FPGAhammer: Remote Voltage Fault Attacks on Shared FPGAs, suitable for DFA on AES

Jonas Krautter, Dennis R. E. Gnad and Mehdi B. Tahoori

Karlsruhe Institute of Technology (KIT), Germany,
{jonas.krautter,dennis.gnad,mehdi.tahoori}@kit.edu

Abstract. With each new technology generation, the available resources on Field Programmable Gate Arrays increase, making them more attractive for partial access from multiple users. They get increasingly adopted as accelerators in various application domains, embedded in shared Systems on Chip or remote cloud services. Thus, some recent works have already explored Denial-of-Service and side-channel attacks, where an FPGA fabric is shared among multiple users. In this work, we show how fault attacks can be launched within an FPGA, through software-provided bitstreams alone. Excessive voltage drops can be generated from legitimate logic mapped into the FPGA to cause timing faults, reaching from spatially and logically isolated partitions of one to another user of the FPGA fabric. To cause this voltage drop, we first show how specific patterns to activate Ring Oscillators can cause timing failures in simple test designs on various FPGA boards. Subsequently, we analyze and adapt an existing fault model for the Advanced Encryption Standard to match the accuracy of our fault attack. In the same multi-user scenario, we show as a proof-of-concept how a successful Differential Fault Analysis attack on an AES module can be launched. We perform experiments on three FPGA boards of the same model and confirm that the attack adapts to all systems and is successful under process variation, but with different susceptibility to faults. The paper is concluded by validating the attack on another platform, and analyzing the vulnerability based on a timing analysis, proving the applicability to different devices.

Keywords: FPGA · DFA · chosen-plaintext · fault attack · on-chip · remote · multi-user · cloud · AES

1 Introduction

Field Programmable Gate Arrays (FPGAs) are increasingly used to accelerate computational hotspots of various applications, both in small Systems on Chip (SoCs), as well as in the data-center. For the same efficiency reasons of virtualizing CPU resources in the cloud, multi-user access to single FPGAs has been proposed in academic and industrial studies [XMHP12, EV12, BSB+14, FVS15], and reconfigurable computers can distribute accelerators from individual tasks in a similar fashion [YB18]. Among others, companies such as Amazon [AWS] and the Alibaba Cloud [Cor] allow users to rent their FPGA computing capacity for their own use, and all major vendors already provide SoCs including FPGAs since several years.

Since the introduction of fault attacks, which break cryptographic implementations without the need for finding algorithmic weakness, this class of attacks has quickly spread to various devices and different types of security related algorithms [BDL97, AK97, BS97]. Fault attacks traditionally require access to the hardware that is attacked. However, the famous rowhammer attack proved that software executed remotely can also cause faults

IACR Transactions on Cryptographic Hardware and Embedded Systems ISSN 2569-2925, Vol. 2018, No. 3, pp. 44–68
DOI:10.13154/tches.v2018.i3.44-68

in memory areas, that are restricted from access by the memory controller [KDK+14, GMM15].

Due to the characteristics of on-chip Power Distribution Networks (PDNs), switching activity on the chip leads to supply voltage fluctuations [ZSZF13]. This peculiarity can affect not only the reliability of the system but also pose a potential security threat. Just recently, it has been shown that activity of a small fraction of FPGA logic can cause a sufficiently excessive voltage drop to crash an entire FPGA, together with an integrated CPU, if the correct pattern of switching activity is applied [GOT17]. However, the authors noted that their FPGA stopped working entirely, before any timing faults could be observed in properly constrained designs. In other related work, it has been shown that voltage fluctuations inside an FPGA can be measured using the existing FPGA primitives, which allows to perform remote power analysis attacks [SGMT18, ZS18].

In this work, we present *FPGAhammer*. In analogy to rowhammer, we also cause faults through repetitive activation patterns, here to affect the supply voltage of an FPGA. This attack is precise enough to inject timing faults in FPGA logic, suitable to target specific encryption rounds of the Advanced Encryption Standard (AES) and perform Differential Fault Analysis (DFA). We carry out and elaborate this attack on various FPGA boards, containing Intel Cyclone V SoCs with different configurable logic sizes. We conclude that it is indeed possible to induce timing faults in a cryptographic core through remote configuration, with a partial bitstream that can be easily generated with official FPGA vendor tools.

In summary, our work makes the following contributions:

- We introduce a new category of software-initiated fault attacks in FPGA systems, possible with remote access to the target only, based on supply voltage drops generated by means of malicious yet legitimate switching activity.
- We establish a generic threat model for an attacker and a victim using a shared FPGA resource in an active fault attack scenario.
- We show that a spatially and logically separated attacker in one region of the FPGA fabric can attack a victim in another region.
- We test and prove the general vulnerability to on-chip voltage drop fault injection on a range of FPGA platforms, and elaborate an automated way to inject faults more precisely.
- We empirically prove that fault injections achieve a precision high enough for a successful DFA and key recovery on the AES, regardless of FPGA model or process variation within the tested devices.

The remaining paper is structured as follows: Section 2 explains the proposed threat model, the related work, and background on how voltage fluctuations occur inside chips. Moreover, we briefly outline the DFA method we apply in our attacks. In Section 3, we present an initial attacker design and describe the behaviour of FPGA boards from different manufacturers under the influence of malicious switching activity. In Section 4, we elaborate on how a fault attack and subsequent DFA can be carried out with the proposed method on an FPGA AES implementation. An overview of the hardware used in the experiments and implementation details are provided. We also present results of analyzing injection rates and key recovery success. We discuss some ideas for future research based on our findings in Section 5 and conclude our work in Section 6.

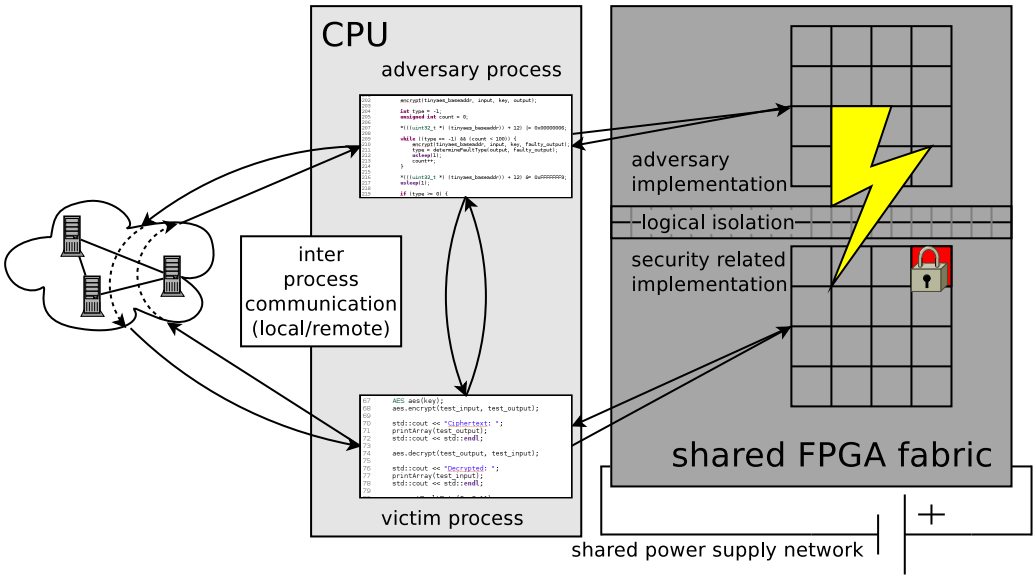


Figure 1: Overview of the threat model considered in this work: Attacker and victim share an FPGA resource with a common power supply network, but isolated, logically disconnected partitions on the fabric

2 Preliminaries

Before elaborating a full attack on the AES, we put our work in the context of other publications, which are relevant to our findings or assume a similar threat model. Moreover, we briefly explain the theoretical background of our experiments and the basics of causing a voltage drop, leading to faults in FPGAs.

2.1 Threat Model

In this section, we further describe the attacker-victim scenario assumed throughout this paper. A brief overview on this scenario is given in Figure 1. We assume the victim and the adversary to have access to a fraction of an FPGA, in which they can load their own arbitrary design, like a cryptographic accelerator. Both attacker and victim have their respective processes in an operating system, and their designs on the FPGA are logically and spatially separated, and follow other common best practices as explained in [HBW+07]. It is also assumed that the respective FPGA fabric is powered by a single common power supply. This scenario includes both data-center applications, in which FPGAs are utilized as standalone accelerators, as well as SoC platforms, in which multiple processes on the CPU can utilize a fraction of the FPGA logic.

We assume the victim to utilize their part of the programmable logic for a security related algorithm, such as a block cipher. A secret key used in this algorithm is either hard-coded onto the FPGA or transferred at runtime.

If we consider a symmetric encryption module, such as an AES implementation on the FPGA, used by the victim, we assume the following **Adaptive-Chosen-Plaintext-Scenario**:

- The adversary can issue arbitrary plaintexts to a public interface of the victim process either locally or remotely through a network.
- The victim outputs the ciphertext of the provided plaintext, encrypted with the secret key, only known to the victim.

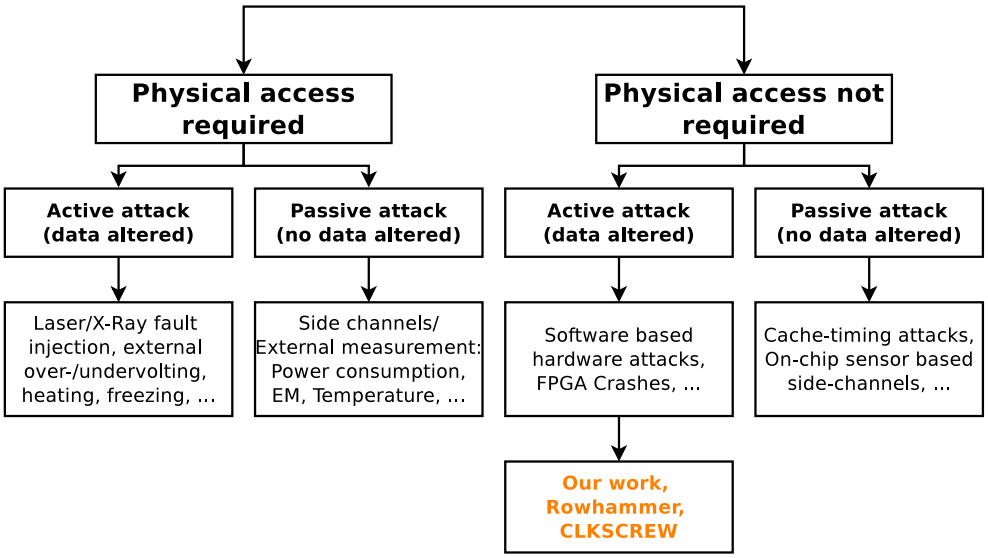


Figure 2: Proposed classification of different hardware attacks with our work categorized as an active attack, that does not require physical access to the attacked device

- The amount of requests is limited only by the attackers computational restrictions, which we assume to be polynomial.

We remark that we assume an attacker that can encrypt arbitrary plaintexts in the generic threat model. However, the actual content of the plaintext is irrelevant for a successful DFA and may even be unknown. The attacker only needs to be able to enforce encryption of the same plaintext twice, where one case is a fault-free encryption and in the other case faults are injected. Therefore, a DFA based attack on AES like the one presented in this paper, can be applied to situations where replaying encryption requests to the target module is possible. Please keep in mind that attacks with faulty ciphertext only are also possible [FJLT13], but are subject to future studies to reveal whether they are feasible by internal voltage-drop based fault generation in FPGAs.

2.2 Related Work

Figure 2 illustrates how our work contributes to the broad area of fault and side-channel attacks. We show attack categories based on physical and without physical access, and further split them into active attacks, such as fault injection or tampering, and passive attacks, such as side-channels.

We would like to mention that side-channel or fault attacks through remote access do not only concern FPGA-based systems. In many devices, there exist opportunities from software to observe system activity or trigger malicious behavior, where some have been used to perform side-channel or fault attacks already [YF14, KGG+18, KDK+14, TSS17]. The specific category of remotely induced fault attacks is not yet widely explored. To the best of our knowledge, only two other attack families can be categorized within this class. One of them is the famous rowhammer attack that introduces faults in DRAM by maliciously crafted access patterns [KDK+14]. These patterns lead to switching activity in the memory that does usually not occur in normal operation, and invokes high stress patterns causing bitflips in adjacent memory. In the end, privilege escalation is possible, which can even be triggered with sandboxed javascript code [GMM15]. Another category is introduced by a work that exploits power management functionality, that is not sufficiently secured against parameter changes with malicious intent, to perform fault attacks in the

same SoC [TSS17]. The introduced faults can be used for DFA on an AES module, finally leading to privilege escalation within the SoC.

Few publications exist, which consider a similar threat model on FPGA-based systems, in which no physical access to the FPGA device is possible or required for an attack [GOT17, SGMT18, ZS18]. In two of these works, side-channel attacks have been shown that are based on measuring a fraction of the internal power consumption of an FPGA within the chip itself. In [SGMT18], voltage fluctuation sensors based on tapped delay lines [ZSZF13] were implemented in FPGA logic, to sufficiently measure transient voltage fluctuations for attacking an AES design in remote FPGA logic on the same chip. On the other hand, [ZS18] showed that even simple ring oscillator-based voltage sensors, that achieve lower sampling rates, are sufficient to attack a simple RSA implementation, running in software on the CPU of a SoC containing FPGA logic, proving that not only the FPGA fabric itself is vulnerable.

The work presented in [GOT17] evaluated the effectiveness of Ring Oscillator (RO) designs for generating critical voltage variations and their potential use in fault attack schemes. The authors were able to cause crashes and system resets, resulting in a Denial-of-Service (DoS) attack on the FPGA, which can be triggered remotely, and for the tested FPGAs requires access to only about 12% of logic resources. However, according to that work, the FPGAs always crashed before timing faults could be observed. We show later, that this might be due to the choice of analyzed FPGAs, and that not all of them are equally vulnerable. Additionally, in an SoC shared between hard CPU cores and FPGA logic, their attack did not only affect the FPGA, but also the CPU, suggesting a shared power supply.

2.3 DFA on AES

As a proof-of-concept for successfully conducting an on-chip fault attack on a cryptographic implementation, we evaluate a DFA method on an FPGA implementation of the block cipher AES. In this work, we attack an implementation with 128 bit key length. The encryption and decryption scheme is based on the circular application of four different operations *SubBytes*, *ShiftRows*, *MixColumns*, and *AddRoundKey* on the data block, which is stored in a four by four byte matrix called the *state*. This circular application is repeated for 10 rounds as defined by the key length of 128 bits.

DFA is based on causing the same plaintext to be encrypted twice – the first time to gain the correct ciphertext and the second time to acquire a faulty ciphertext as the result of fault injection at a specific point of the algorithm. The ciphertext pairs, each consisting of a correct and a faulty ciphertext of the same plaintext, are then evaluated to extract information about the secret key of the cipher.

In 2003, a generic fault attack on Substitution-Permutation Network (SPN) ciphers was introduced, which only requires two ciphertext pairs to recover the original secret key [PQ03]. We apply this attack with multiple adaptions for practical feasibility to use it on an AES module implemented on an FPGA, and therefore elaborate on this method in detail.

The fault model of the attack is a random byte fault on a single byte occurring before the 8th round of the AES algorithm. Before elaborating the eventual attack, which requires only two faulty ciphertext pairs to be successful, we consider a random byte fault before the 9th round. As depicted in the example in Figure 3, a single byte fault on the first byte of the state matrix before the 9th round (as well as on bytes 5, 10 or 15) is propagated and results in four faulty bytes at specific positions in the output ciphertext: 0, 7, 10 and 13. With a single byte fault on one of those bytes, we can therefore compute candidates for four bytes of the last round key. After recovering the entire 10th round key, it is possible to compute the original AES secret key, since the key schedule is invertible. Similar relations exist for the other bytes of the state matrix.

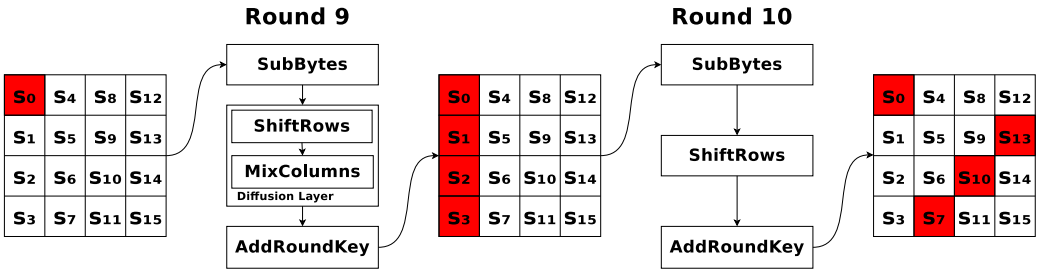


Figure 3: Propagation of a faulty byte in the input of the 9th round of the AES algorithm

Exemplary, we consider single byte faults on the bytes 0, 5, 10 or 15 in the state matrix before round 9, which can be used to recover bytes 0, 7, 10 and 13 of the last round key. We initialize a set $\mathscr{S}$ containing all possible candidates for those bytes of the last round key. This set of possible candidates is continuously reduced with the evaluation of each pair of correct and faulty ciphertext (C, C') by inverting the 10th AES round for the two ciphertexts with a candidate $k \in \mathscr{S}$. The candidate is discarded, if the difference between the two state matrices resulting from the inversion of the ciphertexts is not within the set of possible differences resulting from a single byte fault on bytes 0, 5, 10 or 15 before round 9.

In [PQ03], the number of candidates remaining after two ciphertext pairs was only one (the correct) candidate in 98% of all cases. In the other 2% of cases, only two or a maximum of four key candidates were left. Therefore, to recover the entire round key of round 10 with faults injected before round 9, a minimum of eight ciphertext pairs are required: For each of the four key bytes, two pairs are needed. This can be improved by injecting single byte faults before round 8, which affect four bytes before round 9 and therefore all bytes of the output at once. Then only two ciphertext pairs are required to recover the full AES key.

2.4 Fault Injection using FPGA logic

A functional block in a synchronous FPGA design includes a common clock signal, which is used to synchronize all memory components within the block. This means that combinational paths between registers (D-flipflops) are constrained in their delay by the clock signal. If a signal takes longer than a clock cycle to traverse a combinational block, timing violations may cause the output of the target register to be different from the desired results – a timing fault occurs.

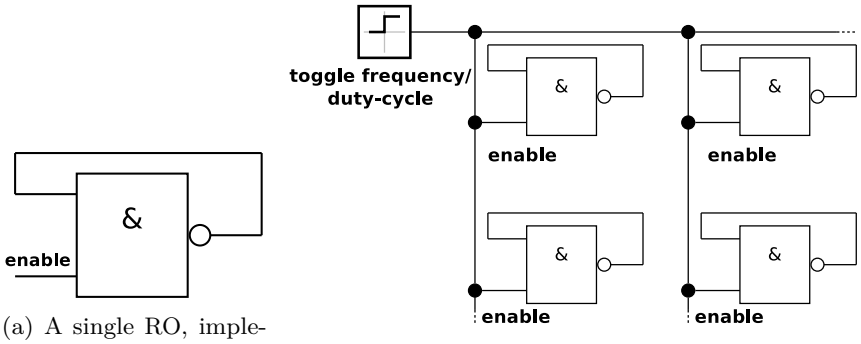
The constraints can be formulated with five parameters [ZDCT13]: The clock cycle time t_{clk}, the internal register delay d_{clk2q}, the setup time t_{setup}, which is the amount of time an input signal has to be stable at a register input, the maximum data propagation time through the combinational logic d_{pMax}, and the clock skew t_{skew}, which is the phase difference of the clock signal between two different registers.

The timing constraints can then be expressed by Equation 1.

$$t_{\text{clk}} > d_{\text{clk2q}} + d_{\text{pMax}} + t_{\text{setup}} - t_{\text{skew}} \tag{1}$$

A higher data propagation time can be achieved by lowering the power supply voltage V_{DD} [ASM07]. The increased delay raises the right hand side of the above equation, leading to a timing violation and a potential fault injection.

To understand how the supply voltage of an FPGA can be decreased with on-chip logic elements, it is necessary to understand how the PDN of an FPGA behaves under the influence of different designs on the fabric. The PDN includes a network from the voltage regulation module on the board down to the internal power rails and every transistor on the FPGA. Generally, the PDN can be modelled as a mesh of resistive, inductive and



(a) A single RO, implemented with a NAND gate

(b) The implemented oscillator grid, used to cause voltage drops

Figure 4: Schematics of the RO implementations

capacitive elements. Therefore, the power supply voltage depends on two parameters: The average static current drawn by the implemented design (IR-drop) and the voltage drop caused by switching activity and inductance (di/dt-drop). The relation is summarized in the *Law of Inductance*: $V_{\text{drop}} = IR + Ldi/dt$. With technology scaling, the effect of static voltage drops has become less relevant compared to the voltage fluctuations caused by switching activity and inductive components [ASM07].

To evoke malicious switching activity on an FPGA and cause an excessive di/dt-drop, we can deploy a massive amount of ROs to generate high frequency current oscillations. In [GOT17], ROs were already used to induce voltage drops high enough to crash FPGA-based systems. It was shown, how a singular activation of a large amount of ROs causes the voltage to drop rapidly by a certain amount and then more slowly return to the original value within about 50 μs for the tested devices. Moreover, it was described, how a drop can be increased significantly, by driving the entire grid of oscillators with a different, slower frequency, constantly enabling and disabling the ROs. A dependency on the duty-cycle of this RO toggle signal was demonstrated as well [GOKT16].

ROs are implemented by connecting any odd number of inverters circularly. An additional enable signal allows enabling and disabling the oscillation, to connect each RO to a common toggle signal. Figure 4a shows the schematic of an RO with an enable signal and a single two-input NAND gate. The frequency of the oscillation depends on the gate delay and the loopback routing from the output of the gate to the input. Since the gate is usually implemented as a single Look-Up Table (LUT) in the FPGA, the oscillation frequency depends mostly on the loopback routing.

3 Provoking Faults in FPGA Designs

Before developing a full DFA attack on the AES, we reproduce results from [GOT17] about provoking crashes on Intel FPGAs, which has only been shown on Xilinx devices so far. Moreover, we evaluate fault injection vulnerability of a broad range of FPGA devices and boards from different manufacturers.

3.1 Initial Design for Causing Voltage Drops

For stressing the PDN and inducing a voltage drop, we deploy an RO grid onto the FPGA fabric, which can be enabled and disabled with a variable toggle signal. A schematic overview on the design with multiple ROs is presented in Figure 4b. The characteristics of the voltage fluctuation caused by the oscillators depend on the toggle signal frequency and duty-cycle.

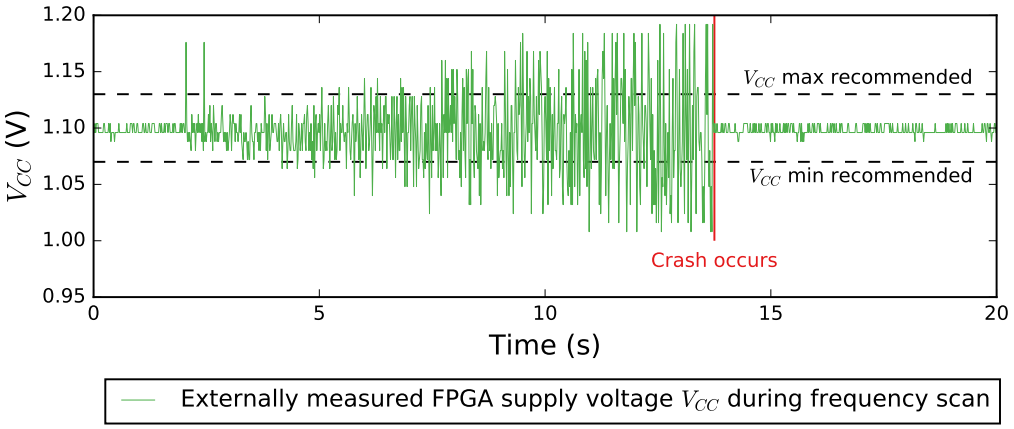


Figure 5: Trace of FPGA supply voltage *VCCINT*, measured externally with an oscilloscope during a frequency sweep leading to a crash of the device

Figure 5 shows a trace of the externally-measured supply voltage of an Intel Cyclone V SoC device, when performing a sweep over different toggle frequencies for the RO activation signal, until the device crashes. We toggle the RO grid with a decreasing frequency, and for each frequency, increase the duty-cycle up to 75% until the voltage fluctuation significantly exceeds the supply voltage limits of the device. After this point it crashes, leading to a hard reset of the included ARM Hard Processor System and a loss of the configuration of the FPGA device.

When developing in Hardware Description Languages (HDLs), the synthesis software from most FPGA manufacturers deems the RO design entirely useless. Thus, it is necessary to prevent the synthesis tools from optimizing the RO grid away. Initially, we conducted experiments with different oscillator designs and found significant differences regarding their effectiveness in causing the supply voltage to drop. We studied the following design options on Intel FPGAs:

a) Implementation using an output pin: The first approach is to connect all ROs through a reduction function to an arbitrary output pin of the FPGA. This approach was discarded immediately, since the routing congestions limit the amount of ROs significantly.

b) Implementation using virtual pins: Secondly, we can declare all outputs of the ROs as virtual pins, which are implemented as a single LUT each on the FPGA.

c) Implementation without additional elements (Bare ROs): Another possibility is to define output connections of each RO to the top-level entity, but not to an output pin. The synthesis software then accepts the fanout-free LUTs without additional elements.

In Figure 6, we present results of comparing the two RO variants with (b) and without (c) virtual output pins, where each implementation has the same amount of logic utilization.

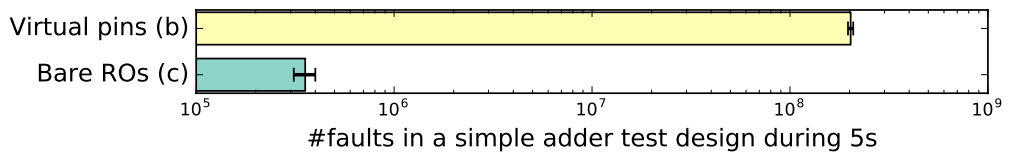


Figure 6: Amount of errors detected in a simple adder test design during 5 seconds of RO toggle activation with respect to the RO implementation option. Tested on DE1-SoC.

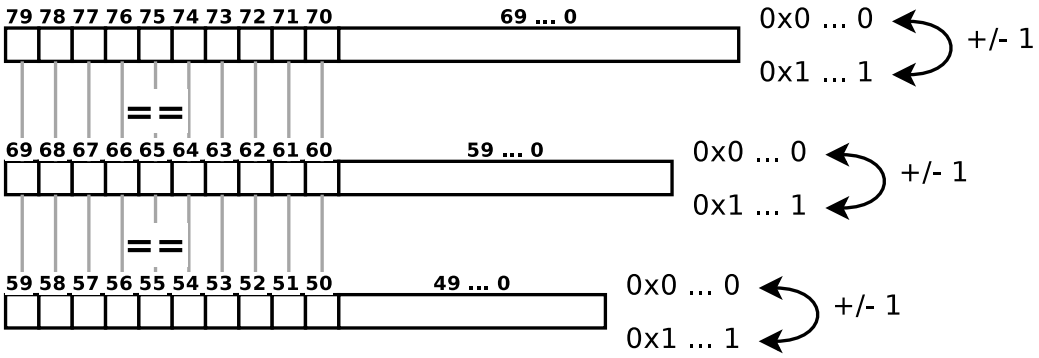


Figure 7: Simple adder design to evaluate fault attack vulnerability

We stressed a simple test design, which is detailed in the next section. We collect the number of faults in a series of 10 trials for 5 seconds each. The amount of recorded errors in the test design proves the higher effectiveness of the virtual pin option. Despite less ring oscillators are used in variant b, the additional interconnect resources connected to each single oscillator cause more faults.

Voltage drops can be further increased by enforcing high interconnect utilization when separating ROs from their respective virtual output pins. However, we have observed that the design which works the best, strongly depends on the used device, especially if devices from different manufacturers are considered.

3.2 Voltage Drop-based Timing Faults in a Simple Test Design

Before applying fault attack analysis to a cryptographic implementation on an FPGA, we analyze the behaviour of various FPGA boards, while stressing them with a massive amount of ROs. We find that all evaluated boards from different manufacturers are susceptible to the oscillator grids in terms of showing unusual behaviour upon activation of the oscillation.

An overview of a simple adder design for the evaluation of fault attack vulnerability is depicted in Figure 7. Three register carry-chains of different lengths are driven in a way that keeps them switching between their maximum and minimum values. When the maximum value is incremented, the resulting overflow requires a carry bit to be propagated from the Least Significant Bit (LSB) to the Most Significant Bit (MSB) through the entire carry-chain. Likewise, decrementing the minimum value causes propagation of a carry bit through the registers.

In the given design, we can now compare n of the uppermost bits of each adder with the correct and expected result. When a voltage drop increases the propagation delay of circuit elements, this comparison shows whether a timing fault occurred in the respective adder.

We then investigate the behaviour of the adder design with different lengths and under various frequencies. In these configurations, it is possible to find a setting, where the timing analysis of the FPGA mapping tools shows a violation of the timing constraints, yet the design works in a normal situation. ROs can then cause timing faults at runtime.

In production-stage cryptographic implementations, users are likely keen to avoid timing-violations reported by the analysis tools during design synthesis, which makes the vulnerability of devices in this situations less relevant to an adversary. We found that on most of the tested platforms it is also possible to inject faults into designs that meet the timing constraints of worst-case estimation models.

In Table 1, we summarize our results across several platforms, which we evaluated regarding the feasibility of fault attacks in designs that do not meet the timing constraints (unmet) and designs that meet the constraints. Although some platforms seem to be not

vulnerable, it is more likely that we simply failed to find the appropriate parameters to activate the oscillators in a way to trigger the necessary voltage drops yet. These initial results promise a possible success regarding the application of fault injection and DFA to cryptographic modules on FPGAs.

Table 1: First results about general vulnerability of different platforms

Vendor	Board	Device	Fault attack possible (constraints unmet)	Fault attack possible (constraints met)
Intel	Terasic DE4	Stratix IV	Yes	Yes
Xilinx	XUP PYNQ-Z1	Zynq-7000	Yes	No
Lattice	iCE40HX8K-B-EVN	iCE40HX8K	Yes	Yes
Intel	Terasic DE1-SoC	Cyclone V SoC	Yes	Yes
Intel	Terasic DE0-Nano-SoC	Cyclone V SoC	Yes	Yes

4 Fault Attack Evaluation on AES

In this subsection, we illustrate the details of performing a full key recovery attack on AES using the RO design for fault injection and the DFA explained in Subsection 2.3 for key recovery. We prove the concept of on-chip fault attacks by evaluating fault injection and key recovery on the Intel Cyclone V SoC chip family. The following subsections describe the general setup and detailed parameters for each experiment and present the acquired results.

We initially detail how we achieve the required fault injection precision with an automated calibration approach. Subsequently, we investigate the general fault injection rate with respect to the amount of ROs in the attacker design and inter-die process variations of three DE1-SoC boards. We continue by evaluating the success rate of a full AES key recovery for 5000 keys. Additionally, we study the dependence of injection rates on the operational frequency of the AES module on the smaller Cyclone V SoC device on the DE0-Nano-SoC board.

4.1 Calibrating the Fault Injection Precision

The DFA attack from [PQ03], which was described in Subsection 2.3, allows to recover a secret AES key with only two pairs of correct and faulty ciphertexts, if the fault is injected according to the fault model of a single byte fault before the 8th round of the AES encryption. In practice, we need to adapt parameters such as frequency, duty-cycle, and activation time of the RO grid to provoke faults at the proper moment of the encryption.

Any fault before the 8th round of the AES leads to all bytes of the faulty output ciphertext to be different from the correct one, whereas any fault after the 9th round leads to less than four bytes to be different. Faults that are injected into the input state matrix of the 9th round are revealed in exactly four bytes of the faulty output ciphertext being different from the correct one. This allows us to verify a successful fault injection using the output ciphertext. Therefore, we decide to aim for injecting faults not before the 8th round of the AES but before the 9th round only.

To make use of the possibility for injection success verification, we develop an automated calibration algorithm, to be executed before evaluating injection rates or attack success for a given design and device. The algorithm allows to use the attacker design in different setups, without the need for finding appropriate parameters in time-consuming trial-and-error experiments manually.

In Figure 8, we present an overview of the full calibration algorithm we use before evaluating injection rates or key recovery success. We adapt the signal for activating the ROs in three parameters: The toggle frequency, the duty-cycle and the delay between starting the encryption and activating the RO grid. On the left side of the flowchart, we depict the process flow on the software side, whereas the right side enlists the actions

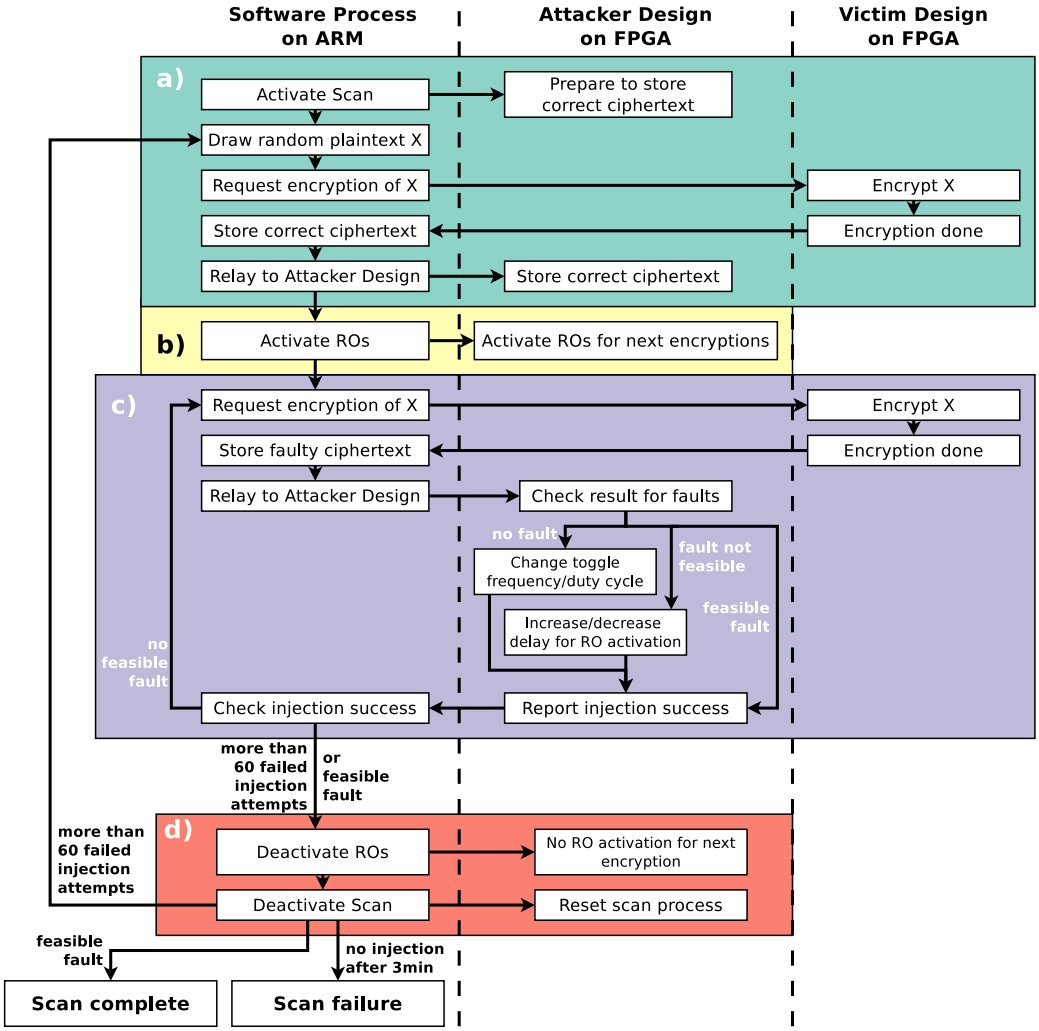


Figure 8: Flowchart of the calibration algorithm to find the appropriate parameters for injecting faults at the desired moment

carried out on the FPGA by both attacker and victim design. The algorithm performs as follows:

a) The attacker activates the calibration process on the FPGA. A random input plaintext is drawn and encrypted without RO activity. The result is stored as the correct ciphertext.

b) Afterwards the fault injection process on the FPGA is activated, to toggle RO activity for the following encryptions.

c) Encryption of the same random plaintext is requested. The attacker design on the FPGA activates the RO grid with an initial frequency and duty-cycle and no activation delay. If no fault is detected, the frequency/duty-cycle are decreased/increased. Duty-cycle is increased for each frequency up to 75%. If a fault is detected, which affects an undesired subset of bytes, the injection occurred too early or too late, and the activation delay is increased/decreased. In any case, the attacker design reports the injection success to the attacker process, which either requests another encryption or continues the process.

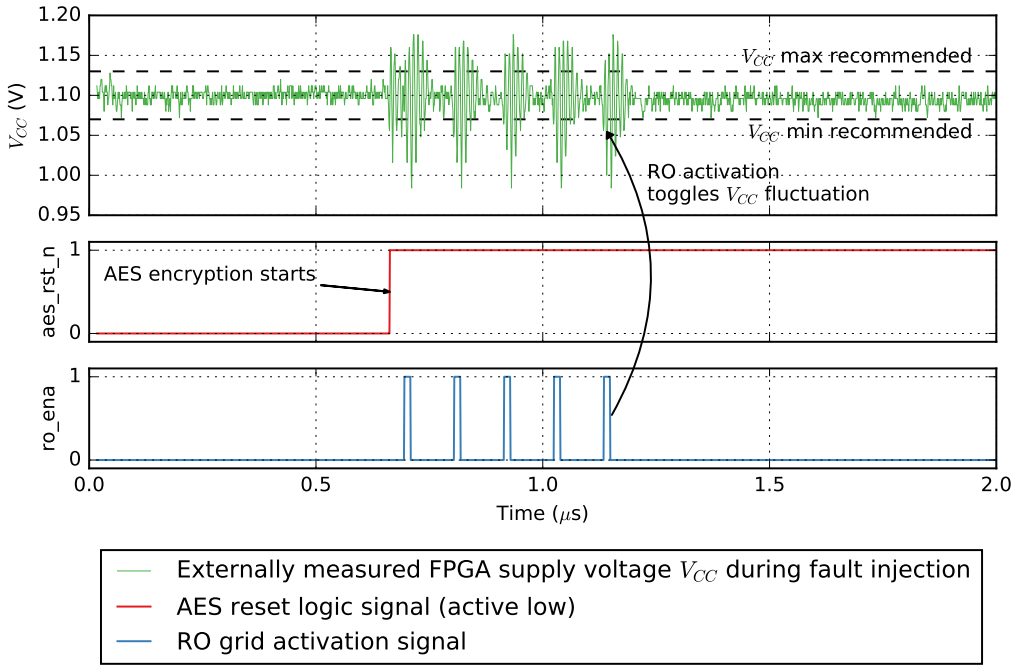


Figure 9: Trace of FPGA supply voltage *VCCINT*, measured externally with an oscilloscope during a single fault injection attempt

d) If the injection was successful or a predefined maximum of injection attempts inj_{max} were unsuccessful, the attacker software deactivates the RO grid and either finishes the successful scan or chooses another random plaintext for fault injection.

During experiments, we determined $inj_{max} = 60$ as an appropriate upper limit regarding the number of encryptions needed until a fault is injected, since not all random plaintexts result in a faulty output even with the ROs active. After a successful calibration process, the three parameters are fixed and used for subsequent fault injections.

In Figure 9, we show an externally acquired trace of the FPGA supply voltage VCCINT during a single fault injection by the attacker design on the FPGA. The AES reset signal (aes_rst_n), which resets the AES encryption module when low, indicates the start of an encryption. To provoke a fault, the attacker design pulses the RO grid (ro_ena signal) with the previously determined frequency, duty-cycle and activation delay. The voltage fluctuations (V_{CC}) cause a critical delay at the desired moment with a higher probability and therefore, a fault is injected.

In conclusion, our approach requires eight instead of only two ciphertexts compared to [PQ03] to recover the secret key, but makes the attack feasible in practice, where a lot more encryption requests are required to have the attacker design affect the AES module at the desired moment. In all subsequent experiments, we apply this variant of the attack, aiming to inject faults before the 9th encryption round. The calibration is executed only once, at the beginning of any evaluation. However, we continue to filter faults that have been injected at an earlier or later encryption round during the collection of ciphertext pairs. This method maximizes key recovery success, although the injection precision achieved by our calibration is very high, as we show in the results.

Furthermore, we remark that the acquired calibration parameters can even be reused on different devices of the same type. Therefore, there is no need to perform specific calibrations on the board on which the fault attack is to be performed. In our experiments on the Terasic DE1-SoC for example, we find that a toggle frequency of 1.16 MHz with a 56% duty-cycle is selected most frequently in a run of 1000 calibrations. Fixing those

Table 2: STA corner timing-models available in the Quartus STA from fastest to slowest model for a given device speed grade at 1100 mV supply voltage

Silicon speed (Process variation)	Operation temperature
Fast	0° C
	85° C
Slow	0° C
	85° C

parameters and reusing the design on a different DE1-SoC board leads to similar or even better fault injection rates, depending on the general vulnerability (process variation) of the board.

Note that this method filters out all faults that are caused at the wrong AES encryption round but does not necessarily discard ciphertexts, which result from fault injections that affect multiple bytes before the 9th round. Some multi-byte faults can also lead to four faulty bytes at the desired positions in the output ciphertext. Therefore, we still have some amount of keys, which can not be recovered during evaluation, because multi-byte faults are not covered by the theoretical fault model of the DFA. Further details regarding unsuccessful key recoveries can be found in the results in Subsection 4.4.

4.2 Hardware and Software Environment

In this subsection, we specify the devices, which have been used in the further experiments, and provide details on the implemented hardware designs and software tools.

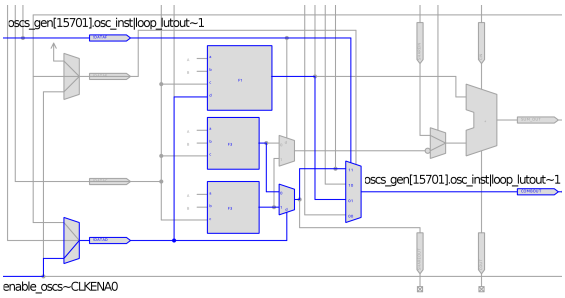
The AES implementation we use as a proof-of-concept in this work is a simple, small module for 128 bit key length encryption. It utilizes around $300 - 400$ registers and about $750 - 850$ LUTs in the tested Cyclone V FPGAs and takes 50 clock cycles to encrypt a given plaintext. The module is not protected against side-channel or fault attacks.

We perform our experiments on systems based on the Intel Cyclone V SoC family, which incorporate an Intel FPGA and a 925 MHz Dual-Core ARM Cortex-A9 processor inside a single die. The 5CSEMA5F31C6 chip is embedded on the Terasic DE1-SoC board. We studied injection rates and attacks on three Terasic DE1-SoC boards of different age and usage history to account for process and aging variation. A smaller variant of the Cyclone V SoC, the 5CSEMA4U23C6N, with only half the amount of logic elements is present in the Terasic DE0-Nano-SoC board, which we investigated as well. The devices are used with their standard, unmodified power supplies.

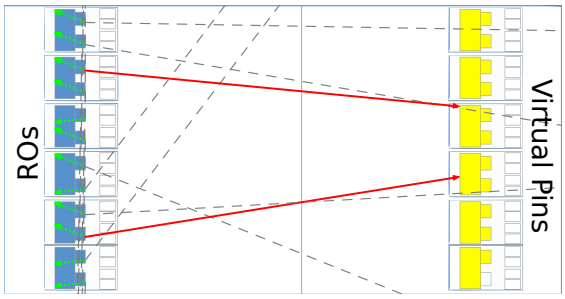
Both boards have an SD card slot, which we use to boot a Linux system and run user applications, that interact with the FPGA fabric, on the ARM processor. We encapsulate the AES cryptomodule as an Intel Avalon Memory-Mapped slave device, which allows access from programs running within the Linux system on the CPU.

The Intel Quartus Prime software offers tools for Static Timing Analysis (STA), which analyzes the design in terms of timing violations under four different models (corners) for a given device with a specific speed grade [MW17]. We enlist the available timing-models in Table 2. The fast/slow classification of silicon for the given device speed grade refers to propagation delay variations caused by intra-die process variation.

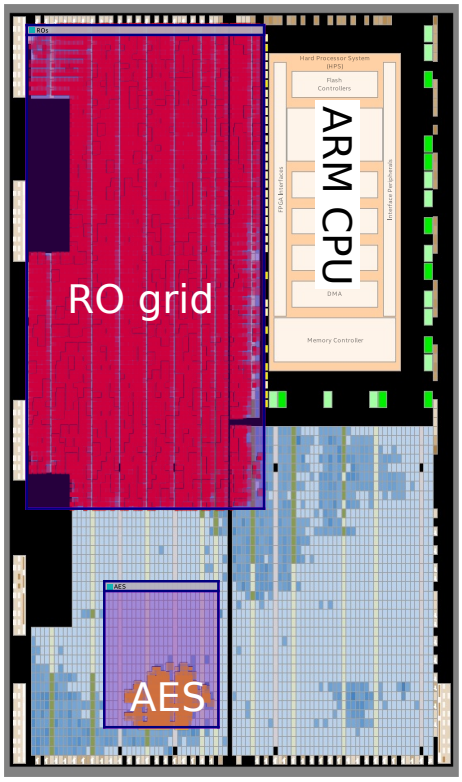
If the timing analysis reports timing violations at the time of implementation, the design is not guaranteed to work reliable under all operation corner cases in terms of temperature and voltage levels, according to official chip specifications as found in the FPGA datasheet. We focus on attacking designs that do not violate any timing constraints, even at the worst-case 85° C corner, but investigate the influence of worst case path slack in designs that violate the constraints as well. For each experiment, we explicitly report whether timing constraints are violated in the respective subsection later.



(a) An RO as implemented during compilation in the LUTs of an ALM of the Intel Cyclone V SoC



(b) ROs in a LAB of the Cyclone V SoC on the left with the interconnect to their respective virtual output pins on the adjacent LAB on the right (continuous lines) and other LABs (dashed lines) as well as loopback routing (dotted lines)



(c) Design for evaluation of fault injection after fitting as displayed in the Quartus Chip Planner on the Terasic DE1-SoC board with the AES module in the bottom area and the ROs grid in the top left region

Figure 10: Implementation details of the RO attack design on the Intel Cyclone V SoC as displayed in the Quartus Chip Planner

We implement the attacker design as a grid of ROs as described in Subsection 2.4. A single RO is composed only of the combinational part of a single Adaptive Logic Module (ALM) on the Cyclone V. The output is directly routed through local interconnect back to the input. This way, we achieve the fastest possible switching frequency for the oscillators. In Figure 10a, an example of how the Intel Quartus Prime software synthesizes and fits a single RO into the bottom part of one ALM can be examined. The used output on the right and input on the top left of the ALM are the same and the additional enable signal is connected to the bottom left input of the ALM.

The Intel Quartus Prime software reports the worst case delay through the LUT to be about 0.08 ns and the loopback routing delay through local interconnect around 0.21 ns. Therefore, assuming a maximum delay of 0.3 ns through gate and loopback, the RO can achieve frequencies of 3 GHz and more.

As explained in Subsection 3.1, an RO-based design with a virtual pin (variant b) is most efficient to provoke critical voltage drops, which is why we choose this design variant for our attack on AES as well. In Figure 10b, we show how several ROs defined as an oscillating LUT and a virtual output pin are mapped into a Logical Array Block (LAB) of the Cyclone V SoC as presented in the Quartus Chip Planner. The schematic shows the loopback routing (dotted lines) of each RO and the routing to their respective virtual output pins, two of which are placed in an adjacent LAB on the right (continuous lines) and some in different regions of the FPGA fabric (dashed lines). The relevant Verilog

code parts for implementing an RO grid on Intel FPGA devices can be found in the Appendices A, B and C.

Moreover, it is necessary to drive the oscillator grid with a very specific frequency and duty-cycle. We therefore add an enable signal to trigger each of the implemented ROs, which is routed through a global clock buffer on the Cyclone V SoC. The use of this type of signal, originally intended for distribution of clock signals on the FPGA, allows to save on routing resources from the toggle frequency control design block to all of the ROs and accelerates the compilation of the entire design significantly.

Since our attack scenario, elaborated in Section 2.1, assumes a shared multi-user FPGA use-case, we constrain each design using the LogicLock feature of the Intel Quartus Prime software to keep victim and attacker design blocks within designated areas of the FPGA fabric. To avoid any variation from other components on the chip, we additionally activate the region reservation parameter of the LogicLock region, that contains the AES module, which prevents the fitter from placing any other logic than the AES module and its Avalon MM encapsulation into this area. Figure 10c shows the ROs mapped into the top left area and the AES module in the bottom region as displayed in the Quartus Chip Planner for the design on the larger Cyclone V SoC on the Terasic DE1-SoC. On the software side, we implement tools for controlling encryption and fault injection to be executed within the Linux system on the ARM core of the Cyclone V SoC. The evaluation of the collected ciphertext pairs and respective DFA is performed on a standard host computer with an Intel i7-7700HQ Quad-Core processor.

4.3 General Fault Injection Efficiency on the DE1-SoC

In order to evaluate the general fault injection efficiency, we first generate bitstreams for different percentages of logic utilization of the attacker logic in the range of 30% to 50% for the DE1-SoC board. The victim module runs at a frequency of 111 MHz, which does not violate any timing constraints as explained in the previous subsection, even in the worst-case corner of the static timing analysis. Then we measure the number of faults occurring for one million encryption requests from a previously generated set of random plaintexts, which are reused for all experiments. We evaluate the experiment on three different Terasic DE1-SoC boards of different age and usage history. The encryption key remains the same for one test series, which is repeated for a second random key for each of the DE1-SoC boards. Before every evaluation, the calibration algorithm as described in Subsection 4.1 is executed, to find optimal parameters for provoking the desired kind of faults, which can be used in the subsequent DFA.

Figure 11 shows the total number of faults out of 1M trials F_{tot}, as well as the number of faults usable for DFA with our described fault model F_{DFA}. Both results are shown dependent on the number of activated ROs in % of available LUTs in the FPGA. We see that both F_{tot} and F_{DFA} initially increase at the same rate, starting from a different minimum amount of required ROs for each board respectively. This proves the effectiveness of the calibration algorithm before each evaluation. On all boards we see, how the calibration algorithm is able to adapt to a variety of different setups with a very high precision. However, if the amount of activated ROs exceeds a certain level, the effect is too strong to allow precise injections. F_{tot} still increases with more ROs, but can affect more than one round or byte per round. Hence, the resulting ciphertext will have more than four byte faults, which can not be used anymore to recover the secret AES key in the used fault model.

4.4 Total Key Recovery Success Rates on the DE1-SoC

Subsequently, we evaluate the success of the full DFA attack including recovery of the secret AES key. This evaluation reflects on the success of our entire algorithmic flow

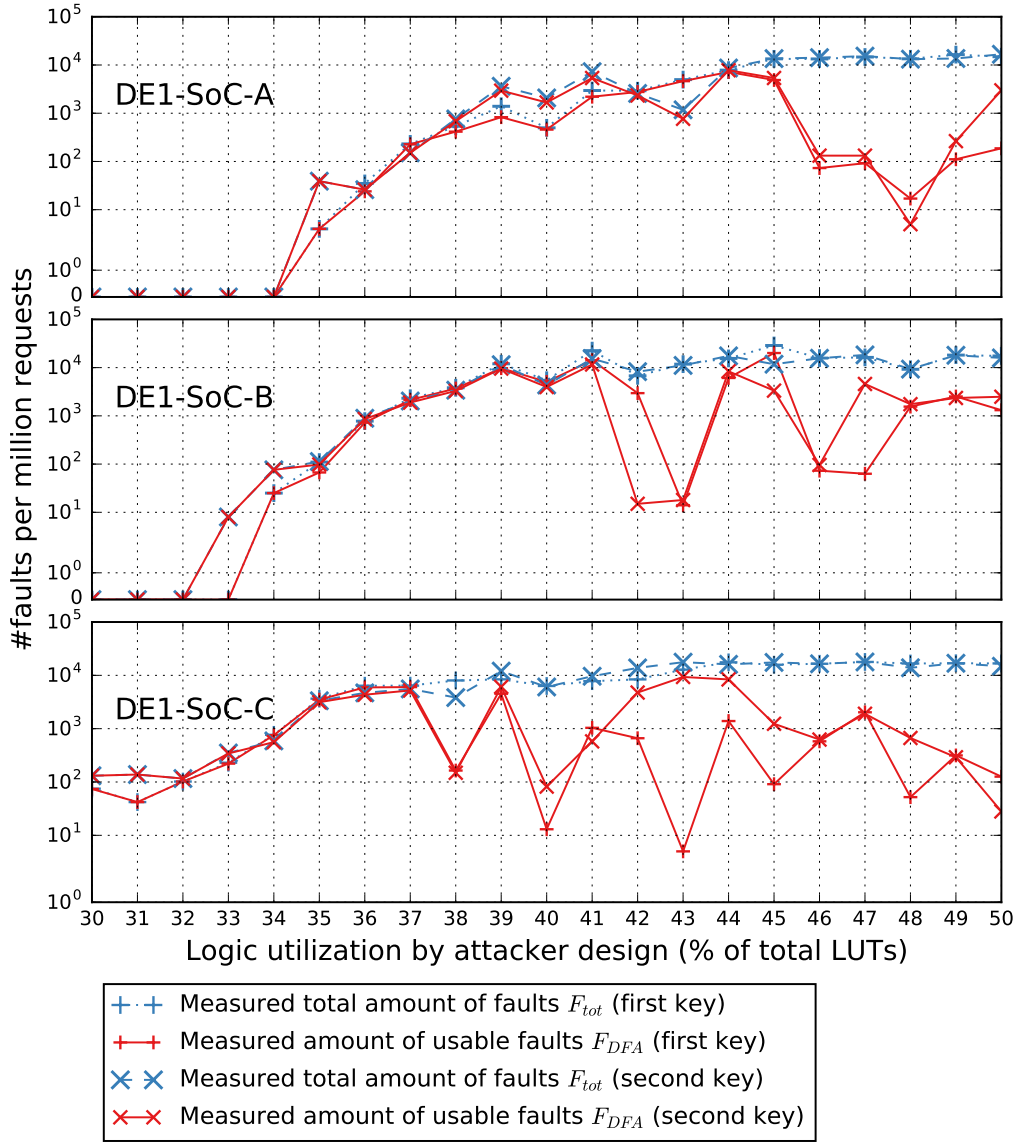


Figure 11: Total measured fault injection rates F_{tot} and measured injection rate of faults usable in DFA F_{DFA} with respect to the amount of logic utilization (percentage of total LUTs) by the attacker design for three different Terasic DE1-SoC boards and two different random encryption keys

of injecting faults before the 9th AES encryption round, the calibration algorithm and filtering of undesired faulty ciphertexts. For each of the three boards, which we already used to investigate fault injection rates, we use the amount of ROs that lead to the highest injection rate F_{DFA} of faults usable for DFA. Again, the AES module has an operating frequency of 111 MHz, where no timing violations are reported by the STA. We generate a set of 5000 random AES keys and collect a minimum of two ciphertext pairs, which exhibit faults at the desired positions, for each four bytes of the last AES round key. The ciphertexts along with each key are stored on the SD card of the board and later transferred to a host computer. After collecting the minimum amount of faults required, we apply the DFA from [PQ03] with the slight adaption of assuming single byte faults before the 9th round instead of the 8th round. In that case, we require a minimum theoretical limit of 8

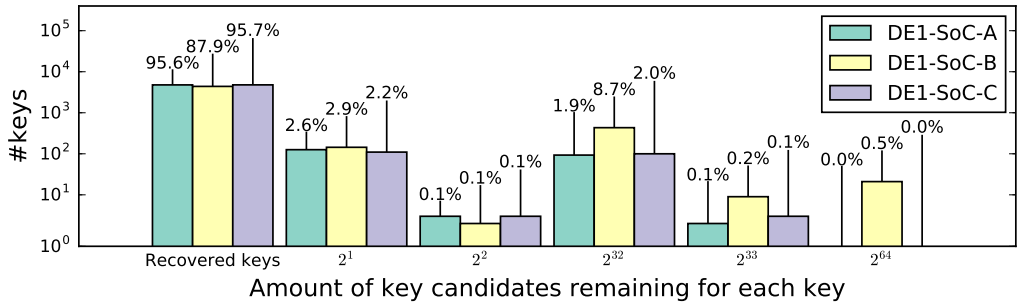


Figure 12: Amount of key candidates remaining for each DFA key recovery attempt on 5000 randomly drawn AES keys on three different DE1-SoC boards

ciphertext pairs per key.

Figure 12 summarizes the results of the key recovery attempts on our three DE1-SoC boards. On all three boards, we are able to deploy the attack completely successful for at least 87.9% of the 5000 random keys. All recovered keys are correctly recovered, so no false positives are encountered. On all boards, we have a small amount of around $2-3\%$ of all keys which can not be recovered, but less than four candidates for the last round key remain. This ratio confirms the results in [PQ03], showing that in about 2% of the cases more than two ciphertext pairs are necessary to recover the AES key. If a sufficiently small amount of key candidates remains, the correct key can be easily recovered with an exhaustive search. We encounter, however, some keys, where more than 2^{32} or even 2^{64} candidates remain. Across all our experiments, an average of 22 usable faults were required to gather the required two ciphertext pairs per four bytes of the round key. To collect these pairs, the attacker design needs to issue 17979 encryption requests on average to the AES module, which took on average 2344 ms. The average time for the evaluation of one attack until key recovery on the described host machine is about 107 ms.

Ultimately, the attack can therefore recover a secret AES key in about 90% of cases. In the remaining cases, fault injection itself fails. Our calibration algorithm and subsequent filtering of faults, which can not be used in DFA, prevents the gathering of faults that have been injected at any other stage of the AES encryption than before the 9th encryption round. However, as mentioned in Subsection 4.1, the method is unable to distinguish some multi-byte from single-byte fault injections. The adapted fault model from [PQ03] assumes single byte faults before the 9th encryption round, which is why key recovery attempts are unsuccessful, if the faulty ciphertext is the result of a multi-byte fault.

4.5 Slack-dependent Fault Injection Vulnerability of the DE0-Nano-SoC

The DE0-Nano-SoC provides about half the amount of logic elements than the DE1-SoC. Since the power supply is, as the experiments suggest, equal or at least similar to the one of the DE1-SoC, we need to utilize a huge percentage of LUTs to attack a design, which does not violate any timing constraints in the four timing models during the Quartus STA. Therefore, we additionally study the fault injection rates using an attacker design which always occupies only exactly 50% of logic resources with respect to different operational frequencies f_{op} for the victim AES module, assuming an FPGA split exactly between two users. The STA reports violation of the two worst-case timing corners (slow silicon speed, $0°/85°$ operating temperature), whereas the timing constraints are fulfilled within the fast silicon speed models.

We implement the evaluation design as described first with an initial clock frequency of $f_{\text{initial}} = 160$ MHz for the AES module and its ARM interconnect. We direct the fitter to use maximum effort to fulfill the given timing constraints for the AES module, therefore

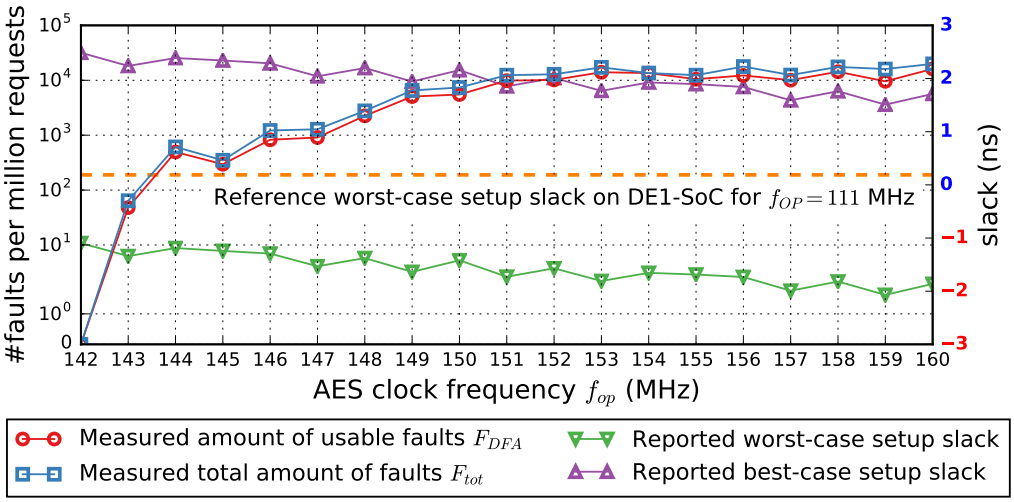


Figure 13: Total measured fault injection rates F_{tot} and measured injection rate of faults usable in DFA F_{DFA} as well as setup slacks reported by Quartus STA for different AES operating clock frequencies f_{op} with preserved design placement but remaining routing randomization on the DE0-Nano-SoC

optimizing placement of logic elements w.r.t. timing constraints. Then we prevent the fitter algorithm from changing the placement of logical components within the partitions for each recompilation of the design. The randomization during the placement algorithm can make a design running on a higher frequency cause less timing violations, than a design running on a lower frequency. However, since the routing can not be fixed completely, the routing algorithm still causes some derivation from the desired outcome.

The operating clock frequency f_{op} for the mapped and placed design with remaining routing variations is decreased in steps of 1 MHz to a frequency of $f_{\text{op}} = 142$ MHz and for each design the fault injection rates for one million total encryption requests are recorded. Furthermore, we note the setup slack values for each design as provided by the STA in the worst-case corner (slowest silicon, 85° C) and in the best-case corner (fastest silicon, 0° C). For reference, we also show the worst-case path slack value for the design running at 111 MHz on the DE1-SoC.

In Figure 13, we show the fault injection results together with the respective slack values at different operating frequencies. The results show that the reported worst-case and best-case slacks for the design do not directly correspond to the respective operating frequency f_{op} linearly, due to the remaining heuristic algorithm in the routing stage. However, the trend is that slack values increase for lower frequencies and decrease for higher frequencies. Both F_{tot} and F_{DFA} increase together with operating frequency f_{op}. However, the increase is more steep within the threshold range 145 MHz $\leq f_{\text{op}} \leq$ 151 MHz. A divergence between F_{DFA} and F_{tot} with increasing frequency is not as significant as in the experiments with respect to logic utilization. Single experiments with $f_{\text{op}} = 170$ MHz imply, however, that the injection becomes less precise with increasing f_{op} as well. We were unable to inject faults into the design running at $f_{\text{op}} = 142$ MHz.

5 Discussion and Future Work

Our results show the effectiveness of provoking voltage drops with ROs and launching a DFA attack. The most effective RO design makes use of virtual pins and benefits from added load through toggling activity on interconnect wires. However, in this work we just elaborated three variants of RO designs to cause high load. As our results suggest,

future work should look into toggling more interconnect wires. Alternatively, we may consider other on-chip methods for provoking a decrease in supply voltage. For example in [BKT10], short circuits are caused by maliciously crafted bitstreams. Those illegal bitstreams can, however, be detected by software tools easily. Furthermore, the calibration algorithm can be replaced by more advanced methods, possibly including machine learning approaches, to make injection as precise as possible. On the other hand, a more generic fault model, that covers all possible occurring faults before the 9th AES encryption round, may also raise the key recovery success rate to 100% [MSS06]. Using self-heating elements for example from [AHT14], we may also increase the temperature of the FPGA remotely, improving the fault injection rates as shown for AVR microcontrollers in [HS14].

Just like in [ZS18], where a passive side-channel attack on the integrated ARM processor of a Xilinx Zynq SoC was carried out through voltage monitors on the FPGA, we also need to consider extensions of our given threat model to hardware that is tightly coupled with the FPGA, and shares a significant part of the power supply network. In [GOT17], crashes of a full SoC could also be provoked from the FPGA part of the system. From that, we can conclude that these systems are basically vulnerable to RO based attacks. In the future, it will have to be proven if fault attacks are also possible in this configuration, or if there are reasons that the integrated CPU will crash before showing any timing violations.

Because these attacks are a risk to any multi-user FPGA applications, proper countermeasures will have to be investigated in the future. Fortunately, a remote attacker that has only access to the FPGA fabric is more limited than an attacker with full physical access. Thus, significantly increasing timing margins might be a sufficient countermeasure to this attack, with the possibly high cost of reducing the speed of the circuit. However, adding arbitrary timing margins just reduces the risk, but does not ensure no timing violations. To give more guarantees, delay elements can be added to an FPGA design, that will invalidate an output result when the design is close to timing faults [SBG+09, ELH+12]. In these cases, the cost of reducing the circuit speed might also be less, with only small area overhead.

Internal sensors allow the user to detect critical path timing failures in their designs, which has been used for dynamic voltage scaling features and detecting transistor aging [EKD+03, AT11]. Similarly, sensing timing failures can enable detection of voltage drops and delay line sensors have been already shown to detect possible fault injection attempts for local fault attacks [ZSZF13].

On the FPGA hardware design level, in the generic threat model of remote attacks on FPGAs, vendors might consider to only allow shared FPGAs when the regions of each user are residing in their own respective voltage island. However, this might defeat the purpose of an efficient way to utilize FPGA resources.

Another idea, that previous works already mentioned briefly, would require to check each bitstream that is loaded to an FPGA [GOT17, ZS18]. By essentially restricting the available basic logic circuits and, for example, prohibit or limit the implementation of combinational loops and ROs, the attack presented in our work could be mitigated. In a more complex attacker design, however, an algorithm to verify the bitstream for potentially malicious implementations would not be limited to polynomial complexity, since attacker logic can also be hidden within legitimate logic, as existing research on *hardware trojans* suggests [KRRT10]. Furthermore, the reduced flexibility of FPGA implementations may be too big of a disadvantage for this restriction to be considered as a countermeasure.

Some of the mentioned ideas might already be sufficient for mitigation, but finally, we also believe that our work is just the beginning, and more effective attacks are yet to come.

6 Conclusion

FPGAs are getting increasingly adopted in larger computing systems, as accelerators in the cloud or Systems on Chip. In such scenarios, multiple isolated users will share a single FPGA fabric and Power Distribution Network. Previous works have already shown power analysis side-channel attacks in this scenario, without requiring dedicated sensors. In this work, we additionally prove fault attacks on shared FPGAs possible by applying a Differential Fault Analysis attack on the Advanced Encryption Standard in a similar scenario, also implemented with standard FPGA tools. We demonstrated that FPGAs will not just crash if voltage drops are injected with Ring Oscillators, but in fact timing faults can be injected with sufficient precision for DFA. First, we showed an effective way to inject timing faults in simple test designs with the focus on reducing required FPGA resource use. Based on this method and given precision of the injections, we adapted an existing fault model for AES, and performed a successful DFA with a fully automated calibration of the injections. We evaluated the general injection rate and precision with respect to the percentage of logic utilization by the attacker design and the operating frequency of the target AES module. Evaluating the injection rates showed how an attacker can provoke sufficient faults for a key recovery, with logic utilization in the range of only 35% to 45% of the Look-Up Tables on a Terasic DE1-SoC board based on an Intel Cyclone V SoC. Since our calibration algorithm allows precise injections, independent of target parameters, we were able to recover at least 90% of secret AES keys from a set of 5000 randomly drawn keys on three different boards. The results in this work highlight the importance of further research, before FPGAs can be adopted widely in multi-user scenarios.

Acknowledgements

We like to thank Amir Moradi from Ruhr-Universität Bochum for allowing us to use his example AES implementation, which was also previously used in our joint work [SGMT18].

References

[AHT14] Abdulazim Amouri, Jochen Hepp, and Mehdi Tahoori. Self-heating thermal-aware testing of FPGAs. In *2014 IEEE 32nd VLSI Test Symposium (VTS)*. IEEE, apr 2014.

[AK97] Ross Anderson and Markus Kuhn. Tamper resistance-a cautionary note. In *Proceedings of the second Usenix workshop on electronic commerce*, volume 2, pages 1–11, 1997.

[ASM07] Karim Arabi, Resve Saleh, and Xiongfei Meng. Power supply noise in SoCs: Metrics, management, and measurement. *IEEE Design & Test of Computers*, 24(3):236–244, may 2007.

[AT11] Abdulazim Amouri and Mehdi Tahoori. A low-cost sensor for aging and late transitions detection in modern FPGAs. In *2011 21st International Conference on Field Programmable Logic and Applications*. IEEE, sep 2011.

[AWS] Amazon ec2 f1 instances.

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults. In *Advances in Cryptology — EUROCRYPT'97*, pages 37–51. Springer Berlin Heidelberg, 1997.

[BKT10] Christian Beckhoff, Dirk Koch, and Jim Torresen. Short-circuits on FPGAs caused by partial runtime reconfiguration. In *2010 International Conference on Field Programmable Logic and Applications*. IEEE, aug 2010.

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *Advances in Cryptology — CRYPTO'97*, pages 513–525. Springer Berlin Heidelberg, 1997.

[BSB+14] Stuart Byma, J Gregory Steffan, Hadi Bannazadeh, Alberto Leon Garcia, and Paul Chow. Fpgas in the cloud: Booting virtualized hardware accelerators with openstack. In *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 109–116. IEEE, 2014.

[Cor] Intel Corporation. Intel fpgas power acceleration-as-a-service for alibaba cloud | intel newsroom.

[EKD+03] Dan Ernst, Nam Sung Kim, Shidhartha Das, Sanjay Pant, Rajeev Rao, Toan Pham, Conrad Ziesler, David Blaauw, Todd Austin, Krisztian Flautner, and Trevor Mudge. Razor: A low-power pipeline based on circuit-level timing speculation. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, pages 7–, Washington, DC, USA, 2003. IEEE Computer Society.

[ELH+12] Sho Endo, Yang Li, Naofumi Homma, Kazuo Sakiyama, Kazuo Ohta, and Takafumi Aoki. An efficient countermeasure against fault sensitivity analysis using configurable delay blocks. In *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, sep 2012.

[EV12] Ken Eguro and Ramarathnam Venkatesan. Fpgas for trusted cloud computing. In *International Conference on Field Programmable Logic and Applications (FPL)*, pages 63–70. IEEE, 2012.

[FJLT13] Thomas Fuhr, Eliane Jaulmes, Victor Lomne, and Adrian Thillard. Fault attacks on AES with faulty ciphertexts only. In *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, aug 2013.

[FVS15] Suhaib A. Fahmy, Kizheppatt Vipin, and Shanker Shreejith. Virtualized FPGA Accelerators for Efficient Cloud Computing. In *CloudCom*, pages 430–435. IEEE, 2015.

[GMM15] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Rowhammer.js: A remote software-induced fault attack in javascript. *CoRR*, abs/1507.06955, 2015.

[GOKT16] Dennis R.E. Gnad, Fabian Oboril, Saman Kiamehr, and Mehdi B. Tahoori. Analysis of transient voltage fluctuations in FPGAs. In *2016 International Conference on Field-Programmable Technology (FPT)*. IEEE, dec 2016.

[GOT17] Dennis R. E. Gnad, Fabian Oboril, and Mehdi B. Tahoori. Voltage drop-based fault attacks on FPGAs using valid bitstreams. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, sep 2017.

[HBW+07] Ted Huffmire, Brett Brotherton, Gang Wang, Timothy Sherwood, Ryan Kastner, Timothy Levin, Thuy Nguyen, and Cynthia Irvine. Moats and drawbridges: An isolation primitive for reconfigurable hardware based systems. In *2007 IEEE Symposium on Security and Privacy (SP'07)*. IEEE, may 2007.

[HS14] Michael Hutter and Jörn-Marc Schmidt. The temperature side channel and heating fault attacks. In *Smart Card Research and Advanced Applications*, pages 219–235. Springer International Publishing, 2014.

[KDK+14] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. In *International Symposium on Computer Architecture (ISCA)*, pages 361–372, Piscataway, NJ, USA, 2014. ACM/IEEE.

[KGG+18] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. *ArXiv e-prints*, January 2018.

[KRRT10] Ramesh Karri, Jeyavijayan Rajendran, Kurt Rosenfeld, and Mohammad Tehranipoor. Trustworthy hardware: Identifying and classifying hardware trojans. *Computer*, 43(10):39–46, Oct 2010.

[MSS06] Amir Moradi, Mohammad T. Manzuri Shalmani, and Mahmoud Salmasizadeh. A generalized method of differential fault attack against aes cryptosystem. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006*, pages 91–100, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[MW17] Minh Mac and Chris Wysocki. Guaranteeing silicon performance with fpga timing models. Technical report, Intel Corporation, 2017.

[PQ03] Gilles Piret and Jean-Jacques Quisquater. A differential fault attack technique against spn structures, with application to the aes and khazad. In Colin D. Walter, Çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2003*, pages 77–88, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[SBG+09] Nidhal Selmane, Shivam Bhasin, Sylvain Guilley, Tarik Graba, and Jean-Luc Danger. WDDL is protected against setup time violation attacks. In *2009 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, sep 2009.

[SGMT18] Falk Schellenberg, Dennis R.E. Gnad, Amir Moradi, and Mehdi B. Tahoori. An inside job: Remote power analysis attacks on FPGAs. In *Proceedings of Design, Automation & Test in Europe (DATE)*, 2018.

[TSS17] Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo. CLKSCREW: Exposing the perils of security-oblivious energy management. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1057–1074, Vancouver, BC, 2017. USENIX Association.

[XMHP12] Yan Xu, Olivier Muller, Pierre-Henri Horrein, and Frederic Petrot. HCM: An abstraction layer for seamless programming of DPR FPGA. In *22nd International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, aug 2012.

[YB18] Sadegh Yazdanshenas and Vaughn Betz. Interconnect solutions for virtualized field-programmable gate arrays. *IEEE Access*, 6:10497–10507, 2018.

[YF14] Yuval Yarom and Katrina Falkner. Flush+ reload: A high resolution, low noise, l3 cache side-channel attack. In *USENIX Security Symposium*, pages 719–732, 2014.

[ZDCT13] Loic Zussa, Jean-Max Dutertre, Jessy Clediere, and Assia Tria. Power supply glitch induced faults on FPGA: An in-depth analysis of the injection mechanism. In *2013 IEEE 19th International On-Line Testing Symposium (IOLTS)*. IEEE, jul 2013.

[ZS18] Mark Zhao and G. Edward Suh. Fpga-based remote power side-channel attacks. In *2018 IEEE Symposium on Security and Privacy (SP)*, volume 00, pages 805–820, May 2018.

[ZSZF13] Kenneth M. Zick, Meeta Srivastav, Wei Zhang, and Matthew French. Sensing nanosecond-scale voltage attacks and natural transients in FPGAs. In *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays - FPGA'13*. ACM Press, 2013.

A Single Ring Oscillator Verilog Source Code

```
module osc ( enablein, dummyout );
  input enablein;
  output dummyout;
  wire enablein_lut;
  wire loop_lut, loop;
  lut_input enable_lutin(enablein, enablein_lut);
  lut_input loop_lutin(loop, loop_lut);
  lut_output loop_lutout(~loop_lut & enablein_lut, loop);
  assign dummyout = loop;
endmodule
```

Listing 1: Single RO module Verilog source code using low-level primitives to implement a two-input NAND gate

B Ring Oscillator Grid Generation Verilog Source Code

```
module osc_array ( clkin, enablein, rstin, dummyout, [...] );
  parameter amount = 10000;
  [...]
  reg enable_oscs = 1'b0;
  wire enable_oscs_g;
  //Global clock buffer routing:
  global enable_oscs_glob (.in(enable_oscs), .out(enable_oscs_g));
  output [amount-1:0] dummyout;
  genvar i;
  generate
    for (i=0; i < amount; i=i+1) begin : oscs_gen
      osc osc_inst(.enablein(enable_oscs_g), .dummyout(dummyout[i]));
    end
  endgenerate
  [...]
endmodule
```

Listing 2: Relevant parts of the RO grid generation Verilog source code with global clock buffer routing and dummy output signals

C Top-Level RO Instantiation Verilog Source Code

```
parameter ro_amount = 10000;
output [ro_amount-1:0] dummyout /* synthesis noprune=1
        altera_attribute="-name VIRTUAL_PIN ON" */;

osc_array osc_array_inst ( .clkin(clk), .rstin(rst_n),
        .enablein(enable_oscs), .dummyout(dummyout), [...] );

defparam osc_array_inst.amount = ro_amount;
```

Listing 3: Instantiation of ROs with virtual pins in the top-level module; To generate bare ROs without virtual pins, the *dummyout* output declaration is removed and the respective port of the *osc_array* module instance left unconnected