) as \( l(x) = \bigoplus l(x_i) \). However, the realization of non-linear functions, e.g., an Sbox, over Boolean masked data is challenging. Following the concept of TI, if the algebraic degree of the underlying Sbox is denoted by \( t \) and the desired security order by \( d \), the minimum number of shares to realize the Sbox under the TI settings is \( n = td + 1 \). Further, such a TI Sbox provides the output \( y = S(x) \) in a shared form \( \langle y_1, \ldots, y_m \rangle \) with at least \( m = \binom{n}{t} \) shares. Note that the bit length of \( x \) and \( y \) (respectively of their shared forms) are not necessary the same since \( S(\cdot) \) might be not a bijection, e.g., in case of DES.

Each output share \( y_{j \in \{1, \ldots, m\}} \) is given by a component function \( f_j(\cdot) \) over a subset of the input shares. To achieve the \( d \)-th-order security, any \( d \) selection of the component functions \( f_{j \in \{1, \ldots, m\}}(\cdot) \) should be independent of at least one input share.

Since the security of masking schemes is based on the uniform distribution of the masks, the output of a TI Sbox must be also uniform as it is used as input in further parts of the implementation. To express the uniformity under the TI concept, suppose that for a certain input \( x \) all possible sharings \( \mathcal{X} = \{ (x_1, \ldots, x^n) | x = \bigoplus x' \} \) are given to a TI Sbox. The set made by the output shares, i.e., \( \{ (f^{1}(\cdot), \ldots, f^{m}(\cdot)) | (x_1, \ldots, x^n) \in \mathcal{X} \} \), should be drawn uniformly from the set \( \mathcal{Y} = \{ (y_1, \ldots, y^m) | y = \bigoplus y' \} \) as all possible sharings of \( y = S(x) \).

This uniformity check process should be individually performed for \( \forall x \in \{0,1\}^s \). We should note that for \( d > 1 \) where \( m > n \) the uniformity cannot be achieved. Hence, some of the registered output shares should be combined to reduce the number of output shares to \( n \). Afterward the uniformity can be examined. For more detailed information we refer to the original articles [BGN+14b, NRS11].
output transition, a register is connected to each LUT output. However, a single register stage in a DRP circuit contradicts the requirement of a constant gate and register transition per clock cycle [MEPP09] as inconstant and data-dependent transitions would result in data-dependent leakage. Therefore, it is required to place an even number of register stages between each two LUTs connected in the circuit. Consequently, a pipeline architecture is formed which prevents glitches by halting the propagation of a signal after each LUT. Figure 5.1(a) shows the timing diagram of a circuit following the rules defined by the GliFreD\textsuperscript{*} scheme.

Similar to many DRP schemes, it is required to place a dual of the circuit. Copying the routing structure is currently the best known way in FPGAs to keep the wire capacitances of the duplicated circuit as equivalent as those of the original circuit. Hence, to perform the circuit dualization, i.e., placing the duplicated circuit, a second horizontally-moved instance of the original circuit is placed on the FPGA. The copy process is performed as described in Section 3.2.

Similar to the GliFreD version described in Chapter 4, an arbitrary LUT configuration is allowed; since both control signals \texttt{CLK} and \texttt{active} should be connected to each LUT, the function \( f \) each LUT can realize is limited to a 4-to-1 look-up table. The output of each LUT can be seen as \( O = \text{active} \cdot \text{CLK} \cdot f(I_2, \ldots, I_5) \), while the corresponding dual function (of the duplicated circuit) becomes \( \overline{O} = \text{active} \cdot \overline{\text{CLK}} \cdot f(\overline{I_2}, \ldots, \overline{I_5}) \). Figure 5.1 shows the GliFreD\textsuperscript{*} pendant of an exemplary function

\[
y = x_0 + x_0x_3 + x_2x_3 + x_3x_4 + x_3x_6 + x_0x_7 + x_2x_7,
\]

whose standard implementation is shown in Figure 5.1(b).

Since the output of each LUT is buffered by a register, the critical path in a GliFreD\textsuperscript{*} circuit is minimized allowing to run the circuit at high frequencies. To this end the delay between the \texttt{CLK} and \texttt{active} signals should be kept minimum (see Figure 5.1(a)), that can be achieved by forcing \texttt{active} signal to be routed through the clock trees. The GliFreD\textsuperscript{*} design methodology offers the ability to transfer a design into a fully-pipelined architecture, hence achieving a high throughput in combination with a high clock frequency. In general, large combinatorial circuits cause glitches which propagate through the whole circuit. Since GliFreD\textsuperscript{*} prevents those glitches, it may also reduce the power consumption. In small combinatorial circuits this benefit is faded and dominated by the increased amount of resources the GliFreD\textsuperscript{*} circuit utilizes. Nevertheless, GliFreD\textsuperscript{*} is a resource-costly solution. The LUT overhead (at most 8) required to form a GliFreD\textsuperscript{*} circuit strongly depends on the original design structure. Compared to the LUT utilization, GliFreD\textsuperscript{*} causes a massive register overhead and hence an increased latency. The register overhead cannot be trivially estimated and depends on the LUT depth, width and the amount of registers in the original design.

5.3 Case Study

5.3.1 KATAN-32

As stated in Section 4.4, the overhead and performance of a GliFreD or GliFreD\textsuperscript{*} circuit depend on the nature of the underlying application. If the target design is made of small combinatorial

\(^1\)I_0 and I_1 are reserved for \texttt{CLK} and \texttt{active}. 
circuits, the overhead of the resulting GliFreD* circuit is minimal. Therefore, KATAN \cite{CDK09}, which benefits from a serialized architecture with small combinatorial logics, is a suitable candidate for our investigations. Further, both first- and second-order uniform TI representations of its non-linear functions are given in \cite{BGN14}, allowing us to develop the design with minimal efforts.

The architecture of our designs is based on those given in \cite{BGN14}. Figure 5.2(a) shows an overview of such a serialized architecture considering KATAN-32 encryption engine with 32-bit plaintext and 80-bit symmetric key. The plaintext and key are serially loaded into the registers, and after 254 clock cycles the ciphertext can be taken from the state register\(^2\). The first-order TI of KATAN-32 with 3 shares (the minimum settings) needs the state (shift) registers to be tripled. Similar to that of \cite{BGN14}, we do not represent the key (and the corresponding shift register) in a shared form. The XOR operations are easily repeated for each share, and the non-linear functions which are limited to the AND/XOR module (involved in function \(f_a\) and \(f_b\))

\(^2\)For more detailed information on the construction of functions \(f_a\) and \(f_b\) in Figure 5.2(a) see \cite{CDK09} and \cite{BGN14}.
Chapter 5. Hiding Higher-Order Leakages

$f_b$ of Figure 5.2(a) need to be realized under the concept of the first-order TI. An AND/XOR function receives a 3-bit input $(a, b, c)$ and gives a single-bit output $y$ as

$$y = a + bc.$$  

Following the concept of direct sharing [BNN+12] the component functions (given in [BGN+14b]) which realize a uniform first-order TI can be derived as

$$f_{i,j}((a^i, b^i, c^i), (a^j, b^j, c^j)) = a^j + b^j c^j + b^i c^i + b^j c^i,$$  (5.2)

where each output share is made by an instance of such a component function as

$$y_1 = f_{1,2}(.., ..),$$
$$y_2 = f_{2,3}(.., ..),$$
$$y_3 = f_{3,1}(.., ..).$$

The same procedure is followed to realize the second-order TI of KATAN-32. First, the minimum number of shares is increased to 5, and all state registers and linear functions need to be repeated accordingly. Further, a second-order TI representation of an AND/XOR module (given in [BGN+14b]) can be derived from Equation (5.2) and the following component function

$$g_{i,j}((a^i, b^i, c^i), (a^j, b^j, c^j)) = b^i c^j + b^j c^i.$$  (5.3)

In such a case, the output shares are made as

$$y_1 = f_{1,2}(.., ..),$$
$$y_2 = f_{1,3}(.., ..),$$
$$y_3 = f_{1,4}(.., ..),$$
$$y_4 = f_{5,1}(.., ..),$$
$$y_5 = f_{2,5}(.., ..),$$

and

$$y_6 = g_{2,3}(.., ..),$$
$$y_7 = g_{2,4}(.., ..),$$
$$y_8 = g_{3,4}(.., ..),$$
$$y_9 = g_{3,5}(.., ..),$$
$$y_{10} = g_{4,5}(.., ..).$$

As mentioned before, in a second-order case the output shares should be combined after being registered in order to reduce the number of shares back to 5. In this case, the reduction is done as

$$z_{i\in\{1,..,4\}} = y_i,$$
$$z^5 = y_5 + y_6 + y_7 + y_8 + y_9 + y_{10},$$

thereby achieving a uniform second-order TI of the AND/XOR module [BGN+14b]. For more clarification the formulas for all the component functions are given in Appendix 10.

5.3.2 PRESENT

As the second target we selected the PRESENT cipher [BKL+07] to be implemented in a round-based fashion. As Figure 5.2(b) shows, 16 instances of the Sbox in addition to the PLayer operate in parallel to compute one cipher round. The reason for choosing such a target is to have an application for GhFreD with large combinatorial circuit compared to that of KATAN. Also, due to a possibility to decompose the PRESENT Sbox – as we express below – we are able to develop its uniform first- and second-order TI representations. We should note that we have not selected the AES as a target because its first-order TI (in [BGN+14a] and [MPL+11]) can only be realized by remasking (requiring multiple fresh mask bits per clock cycle) and furthermore there is not yet a clear roadmap how to realize its second-order TI.
5.3. Case Study

Figure 5.2: Architecture of the case studies: first \((d = 1)\) and second \((d = 2)\) order TI.

Similar to the case of KATAN, the first-order (respectively second-order) TI of the targeted PRESENT architecture employs a 3-share (respectively 5-share) Boolean masking. The \texttt{PLayer} (realized by routing in the round-based architecture) is repeated on each share, and the key XOR is applied on only one share as the 80-bit key is not represented in a shared form. Clearly, the remaining part is the TI representation of the PRESENT Sbox. Previously Poschmann et al. \cite{PMK11} have shown a decomposition and a uniform first-order TI of such an Sbox. However, below we represent another decomposition allowing us to develop its both first- and second-order uniform TI representations.

The PRESENT Sbox \(S(x) = y\) is a cubic bijection (i.e., with algebraic degree \(t = 3\)) leading to minimum \(n = 4\) and \(n = 7\) shares in the first- and second-order TI settings respectively. Therefore, it is preferable to decompose the Sbox into two (at most) quadratic bijections \(F\) and \(G\), in such a way that \(S(x) = F(G(x))\) (i.e., \(S = F \circ G\)). If so, each \(F\) and \(G\) can be shared with \(n = 3\) and \(n = 5\) (for first- and second-order TI). According to the classifications given in \cite{BNN15}, the PRESENT Sbox belongs to the cubic class \(C_{266}\). It means that there exist affine transformations \(A\) and \(B\), where \(S(x) = B(C_{266}(A(x)))\). In other words, \(S\) and \(C_{266}\) are affine equivalent. To find the affine functions, the algorithm given in \cite{BCBP03} can be used; indeed, there exist four such two affine functions. Also, as stated in \cite{BNN15}, \(C_{266}\) can be decomposed into two quadratic bijections. One of the possibilities is \(Q_{294} \times Q_{299}\). It means that there exist three affine functions \(A_1\), \(A_2\), \(A_3\), where \(C_{266} = A_3 \circ Q_{299} \circ A_2 \circ Q_{294} \circ A_1\). Since \(C_{266}\) and \(S\) are affine equivalent, there exist also three affine functions to decompose the PRESENT Sbox as

\[
S(x) = A_3 \left( Q_{299} \left( A_2 \left( Q_{294} \left( A_1(x) \right) \right) \right) \right). \tag{5.4}
\]
We have found 229,376 such 3-tuple affine bijections, and we have selected one of the most simplest solutions with respect to the number of terms in their Algebraic Normal Form (ANF) directly affecting the size of the corresponding circuit.

The next step is to provide the uniform first-order TI of the quadratic bijections \( Q_{294} \) and \( Q_{299} \) which can be easily achieved by direct sharing [BNN+15]. For \( Q_{294} : 0123456789BAEFDC \) we can write

\[
e = a + bd, \quad f = b + cd, \quad g = c, \quad h = d,
\]

with \( \langle a, b, c, d \rangle \) the 4-bit input, \( \langle e, f, g, h \rangle \) the 4-bit output, and \( a \) and \( e \) the least significant bits. The component functions of the first-order TI of \( Q_{294} \) can be derived by

\[
f_{Q_{294}}^{1,2}(\langle a', b', c', d' \rangle, \langle a'', b'', c'', d'' \rangle) = \langle e, f, g, h \rangle \text{ as}
\]

\[
e = a' + b'd' + d'b' + b'd'
\]

\[
f = b' + c'd' + d'c' + c'd'
\]

\[
g = c' + (b'd' + d'b' + b'd') + (c'd' + d'c' + c'd')
\]

\[
h = d'.
\]

(5.6)

The three 4-bit output shares provided by \( f_{Q_{294}}^{2,3}(\ldots), f_{Q_{294}}^{3,1}(\ldots) \) and \( f_{Q_{294}}^{1,2}(\ldots) \) make a uniform first-order TI of \( Q_{294} \).

Following the same principle for \( Q_{299} : 012345678ACEB9FD \) as

\[
e = a + ad + cd, \quad f = b + ad + bc + cd, \quad g = c + bd + cd, \quad h = d,
\]

we can define the component function \( f_{Q_{299}}^{1,3}(\langle a', b', c', d' \rangle, \langle a'', b'', c'', d'' \rangle) = \langle e, f, g, h \rangle \) as

\[
e = a' + (a'd' + d'a' + a'd') + (c'd' + d'c' + c'd')
\]

\[
f = b' + (a'd' + d'a' + a'd') + (b'd' + d'b' + b'd') + (c'd' + d'c' + c'd')
\]

\[
g = c' + (b'd' + d'b' + b'd') + (c'd' + d'c' + c'd')
\]

\[
h = d'.
\]

(5.8)

Similarly, three 4-bit output shares provided by \( f_{Q_{299}}^{2,3}(\ldots), f_{Q_{299}}^{3,1}(\ldots) \) and \( f_{Q_{299}}^{1,2}(\ldots) \) make a uniform first-order TI of \( Q_{299} \).

Since the affine transformations \( A_1, A_2, A_3 \) do not change the uniformity and should be applied on each 4-bit share separately, the decomposition in Equation (5.4) provides a 3-share uniform first-order TI of the PRESENT Sbox. It should be noted that registers are required to be placed between the component functions of \( Q_{294} \) and \( Q_{299} \) to avoid the propagation of the glitches (see Figure 5.3). Note that the affine function \( A_2 \) can be freely placed before or after the intermediate register.

For the second-order TI representations, in addition to the above expressed component functions, we define \( g_{Q_{294}}^{1,2}(\langle a', b', c', d' \rangle, \langle a'', b'', c'', d'' \rangle) = \langle e, f, g, h \rangle \) as

\[
e = d'b' + b'd'
\]

\[
f = d'c' + c'd'
\]

\[
g = 0
\]

\[
h = 0.
\]

(5.9)

The 4-bit output shares \( y^{\ell} \in \{1, \ldots, 10\} \) are provided by

\[
y^{1} = f_{Q_{294}}^{2,3}(\ldots), \quad y^{2} = f_{Q_{294}}^{3,4}(\ldots), \quad y^{3} = f_{Q_{294}}^{4,5}(\ldots), \quad y^{4} = f_{Q_{294}}^{5,1}(\ldots),
\]

\[
y^{5} = f_{Q_{294}}^{1,2}(\ldots), \quad y^{6} = f_{Q_{294}}^{4,2}(\ldots), \quad y^{7} = g_{Q_{294}}^{2,4}(\ldots), \quad y^{8} = g_{Q_{294}}^{2,5}(\ldots),
\]

\[
y^{9} = g_{Q_{294}}^{2,5}(\ldots), \quad y^{10} = g_{Q_{294}}^{1,3}(\ldots).
\]

(5.10)
Figure 5.3: A first-order TI of the PRESENT Sbox: \( S(x) = y \).

After a clock cycle, when \( y^i \in \{1, \ldots, 10\} \) are stored in dedicate registers, the output shares should be combined as

\[
    z^i \in \{1, \ldots, 5\} = y^i + y^{i+5},
\]

which provides the uniform second-order TI of \( Q_{294} \).

The same procedure is valid in case of \( Q_{299} \) considering the component function

\[
    g^{ij}_{Q_{299}} \left( (a^i, b^i, c^i, d^i), (a^j, b^j, c^j, d^j) \right) = (e, f, g, h)
\]

as

\[
    e = d^i a^j + d^i c^j + a^i d^j + c^i d^j \\
    f = d^i a^j + d^i b^j + d^i c^j + a^i d^j + b^i d^j + c^i d^j \\
    g = d^i b^j + d^i c^j + b^i d^j + c^i d^j \\
    h = 0.
\]

By changing the indices from \( Q_{294} \) to \( Q_{299} \) in Equations (5.10) and later applying the reduction in Equation (5.11), a uniform second-order TI of \( Q_{299} \) is achieved. Hence by means of these component functions in addition to the affine transformations, we can realize a uniform second-order TI of the PRESENT Sbox. Figure 5.4 shows the graphical view of such a construction, and all the required formulas are given in Appendix 11. Note that the registers after the affine function \( A_2 \) can instead be placed before \( A_2 \) right after the reduction from 10 to 5 shares.

5.3.3 Implementation

Based on the specifications given above and considering a Spartan-6 FPGA (indeed the XC6SLX75 of SAKURA-G [sak]) we implemented six designs. The first three ones are different profiles of KATAN-32, and the next three designs realize the encryption of PRESENT with a round-based architecture. For each of the targeted cipher we implemented:

- the first-order TI, i.e., KATAN-1st and PRESENT-1st profiles,
- the second-order TI, i.e., KATAN-2nd and PRESENT-2nd profiles, and
- the first-order TI with GliFreD, i.e., KATAN-1st-G and PRESENT-1st-G profiles.

Although we did not consider any constraints on placement and routing of the four non-GliFreD\(^*\) profiles, following the principles of GliFreD\(^*\) the corresponding profiles have been
realized by first defining an area on the target FPGA, where the component of the original part of the GliFreD* circuit should be placed. After finishing the placement and routing, the corresponding dual circuit, i.e., the duplicated part of the GliFreD* circuit, has been cloned and dualized by means of the RapidSmith tool [LPL+12] (see Section 3.2). As a reference, the circuits shown in Figure 5.1 are the normal and GliFreD* realizations of the least significant bit $e$ of Equation (5.8).

Due to its serialized ring architecture, the $\text{KATAN-1st-G}$ profile does not form a pipeline. The most important difference between such a profile and its original one ($\text{KATAN-1st}$), is on the one hand, the number of required clock cycles to finish an encryption (i.e., latency) which is doubled and, on the other hand, the raised achievable clock frequency due to the minimal LUT depth. The max LUT depth in GliFreD* circuits is one which results in a short critical path. However, the $\text{PRESENT-1st-G}$ profile is implemented in a fully-pipelined way, so that the round-based architecture is able to hold 11 different cipher states. Hence, after $32 \times 11 \times 2 = 704$ clock cycles, 11 encryptions with the same key are performed. The pipelined architecture naturally increases the register utilization of the components but provides a much higher throughput.

Table 5.1 compares the overhead and performance of different design profiles. It indeed gives an overview on the disadvantage (area and time overheads) as well as the advantage (throughput) of employing GliFreD* with respect to two different design architectures, i.e., a fully-serialized one which is register oriented ($\text{KATAN-1st-G}$) and a round-based one which is combinatorial oriented ($\text{PRESENT-1st-G}$). As shown by Table 5.1, although the resource utilization and the latency of the GliFreD* profiles are drastically increased, the throughput is still kept comparable with the original design profiles. Such achievements are mainly due to the naturally-minimized critical paths in the GliFreD* designs allowing a high clock frequency.
Table 5.1: Details about the implemented profiles. The values given in this table are taken from the post route synthesis report of Xilinx ISE 14.7.

<table>
<thead>
<tr>
<th>Profile</th>
<th>Resources</th>
<th>Frequency (MHz)</th>
<th>Latency (#clock)</th>
<th>Pipeline (stage)</th>
<th>Throughput (MBit/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>KATAN-1st</td>
<td>34</td>
<td>225.38</td>
<td>273</td>
<td>1</td>
<td>26.42</td>
</tr>
<tr>
<td>KATAN-2nd</td>
<td>65</td>
<td>321.54</td>
<td>273</td>
<td>1</td>
<td>37.69</td>
</tr>
<tr>
<td>KATAN-1st-G</td>
<td>114</td>
<td>438.21</td>
<td>546</td>
<td>1</td>
<td>25.68</td>
</tr>
<tr>
<td>PRESENT-1st</td>
<td>808</td>
<td>206.61</td>
<td>64</td>
<td>2</td>
<td>413.22</td>
</tr>
<tr>
<td>PRESENT-2nd</td>
<td>2245</td>
<td>203.46</td>
<td>128</td>
<td>4</td>
<td>406.92</td>
</tr>
<tr>
<td>PRESENT-1st-G</td>
<td>5442</td>
<td>458.09</td>
<td>704</td>
<td>11</td>
<td>458.09</td>
</tr>
</tbody>
</table>

5.4 Practical Evaluation

In addition to the performance and overhead figures given in Section 5.3.3, we practically examined the ability of each of our six developed designs to avoid side-channel leakages. Again, the practical evaluation of the previously presented methods are done by Amir Moradi.

The experimental platform is a SAKURA-G [sak] equipped with a Xilinx Spartan-6 FPGA. The side-channel leakages have been measured by collecting power consumption traces of the underlying FPGA by means of a Teledyne LeCroy HRO 66Zi digital oscilloscope at a sampling frequency of 500 MS/s and a limited bandwidth of 20 MHz. Due to the low peak-to-peak amplitude of the signals we also made use of the amplifier embedded on the SAKURA board. For all six design profiles, the target FPGA operated at a frequency of 24 MHz during the collection of the power traces. Our intuition on the measured power traces from our platform is that the traces are heavily filtered by the measurement setup including the shunt resistor, chip packaging, PCB, and probes. Measuring the power traces with a high bandwidth (> 20 MHz) leads to higher electrical noise. We have examined this behavior and observed leakages easier when the bandwidth is limited. Note that this intuition does not hold true in case of EM measurements.

It is noteworthy that such a frequency of operation has intentionally been taken in order to: i) cover the full power trace length in the measurements as the KATAN profiles need 254 clock cycles after data being loaded (respectively 508 for KATAN-1st-G), and ii) cause the power peaks of adjacent clock cycles slightly overlap each other. The later has been considered with respect to the note given in [Rep15] that the second-order TI can still be vulnerable to a second-order bivariate attack. Recalling the techniques introduced in [MM13], employing certain amplifiers or running the device at a high clock frequency leads to converting multivariate leakages to univariate. It has been shown in [SMG15] that a second-order TI design actually can exhibit a univariate second-order leakage if the measurement setup is employed by certain components, e.g., Direct Current (DC) blockers and/or amplifiers. Hence, operating the device at 24 MHz allows us to easily cover the long traces in the measurements and provide particular situations, where second-order TI profiles may demonstrate second-order leakage.

As the evaluation metric we employed the leakage assessment methodology of [GJJR11, SM15] which is based on the Student’s $t$-test. The reason for such a choice is twofold. First, the $t$-test can examine the existence of detectable leakages without performing any key-recovery attack, which significantly eases the evaluation process particularly where higher-order leakages
using millions of traces should be examined. Moreover, the efficiency of the state-of-the-art key-recovery attacks strongly depends on the targeted intermediate value and the underlying (power) model. Second, the same leakage assessment technique (more precisely the non-specific \( t \)-test also known as fixed vs. random test) has been used to examine the resistance of different threshold implementations (for example, see [BGN+14b] and [SMG15]). In order to keep our evaluations comparable with the former ones, we trivially employed the same evaluation method.

![Sample Trace](image1)
![PRNG off (10,000 traces)](image2)
![First-order](image3)
![Second-order](image4)
![Third-order](image5)

Figure 5.5: **KATAN**-1st profile, sample trace and non-specific \( t \)-test results using 1,000,000 traces.

It is noteworthy that all the tests we performed here are based on a univariate scenario. In other words, we did not run any combination function on different sample points of each collected power trace. Further, we followed the same principle explained in Section 2.2.3 to conduct the tests at higher orders. Indeed, these preprocesses required for higher-order evaluations are with the respect to the centered and standardized higher-order statistical moments.
We start our evaluations with KATAN-1st profile. Figure 5.5(a) shows a corresponding sample power trace. Note that the collected power traces do not cover a time period when plaintext and key are serially loaded into the shift registers. In order to have an overview about the quality of the measurement setup and verify the employed evaluation metric, for the first analysis we turned the Pseudo-Random Number Generator (PRNG) off thereby forcing all masks used for sharing the plaintexts to zero. As shown by Figure 5.5(b), the first-order $t$-test shows clear detectable leakages using a few 10,000 traces. By keeping the PRNG active and conducting the same non-specific $t$-tests up to third-order using 1,000,000 traces we observed the curves shown by Figure 5.5(c), 5.5(d), and 5.5(e), which indeed confirm the first-order resistance and vulnerability at the second and third orders, as expected.

Figure 5.6: KATAN-2nd profile, sample trace and non-specific $t$-test results using 100,000,000 traces.
For the KATAN-2nd profile we had to collect much more traces to be able to observe the higher-order leakages. It is due to the high order of sharing, i.e., at least 5 shares (see Section 5.2.1) in case of a second-order TI. In fact, we observed the fourth- and fifth-order leakages using approximately 100,000,000 traces, as shown in Figure 5.6. However, in order to examine the issue reported in [Rep15] (by operating the target at 24 MHz) we continued the collection of the traces up to 500,000,000, but we have not observed any second-order leakage while the fourth- and fifth-order leakages became detectable – expectedly – with higher confidence. We should here refer to the issue addressed in [Rep15] and the detectable second-order leakage reported in [SMG15]. Based on the explanations of [Rep15], a second-order bivariate leakage should be detectable, but such a bivariate leakage is not necessarily detectable from the consecutive clock cycles that can additively be combined by means of an amplifier or running the device at a high clock frequency [MM13]. In case of the application of [SMG15], apparently the consecutive clock cycles exhibit such a bivariate leakage, but it is not hold true for the serialized KATAN architecture. Further, compared to our design profiles the constructions in [SMG15] make use of a kind of remasking which is a different methodology to ensure the uniformity.

Following the same scenario we performed the evaluations on the KATAN-1st-G profile and collected 1,000,000,000 traces to perform the same \( t \)-tests at up to third order. The corresponding results which are depicted in Figure 5.7 indeed confirm the effectiveness of the underlying hiding technique to significantly harden the higher-order attacks. The result of this profile can be compared to that of the KATAN-1st profile (Figure 5.5), where 1,000,000 traces are adequate to observe the second- and third-order leakages.

The same leakage assessment technique has been conducted on the three profiles of the round-based PRESENT architecture, and the corresponding results are shown in Figure 5.8, Figure 5.9, and Figure 5.10. For the PRESENT-1st profile we required 10,000,000 trace to observe the second- and third-order leakages. Respectively 300,000,000 traces were necessary for the PRESENT-2nd profile to exhibit fourth- and fifth-order leakages. We should again bring the reader’s attention to the infeasibility to observe a second-order leakage from the PRESENT-2nd profile. We indeed continued our evaluations on this profile by measuring 1,000,000,000 traces as well as with different fixed inputs (with respect to the non-specific \( t \)-tests), but in none of the tests we observed a detectable second-order leakage. As an example, we give the results of one of such tests with 1,000,000,000 traces in Appendix 12, where the third-order leakage also becomes detectable. Finally, similar to the KATAN GliFreD design we collected 1,000,000,000 traces and conducted the same non-specific \( t \)-tests on the PRESENT-1st-G profile, which still shows robustness to avoid the leakages to be detectable at first, second, and third orders.

### 5.5 Conclusion/Discussion

Comparing the presented practical results, at the first glance it can be noticed that the GliFreD* profiles consume more energy than the other corresponding profiles. They also increase the number of required clock cycles (latency) particularly in case of the PRESENT design as its combinatorial circuit has a longer depth compared to the KATAN design. However, their achievement, i.e., hiding the higher-order leakages to make the higher-order attacks practically infeasible, is confirmed. Hence, it can be concluded that the combination of such a power-equalization technique and a proper masking scheme (i.e., first-order TI) gives a high level of confidence to argue the practical infeasibility of the key-recovery attacks.
Our comparisons are limited to the second-order TI of KATAN and PRESENT, which can be extended to higher-order TI designs. However, by increasing the desired order of security the number of shares and the required internal PRNGs respectively increase (e.g., at least 7 and 9 shares for third- and fourth-order TI). Note that the numbers given in Table 5.1 exclude the area required for the PRNGs.

Nonetheless, due to the local separation of false and true parts in GliFreD* circuits, the resistance of our proposed method against higher-order EM attacks is still an open question and should be addressed in the future. Further, GliFreD* is exclusively designed for FPGAs and uses the fixed LUT structure to realize Boolean functions of a circuit. Transforming this logic style naively to ASIC may not lead to the expected results especially with respect to the area overhead. The idea of combining TI with DRP styles can be adopted for ASICs by employing one of the logic styles designed for ASICs in addition to a customized router.
Chapter 5. Hiding Higher-Order Leakages

(a) Sample Trace
(b) First-order
(c) Second-order
(d) Third-order

Figure 5.7: KATAN-1st-G profile, sample trace and non-specific $t$-test results using 1,000,000,000 traces.

(a) Sample Trace
(b) First-order
(c) Second-order
(d) Third-order

Figure 5.8: PRESENT-1st profile, sample trace and non-specific $t$-test results using 10,000,000 traces.
Figure 5.9: PRESENT-2nd profile, sample trace and non-specific t-test results using 300,000,000 traces.
Figure 5.10: PRESENT-1st-G profile, sample trace and non-specific t-test results using 1,000,000,000 traces.
Part III

PUFs on Reconfigurable Hardware
Chapter 6

Recovering Initial SRAM Values

In this chapter we provide a method to use the BRAM components of Xilinx 7 series devices as SRAM-based PUFs. The proposed method is applied after the power-up reset of the memory cells and does not need a hardware reset to generate the PUF responses.

Contents of this Chapter

6.1 Introduction ................................. 87
6.2 Bypass BRAM Reset ......................... 88
6.3 Evaluation ................................. 89
6.4 Conclusion ................................. 93

6.1 Introduction

In the past years a large number of PUF variants have been proposed as some of them are noted in Section 2.3.4. Despite the wealth of PUF constructions, an open research question is which of them provide the best properties for use in real-world applications. The straightforward design of SRAM-based PUFs which simply evaluate initial states of memory cells has been identified as promising and has already been emerged to IP cores for integration into ASICs. However, transferring this concept to FPGAs – at least for those of the market leader Xilinx – is not possible since all SRAM-based BRAMs in Xilinx FPGAs are automatically cleared on power-up [SGB+12], destroying the desired initial information bits. Due to this start-up procedure, it is therefore widely assumed that it is not possible to implement SRAM PUFs in Xilinx FPGAs.

The SRAM PUF concept proposed in [GKST07] has been studied quite well by the PUF community with promising results. Unfortunately, the power-up reset of the FPGAs deletes the initial SRAM values and hinders the usage of this concept in FPGAs. Nonetheless, motivated by great properties and results SRAM PUFs have, some research was done to implement this PUF construction on FPGAs. For instance, Sander et al. [SGB+12] read out the SRAM cells of unused configuration memory of Xilinx Virtex-5 devices which are not reset during power-up. Another approach was proposed by Güneysu in [Gün12]. He used the dual-port ability of Xilinx Spartan-3 BRAMs and enforced a write collision by writing LO value from one port and HI from the other port into the same SRAM cell. This set the cell into an undefined metastable state and the cell followed the concept of an SRAM PUF. Maes et al. realized that the SRAM cells holding the initial FF values of Xilinx Virtex-II Pro are not reset during the
device power-up. In [MTV08] they proposed a method to read out the memory content of those device types.

**Contribution**

In this chapter, we present a novel way to implement SRAM PUFs on Xilinx FPGAs that overcomes the initial clearing of all SRAM-based memory blocks during system start-up. In the 7 series devices manufactured at 28nm technology, Xilinx has added the ability to shut off unused SRAM memory blocks to reduce the power consumption to a minimum. We exploit the availability of these power gates to switch off a block memory while in operation and re-enabling it afterwards without explicit initialization. The deactivation and subsequent activation procedure can be done by partial reconfiguration of corresponding parts of the device. Since only small parts need to be replaced, this can be easily done internally by using the integrated Internal Configuration Access Port (ICAP) or Processor Configuration Access Port (PCAP) interface. Regaining a portion of uninitialized SRAM block in Xilinx FPGAs, we demonstrate that the straightforward construction of SRAM PUFs is indeed possible with the 7 series of Xilinx FPGA.

### 6.2 Bypass BRAM Reset

The fundamental obstacle Xilinx FPGAs have – from the SRAM PUF point of view – is the global power-up reset and the autonomously triggered initialization of BRAM cells. To enable an SRAM PUF, an efficient work-around for the reset has to be found. For the work-around we make use of the power-gating feature in Xilinx FPGAs that enables to disconnect not instantiated BRAM from the power supply network. This feature became available with the 7 series [Xil13b]. In combination with partial reconfiguration, this feature is exploited to disconnect and connect BRAM blocks to the power network and reveal the initial state of their SRAM cells.

An evaluation setup to bypass memory initialization can be described in five steps.

1. The FPGA is powered and the power-up reset is triggered.
2. A bitstream, running a design to read BRAM content, is loaded into the FPGA.
3. A partial bitstream with BRAM instances and initialization values is loaded into the FPGA. This process preloads the BRAMs with initialization values of our choice. The effort of this step can also be integrated into the former and is just separated to reset the BRAMs in a repetitive measurement setting.
4. A second partial bitstream is loaded into the FPGA which disables BRAMs and disconnect them from the power network.
5. A manipulated third partial bitstream is loaded into the FPGA with instantiated BRAMs but without initial values to skip the initialization. Loading this bitstream reconnects the BRAMs to the power network but does not overwrite any content contained in the memory. At this point in time the PUF responses are available in the BRAMs.
6.3 Evaluation

After power-up of the FPGA, the steps 3-5 could potentially be used to create an SRAM PUF and also repeated without any intermediate hardware reset of the device. We refer to steps 3-5 in the remainder of this chapter as reconfiguration loop.

Next, we briefly describe how we modified the configuration bitstreams to prevent the initialization of SRAM block memories. A more detailed description of the bitstream layout can be found in Section 2.1.4. All Xilinx bitstreams are organized in frames. Each frame belongs to a special register to which it is written during the FPGA configuration. Relevant registers for this chapter are FDRI that contains the configuration frames and the FAR to store the frame addresses mapping configuration data to a specific place on the FPGA. Addresses are typically not explicitly included in the frame format and therefore implicitly incremented. As explained in [Xil13a], initial BRAM values are placed in separate frames at the end of an FDRI block. To finally remove the initial values from a configuration bitstream, the frames with the initialization data can be simply cut out of the FDRI operands. It is also necessary to compute a new word count for the remaining frames replacing the previous value in the instruction header. For partial bitstreams, BRAM initialization frames are sometimes also located in a separate FDRI write instruction. In this case the complete instruction can be removed to instantiate the BRAM without any initialization. Note that due to the manipulation, the CRC validation will fail and the FPGA will not be configured. The CRC value needs either to be updated or the CRC check has to be disabled (“bitgen -g crc:disable”).

6.3 Evaluation

To evaluate the quality of a PUF class, both reliability and uniqueness of the responses are essential quality criteria. Further, relevant PUF metrics like unpredictability, physical, or mathematical unclonability are not considered in this section. We now provide results for the proposed SRAM-based PUF on Xilinx Zynq FPGAs.

6.3.1 Memory Decay and Flip Direction

As depicted in Section 6.2, the BRAMs under test are always initialized with a given value in the first place. Skorobogatov has shown in his work [Sko02] that shortly after a power-down the SRAM content is still available. This effect is known as remanence of memory cells. To evaluate the influence of this effect on the BRAM, we performed the following test. BRAMs are initialized either with 0 or 1 as described above. A complete reconfiguration loop (steps 3-5) is performed to produce the response bits. For subsequent measurements all SRAM cells are reset to their initial state. After extracting the response bits, the flipped SRAM cells can be easily identified. Figure 6.1 shows the result for 70 × 32K-BRAM on a Zynq 7020 (revision 0). We repeated our experiments with increasing waiting times between reconfiguration cycles as given in Figure 6.1. Interestingly, we did not see any impact of the remanence effect. Rather more, it shows that the number of SRAM cells changing their state is correlated to the temperature of the device. We finally conclude that the SRAM cells are more affected by the environmental temperature than the time the BRAM was turned off.

Another aspect that is clear in Figure 6.1 is that only about 1.4% of the SRAM cells changed their content while other devices of the same revision showed other switching probabilities. The switching probabilities for each tested device can be found in Table 6.1. This value strongly
Chapter 6. Recovering Initial SRAM Values

Figure 6.1: The amount of SRAM cells changing their state, based on the time the BRAM was disconnected from the power network.

differ between tested Zynqs of the same and different chip revisions. Devices of later revisions (1 and 2) come up with little activity even with long turn off times of the BRAM, so that the remainder of the analysis focuses on devices of revision 0. The reduction of content losing SRAM cells of later revisions is not unusual. A common reason for the changed behavior is improved manufacturing processes.

Comparing the results of Figure 6.1(a) and 6.1(b) and taking the high correlation between temperature and the number of flipped cells into count, both initial values show approximately the same amount of SRAM cells that changed their content. In a more detailed analysis the

<table>
<thead>
<tr>
<th>ZedBoard ID</th>
<th>Revision</th>
<th>% Flips&lt;sub&gt;Init=0&lt;/sub&gt;</th>
<th>% Flips&lt;sub&gt;Init=1&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>446435</td>
<td>0</td>
<td>8.1287%</td>
<td>7.9526%</td>
</tr>
<tr>
<td>446489</td>
<td>0</td>
<td>1.6688%</td>
<td>1.3839%</td>
</tr>
<tr>
<td>447633</td>
<td>0</td>
<td>1.4867%</td>
<td>1.5487%</td>
</tr>
<tr>
<td>471018</td>
<td>1</td>
<td>0.0078%</td>
<td>0.0040%</td>
</tr>
<tr>
<td>492968</td>
<td>2</td>
<td>0.0005%</td>
<td>0.0019%</td>
</tr>
</tbody>
</table>

Comparing the results of Figure 6.1(a) and 6.1(b) and taking the high correlation between temperature and the number of flipped cells into count, both initial values show approximately the same amount of SRAM cells that changed their content. In a more detailed analysis the
addresses of affected SRAM cells of both initial states are extracted and compared to each other. The address comparison shows just a little overlap. This concludes that the SRAM cells keep or switch their content based on their initial state. Hence, the gathered entropy per BRAM can be increased by evaluating the SRAM cells for both initial values.

6.3.2 Unpredictability of SRAM Cells

To show the unpredictability of the SRAM-based Xilinx PUFs, the addresses of switching SRAM cells must be independent from other cells of the same and different devices.

To verify the independence of SRAM cells on the same device, the BRAMs with toggled bits at the same address are counted. The result for transitions from 0 to 1 is listed in Histogram 6.2(a). In case of a total independence an almost horizontal line for the bar edges is expected. Figure 6.2(a) shows that with exception of the 16 highest and 16 lowest addresses, the switches are almost evenly distributed. With a switching probability \( p \) for each bit and \( x \) representing the counted flips for one address of 70 BRAM blocks, the given Binomial Distribution \( f_{\text{bino,intra}} \) is expected. Figure 6.2(b) shows the measured distribution for the bit addresses and the Binomial Distribution given by the red line.

\[
f_{\text{bino,intra}}(x, 70, p) = \binom{70}{x} \cdot p^x \cdot (1 - p)^{70-x} \cdot 32 \cdot 1024 \quad \text{for} \quad 1 \leq x \leq 70
\]

(a) Address count for switched bits in one device (ID: 446435)

(b) Binomial Distribution for 70× 32K-BRAM in one device (ID: 446435)

Figure 6.2: The distribution of the SRAM cells changing their state (1V, 23°C).

The almost equal distribution of flipped bits over the address space in each BRAM and the high similarity to the expected Binomial Distribution concludes that the SRAM cells flip independently of each other in a device.

The exceptions are the 16 highest and 16 lowest addresses. The transitions from 0 to 1 occur on some of these addresses with a probability of about 50% which seems to be ideal. Further experiments have shown, however, that the inter-distance of these bits is still unbalanced making them for some portions of the RAM predictable. Therefore, such SRAM cells have to be handled carefully or better completely removed from the PUF response.
To show the independence of SRAM cells on different devices, the switching characteristics for each BRAM block and address on 3 devices are compared. More precisely, for each address and BRAM the flips of 3 devices are counted. Ideally, for a switching probability $p$ of an address, the given Binomial Distribution $f_{\text{bino, inter}}$ is expected if the switches for each address in each BRAM is counted separately.

$$f_{\text{bino, inter}}(x, 3, p) = \binom{3}{x} \cdot p^x \cdot (1 - p)^{3-x} \text{ for } 1 \leq x \leq 3$$

The analyzed measurement result also fits exactly into the statistical distribution for the switching behavior of three devices. This concludes that SRAM cells on different devices show an independent switching characteristic, and hence the BRAM PUF acts device independently.

### 6.3.3 Reliability

As previously noted, a common metric to evaluate the reliability of SRAM PUFs is the intra-distance. The intra-distance is the distance between two responses of the same challenge and PUF instance. The distance can be any well-defined distance metric. In this chapter the fractional HD is used. The ideal intra-distance for a PUF class is zero. The behavior of a lot of PUF classes is influenced by environmental conditions. Therefore, the intra-distance tests are performed under different environmental conditions. More precisely, the temperature and supply voltage under which the PUF operates are changed. To get a more reliable result, the PUF responses are generated 100 times for each environmental aspect and set in relation to a reference value under nominal conditions. Figure 6.3 shows the result of the intra-distance test. Beside the mean value, the standard derivation is given for each condition. It is clear to see that under nominal and stable environmental conditions, the behavior of the SRAM cells shows an acceptable reliability. The temperature as well as the supply voltage highly influence the cells which results in noisy PUF responses.

![Figure 6.3: The mean fractional intra-distance as well as the standard derivation for different environmental conditions. Note that the device under test (ID: 446489) shows a little flip-rate of about 1.5%.](image-url)
6.3.4 Uniqueness

Beside reliability, the uniqueness is also one of the most important properties of a PUF class. To evaluate the uniqueness, the inter-distance is the usual metric. The inter-distance is defined as the distance between two responses for the same challenge produced by different PUF instances. As a distance metric we also use the fractional HD. The histogram for the inter-distance is given in Figure 6.4. Ideally, the inter-distance is distributed around 50%. As stated before, a lot of SRAM cells in the BRAMs keep their initial state. So, many SRAM cells of compared BRAM blocks come up with the same value which lowers the inter-distance between responses. In general, a Binomial Distribution is expected for the inter-distance. Unlike this expectation, Figure 6.4 shows two binomial-like distributions. This is due to the strong variation of the flip probability (Table 6.1) and the lower number of devices under test.

![Histogram](image)

Figure 6.4: The inter-distance between three devices of revision 0, and 70 BRAMs.

6.4 Conclusion

We have shown in this chapter that the availability of the power-gate feature at the BRAMs of Xilinx 7 series devices can be used to generate an SRAM PUF even on devices that typically run memory initialization on power-up. It turned out that most of the in BRAM located SRAM cells keep their initial value, which drastically decreases the gathered entropy of PUF responses. Additionally, just little environmental variations have a strong impact on the PUF response. However, the evaluation of the presented SRAM PUFs is preliminary and uses only three devices. Future tests with a larger set of devices and an inclusion of other tests like aging would evaluate the PUF quality more comprehensively. We are confident that our results can also be applied to other device families besides Zynq. The significant deviations of results obtained from different chip revisions of the same type of FPGA is a further aspect that needs to be addressed in future works.
Chapter 7

Implementation of Ring-Oscillator PUFs

In this chapter we analyze the RO PUF architecture used by Maes et al. in [MHV12] with respect to its noted structural bias and the influence of surrounding logic. Additionally, we propose FPGA-optimized RO instances that are able to either implemented with lower frequencies to stabilize the RO or with reduced area consumption. In this context we found that counters stop to work on some devices and tried to further analyze that phenomena.

Contents of this Chapter

7.1 Introduction ................................. 95
7.2 Case Study .................................. 96
7.3 Implementation Issues ....................... 97
7.4 Efficient Implementation Techniques .......... 100
7.5 Evaluation .................................. 102
7.6 Conclusion .................................. 104

7.1 Introduction

One of the most popular PUFs on FPGAs is the RO PUF [GCvdD02]. It is based on the observation that the frequency of a RO heavily depends on process variations. In the classic RO PUF, two ROs are compared to generate a single bit. By using multiple of these RO pairs a device-specific key can be generated. The disadvantage of this method is that a large number of ROs is needed to generate a key. To reduce this overhead, Yin and Qu proposed not to compare two ROs to generate one bit but to sort n ROs according to their frequencies to generate up to $\log(n!)$ bit-entropy [YQ10]. This greatly reduces the number of needed ROs. However, to build a secure and reliable key generator, a PUF on its own is not enough since the PUF responses are not completely reliable and also do not have full bit-entropy. A so-called Fuzzy Extractor is therefore needed to generate secure and reliable keys from noisy PUF responses [DRS04]. One of the few examples of a fully functional PUF-based key generator was introduced at [MHV12]. This construct, called PUFKY, uses 848 ROs and 2052 bits of helper data to generate a 128 bit full-entropy key.
Chapter 7. Implementation of Ring-Oscillator PUFs

Contribution

While PUFKY shows great promise as a secure key generation architecture, some optimizations can be applied to improve the concept. First, the RO instances consume most of the resources used by PUFKY and hence optimizations in these components will result in high area savings. Furthermore, as noted in [MHV12], the RO output is strongly bias due to the structure of the RO instances.

We therefore made an exhaustive analysis of the PUFKY design and searched for possible optimization. The main findings and contributions of this chapter can be summarized as follows:

- We introduce an optimized RO design that can save up to 50% of area compared to previous designs.
- We show that in practice RO PUFs are susceptible to counter failures. This phenomenon has large security implications for RO PUFs but has not been discussed in this context before.
- We study these counter failures and show that they are device- as well as environmental-dependent. This makes it extremely challenging to verify RO PUFs on FPGAs and raises questions about their reliability.
- We furthermore show that unrelated IP cores can greatly reduce the entropy of the PUF once they are activated. This is even true for IP cores that are placed in opposite corners of the FPGA.

7.2 Case Study

As a case study for our work, we analyzed the code of PUFKY which was published at [MHV12]. PUFKY is based on two major components: the RO PUF architecture, which is used as an entropy source, and the post-processing component that consists of a Fuzzy Extractor for the error correction and entropy extraction. This chapter only focuses on the RO PUF architecture, hence a brief description of this component is given in the following. The RO PUF architecture (depicted in Figure 7.1) consists of a RO grid, a multiplexer, a counter, a normalization step, a Lehmer-Gray encoder and an entropy compression component.

To extract the hidden device-specific information from multiple ROs, PUFKY uses a resource-friendly approach based on an idea proposed in [YQ10]. In this concept a grid of ROs (64 \times 16) with each column sharing a counter is generated. The frequencies of a full row (i.e., a batch of 16 ROs) are used to generate a multi-bit response which is in general the encoded order of the RO frequencies. There are several ways to represent an order of elements in binary format. For PUFKY, a Lehmer-Gray encoder is used which produces a 49 bit output. Please note that the maximum entropy of ordering 16 elements is $\log_2(16!) = 44.3$, and therefore the 49 bit response does not have full bit-entropy. To increase the bit-entropy, a simple entropy compression is performed by XORing the bits with the least entropy. The result of this compression is a 40 bit or 42 bit value with a maximum bit-entropy of 0.9795 or 0.9878 respectively. In general, ROs are highly affected by environmental conditions like temperature and voltage. The time period, the ROs oscillate, is determined by a reference RO. In this way, environmental changes affect the reference RO as well as the PUF ROs which helps to stabilize the counter values.
7.3 Implementation Issues

7.3.1 Structural Bias

A strong structural bias was observed in PUFKY [MCMS10], and the authors proposed a normalization step to counter this structural bias. This normalization step is done by simply subtracting a normalization term from the RO frequency counters. In the following we will analyze the source of this structural bias and its implications in more detail. In Figure 7.2(a), the frequency values of the RO grid used by the original PUFKY implementation are shown. One can clearly see the noted strong structural bias that follows a regular pattern. A closer look at the implementation at a lower level reveals the reason for the bias: a RO in PUFKY is instantiated in half of a slice that is routed using the Xilinx tools. The ROs in a batch are made of different slice types that are not routed equally. In particular, these slice types are internally different and have therefore different delay characteristics. Furthermore, an equivalent routing with different slice types is hard to achieve due to the non-public switch matrix structure and the different wire lengths between switch matrix and slice types. The different slice types and the routing are clearly the main sources of this structural bias.

In PUFKY, precomputed normalization terms are subtracted from the RO values to overcome this structural bias. In Figure 7.2(b), the normalized frequencies are plotted, and it seems that the normalization gets rid of the structural bias. However, not only the mean values but also the variances of the ROs show a structural bias. Figure 7.3 shows the variances of the 64 batches after normalization. Please note that structural variance also reduces the entropy of the sorted values. This is due to the fact that ROs with a high variance are more likely to either have large or low value while the ROs with low variance will be closer to the mean. But the normalization
only changes the mean value of a distribution but not the variance, and therefore a different strategy is needed to get rid of the structural variances.

![Grid frequency of PUFKY](image1.png)

(a) Grid frequency of PUFKY

![Grid frequency of PUFKY after normalization](image2.png)

(b) Grid frequency of PUFKY after normalization

Figure 7.2: Untouched and normalized RO grid frequency of PUFKY.

Fortunately, there is a rather simple solution to reduce the structural bias as well as the structural variance. One simply has to build the batches over the same slice types and the same routing. In such a construction the structural bias within a batch is minimal, and hence the variance of each RO in a batch is similar as well. For PUFKY this can be done by simply building the batches vertically instead of horizontally.

![Grid variance of PUFKY after normalization](image3.png)

Figure 7.3: Grid variance of PUFKY after normalization.

### 7.3.2 Malicious IP

Not only the RO PUF design itself can lead to bias. It is known that surrounding logic can influence the behavior of the RO [MCMS10, MSE10]. There are different reasons why surrounding logic can impact the ROs. First of all, the surrounding active logic generates local heat so that the switching speed of near inverters is increased. A second reason can be cross talk of RO wires and the logic. A third reason is a slight voltage drop caused by surrounding logic at its
transitions. An intuitive way to minimize the effect on the RO is an adequate spacing between the RO and the surrounding logic. However, the size of an adequate spacing, so that the logic has no impact on the RO behavior, and how big this impact can be, has not been analyzed yet. We therefore conducted an experiment to see how much impact an IP core that is placed on the same FPGA but without direct connection to the PUF can have. For this, we implemented a core that consists of XORs with looped output signals which produces a glitch core. Activating the core generates a high local temperature increase and high voltage drop. The glitch core is implemented in an area of $27 \times 27$ slices and can be activated by an external signal. The size of the glitch core was inherently given by the synthesizer, since a larger core was not routable by the toolflow. The RO grid and the glitch core are placed on the opposite corners of the FPGA. In case of an inactive glitch core, the RO PUF showed a similar behavior as the original design. However, once the glitch core was active, the frequencies of the RO changed drastically as can be seen in Figure 7.4(a) compared to Figure 7.2(b). When the core is active, the frequencies of the ROs show a visible slope towards the right. The security implication of this change can be seen in Figure 7.4(b) which shows the uniqueness of the PUF design before and after the glitch core was activated. Hence, even though the glitch core and the RO grid have been placed on opposite corners of the FPGA, the impact of the glitch core was large enough to greatly reduce the entropy of the RO PUF. This shows that there are no upper bounds on the required security margin and the entire FPGA design (including third party IP cores) has to be considered during PUF analysis.

![Figure 7.4: Impact of the glitch core on the RO grid.](image)

(a) Grid frequency of PUFKY after normalization with noisy core in FPGA corner  
(b) Uniqueness with glitch core

### 7.3.3 Counter Failures

Due to the potential impact of surrounding control logic, we inserted a spacial security margin around the RO grid of 6 slices (this number was arbitrary chosen) in which no logic was placed. We then evaluated this PUF design using 22 Atlys boards from Digilent which are equipped with a Xilinx Spartan-6 and expected a similar behavior on all boards. However, we observed something curious: in one of these 22 boards a few ROs returned a frequency count value of one. Figure 7.5 illustrates the RO frequency counts for this specific board. As one can see, the
behavior of this board is clearly out of norm and it seems that the counter does not sample the oscillating RO output correctly. A result like this can have large implication on the security of the PUF-based key generation. If multiple or all counters fail within a batch, the entropy is greatly reduced and weak keys are generated. This important phenomenon will be discussed in more detail in Section 7.5.

![Figure 7.5: One board failed in counting correctly.](image)

### 7.4 Efficient Implementation Techniques

Implementing a large grid of ROs in an efficient and reliable way on FPGAs is a non-trivial task. In this section we introduce an optimized design strategy for implementing ROs on FPGAs with a Spartan-6 like architecture. Three different designs are introduced, each with different optimizations. These designs are then compared and analyzed with respect to the aforementioned counter failures.

The RO structure defined in the reference implementation of PUFKY uses half a slice per RO. However, there is room for improvement since in such a design a lot of the available resources within an LUT are unused. In particular, a single LUT of a Spartan-6 is composed of two 5-input LUTs and one multiplexer as can be seen in Figure 2.1(b). If a logical 1 is applied to the last multiplexer stage, the two output pins are completely independent and do not share any resources. Therefore, the delay characteristics of the two 5-to-1 LUTs are also independent. Thus, both output pins can be used to construct the RO. This way two RO elements (i.e., inverter, AND gate, or buffer) can be instantiated within a single LUT. This idea is the main reason why our design achieves an area reduction of up to 50%. In addition to the use of both output pins, we also made optimizations in regard to the routing, which is important for RO PUFs on FPGAs. A total of three different designs with different optimization goals and tradeoffs are tested.

- **0Local** places all components of an RO into a single slice and only uses local routing resources, similar to the design proposed by Merli et al. [MSE10].
7.4. Efficient Implementation Techniques

- **OSingle** places half of its components into a slice of horizontally neighboring CLBs. In contrast to **OLocal**, this design uses single wires instead of local wires to form the RO loop. This routing aims to slow down the design with minimal area increase.

- **OSingle** slow doubles the number of delay elements (LUTs) per RO and uses neighboring CLBs that are connected via single wires. The goal of this design is to drastically reduce the oscillation frequency.

![Figure 7.6: Visualization of the proposed RO structure using both LUT output pins. Based on the used design, either one (i.e., **OSingle** slow) or two ROs (i.e., **OLocal** or **OSingle**) are constructed from the given resources.](image)

![Figure 7.7: The chained structure of **OSingle** and **OSingle** slow to build the RO grid.](image)

The new designs **OLocal** and **OSingle** utilize just two LUTs and two latches to form two ROs, while **OSingle** slow consumes the same resources to form a single RO. Figure 7.6 visualizes the instantiation of the ROs. The dedicated routing ensures that there is no overlap between wires from vertically adjacent ROs so that no crosstalk effect is possible, and the vertically adjacent ROs can be used in a batch. As noted before, **OLocal** and **OSingle** differ only in the placement of their elements. **OSingle** has moved half of its components to a horizontal neighbored CLB as depicted in Figure 7.7. The idea behind this design was to use the longer routing and hence additional switching stages in the FPGA to slow down the design without using additional LUTs and latches. For a full resource utilization of a CLB, the single wires in both horizontal directions are used which results in a chain of ROs. Please note that a CLB consists of two slices and due to the shifted structure of the **OSingle** design, only one slice of the right and leftmost CLB is used as an RO. Hence, in each chain, two slices remain unused. Furthermore, in a Spartan-6 (XC6SLX45-2CSG324) FPGA a large routing line goes through the grid of CLBs.
after 2, 5, or 6 CLBs. If these routing lines are unused, this is not a problem as long as only ROs that cross the same routing lines are used to form a batch. However, if there are signals routed through these lines, there might be potential crosstalk effects. To be conservative, we chose to stop each RO chain before such routing lines even though it increased the area overhead by 16 CLBs. Therefore the number of used CLBs is higher for the OSingle although the same number of slices as in OLocal are used. The resulting resource consumption and RO frequencies of the different designs for a 64 × 16 RO grid can be found in Table 7.1.

Table 7.1: A resource and frequency comparison of the given design structures to build a 64 × 16 RO grid. Note that the given mean frequencies are based on the active time-period of the counters which highly depends on the reference oscillator. The active time of a representative board was measured by an oscilloscope and set to 61 µs for the calculations.

<table>
<thead>
<tr>
<th>Design</th>
<th>RO / Slice</th>
<th>total Slices</th>
<th>used CLBs</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference</td>
<td>2</td>
<td>512</td>
<td>256</td>
<td>306.74 MHz</td>
</tr>
<tr>
<td>OLocal</td>
<td>4</td>
<td>256</td>
<td>128</td>
<td>433.96 MHz</td>
</tr>
<tr>
<td>OSingle</td>
<td>4</td>
<td>256</td>
<td>160</td>
<td>348.49 MHz</td>
</tr>
<tr>
<td>OSingle_slow</td>
<td>2</td>
<td>512</td>
<td>320</td>
<td>184.46 MHz</td>
</tr>
</tbody>
</table>

7.5 Evaluation

In the following, the different designs are analyzed in regard to their uniqueness, reliability, and counter failures using 22 Digilent Atlys boards equipped with Spartan-6 (XC6SLX45-2CSG324) FPGAs. For each of the three designs three different placements are used to study its impact:

- P1 does not have any security margin around the RO grid so that the synthesizer can do its best to place the multiplexers and counters.
- P2 has a security margin of 6 slices set around the RO grid.
- P3 is the extreme case where the RO grid and counter multiplexer combination are constrained to be placed in opposite corners of the FPGA.

For the proposed RO structures we calculated the reliability and uniqueness of the Lehmer-Gray output under nominal conditions as done in the PUFKY paper [MHV12]. The analysis results can be seen in Figure 7.8. Note that we excluded the boards with counter failures from this evaluation. With respect to the reliability and uniqueness, the RO structure does not seem to have a significant impact. We assume that the slight variations result from the small number of devices under test. As noted before, we observed that for some designs on some boards the counters failed to count. In this context we define a counter as failed when the number of sampled oscillations, i.e., the counter value after the evaluation phase, has not reached a defined threshold. We set this threshold in our experiments to half of the average frequency of each design. Please note that most counter failures resulted in a value of 1 or 2. For each board,
design, and placement combination, we measured the RO grid 100 times. The results of this analysis can be found in Table 7.2.

As expected, the OLocal implementation running with the highest frequency (433.96 MHz) had the most counter failures of all designs. On some boards some counters fail for position \( P1 \) while for others all counters work correctly. When the counters are moved further away in \( P2 \), the number of counter failures increases drastically. When the counters are placed in the opposite corner in \( P3 \) nearly all counters fail. For the OSingle implementation and position \( P1 \), not a single failure was observed. Hence, the smaller frequency of 348.49 MHz had the intended effect of a reduced counter failure rate. However, the trend that the counter failures increase if you place the counter further away from the RO grid also holds for this design: counter failures appeared for \( P2 \), and failures increased for \( P3 \). In the OSingle slow design the RO frequency is roughly halved compared to the OSingle implementation with a frequency of 184.46 MHz. Therefore, one expects no counter failures for this design. It works as expected on most boards, and no counter failures were observed for positions \( P2 \) and \( P3 \). However, on four boards we observe counter failures for position \( P1 \) with a strange failure behavior. For board 15, multiple batches failed, which means that for certain challenges all counters refuse to work as illustrated in Figure 7.9(a). Additionally, one outlier of the measurement is a counter value that is about 50% faster than the average. But this behavior could not be observed on each run. In other runs the board worked like expected which is illustrated in Figure 7.9(b).

Our results show that the problem of counter failures clearly depends on the used RO design and their oscillation frequency as well as on the placement of the counters themselves and the surrounding logic. But the most important observation is that process variations also have a strong impact that should not be underestimated. The best example for this is the OSingle slow design for position \( P1 \). Since the RO frequency of this design is only half of that of OSingle, one expects this design to be robust. However, in four boards nevertheless counters failed in position \( P1 \), while for position \( P3 \) no counters failed. The general conclusion to this experiment is that one can increase the reliability of RO PUFs with slower ROs, but it does
Table 7.2: Summarized counter failures. At maximum, the counter for all 1024 ROs can fail.

<table>
<thead>
<tr>
<th>Identifier (ID)</th>
<th>OLocal $P_1$</th>
<th>OLocal $P_2$</th>
<th>OLocal $P_3$</th>
<th>OSingle $P_1$</th>
<th>OSingle $P_2$</th>
<th>OSingle $P_3$</th>
<th>OSingle_slow $P_1$</th>
<th>OSingle_slow $P_2$</th>
<th>OSingle_slow $P_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>18-19</td>
<td>951-959</td>
<td>0</td>
<td>0</td>
<td>465-480</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>10-16</td>
<td>958-962</td>
<td>0</td>
<td>0</td>
<td>420-433</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>3</td>
<td>856-867</td>
<td>0</td>
<td>0</td>
<td>235-245</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>9-10</td>
<td>951-956</td>
<td>0</td>
<td>0</td>
<td>410-422</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>6-7</td>
<td>880-896</td>
<td>0</td>
<td>0</td>
<td>274-291</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>12-15</td>
<td>103-105</td>
<td>1023-1024</td>
<td>0</td>
<td>6-7</td>
<td>914-925</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>21-22</td>
<td>969-971</td>
<td>0</td>
<td>0</td>
<td>502-523</td>
<td>1003-1024</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0-5</td>
<td>77-81</td>
<td>1019-1020</td>
<td>0</td>
<td>4</td>
<td>851-857</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>11-13</td>
<td>101-105</td>
<td>1024</td>
<td>0</td>
<td>5</td>
<td>912-917</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>6-7</td>
<td>83-88</td>
<td>1021-1022</td>
<td>0</td>
<td>4</td>
<td>873-882</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>1-2</td>
<td>78-82</td>
<td>1017</td>
<td>0</td>
<td>3</td>
<td>831-839</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>10-11</td>
<td>946-953</td>
<td>0</td>
<td>0</td>
<td>404-420</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>5</td>
<td>901-910</td>
<td>0</td>
<td>0</td>
<td>284-303</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>8</td>
<td>91-94</td>
<td>1022-1023</td>
<td>0</td>
<td>3-5</td>
<td>895-898</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>42</td>
<td>1000-1002</td>
<td>0</td>
<td>0</td>
<td>649-663</td>
<td>0-143</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>15-17</td>
<td>112-113</td>
<td>1024</td>
<td>0</td>
<td>11</td>
<td>943-947</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>5-6</td>
<td>94-96</td>
<td>1023</td>
<td>0</td>
<td>4-5</td>
<td>879-887</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>7-8</td>
<td>877-889</td>
<td>0</td>
<td>0</td>
<td>238-256</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>5</td>
<td>104-106</td>
<td>1024</td>
<td>0</td>
<td>7-8</td>
<td>905-911</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>2</td>
<td>757-767</td>
<td>0</td>
<td>0</td>
<td>156-163</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>9-12</td>
<td>108-111</td>
<td>1024</td>
<td>0</td>
<td>7</td>
<td>924-929</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>8-9</td>
<td>93-97</td>
<td>1023</td>
<td>0</td>
<td>5-8</td>
<td>900-905</td>
<td>0-320</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

not give any guarantees on the design functionality. Hence, without novel methods to test and simulate the counter behavior of RO PUFs on FPGAs, only an extensive large scale analysis among various environmental conditions could guarantee its correct functioning.

### 7.6 Conclusion

RO PUFs were proposed over a decade ago and many papers about this topic have been published. However, a lot of open research questions remain that we summarize briefly in this section with a focus on the new problems that arose as part of our conducted experiments. We have separated the open research questions according to the three main PUF properties: reliability, uniqueness, and implementation costs.

**Reliability:** Usually, the reliability of a PUF is either analyzed using experiments directly at the bit-level, or a model based on the frequency changes of ROs in different environmental conditions is used to determine the reliability via simulations. However, such analyses do not take into consideration the counter failure problem that we observed in our experiments. These counter failures are considerably different compared to changes in the RO frequency and have
big security implications. We argue that the problem of counter failures is maybe the hardest research problem to solve for FPGA-based RO PUFs: usually, circuit-level simulations across a variety of process and environmental corners would be conducted to understand the source of the failures and to determine parameters that guarantee a correct functioning of the counters. However, due to the fact that accurate circuit level models of an FPGA are not publicly available, this is not possible for FPGA-based RO PUFs. Hence, the question of how to build reliable and efficient RO PUFs without counter failures is an important but difficult open research problem.

**Uniqueness:** In this chapter we analyzed the dominating source of structural bias and showed how the impact of this structural bias can be reduced greatly by careful routing as well as sorting of the ROs. It is also important to note that our analysis shows that RO PUFs not only have a structural bias in the mean but also in the variance. This has not been considered in this context before, but it clearly needs to be taken into account when determining the entropy of a RO PUF design such as PUFKY. Furthermore, our results with the glitch core once more show that other IP cores can have great impact on the entropy of the RO PUF. How to determine this impact and how to make sure that other IP cores do not reduce the entropy of the RO PUF is an interesting and important open research question. One problem in analyzing the entropy of a PUF design is that to get meaningful data, a large-scale analysis with many boards is needed. Even the largest conducted experiment, which consisted of 100 boards [MCMS10], is not really enough to accurately determine the entropy. In general, how to determine the exact entropy of a RO PUF response is another important open research problem.

**Implementation Costs:** As PUFKY showed, to derive a cryptographic key from a RO PUF, a large grid of ROs is needed. Hence, techniques to reduce the RO area are interesting. Furthermore, we showed how the area of ROs can be greatly reduced by making full use of the available resources on a Spartan-6. Similar optimizations are likely possible for other FPGA architectures as well. However, an area-optimized RO design that suffers from counter failures is not helpful. The underlying research problem is therefore not to construct small RO PUFs, but rather to construct small RO PUFs that do not suffer from counter failures.

The first step should be an in-depth analysis of the counter failure problem. Following this, new designs and design strategies need to be developed that can guarantee the proper functioning of the counters among all process corners and environmental conditions.
Chapter 8

Large-Scale Analysis of Oscillation-Based PUFs

PUFs have gained a lot of research attention in recent years with many different
PUFs having been proposed. Several of these proposals were aimed specifically at
FPGA implementations. However, often these PUFs were evaluated and imple-
mented for different (and often old) FPGA families with different metrics. Missing
implementation details in many papers further hamper a fair analysis, as small de-
tails such as the exact routing can have significant impact on the PUF performance.
In this chapter we aim to overcome these problems by providing a fair comparison of
some of the most promising Weak PUFs for FPGAs, the classic RO PUF, the Loop
PUF and the TERO PUF.

Contents of this Chapter

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1</td>
<td>Introduction</td>
<td>107</td>
</tr>
<tr>
<td>8.2</td>
<td>Setup</td>
<td>108</td>
</tr>
<tr>
<td>8.3</td>
<td>Response Generation</td>
<td>108</td>
</tr>
<tr>
<td>8.4</td>
<td>Implementation and Analysis</td>
<td>109</td>
</tr>
<tr>
<td>8.5</td>
<td>Comparison of the PUF Constructs</td>
<td>111</td>
</tr>
<tr>
<td>8.6</td>
<td>Conclusion</td>
<td>118</td>
</tr>
</tbody>
</table>

8.1 Introduction

Secure key generation and storage are challenging problems in embedded security applications. In recent years, Physical Unclonable Functions (PUFs) have gained increasing attention as a new way to generate and store cryptographic keys. The most popular PUF used for key generation on FPGAs is the RO PUF [GCvDD02], which can be realized efficiently on FPGAs.

Recently, a new PUF construct called TERO PUF was introduced which is also easy to implement on FPGAs and seems to have good properties [BNCF14]. Similarly, the Loop PUF is another promising PUF that has been proposed and can also be implemented on FPGAs [CDGB12]. However, a fair comparison of these PUF constructions has not been performed yet. So far these PUF constructs have been implemented on different FPGA technologies under different environmental conditions and analyzed with different metrics.

Furthermore, the originally proposed Loop PUF is a so-called Strong PUF as opposed to a Weak PUF, which means that it has an exponential challenge space. While this is great for challenge-and-response protocols, for security reasons this can be problematic since it leaves the
Chapter 8. Large-Scale Analysis of Oscillation-Based PUFs

PUF vulnerable to machine learning attacks, and the generated responses are likely correlated. How problematic it can be to use a Strong PUF for key generation was, for example, shown in [BuG15], where a k-sum PUF could be modeled by only using the helper data generated during key generation. Hence, a Weak PUF is the more conservative choice for PUF-based key generation. In this chapter we show how the Loop PUF can be used as a Weak PUF in a way that it cannot be modeled using machine learning while remaining easy to implement. The main contribution is an in-depth and fair analysis of the three PUFs (RO, TERO, and Loop) using a large-scale measurement setup consisting of 100 modern Xilinx Artix-7 FPGAs (XC7A35T-1CPG236C) and a precisely controllable climate chamber (CTS C -40/100). All PUFs were carefully implemented and area-optimized with the same design goals and techniques, and the same fair metrics were applied to the PUF construction.

8.2 Setup

The setup we built consists of a laptop, 100 Digilent BASYS-3 developer boards, the CTS C -40/100 climate chamber, USB hubs, an external power supply, and self designed PCBs with an ATxmega16A4U to build a programmable power supply in order to individually power the BASYS-3 boards. Each BASYS-3 board is equipped with a Xilinx Artix-7 (XC7A35T-1CPG236C) FPGA. The boards are cubically stacked to fit into our climate chamber together with the USB hubs and the power supply boards. All BASYS-3 boards are connected to the USB hubs so that just a single USB cable is connected to the Laptop. The power supply boards are programmable via USB and provide external power to each BASYS-3. The Laptop is further connected via Ethernet to the climate chamber and is able to set and get the chamber temperature. On top of this setup we developed some scripts to fully automate the PUF measurements.

8.3 Response Generation

The three PUF types mentioned (RO, Loop, and TERO) make use of an asynchronous counter for the output. Furthermore, the counter values need to be post-processed to generate the actual binary response bits. In this chapter we use three different methods. The first is the classical approach used by the RO PUF proposed in [GCvDD02], which compares two counter values, and the response bit is simply the sign of the counter difference. In the following, this approach is referred to as 2Comp. If a counter value is used more than once, this again leads to shared entropy usage that could be exploited by an attacker [RSS+10, GTS15], but if it is only used once, then only one response bit is generated per counter pair. Therefore, to increase the extracted entropy, the exact sorting of \( n \) counter values or frequencies is used as a response [YQ10] to maximize the extracted response entropy without reusing entropy. Again, we follow the PUFKY [MHV12] approach, which uses this idea paired with Lehmer/Gray encoding of the ordering to generate the PUF response bits.

In the first step the sorting is performed by Lehmer encoding. It sequentially checks for \( n \) frequencies \( f_i \), how many frequencies with lower indices are smaller than the currently selected frequency, i.e., \( s_i = |\{ f_j | f_j < f_i \land j < i \}| \). This way a sorting of \( n \) elements can be uniquely represented with \( n - 1 \) numbers \( s_2, \ldots, s_n \). These numbers \( s_i \) are Gray-encoded, which guarantees
that consecutive numbers only differ by one bit, which is helpful to decrease the bit unreliability in case the order varies.

Note that based on the encoding, some of the response bits are biased. The maximum entropy that can be extracted by sorting a batch of \( n \) frequency values is \( \log_2(n!) \). In this chapter we will use a batch size of \( n = 16 \) as in PUFKY. After Lehmer/Gray encoding, 49 response bits are generated from the 16 frequency values within one batch that have a maximum entropy of \( \log_2(n!) \approx 44 \). For more detail, see [MHV12]. A third approach was used in [BNCF14], which uses some counter bits directly as the PUF response. In this case the raw counter bits have to be carefully checked since the reliability as well as the uniqueness strongly vary for each bit position.

8.4 Implementation and Analysis

In this section we provide details of our implementations and give reasons for design decisions. We only implemented the PUFs and their counters on the FPGA while all post-processing such as Lehmer/Gray encoding was performed in software.

In contrast to the description in Section 2.1.1, an Artix-7 holds just two slice types, SliceM and SliceL. It is important to note that the layout of some CLBs is different even if they are of the same type (holding the same slice types), because the location of the switch matrix changes. In our implementations we stick with the same CLB types to generate more PUFs of the same layout within one design and ensure that only PUFs with identical layouts are used for the response generation. We also make use of local routes to form the PUF instances.

8.4.1 Ring Oscillator PUF

The RO PUF is one of the PUF constructions hat have been analyzed and referred to the most. Hence, the RO PUF has been implemented in several different ways. In Chapter 7, the most area-efficient RO implementation for modern Xilinx FPGAs was presented that uses the the 6-to-2 LUT components provided by Xilinx FPGAs.

We therefore adapted this implementation technique and made use of the 6-to-2 LUT components and latches as well to build an efficient RO structure with just local wires on the Artix-7. As noted in Chapter 7, the concept requiring the fewest resources (\( \text{OLocal} \)) results in an unreliable counter, i.e., for some PUF instances the counter did not correctly measure the frequency. We observed the same behavior on the Artix-7 boards when implementing a single RO using \( \frac{1}{4} \) slices. We therefore doubled the resources spent for an RO, as given in Figure 8.1, to decrease the frequency of the RO and thereby significantly decrease the observed counter failures, which in turn resulted in an increased reliability and uniqueness.

For the RO PUF we instantiated 16 \( \times \) 80 ROs in one design\(^1\). Only ROs located in the same LUT, slice, and CLB type should be part of a batch because otherwise the layouts of the ROs will not be identical and hence the frequencies will be biased. In our case this means that we have four different types of ROs, as we always use the same CLB type and implement one RO in half a slice.

\(^1\)We chose 16 \( \times \) 80 as this is the magnitude of PUF responses needed to generate a 128-bit secret key with a secure fuzzy extractor [MHV12].
8.4.2 Loop PUF

The Loop PUF compares the frequency differences induced by the delay stages of the loop for different challenges, which requires a fixed layout for each delay stage. Note that the internal routing and layout of a delay element do not need to be identical, i.e., the delay elements are allowed to be significantly different for their challenge bit. What important is that for the same challenge word the layout of different delay stages is identical. Similarly, the routing between stages does not need to be identical, which significantly decreases the implementation complexity of the Loop PUF on an FPGA.

The Artix-7 structure allows several techniques to implement these delay elements. One is to use just the LUT internal wires, which means that the paths within a delay element only differ inside a LUT as shown in Figure 8.2(a). This idea was first used in [MKD10] to implement the Programmable Delay Lines (PDLs) of an Arbiter-like PUF construction. We further call this profile Loop-PDL. Our second approach uses two different paths built by local routes. Hence one LUT, based on a challenge bit, routes the input signal to one of these paths and a second LUT is used to combine both paths again to yield a single data signal. This profile is depicted in Figure 8.2(b) and will be referred to as Loop-Wire. The LUTs of an Artix-7 are followed by a latch. In case of Loop-Wire (and Loop-PDL) the latches are not used. Including a latch into the signal path increases the variance of the signal propagation delay since the transistors of the latch are also affected by manufacturer process variations. Hence, we built a third profile called Loop-Latch depicted in Figure 8.2(c), by pushing the signal through transparent latches. The routes used to build a delay element in Loop-Wire and Loop-Latch are just local routes, and we fixed these for all PUF instances. Loop-Latch and Loop-Wire consume the same resources on the slice level, i.e., 1/2 slices, while Loop-PDL is instantiated in 1/4 slices.

In the second step the delay elements have to be combined to a loop. To build an area-efficient design, the requirements of layout-equivalent delay stages and the FPGA structure basically allow for two approaches. First, a big loop with stages of different delay element types is built (this again is due to the layout of the CLB and slices within the Artix-7). We followed that strategy and built a big loop of \( n = 16 \) stages with \( m = 4 \) delay elements (with different layouts) per stage.

We also built smaller loops consisting of only \( n = 16 \) stages and \( m = 1 \) delay element. To use the full resources available in a CLB, four loops are placed in an interleaved manner, each with a different delay stage layout. The interleaved approach of shorter loops resulted in considerably better reliability while meeting the same area requirements. Hence, we chose this interleaved method for our three final designs, each consisting of 80 loops with 16 stages each.
8.5. Comparison of the PUF Constructs

In this section we are going to compare the different PUF types with respect to their unreliability and uniqueness. It is important to note that the way the responses are generated strongly influences the unreliability and uniqueness values. In this analysis we opted to use the Lehmer/Gray encoding as described in Section 8.3 for response generation to compare all
three PUF types. For this we sorted the responses in 80 batches, each consisting of 16 responses from identical PUFs (i.e., with the same routing and slice types) located next to each other. Then Lehmer/Gray encoding was applied to each of these batches to generate $80 \times 49$ response bits. To compute the unreliability, each PUF instance was measured under nominal temperature ($22^\circ\text{C}$) as a reference value and again for different temperatures. The unreliability is simply given as bit unreliability, which is defined as the mean HD in percent per bit between the different measurements of the same PUF instance. To compute the uniqueness, the average HD of the responses between each of the 100 different PUF instances (i.e., FPGA boards) was computed.

8.5.1 Reliability

The first important aspect that needs to be addressed when different counter-based PUF constructions are compared is the run time of the PUFs. This aspect is unfortunately often neglected in PUF papers but, as we will see, is important. The oscillation frequency is determined by the implementation, which is a constant for all instances, the instance-dependent process variations, environmental noise (e.g., due to temperature or supply voltage differences), and random temporal noise (e.g., temporal fluctuations in the power supply).

The latter type of noise is not constant and can be averaged out by repeatedly measuring the PUFs’ entropy source. This is well known, and averaging (also called temporal majority voting) has been widely used as a mean to increase reliability [DGSS15]. For the RO and Loop PUF, increasing the evaluation time essentially increments the number of times the entropy source is evaluated and hence decreases temporal noise.

Similar to the RO and Loop PUFs, averaging can greatly increase the reliability of the TERO PUF. However, since a TERO PUF only oscillates for a short period of time before ending up in a stable state, averaging was not done by increasing the run time but by repeatedly evaluating the TERO PUF. In [BNCF14], a total of $2^{18}$ PUF evaluations were averaged to generate the response. However, the run time of the TERO PUF also affects reliability. Most of the TERO circuits settle in a stable state after a short time, and ideally the run time should be chosen in such a way that all TERO circuits have settled. But some of the circuits are balanced so that they keep oscillating for quite a long time. Setting the run time to the maximum oscillating time would drastically increase the run time of the PUF while only marginally increasing the reliability. We performed a reliability experiment on TERO by providing increasing time for settling while averaging an adequate number of $2^{12}$ PUF evaluations. We determined that for our implementation a settle time of $2^5$ clock cycles (at 100 MHz) results in a good time-reliability ratio.

We performed an experiment in which we changed the run time from $2^5$ up to $2^{20}$ clock cycles (320 ns to 10.49 ms) and plotted the resulting uniqueness and reliability values in Figure 8.4. As one can see in Figure 8.4(a), the TERO PUF has near-optimal uniqueness from the beginning. The RO PUF, on the other hand, has a lower uniqueness for short run times. This is due to the simple fact that if the run time is too short, the frequency of the RO cannot be measured with sufficient accuracy. The intensity of this phenomenon is greatly increased for the Loop PUF. The frequency of the Loop PUF is considerably smaller than the frequency of the RO PUF, and hence it takes many more clock cycles until the frequency can be determined with high enough precision. Hence, to get a good uniqueness, at least $2^{16}$ clock cycles are required for the Loop
8.5. Comparison of the PUF Constructs

Figure 8.4: In (a) the average uniqueness for a 49-bit Lehmer/Gray word is shown for the three PUF constructs with different run times. For an ideal PUF, this uniqueness should be 23.11, which the TERO PUF achieves even with a run time of $2^5$ clock cycles. In (b) the average unreliability of the PUF construct for Lehmer/Gray words in % per bit is shown for different run times.

PUF compared to $2^5$ for the TERO and $2^8$ for the RO PUF. In general, the uniqueness values of the RO and Loop PUF never completely reach that of the TERO PUF.

Figure 8.4(b) shows the unreliability of the PUF constructs for different run times. For the RO and Loop PUFs, the unreliability first increases up to a certain point before decreasing again. This is due to the fact that a low uniqueness usually results in a high reliability and hence when the uniqueness increases, the unreliability also increases at first. Once a high uniqueness is reached, the averaging effect of the run time dominates, the noise decreases and the unreliability decreases as well. The figures also show that the RO construction shows the lowest unreliability of the three designs for longer run times. TERO, on the other hand, has a lower unreliability for short run times of up to about $2^{11}$ clock cycles.

The Loop PUF depicts the worst reliability results in this analysis, but it is extremely interesting to see that in general the unreliability of the Loop PUF is high while the temperature seems to hardly have any impact.

For further analysis we fixed the run time to $2^{17}$ clock cycles (1.31 ms) per measurement since longer run times resulted just in slight decreases in unreliability for each PUF construction, and the uniqueness of all PUF types reached their maximum value. We measured our PUF constructs for the fixed run time at different temperatures (-22°C, 0°C, $2 \times 22°C$, 44°C) to get a better understanding of the different PUFs’ stability in the presence of temperature changes. The nominal temperature is defined as 22°C, and each measurement is compared with the nominal results. The values given in Table 8.1 are the mean bit unreliability values in % per bit for the noted temperatures. As one can see, the RO PUF produces the most reliable results. Surprisingly, the Loop PUF is the most unreliable but at the same time also
the most temperature-independent design. This phenomenon will be discussed in greater detail in Section 8.5.3.

Table 8.1: Unreliability of the PUF constructions for Lehmer/Gray encoding and 2Comp for different temperatures in % per bit.

<table>
<thead>
<tr>
<th>PUF class</th>
<th>Bit unreliability (Lehmer/Gray) [%]</th>
<th>Bit unreliability (2Comp) [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-22°C</td>
<td>0°C</td>
</tr>
<tr>
<td>RO</td>
<td>7.30</td>
<td>4.64</td>
</tr>
<tr>
<td>TERO</td>
<td>12.78</td>
<td>8.71</td>
</tr>
<tr>
<td>Loop-Latch</td>
<td>12.91</td>
<td>11.74</td>
</tr>
<tr>
<td>Loop-Wire</td>
<td>20.49</td>
<td>19.58</td>
</tr>
<tr>
<td>Loop-PDL</td>
<td>16.02</td>
<td>15.62</td>
</tr>
</tbody>
</table>

Table 8.2: Uniqueness and bias of the tested PUF constructions at nominal temperature. The indicated bias values distinguished between “Batch”, which is the bias between all PUF instances grouped into a batch and “Neighbors”, which takes just the bias of physically neighboring PUF instances into account.

<table>
<thead>
<tr>
<th>PUF class</th>
<th>Mean Uniqueness</th>
<th>Mean Bias</th>
<th>Max Bias</th>
<th>Mean Bias/2Comp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Lehmer/Gray bits (from 49)</td>
<td>(Batch) %</td>
<td>(Batch) %</td>
<td>(Neighbors) %</td>
</tr>
<tr>
<td>Glitch off</td>
<td>RO</td>
<td>22.97</td>
<td>3.08</td>
<td>9.24</td>
</tr>
<tr>
<td></td>
<td>TERO</td>
<td>23.11</td>
<td>0.48</td>
<td>1.63</td>
</tr>
<tr>
<td></td>
<td>Loop-Latch</td>
<td>22.98</td>
<td>4.10</td>
<td>11.31</td>
</tr>
<tr>
<td></td>
<td>Loop-Wire</td>
<td>22.42</td>
<td>8.26</td>
<td>21.1</td>
</tr>
<tr>
<td></td>
<td>Loop-PDL</td>
<td>23.05</td>
<td>5.4</td>
<td>17.7</td>
</tr>
<tr>
<td>Glitch on</td>
<td>RO</td>
<td>22.2</td>
<td>13.2</td>
<td>31.6</td>
</tr>
<tr>
<td></td>
<td>TERO</td>
<td>23.12</td>
<td>0.6</td>
<td>1.7</td>
</tr>
<tr>
<td></td>
<td>Loop-Latch</td>
<td>18.01</td>
<td>27.2</td>
<td>47.2</td>
</tr>
<tr>
<td></td>
<td>Loop-Wire</td>
<td>14.83</td>
<td>34.4</td>
<td>49.8</td>
</tr>
<tr>
<td></td>
<td>Loop-PDL</td>
<td>20.68</td>
<td>15.6</td>
<td>36.8</td>
</tr>
</tbody>
</table>

8.5.2 Uniqueness

Other than reliability, uniqueness is the most important property of a Weak PUF as it indicates the entropy of the generated responses. Figure 8.4(a) suggests that all PUF designs achieve an equally near-perfect uniqueness. In this section we therefore would like to examine the uniqueness of the PUFs in more detail. As mentioned, for the Lehmer/Gray encoding the PUF responses are sorted into 80 batches \{f_1, \ldots, f_{16}\} each of size 16. Ideally, the probability that
8.5. Comparison of the PUF Construct

$f_i$ is larger than $f_j$ should be 0.5 for all $i, j \in \{1, \ldots, 16\}$. This probability $P(f_i > f_j)$ is depicted on the left side in Figure 8.5. We call the derivation of this probability “bias”. It is basically the same as the bias if one computed the PUF responses by comparing two PUF responses (noted in Section 8.3 as 2Comp). We computed this bias for every $i$ and $j$ for each PUF construction with $\text{mean}(0.5 - P(f_i > f_j))$. Hence, the maximum bias can be 50%. As one can see in Table 8.2 and Figure 8.5, the TERO PUF has nearly no bias. In comparison to this, the RO PUF clearly shows a much larger bias that depends on the position of the PUFs. Since all 16 PUFs of one batch are located in a row, the larger the difference between $i$ and $j$, the further away they are located on the FPGA. This directly results in a higher bias which is inline with earlier findings that showed that the frequency of an RO depends on its location on the FPGA [uBG16, MCMS10]. The Loop PUF shows the strongest bias of the three PUFs, as can be seen in Figure 8.5(e).

One problem with PUFs on FPGAs is that the surrounding logic can influence the PUF’s behavior. This is especially problematic for PUF IP cores since they need to be re-evaluated based on the design they are integrated into. In order to test the vulnerability of the PUFs to surrounding logic, we repeated the experiment proposed in Section 7.3.2, in which a so-called “glitch core” generating a large amount of switching activity is placed in the top right corner of the FPGA. On the left side of Figure 8.5 this glitch core is inactive while on the right side it is active during a PUF evaluation. As can be seen, this glitch core drastically increases the bias in the RO and Loop PUF and hence decreases the entropy. In comparison, the uniqueness of the TERO PUF seems not to be influenced by the glitch core. It should be noted though that the switching activity of the glitch core is extreme and should be considered as a worst-case scenario. To give the reader an intuition of the amount of switching activity of the core, during testing the BASYS-3 board was powered via an USB cable, and it turned off due to the increased power consumption of the glitch core.

Table 8.2 summarizes the uniqueness results with an active and inactive glitch core. Out of the three PUF constructions examined, the TERO PUF showed the best uniqueness properties. Especially in the presence of the glitch core (see Fig. 8.5) its results were considerably better than those of RO and Loop PUFs. The results also show that looking only at the uniqueness after Lehmer/Gray encoding might suggest a much higher entropy than what actually exists in the PUF (compare Figure 8.5 and Table 8.2). One reason for this is the encoding into blocks of 49 bits. To precisely measure the uniqueness and entropy of these batches, a much larger number of batches is required than the $80 \times 100$ we used in our experiment. How to determine the actual entropy within a sample size in the magnitude of our measurement setup, is an interesting and open research problem since our setup of 100 FPGAs is actually one of the largest in the community.

8.5.3 Analysis of the Loop PUF

Typically, an environmental noise has a large impact on the reliability of a PUF, in particular noise due to temperature variations. However, as can be seen in Table 8.1 and Figure 8.4(b), the Loop PUF has a large temporal random noise while it is not affected much by temperature variations. The reason why the Loop PUF has such a large temporal noise is its architecture. For the response generation, the Loop PUF compares the frequencies $f_i$ and $f_j$. The loop configurations of the measured frequencies differ just in stage $i$ and stage $j$. 

115
Figure 8.5: The bias analysis for glitch off and on.
8.5. Comparison of the PUF Constructs

Hence, by comparing two frequencies measured under the same environmental conditions, the environmental noise within all stages that are not \( i \) or \( j \) and the environmental noise independent from a challenge bit cancel each other out. Furthermore, the process variation induced differences also cancel each other out. Hence, the frequency differences contains (i) the challenge-dependent process variations within stages \( i \) and \( j \), (ii) the environmental noise within stages \( i \) and \( j \), and (iii) the temporal random noise for all parts of the Loop PUF.

Compared to the RO PUF, in which all parts of the RO have process variations that add uniqueness as well as environmental noise, in a Loop PUF only a small part of the loop adds uniqueness and suffers from environmental noise, while every part of the loop contributes to temporal noise. This explains why the reliability of the Loop PUF is much smaller than that of the RO PUF due to the magnitudes larger temporal random noise in comparison to the PUF’s uniqueness. This effect is amplified for larger loops, which provides a reason for the decreased reliability of our \( m = 16 \) and \( n = 4 \) Loop PUF profile.

We considered different designs for the Loop PUF to identify entropy-rich components on the FPGA. Hence, we compared the reliability and uniqueness of Loop-Latch, Loop-Wire and Loop-PDL. Table 8.2 indicates that the mean uniqueness values of the designs are quite similar. But looking at the maximum and mean bias, one can see that the Loop-Latch performs best with the Loop-Wire performing worst. Similarly, the Loop-Latch is clearly the most reliable while Loop-Wire is the most unreliable of the three PUF profiles. Interestingly, for nominal temperature the reliability of Loop-Wire and Loop-PDL are similar, but Loop-Wire is much more vulnerable to temperature changes. Note that in Loop-Latch the local routing is similar to Loop-Wire. Hence, one can summarize that in this experiment the routing has negative impact on the PUFs as their temperature variations are larger than the introduced process variations. Furthermore, the latches seem to have a large amount of process variations, and they greatly increase the uniqueness as well as the reliability of the Loop PUF. Hence, at least for the Artix-7, including latches into a PUF design (like we also did in the RO and TERO PUF) is advisable.

8.5.4 Other Response Generation Schemes

In this chapter we focused on the Lehmer/Gray encoding scheme for response generation as it is the most efficient scheme and, e.g., is used in [MHV12]. However, in the original TERO PUF publication it was proposed to directly use two counter bits as the response, which is also fairly efficient. We also computed the uniqueness and reliability of the TERO PUF when the counter bits are used as a response. The result of this analysis can be seen in Figure 8.6(a) for the TERO PUF. Counter bit 19 achieves a uniqueness of 0.46 per bit, which is similar to Lehmer/Gray encoding \((23^{11}/49 = 0.47)\). However, the unreliability of 10.4\% – 17.6\% for counter bit 19 is already considerably higher than what occurs with Lehmer/Gray encoding, or if two PUFs are compared to each other. Hence, response generation based on comparing counter values seems to be the more efficient approach for TERO. We also repeated this analysis for the RO PUF, and the results can be seen in Figure 8.6(b). The high temperature dependency of the RO frequency makes this type of response generation worse. Some counter bits even flip with a probability of over 80\% when the temperature is changed.
Figure 8.6: Uniqueness and unreliability when using the counter bit directly as a response for each bit.

8.6 Conclusion

In this chapter we compared three Weak PUF architectures in detail. It turned out that the RO PUF shows the best reliability results while TERO clearly leads in terms of run time and performs almost independent of the surrounding logic. The Loop PUF works almost independent of temperature conditions but suffers significantly from temporal noise. It is also noteworthy that the commonly used uniqueness metric is rather misleading and does not reflect the entropy of a certain type of PUF. In general, estimating the entropy of PUFs is an interesting and open research problem.
Part IV

Conclusion
Chapter 9

Conclusion and Future Work

This chapter summarizes the results of this thesis and discusses still open and future topics related to the presented works.

Contents of this Chapter

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1</td>
<td>Conclusion</td>
<td>121</td>
</tr>
<tr>
<td>9.2</td>
<td>Future Work</td>
<td>122</td>
</tr>
</tbody>
</table>

9.1 Conclusion

In this thesis we have taken a closer look at DRP logic styles and PUF constructions for FPGAs. Addressing DRP schemes, we have shown that previous works [TV04, BGF+10, MI14, LMT+09, NBD+10, YS07, KV10, HdlTR11] do not fully succeed to transfer the concept to FPGAs, even the duplication concept on WDDL proposed in [YS07] which has been pretended by the community to be the best performing DRP scheme on FPGAs. The lack of a flawless DRP scheme motivated us to propose a logic style which avoids the known major pitfalls (glitches, early propagation, and different wire capacitances of coupled signals) and hence, further reduces the SNR. The SNR reduction achieved by GliFreD hides the leakage in the power traces and hardens a given algorithm against power analysis attacks up to a certain level but still does not fully prevent the first-order leakage. The literature notes that hiding techniques should not be used as sole countermeasures to harden circuits against SCA [MOP07]. Therefore, we have analyzed GliFreD in combination with a proper first-order resistant masking scheme. It has been shown that the signal reduction given by the power-equalization has an equivalent effect for higher-order SCA of increasing the noise level, and hence univariate SCA are demonstrated to be practically unfeasible on circuits combining proper power-equalization and masking schemes.

The implementation of a PUF class on FPGAs is a non-trivial task since the fixed structure, limited resources and predefined initial sequences harden a proper instantiation for the majority of previously proposed PUF classes or limit them to a specific device family. The Xilinx specific power-up reset [Xil13a], for instance, deletes the initial values of SRAM cells and thus hinders the application of the classical SRAM PUF concept. Nonetheless, we proposed in this work a work-around to recover some initial SRAM values to bring back the SRAM PUF concept to Xilinx FPGAs. It is worth to mention that the gained results heavily depend on the chip revision of the devices under test. The RO PUF as the most popular PUF concept on FPGAs is still not fully analyzed as we have also shown in this thesis. Especially, the lack of knowledge...
about the transistor and metal layer of commercial FPGAs makes it hard to tell which side-effects are going to influence this PUF class. We also provided implementation concepts for the RO, Loop, and TERO PUF with respect to minimal area consumption. All proposed implementations have been optimized for the Xilinx 7 Series. This enabled a fair comparison of these PUF constructions with a unified metric, the same environmental conditions, and the same underlying device technology. To perform the evaluation, we built a large scale setup of 100 Artix-7 FPGAs in order to automate the evaluation process and gain statistically significant results.

9.2 Future Work

In the following, open research topics and still open questions with focus on power-equalization schemes and PUFs on FPGAs are discussed.

9.2.1 Power-Equalization

DRP logic styles are just able to reduce the SNR and are not capable to fully remove the data dependency in the power consumption. One reason for the remaining leakage is the variation of wire capacitances and transistor switching speeds induced during the manufacturing process. Hence, a device-dependent design adaption that takes those variations into account would be an option to further push the limits of DRP and improve this concept.

The duplication concept proposed in [YS07] seems to be the best known way to balance coupled routes. To protect a cryptographic primitive with a DRP scheme that follows the duplication concept against EM analysis, an interleaved placement of the two circuits is inevitable. The limited routing resources provided by commercial FPGAs harden this task, especially for more complex design structures. In this context, an adapted FPGA structure supporting the interleaved placement and routing would be very beneficial for DRP schemes. Additionally, other device adaptions like LUTs with buffered output may help to improve the DRP concept on FPGAs. Hence, a detailed survey on beneficial FPGA adaptions may help to design future FPGA structures which are more capable for DRP schemes.

In Section 2.2.4 a comparison between promising FPGA-based DRP schemes is given. Nevertheless, the comparison does only take the resource consumption and theoretical flaws into account but does not analyze the practical achievable security level. Hence, both parameters will define the scheme efficiency, and a more detailed analysis would be helpful for hardware developers to make the right choice for their needs. The schemes given in Section 2.2.4 have been proposed over several years, tested on different device technologies, evaluated with different setups and evaluation metrics. To perform a fair comparison on the practically achievable security level, the noted parameters (device family, measurement setup and evaluation metrics) need to be unified. The adaption of the schemes to other device families is a non-trivial task since the device structure and components have changed and some of the proposed techniques may not work as expected on other devices.

As depicted in this thesis, the combination of masking and DRP schemes can result in circuits with a high resistance against SCA. Since masking schemes depend on the underlying algorithm, it would be of major interest to analyze the combination of DRP schemes with generic countermeasures like noise generation, shuffling, dynamic reconfiguration or clock ran-

Like all SCA countermeasures, DRP schemes trade resources for resistance. It has been shown that GliFreD provides, on the one side, promising security properties but also requires a lot resources, especially a high FF utilization. Further, the latency of a circuit transferred to GliFreD is also heavily increased. Thus, looking for more resource-friendly and low-latency alternatives with an equal or even higher security level can be one goal for the future.

The process variation as well as the dynamic power consumption exploited by SCA strongly vary between processing technologies. In other words, the achievable SNR reduction and hence the resistance against SCA power-equalization schemes can provide strongly rely on the underlying technology. An analysis of DRP schemes on different technologies in order to identify preferable process technologies is another still open task.

As shown in this work, DRP schemes can achieve promising results on FPGAs. Nevertheless, the achievable resistance level of DRP logic styles highly depends on the underlying hardware structure which is given by the FPGA manufacturer. In conclusion, the future of DRP schemes on FPGAs will be highly influenced by the manufacturer itself and the hardware changes they make on future device families.

9.2.2 PUFs

A general question not solved by the PUF community yet is the proper entropy estimation of a PUF class. Currently, the entropy is often estimated by analyzing the challenge and response behavior for a large set of PUF instances which is a rather inefficient process (with respect to the time and cost) for a statistically large set of instances. The bit entropy of a PUF class is important since the post-processing steps further reduce the entropy in the responses [BGV+12]. The exact entropy loss of the entropy source and the post-processing is hence an unsolved and open research question.

As known, the variances induced during the manufacturing process of a hardware device define the challenge and response behavior of a PUF class. The publicly available PUF analysis does only take samples into account which had been processed by unknown manufacturing lines or have been taken from unknown positions of the silicon wafers. The manufacturing lines, the chip positions on the wafer, and even more parameters can induce biases to the PUF classes that had not been characterized yet. Hence, further analysis on the manufacturing parameters are required to further elaborate the quality of proposed PUF classes. This effect can be seen in Chapter 6 since the results highly depend on the device revision.

Since PUFs hold secret information which is often part of a security concept, the protection of those PUF classes against physical attacks is currently just barely taken into account. To use PUFs in the field, countermeasures against physical attacks are inevitable. Further, for a device-specific key generation the resources of a PUF class as well as the post-processing steps have to be taken into account. The post-processing strongly depends on the quality of the PUF instances. PUFKY [MHV12] was published in 2012 and is the last publication which does not consider both components in a nutshell. Providing some numbers for a state-of-the-art concept which also takes the resistance against physical attacks into account would be of major interest.

Similar to DRP schemes, PUFs make use of the underlying hardware structure of the FPGA. With this in mind, it is clear to see that the future of PUFs on FPGAs with respect to the
Chapter 9. Conclusion and Future Work

quality or efficiency is highly influenced by the manufacturer itself, the process technology, and parameters that are used for future device families.
Part V

Appendix
Chapter 10

The KATAN AND/XOR Module

First-order Component Functions

\[ y^1 = f^{1,2}(a^1, b^1, c^1), (a^2, b^2, c^2)) = a^2 + b^2 c^2 + b^1 c^2 + b^2 c^1 \] (10.1)
\[ y^2 = f^{2,3}(a^2, b^2, c^2), (a^3, b^3, c^3)) = a^3 + b^2 c^3 + b^3 c^2 + b^3 c^2 \] (10.2)
\[ y^3 = f^{3,1}(a^3, b^3, c^3), (a^1, b^1, c^1)) = a^1 + b^1 c^1 + b^1 c^1 + b^3 c^1 \] (10.3)

Second-order Component Functions

\[ y^4 = f^{1,2}(a^1, b^1, c^1), (a^2, b^2, c^2)) = a^2 + b^2 c^2 + b^1 c^2 + b^2 c^1 \] (10.4)
\[ y^5 = f^{1,3}(a^1, b^1, c^1), (a^3, b^3, c^3)) = a^3 + b^3 c^3 + b^1 c^3 + b^3 c^1 \] (10.5)
\[ y^6 = f^{1,4}(a^1, b^1, c^1), (a^4, b^4, c^4)) = a^4 + b^4 c^4 + b^1 c^4 + b^4 c^1 \] (10.6)
\[ y^7 = f^{2,1}(a^2, b^2, c^2), (a^1, b^1, c^1)) = a^1 + b^1 c^1 + b^5 c^3 + b^1 c^5 \] (10.7)
\[ y^8 = f^{2,3}(a^2, b^2, c^2), (a^3, b^3, c^3)) = a^5 + b^5 c^5 + b^2 c^5 + b^5 c^2 \] (10.8)
\[ y^9 = g^{2,3}(a^2, b^2, c^2), (a^3, b^3, c^3)) = b^2 c^3 + b^2 c^2 \] (10.9)
\[ y^{10} = g^{2,4}(a^2, b^2, c^2), (a^4, b^4, c^4)) = b^4 c^4 + b^4 c^4 \] (10.10)
\[ y^{11} = g^{3,4}(a^3, b^3, c^3), (a^4, b^4, c^4)) = b^3 c^4 + b^4 c^3 \] (10.11)
\[ y^{12} = g^{3,5}(a^3, b^3, c^3), (a^5, b^5, c^5)) = b^5 c^5 + b^5 c^3 \] (10.12)
\[ y^{13} = g^{4,5}(a^4, b^4, c^4), (a^5, b^5, c^5)) = b^4 c^5 + b^5 c^4 \] (10.13)
Chapter 11

PRESENT Sbox

Decomposition

The PRESENT Sbox $S((a, b, c, d)) = (e, f, g, h):c56b90ad3ef84712$ with ANF

\[
\begin{align*}
e &= a + c + d + bc \\
f &= b + d + bd + cd + abc + abd + acd \\
g &= 1 + c + d + ab + ad + bd + abd + acd \\
h &= 1 + a + b + d + bc + abc + abd + acd
\end{align*}
\]

is decomposed to $A_3 \circ Q_{299} \circ A_2 \circ Q_{294} \circ A_1$ with

\[
\begin{align*}
e &= b \quad f = b + d \quad g = a \quad h = b + c, \\
A_2 &= 91e63b4cd5a27f08 \quad e = 1 + b \quad f = b + c \quad g = b + d \quad h = 1 + a + c, \\
A_3 &= 0145efab89cd6723 \quad e = a \quad f = c \quad g = b + c \quad h = c + d.
\end{align*}
\]

First-order Component Functions

The first stage $Q_{294}$

\[
\begin{align*}
y^1 &= f_{Q_{294}}^{2,3} ((a^2, b^2, c^2, d^2), (a^3, b^3, c^3, d^3)) = (e, f, g, h) \\
e &= a^2 + b^2d^2 + d^2b^3 + b^2d^3 \\
f &= b^2 + c^2d^2 + d^2c^3 + c^2d^3 \\
g &= c^2 \\
h &= d^2
\end{align*}
\]

\[
(11.1)
\]

\[
\begin{align*}
y^2 &= f_{Q_{294}}^{3,1} ((a^3, b^3, c^3, d^3), (a^1, b^1, c^1, d^1)) = (e, f, g, h) \\
e &= a^3 + b^3d^3 + d^3b^1 + b^3d^1 \\
f &= b^3 + c^3d^3 + d^3c^1 + c^3d^1 \\
g &= c^3 \\
h &= d^3
\end{align*}
\]

\[
(11.2)
\]

\[
\begin{align*}
y^3 &= f_{Q_{294}}^{1,2} ((a^1, b^1, c^1, d^1), (a^2, b^2, c^2, d^2)) = (e, f, g, h) \\
e &= a^1 + b^1d^1 + d^1b^2 + b^1d^2 \\
f &= b^1 + c^1d^1 + d^1c^2 + c^1d^2 \\
g &= c^1 \\
h &= d^1
\end{align*}
\]

\[
(11.3)
\]
The second stage $Q_{299}$

$$y^1 = f^{2,3}_{Q_{299}}\left((a^2, b^2, c^2, d^2), (a^3, b^3, c^3, d^3)\right) = \langle e, f, g, h \rangle$$

\begin{align*}
  e &= a^2 + (a^2 b^2 + d^2 b^3 + b^2 d^3) \\
  f &= b^2 + (a^2 d^2 + d^2 a^3 + a^2 b^3) + (c^3 d^2 + a^2 c^2 + d^2 a^3) \\
  g &= c^2 + (b^2 d^2 + d^2 b^3 + b^2 d^3) + (c^2 d^2 + d^2 c^2 + c^2 d^3)
\end{align*}

$$h = d^2$$

(11.4)

$$y^2 = f^{3,1}_{Q_{299}}\left((a^3, b^3, c^3, d^3), (a^1, b^1, c^1, d^1)\right) = \langle e, f, g, h \rangle$$

\begin{align*}
  e &= a^3 + (a^3 d^3 + d^3 a^1 + a^3 d^1) + (c^3 d^3 + d^3 c^1 + c^3 d^1) \\
  f &= b^3 + (a^3 d^3 + d^3 a^1 + a^3 d^1) + (b^3 d^2 + d^3 b^1 + b^3 d^1) + (c^3 d^3 + d^3 c^1 + c^3 d^1) \\
  g &= c^3 + (b^3 d^3 + d^3 b^1 + b^3 d^1) + (c^3 d^3 + d^3 c^1 + c^3 d^1)
\end{align*}

$$h = d^3$$

(11.5)

$$y^3 = f^{1,2}_{Q_{299}}\left((a^1, b^1, c^1, d^1), (a^2, b^2, c^2, d^2)\right) = \langle e, f, g, h \rangle$$

\begin{align*}
  e &= a^1 + (a^1 d^1 + d^1 a^2 + a^1 d^2) + (c^1 d^1 + d^1 c^2 + c^1 d^2) \\
  f &= b^1 + (a^1 d^1 + d^1 a^2 + a^1 d^2) + (b^1 d^1 + d^1 b^2 + b^1 d^2) + (c^1 d^1 + d^1 c^2 + c^1 d^2) \\
  g &= c^1 + (b^1 d^1 + d^1 b^2 + b^1 d^2) + (c^1 d^1 + d^1 c^2 + c^1 d^2)
\end{align*}

$$h = d^3$$

(11.6)

Second-order Component Functions

The first stage $Q_{294}$

$$y^1 = f^{2,3}_{Q_{294}}\left((a^2, b^2, c^2, d^2), (a^3, b^3, c^3, d^3)\right) = \langle e, f, g, h \rangle$$

\begin{align*}
  e &= a^2 + b^2 d^2 + d^2 b^3 + b^2 d^3 \\
  f &= b^2 + c^2 d^2 + d^2 c^3 + c^2 d^3 \\
  g &= c^2
\end{align*}

(11.7)

$$y^2 = f^{3,4}_{Q_{294}}\left((a^3, b^3, c^3, d^3), (a^4, b^4, c^4, d^4)\right) = \langle e, f, g, h \rangle$$

\begin{align*}
  e &= a^3 + b^3 d^3 + d^3 b^4 + b^3 d^4 \\
  f &= b^3 + c^3 d^3 + d^3 c^4 + c^3 d^4 \\
  g &= c^3
\end{align*}

(11.8)

$$y^3 = f^{4,5}_{Q_{294}}\left((a^4, b^4, c^4, d^4), (a^5, b^5, c^5, d^5)\right) = \langle e, f, g, h \rangle$$

\begin{align*}
  e &= a^4 + b^4 d^4 + d^4 b^5 + b^4 d^5 \\
  f &= b^4 + c^4 d^4 + d^4 c^5 + c^4 d^5 \\
  g &= c^4
\end{align*}

(11.9)

$$y^4 = f^{5,1}_{Q_{294}}\left((a^5, b^5, c^5, d^5), (a^1, b^1, c^1, d^1)\right) = \langle e, f, g, h \rangle$$

\begin{align*}
  e &= a^5 + b^5 d^5 + d^5 b^6 + b^5 d^6 \\
  f &= b^5 + c^5 d^5 + d^5 c^6 + c^5 d^6 \\
  g &= c^5
\end{align*}

(11.10)

$$y^5 = f^{1,2}_{Q_{294}}\left((a^1, b^1, c^1, d^1), (a^2, b^2, c^2, d^2)\right) = \langle e, f, g, h \rangle$$

\begin{align*}
  e &= a^1 + b^1 d^1 + d^1 b^2 + b^1 d^2 \\
  f &= b^1 + c^1 d^1 + d^1 c^2 + c^1 d^2 \\
  g &= c^1
\end{align*}

(11.11)
\[ y^6 = g_{Q_{299}}^{2.4}(a^2, b^2, c^2, d^2, \{a^4, b^4, c^4, d^4\}) = (e, f, g, h) \]
\[ e = d^2 b^4 + b^2 d^4 \quad f = d^2 c^4 + c^2 d^4 \quad g = 0 \quad h = 0 \]  
(11.12)

\[ y^7 = g_{Q_{299}}^{3.5}(a^3, b^3, c^3, d^3, \{a^5, b^5, c^5, d^5\}) = (e, f, g, h) \]
\[ e = d^3 b^5 + b^3 d^5 \quad f = d^3 c^5 + c^3 d^5 \quad g = 0 \quad h = 0 \]  
(11.13)

\[ y^8 = g_{Q_{299}}^{4.1}(a^4, b^4, c^4, d^4) = (e, f, g, h) \]
\[ e = d^4 b^4 + b^4 d^4 \quad f = d^4 c^4 + c^4 d^4 \quad g = 0 \quad h = 0 \]  
(11.14)

\[ y^9 = g_{Q_{299}}^{5.2}(a^5, b^5, c^5, d^5) = (e, f, g, h) \]
\[ e = d^5 b^5 + b^5 d^5 \quad f = d^5 c^5 + c^5 d^5 \quad g = 0 \quad h = 0 \]  
(11.15)

\[ y^{10} = g_{Q_{299}}^{6.3}(a^6, b^6, c^6, d^6) = (e, f, g, h) \]
\[ e = d^6 b^6 + b^6 d^6 \quad f = d^6 c^6 + c^6 d^6 \quad g = 0 \quad h = 0 \]  
(11.16)

The second stage $Q_{299}$

\[ y^1 = f_{Q_{299}}^{2.3}(a^2, b^2, c^2, d^2, \{a^3, b^3, c^3, d^3\}) = (e, f, g, h) \]
\[ e = a^2 + (a^2 d^2 + d^2 a^3 + a^2 d^3) + (c^2 d^2 + d^2 c^3 + c^2 d^3) \]
\[ f = b^2 + (a^2 d^2 + d^2 a^3 + a^2 d^3) + (b^2 d^2 + d^2 b^3 + b^2 d^3) + (c^2 d^2 + d^2 c^3 + c^2 d^3) \]
\[ g = c^2 + (b^2 d^2 + d^2 b^3 + b^2 d^3) + (c^2 d^2 + d^2 c^3 + c^2 d^3) \quad h = d^2 \]  
(11.17)

\[ y^2 = f_{Q_{299}}^{3.4}(a^3, b^3, c^3, d^3, \{a^4, b^4, c^4, d^4\}) = (e, f, g, h) \]
\[ e = a^3 + (a^3 d^3 + d^3 a^4 + a^3 d^4) + (c^3 d^3 + d^3 c^4 + c^3 d^4) \]
\[ f = b^3 + (a^3 d^3 + d^3 a^4 + a^3 d^4) + (b^3 d^3 + d^3 b^4 + b^3 d^4) + (c^3 d^3 + d^3 c^4 + c^3 d^4) \]
\[ g = c^3 + (b^3 d^3 + d^3 b^4 + b^3 d^4) + (c^3 d^3 + d^3 c^4 + c^3 d^4) \quad h = d^3 \]  
(11.18)

\[ y^3 = f_{Q_{299}}^{4.5}(a^4, b^4, c^4, d^4, \{a^5, b^5, c^5, d^5\}) = (e, f, g, h) \]
\[ e = a^4 + (a^4 d^4 + d^4 a^5 + a^4 d^5) + (c^4 d^4 + d^4 c^5 + c^4 d^5) \]
\[ f = b^4 + (a^4 d^4 + d^4 a^5 + a^4 d^5) + (b^4 d^4 + d^4 b^5 + b^4 d^5) + (c^4 d^4 + d^4 c^5 + c^4 d^5) \]
\[ g = c^4 + (b^4 d^4 + d^4 b^5 + b^4 d^5) + (c^4 d^4 + d^4 c^5 + c^4 d^5) \quad h = d^4 \]  
(11.19)

\[ y^4 = f_{Q_{299}}^{5.1}(a^5, b^5, c^5, d^5, \{a^1, b^1, c^1, d^1\}) = (e, f, g, h) \]
\[ e = a^5 + (a^5 d^5 + d^5 a^1 + a^5 d^1) + (c^5 d^5 + d^5 c^1 + c^5 d^1) \]
\[ f = b^5 + (a^5 d^5 + d^5 a^1 + a^5 d^1) + (b^5 d^5 + d^5 b^1 + b^5 d^1) + (c^5 d^5 + d^5 c^1 + c^5 d^1) \]
\[ g = c^5 + (b^5 d^5 + d^5 b^1 + b^5 d^1) + (c^5 d^5 + d^5 c^1 + c^5 d^1) \quad h = d^5 \]  
(11.20)

\[ y^5 = f_{Q_{299}}^{6.2}(a^6, b^6, c^6, d^6, \{a^2, b^2, c^2, d^2\}) = (e, f, g, h) \]
\[ e = a^6 + (a^6 d^6 + d^6 a^1 + a^6 d^1) + (c^6 d^6 + d^6 c^1 + c^6 d^1) \]
\[ f = b^6 + (a^6 d^6 + d^6 a^1 + a^6 d^1) + (b^6 d^6 + d^6 b^1 + b^6 d^1) + (c^6 d^6 + d^6 c^1 + c^6 d^1) \]
\[ g = c^6 + (b^6 d^6 + d^6 b^1 + b^6 d^1) + (c^6 d^6 + d^6 c^1 + c^6 d^1) \quad h = d^6 \]  
(11.21)
\[ y_6 = g_{Q_{299}}^4((a^2, b^2, c^2, d^2), (a^4, b^4, c^4, d^4)) = (e, f, g, h) \]
\[ e = d^2 a^4 + d^2 c^4 + a^2 d^4 + c^2 d^4 \]
\[ f = d^2 a^4 + d^2 b^4 + d^2 c^4 + a^2 d^4 + b^2 d^4 + c^2 d^4 \]
\[ g = d^2 b^4 + d^2 c^4 + b^2 d^4 + c^2 d^4 \]
\[ h = 0 \] (11.22)

\[ y_7 = g_{Q_{299}}^5((a^3, b^3, c^3, d^3), (a^5, b^5, c^5, d^5)) = (e, f, g, h) \]
\[ e = d^3 a^5 + d^3 c^5 + a^3 d^5 + c^3 d^5 \]
\[ f = d^3 a^5 + d^3 b^5 + d^3 c^5 + a^3 d^5 + b^3 d^5 + c^3 d^5 \]
\[ g = d^3 b^5 + d^3 c^5 + b^3 d^5 + c^3 d^5 \]
\[ h = 0 \] (11.23)

\[ y_8 = g_{Q_{299}}^4((a^1, b^1, c^1, d^1), (a^4, b^4, c^4, d^4)) = (e, f, g, h) \]
\[ e = d^1 a^4 + d^1 c^4 + a^1 d^4 + c^1 d^4 \]
\[ f = d^1 a^4 + d^1 b^4 + d^1 c^4 + a^1 d^4 + b^1 d^4 + c^1 d^4 \]
\[ g = d^1 b^4 + d^1 c^4 + b^1 d^4 + c^1 d^4 \]
\[ h = 0 \] (11.24)

\[ y_9 = g_{Q_{299}}^5((a^2, b^2, c^2, d^2), (a^5, b^5, c^5, d^5)) = (e, f, g, h) \]
\[ e = d^2 a^5 + d^2 c^5 + a^2 d^5 + c^2 d^5 \]
\[ f = d^2 a^5 + d^2 b^5 + d^2 c^5 + a^2 d^5 + b^2 d^5 + c^2 d^5 \]
\[ g = d^2 b^5 + d^2 c^5 + b^2 d^5 + c^2 d^5 \]
\[ h = 0 \] (11.25)

\[ y_{10} = g_{Q_{299}}^3((a^1, b^1, c^1, d^1), (a^3, b^3, c^3, d^3)) = (e, f, g, h) \]
\[ e = d^1 a^3 + d^1 c^3 + a^1 d^3 + c^1 d^3 \]
\[ f = d^1 a^3 + d^1 b^3 + d^1 c^3 + a^1 d^3 + b^1 d^3 + c^1 d^3 \]
\[ g = d^1 b^3 + d^1 c^3 + b^1 d^3 + c^1 d^3 \]
\[ h = 0 \] (11.26)
Chapter 12

Evaluation Results of PRESENT-2nd by 1 Billion Traces

Figure 12.1: PRESENT-2nd profile, sample trace and non-specific t-test results using 1,000,000,000 traces.
Bibliography


[CBM16] Abdelkarim Cherkaoui, Lilian Bossuet, and Cédric Marchand. Design, Evaluation, and Optimization of Physical Unclonable Functions Based on Transient Ef-


International Conference on Reconfigurable Computing and FPGAs - ReConFig 2012, Cancun, Quintana Roo, Mexico, December 5-7, 2012, Proceedings, pages 1–6. IEEE Computer Society, 2012. 26, 43, 51


[LMT+09] Victor Lомнё, Philippe Maurine, Lionel Torres, Michel Robert, Rafael Soares, and Ney Calazans. Evaluation on FPGA of Triple Rail Logic Robustness Against DPA and DEMA. In Design, Automation and Test in Europe - DATE 2009, Nice,


[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. Power Analysis Attacks: Revealing the Secrets of Smart Cards. Springer, 2007. 18, 45, 62, 63, 121


[OMPR05] Elisabeth Oswald, Stefan Mangard, Norbert Pramstaller, and Vincent Rijmen. A Side-Channel Analysis Resistant Description of the AES S-Box. In Fast Software


Bibliography


Bibliography


[Xil15] Xilinx. *Spartan-6 FPGA Clocking Resources*, June 2015. 9, 54


List of Abbreviations

**AES** Advanced Encryption Standard
**ANF** Algebraic Normal Form
**API** Application Programming Interface
**ASIC** Application Specific Integrated Circuit
**AWDDL** Asynchronous Wave Dynamic Differential Logic
**BCDL** Balanced Cell-based Dual-rail Logic
**BRAM** Block RAM
**CEMA** Correlation Electromagnetic Analysis
**CLB** Configurable Logic Block
**CMOS** Complementary Metal Oxide Semiconductor
**CPA** Correlation Power Analysis
**CRC** Cyclic Redundancy Check
**DAWDDL** Double Asynchronous Wave Dynamic Differential Logic
**DES** Data Encryption Standard
**DC** Direct Current
**DDPL-noEE** Double Dual-Rail Precharge Logic without Early Evaluation
**DEMA** Differential Electromagnetic Analysis
**DPL-noEE** Dual-Rail Precharge Logic without Early Evaluation
**DPA** Differential Power Analysis
**DRP** Dual-Rail Precharge
**DRSL** Dual-Rail Random Switching Logic
**DSP** Digital Signal Processor
**DWDDDL** Double Wave Dynamic Differential Logic
**EM** Electromagnetic
**EP** Early Propagation
**FAR** Frame Address Register
Abbreviations

FDRI Frame Data Register, Input Register
FF Flip Flop
FPGA Field Programmable Gate Array
GliFreD Glitch-Free Duplication
HD Hamming Distance
HDL Hardware Description Language
HW Hamming Weight
ICAP Internal Configuration Access Port
ID Identifier
iMDPL improved Masked Dual-Rail Pre-charge Logic
I/O Input/Output
IOB Input/Output Block
IP Intellectual Property
ISE Integrated Synthesis Environment
IT Information-Theoretic
LUT Look-Up Table
MCP-DPA Moments-Correlating Profiled DPA
MDPL Masked Dual-Rail Pre-charge Logic
MI Mutual Information
µC microcontroller
NOP No Operation
PA-DPL Precharge Absorbed Dual-Rail with Precharge Logic
PCAP Processor Configuration Access Port
PCB Printed Circuit Board
PDF Probability Density Function
PDL Programmable Delay Line
PI Perceived Information
PIP Programmable Interconnect Point
PLL Phase Locked Loop
POK Physically Obfuscated Key
PRNG Pseudo-Random Number Generator
PUF  Physical Unclonable Function
RAM  Random Access Memory
RO   Ring Oscillator
SABL Sense Amplified Based Logic
SAT  Satisfiability
SCA  Side-Channel Analysis
SDDL Separated Dynamic Differential Logic
SEMA Simple Electromagnetic Analysis
SM   Switch Matrix
SNR  Signal to Noise Ratio
SPA  Simple Power Analysis
SRAM Static Random Access Memory
TERO Transient Effect Ring Oscillator
TI   Threshold Implementation
USB  Universal Serial Bus
VHDL VHSIC (Very High Speed Integrated Circuit) Hardware Description Language
WDDL Wave Dynamic Differential Logic
XDL  Xilinx Design Language
# List of Figures

2.1 The internal LUT structure of the Xilinx Spartan-6 FPGA. ........................... 8
2.2 Visualization of the hardwired SMs interconnection structure of the Xilinx
   Spartan-6 device family. .................................................. 9
2.3 WDDL AND Gate. ................................................................ 22
2.4 DPL-noEE AND Gate. .......................................................... 22
2.5 AWDDL AND Gate. ............................................................ 23
2.6 Triple-Rail Gate. ................................................................ 24
2.7 BCDL Gate. ....................................................................... 25
2.8 DWDDL Gate. .................................................................... 25
2.9 SDDL Gate. ....................................................................... 26
2.10 PA-DPL Gate. ..................................................................... 26
2.11 The SRAM cell on logic level. ................................................. 30
2.12 The Butterfly PUF. ............................................................. 31
2.13 Architecture of a classical RO PUF. ........................................ 32
2.14 The TERO PUF. ............................................................... 34
2.15 Conceptual design of the Loop PUF. ........................................ 35
2.16 Architecture of a Bistable Ring PUF. ....................................... 35
2.17 Architecture of an Arbiter PUF. ............................................. 36
2.18 Arbiter-like PUF based on PDLs. ............................................ 37
3.1 Example of a fully routed circuit and its duplication. ......................... 44
3.2 Timing diagrams of the circuit of Figure 3.1. ...................................... 45
3.3 Architecture of the full AES encryption core with key expansion and control logic. 47
3.4 A sample trace and the t-test results using 1 000 000 traces. ................... 48
4.1 GliFreD construction and behavior. ............................................. 53
4.2 Transformation process of a symmetric circuit into an systolic circuit without loop. 57
4.3 Transformation process of a symmetric circuit into an systolic circuit with loop. 58
4.4 The architecture of the round-based AES core. ................................. 60
4.5 Sample power trace of each profile. ............................................. 61
4.6 Signal-to-Noise ratio curves for all plaintext bytes. ............................ 63
4.7 Perceived information curves based on the first plaintext byte. ............... 64
4.8 CPA curves targeting the first key byte. ....................................... 65
4.9 Moment-Correlating DPA results targeting the first key byte. ............... 66
5.1 An exemplary function implemented in a standard 6-to-1 LUT architecture and
   its GliFreD* representation including the timing diagram. ....................... 71
5.2 Architecture of the case studies: first \((d = 1)\) and second \((d = 2)\) order TI. .......... 73
5.3 A first-order TI of the PRESENT Sbox: \(S(x) = y\). .......................... 75
List of Figures

5.4 A second-order TI of the PRESENT Sbox: \( S(x) = y \). .......................... 76
5.5 KATAN-1st profile, sample trace and non-specific \( t \)-test results using 1,000,000 traces. .......................................................... 78
5.6 KATAN-2nd profile, sample trace and non-specific \( t \)-test results using 100,000,000 traces. ......................................................... 79
5.7 KATAN-1st-G profile, sample trace and non-specific \( t \)-test results using 1,000,000,000 traces. ....................................................... 82
5.8 PRESENT-1st profile, sample trace and non-specific \( t \)-test results using 10,000,000 traces. .......................................................... 82
5.9 PRESENT-2nd profile, sample trace and non-specific \( t \)-test results using 300,000,000 traces. .......................................................... 83
5.10 PRESENT-1st-G profile, sample trace and non-specific \( t \)-test results using 1,000,000,000,000 traces. ........................................ 84

6.1 The amount of SRAM cells changing their state, based on the time the BRAM was disconnected from the power network. ................................. 90
6.2 The distribution of the SRAM cells changing their state (1V, 23°C). ................. 91
6.3 The mean fractional intra-distance as well as the standard derivation for different environmental conditions. Note that the device under test (ID: 446489) shows a little flip-rate of about 1.5%. ..................................................... 92
6.4 The inter-distance between three devices of revision 0, and 70 BRAMs. ................ 93

7.1 The RO PUF architecture of PUFKY. Note that this representation swapped the columns and rows of the RO grid. ............................................... 97
7.2 Untouched and normalized RO grid frequency of PUFKY. ............................ 98
7.3 Grid variance of PUFKY after normalization. ............................................. 98
7.4 Impact of the glitch core on the RO grid. .................................................. 99
7.5 One board failed in counting correctly. .................................................... 100
7.6 Visualization of the proposed RO structure using both LUT output pins. Based on the used design, either one (i.e., OSingle_sLOW) or two ROs (i.e., OLocal or OSingle) are constructed from the given resources. .................................. 101
7.7 The chained structure of OSingle and OSingle_sLOW to build the RO grid. .... 101
7.8 Analysis of uniqueness and reliability of the new RO designs for position \( P1 \) each with 49-bit PUF responses after Lehmer-Gray encoding but without any entropy compression. 22 boards were used with 64 batches each. ......................... 103
7.9 Frequencies of RO grid. ................................................................. 105

8.1 Logical layout of the RO PUF implemented in 1/2 slices. ............................ 110
8.2 Schematic of the different delay element profiles. .................................... 111
8.3 Schematic implementation of TERO in 1/2 slices. .................................... 111
8.4 In (a) the average uniqueness for a 49-bit Lehmer/Gray word is shown for the three PUF constructs with different run times. For an ideal PUF, this uniqueness should be 23.11, which the TERO PUF achieves even with a run time of \( 2^n \) clock cycles. In (b) the average unreliability of the PUF construct for Lehmer/Gray words in % per bit is shown for different run times. ................................. 113
8.5 The bias analysis for glitch off and on. .............................................. 116
8.6 Uniqueness and unreliability when using the counter bit directly as a response for each bit. ............................... 118

12.1 PRESENT-2nd profile, sample trace and non-specific t-test results using 1,000,000 traces. ................................................ 133
List of Tables

2.1 The type 1 header format. ............................................. 11
2.2 The type 2 header format. ............................................. 11
2.3 Estimated post-synthesis resources and properties of the most common logic style available for FPGAs. .................................................. 27
3.1 Resource overview of the implemented AES cores. ............... 47
4.1 Property and resource comparison of GliFreD with an unmodified Canright design. 59
5.1 Details about the implemented profiles. The values given in this table are taken from the post route synthesis report of Xilinx ISE 14.7. 77
6.1 Flip probability for the devices under test. .......................... 90
7.1 A resource and frequency comparison of the given design structures to build a 64 × 16 RO grid. Note that the given mean frequencies are based on the active time-period of the counters which highly depends on the reference oscillator. The active time of a representative board was measured by an oscilloscope and set to 61 µs for the calculations. 102
7.2 Summarized counter failures. At maximum, the counter for all 1024 ROs can fail. 104
8.1 Unreliability of the PUF constructions for Lehmer/Gray encoding and 2Comp for different temperatures in % per bit. .................................................. 114
8.2 Uniqueness and bias of the tested PUF constructions at nominal temperature. The indicated bias values distinguished between “Batch”, which is the bias between all PUF instances grouped into a batch and “Neighbors”, which takes just the bias of physically neighboring PUF instances into account. 114
About the Author

Author information as of March 2017

Personal Data

Name Alexander Wild
E-Mail alexander.wild@rub.de
Place of birth Bad Mergentheim, Germany

Education

Ruhr-Universität Bochum, Germany 11/2012 - 12/2016
  Doctoral Candidate, Hardware Security Group / Chair for Embedded Security.
  External Member of the Research Training Group, UbiCrypt.

Ruhr-Universität Bochum, Germany 09/2006 - 07/2012
  Diploma in Engineering Program “Security in Information Technology”.

Informationstechnisches Gymnasium,
  Bad Mergentheim, Germany 09/2002 - 07/2005
  Higher Education Entrance Qualification.

Intern

KPMG AG, Essen, Germany 06/2011 - 08/2011
  IT Security Advisory.
Publications and Academic Activities

Peer-Reviewed Publications in Conference Proceedings and Journals


Publications and Academic Activities

ReConFig 2014, Cancun, Quintana Roo, Mexico, December 8-10, 2014, Proceedings, pages 1–6. IEEE Computer Society, 2014


Invited Talks


Participation in Selected Conferences, Workshops and Summer Schools

- 7. IAV-Tagung Simulation und Test für die Fahrzeugtechnik (Berlin, Germany)
- HOST 2016 (McLean, VA, USA)
- TRUDEVICE - WG Meeting 2016 (Dresden, Germany)
- Xilinx Security Working Group (XSWG) 2015 (Stuttgart, Germany)
- CHES 2015, (Saint-Malo, France)
- 22. Krypto-Tag 2015 (Munich, Germany)
- Threshold Implementation (TI) day 2015 (Leuven, Belgium)
- HOST 2015 (McLean, VA, USA)
- COSADE 2015 (Berlin, Germany)
- ReConFig 2014 (Cancun, Mexico)
- FPL 2014 (Munich, Germany)
- TRUDEVICE - WG Meeting 2013 (Freiburg, Germany)
- CHES 2013 (Santa Barbara, CA, USA)
- CRYPTO 2013 (Santa Barbara, CA, USA)
- ACNS 2013 (Banff, Alberta, Canada)
- Crypto for 2020 2013 (Tenerife, Spain)
- 17. Krypto-Tag 2012 (Heidelberg, Germany)
- ECRYPT Summer School on Challenges in Security Engineering 2012, (Bochum, Germany)