CPU to FPGA Power Covert Channel in FPGA-SoCs

Mathieu Gross[1], Robert Kunzelmann[1], and Georg Sigl[1,2]

[1] Technical University of Munich, Chair of Security in Information Technology, TUM School of Computation Information and Technology, Thresienstr. 90, 80333 Munich, Germany
{mathieu.gross,robert.kunzelmann,sigl}@tum.de
[2] Fraunhofer Research Institution AISEC, Lichtenbergstr. 11, 85748 Garching by Munich, Germany
georg.sigl@aisec.fraunhofer.de

Abstract. FPGA-SoCs are a popular platform for accelerating a wide range of applications due to their performance and flexibility. From a security point of view, these systems have been shown to be vulnerable to various attacks, especially side-channel attacks where an attacker can obtain the secret key of a cryptographic algorithm via laboratory measurement equipment or even remotely with sensors implemented inside the FPGA logic itself. Fortunately, a variety of countermeasures on the algorithmic level have been proposed to mitigate this threat. Beyond side-channel attacks, covert channels constitute another threat which enables communication through a hidden channel. In this work, we demonstrate the possibility of implementing a covert channel between the CPU and an FPGA by modulating the usage of the Power Distribution Network. We show that this resource is especially vulnerable since it can be easily controlled and observed, resulting in a stealthy communication and a high transmission data rate. The power usage is modulated using simple and inconspicuous instructions executed on the CPU. Additionally, we use Time-to-Digital Converter sensors to observe these power variations. The sensor circuits are programmed into the FPGA fabric using only standard logic components. Our covert channel achieves a transmission rate of up to 16.7 kbit/s combined with an error rate of 2.3%. Besides a good transmission quality, our covert channel is also stealthy and can be used as an activation function for a hardware trojan.

Keywords: FPGA-SoC · covert channel · Power Distribution Network · on-chip power sensors · hardware trojan

1 Introduction

Field Programmable Gates Arrays (FPGAs) are commonly used to accelerate computations in hardware. Currently, FPGAs can be found in a variety of platforms from the Cloud as extension cards [1, 2] to Embedded Systems, in so-called System-On-Chips (SoCs). FPGA-SoCs consist of multiple processing units, an

FPGA, and memory elements located in the same chip. One main characteristic of FPGA-SoCs is the sharing of resources such as memory and peripherals between the heterogeneous computation units they integrate. Besides providing advantages for a designer, this lack of isolation between resources has also been used to mount powerful attacks on FPGA-SoCs [8, 10, 13, 17, 31]. In this work, we focus on the shared Power Distribution Network (PDN) which is contained in FPGA-SoCs. Previous works have demonstrated the possibility of mounting side-channel attacks via Ring Oscillators (ROs) [31, 7] or Time-to-Digital Converters (TDCs) [23, 8] acting as on-chip power sensors implemented inside the FPGA fabric. By using one of these primitives, AES secret keys can be extracted from a victim located inside the FPGA [31, 7, 5] or in a software implementation running on an ARM Cortex-A9 CPU [8]. Besides side-channels, covert channels, which consist of building a hidden communication channel through a medium which is not intended for this purpose have been shown to be a threat to FPGAs and traditional computers. Covert channels in FPGAs have been implemented through logical resources or shared connections [22, 26]. These two types of covert channels can be mitigated via proper usage of isolation mechanisms provided by the FPGA manufacturer [3]. Furthermore, covert channels implemented through micro-architectural events can also threaten FPGA-SoC systems, especially if the FPGA can access the CPU caches [14, 20]. More generic covert channels requiring fewer assumptions and bypassing logical isolation mechanisms in the context of FPGA-SoC exploit temperature [12, 27] and power fluctuations [5, 4, 32]. A covert channel implemented via the modulation of the PDN has been considered in the work of [4, 5] with ROs acting as PDN stressors and ROs [4]/TDCs [5] acting as receivers. Additionally, a covert channel exploiting the power supply leakage between a desktop PC's CPU and an FPGA connected to a PCIe acceleration card has been demonstrated in [4]. In order to stress the power supply effectively, the authors have run multiple threads of the `stress` function available on Linux on different CPU cores. By stressing the PDN with this methodology, they implemented a covert channel achieving a transmission rate up to 6.1 bit/s combined with a 97% transmission accuracy.

1.1 Our Contribution

Similarly to [4], we consider a covert channel between a CPU and an FPGA. However, our work consider the SoC scenario where the CPU and FPGA are located on the same chip and present applications specific to this setup. By investigating this scenario, we show that the PDN can be effectively modulated from the CPU via a sequence of divisions and `nanosleep` operations running in a single thread on one CPU core. In addition, to being a stealthier implementation, our covert channel also achieves a higher transmission rate of up to 16.7 kbit/s and a corresponding bit error rate of 2.3% without requiring an explicit synchronization between the transmitter and the receiver. As an application use-case, we discuss the usage of the covert channel for the activation of a hardware trojan.

1.2 Structure of this Work

The remainder of this work is organized as follows: Section 2 explains the background information such as the threat model and the vulnerabilities resulting from a shared PDN in FPGA-SoCs. Section 3 describes the implementation of the power covert channel between the CPU and the FPGA. Section 4 characterizes the covert channel implemented in this work. We compare our covert channel to similar work and discuss its limitations and countermeasures in section 5. Finally, section 6 concludes this work.

2 Background

This section contains the background information related to this work. It first introduces the threat model and the assumptions we made for the implementation of the covert channel. Finally some information regarding Manchester code, the code we used for encoding a message through the PDN are introduced.

2.1 Threat Model

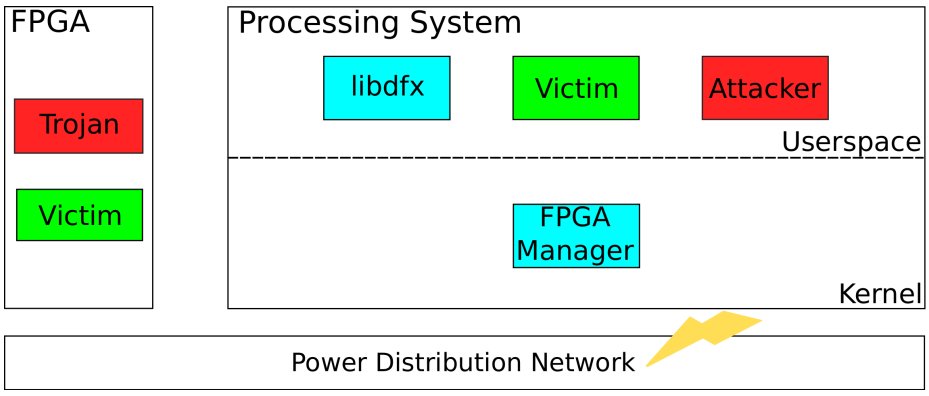


Fig. 1: Threat Model for the CPU to FPGA power covert channel

The attacker model considered in this work is depicted in figure 1. We consider a scenario where an attacker capable of executing unprivileged code on one CPU core wants to communicate with a trojan module located in the FPGA fabric. The trojan purpose is to mount an attack on the CPU, which cannot be mounted from unprivileged code executed on the CPU due to the isolation mechanisms of the operating system, lack of attack primitives available in software or due to primitives that require higher privileges for being used. Such attacks include direct memory access attacks, which have been shown to be feasible on FPGA-SoCs from the FPGA fabric because of the poor protection of memory interfaces

that are accessible from the FPGA logic [13, 10]. Another scenario could be, an unprivileged software adversary who wants to mount side-channel through power sensors implemented in the FPGA logic [8] or use special circuits such as ring-oscillators for generating voltage drops that can compromise software execution on a CPU [19, 11]. For executing one of the previously mentioned attacks, an attacker can reprogram partially the FPGA from the userspace via the *libdfx* library [29], that interacts with the *FPGA Manager* kernel driver [30]. On Xilinx FPGA-SoCs, the FPGA reconfiguration interfaces are considered as trusted under secure boot assumption [28]. Therefore, the runtime reconfiguration of the logic does not force the usage of encrypted or authenticated bitstream load even after secure boot. This relaxed trust assumption contributes to ease the hardware trojan insertion for those platforms. Once the attacker has placed a malicious IP inside the FPGA logic, she needs to communicate with the FPGA for activating the trojan in a covert way. In an FPGA-SoC, the CPU can use the AXI bus for communicating the activation signal, however this communication channel is not coverted and is therefore not suitable for activating a trojan. Furthermore, a suspicious communication on the AXI bus can be easily blocked by a firewall placed on the AXI bus as proposed in [13]. For that purpose, we present a methodology to covertly communicate between the CPU and the FPGA with the help of a PDN modulator software running on the CPU (see section 3.2) and a decoding logic implemented within the FPGA logic (see section 3.3).

2.2 Security Vulnerabilities Resulting from a Shared PDN

The power dissipation of a chip can be divided into a static part which is proportional to the current and its variation and a dynamic part which is influenced by the toggling of the transistors. Like every power supply, the PDN cannot deliver a constant voltage to an FPGA-SoC. Instead, the delivered voltage is dependent on the current demand. Previous works have exploited this property to mount fault and denial-of-service attacks based on voltage-drop by using power-wasting circuits in FPGAs [17, 18].

In order to observe the voltage variations resulting from the PDN, the inversely proportional relation between the supply voltage and the propagation delay of a signal can be used. The propagation delay variation of a clock signal can be measured in a so-called delay-line circuit, which acts as a voltage sensor. One example of such a circuit is the TDC circuit [6] represented in figure 2.

The TDC depicted in figure 2 measures the propagation delay of a clock signal inside a circuit consisting of an initial delay and a chain of delay elements, which constitute the delay line. Latches and registers are used to depict the propagation of the clock signal inside the delay line during one clock period. The delay line state is then reflected as a thermometer code inside the registers. In case of a voltage decrease, the clock signal propagates less inside the delay line, which is seen by a decrease in the delay line state's Hamming Weight. Inversely, if the voltage raises, the delay line state's Hamming Weight increases. By using this principle, TDCs can be effectively used as voltage sensors and have been used

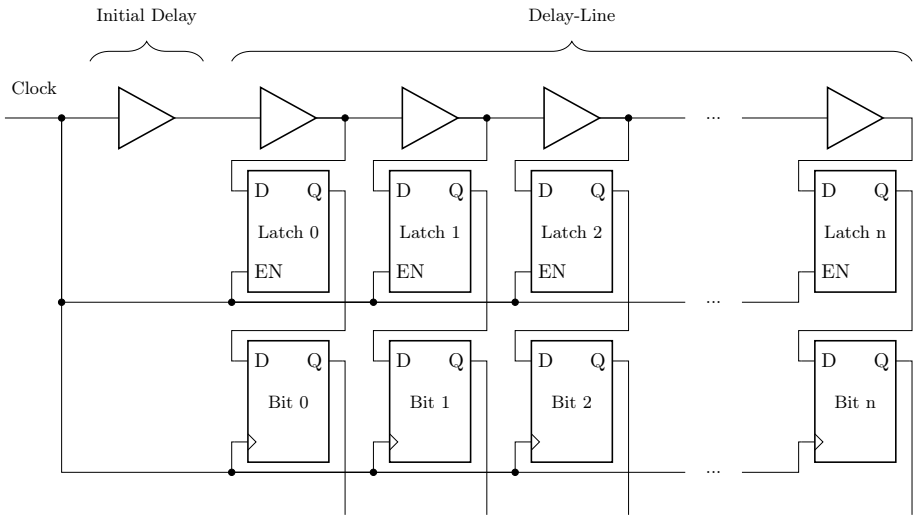


Fig. 2: Schematic of an n-bit delay-line TDC with an initial delay unit

for side-channel attacks against a cryptographic core located in the FPGA [23] or a software implementation running on a CPU [8].
Covert channels implemented through the PDN exploit both capabilities of stressing and observing the PDN through logic circuits implemented within the FPGA. In the work of [5, 4], ROs have been used as PDN stressors and TDCs respectively ROs as PDN observers. Power covert channels were also demonstrated to be feasible between a PC's CPU and an FPGA mounted on an acceleration card in [4]. By modulating the shared power supply usage via the Linux `stress` and `sleep` functions, the authors implemented a covert channel with a transmission rate of up to 6.1 bit/s and a 97% transmission accuracy. In this work, a similar scenario is evaluated in the SoC context, with the FPGA and CPU co-located on the same chip.

2.3 Manchester Code

The transmission of a message inside the PDN requires an encoding scheme. For FPGA platforms, the On-Off keying encoding has been demonstrated to be efficient for a temperature covert channel [27]. In the context of a PDN covert channel, this simple encoding scheme has also shown to be efficient for a pure FPGA-to-FPGA power covert channel [5]. For a more generic power covert channel implementation involving FPGAs, Giechaskiel et al. suggest the usage of the Manchester code which is less prone to transmission errors [4]. Based on this evaluation, we opted for the Manchester encoding scheme for implementing a PDN covert channel in this work. The Manchester code defines a format to physically represent bits on a transmission line. In the Manchester code, logical zeros are encoded in a falling signal edge, whereas logical ones are represented

by rising signal edges. The level transition occurs in the middle of the bit-period i.e., it is aligned with the rising clock edge of a shared clock signal. Due to this alignment capability, the Manchester code is a so-called self-clocking code. The clock signal can be extracted from the data signal itself. Thus, depending on the implementation, a shared clock signal or synchronized clocks on the transmitter and receiver side are not required [24].
In this work, we have implemented a covert channel using the shared PDN, meaning only a single transmission line for communication is available. Hence, we make use of the self-clocking property of the Manchester code, synchronizing the transmitter and receiver without a shared clock signal.

3 Power Covert Channel Implementation

This section describes the implementation details of the PDN covert channel presented in this work. After a brief description of the experimental setup, a more detailed explanation of the transmitter and receiver design is presented.

3.1 Experimental Setup

The experimental setup used in this work is the Pynq-Z1 board from Digilent. This platform contains a Xilinx Zynq-7000 SoC which features a dual-core ARM Cortex-A9 CPU running at 650 MHz together with a Xilinx Artix-7 FPGA clocked at 300 MHz in our experiments. The SoC is connected to an external 512 MB DDR3-RAM chip. We supplied the board with power through micro USB instead of using an external power supply. The transmitter software is running on an Ubuntu 18.04 operating system. For an evaluation of the transmission quality, we read the decoded bitstream from Linux and store them inside logfiles. These logfiles are then downloaded for an offline analysis on a standard PC.

3.2 Transmitter Design

The transmitter software aims at modulating the usage of the PDN by varying the CPU load. In [4], the open-source application `stress` has been used for generating voltage drops on the shared power supply by imposing load on several CPU cores during a given duration with matrix multiplication operations. We verify that this methodology can also be applied to our platform, however it has the downside of leading to a low transmission rate, which is inherent to the usage of the `stress` tool. The tool indeed needs to be run in seconds granularity which prevents the achievement of a high transmission rate. Initial experiments revealed that a sequence of division instructions were sufficient for generating a voltage drop which is significant enough for implementing a covert communication. After testing several strategies for encoding a high voltage level, we opted for the usage of the `nanosleep` function. During the execution of `nanosleep` we observed an initial voltage drop for 15 µs followed by a raised and a constant

high level which depends on the given duration and a final voltage drop for another 15 µs (see figure 3). Fortunately, this behavior matches the Manchester code specification which requires a level transition after either a half or a full period. The PDN response to `nanosleep` prevents the transmission of discrete bits by iterating through the bitstream to transmit. Instead of that, we used the translation table (see table 1) which takes the current bit value to transmit, its predecessor and successor to encode a sequence of instructions which modulates the PDN according to a Manchester encoding (see section 2.3). Additionally, for a bitstream of size n we assume that the predecessor of the first bit is 0 and the successor of the last bit is 1. The sleeping durations of 30 µs and 60 µs, as well as the number of divisions, are derived from the low voltage level between two sequential `nanosleep` function calls. From figure 3 the duration of this voltage level can be read off to be 30 µs. As a result, the period of the Manchester signal must be double this time. Thus, the software must modulate high voltages for 30 µs and 60 µs by executing the `nanosleep` function with the respective duration. Low voltage levels of 30 µs and 60 µs are modulated by the delay between two `nanosleep` executions and an extension of this delay by executing 1200 integer divisions. Figure 4 depicts the waveform corresponding to the transmission of the bitstream *011*, which is obtained by executing instructions on one ARM Cortex-A9 core following the encoding contained in table 1. In conformity to the Manchester code specification, only transitions occurring at the middle of a bit-period carry information (cf. section 2.3). Therefore, the falling edge occurring after 120 µs in figure 4 does not encode a bit.

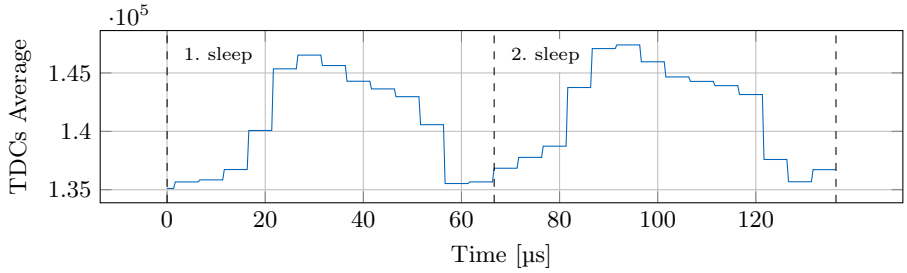


Fig. 3: Averaged TDCs' measurements during two consecutive `nanosleep` executions using Linux

3.3 Receiver Design and Message Decoding

This section describes the components involved in the receiver block. The receiver logic running at 300 MHz consists of TDC sensors and a decoding logic which is represented in figure 5. A particular focus is made on the FSM, which is used for detecting edges corresponding to a valid bit transmission and the corresponding bit value according to the Manchester code specification (see section 2.3).

Predecessor	*Value*	*Successor*	*Instruction*
0	0	don't care	sleep for 30 µs
0	1	0	divide 1200 times, sleep for 60 µs
0	1	1	divide 1200 times, sleep for 30 µs
1	0	don't care	-
1	1	0	sleep for 60 µs
1	1	1	sleep for 30 µs

Table 1: Translation of a bit under consideration of its direct neighbors into a instruction list, used for voltage modulation

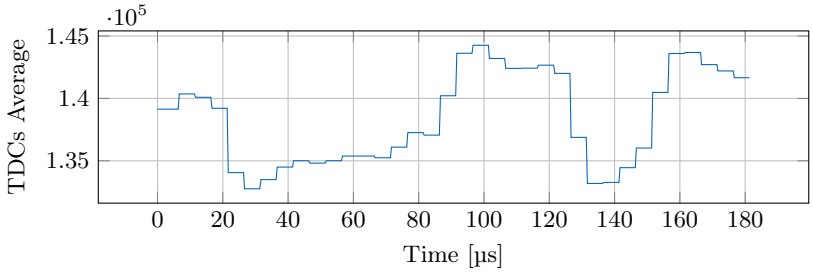


Fig. 4: Signal waveform resulting from the execution of the instruction list translated from the bitstream *011*

3.3.1 PDN monitoring: For monitoring the variations of the PDN, we use TDC sensors. One TDC sensor consists of a chain of 16 CARRY-4 elements as delay-line and 2 LUTs-6 elements as initial delay (see figure 2 and section 2.2 for the explanation of the PDN monitoring via TDC sensors). The output of a TDC sensor is further fed into an encoder so that it can be represented in binary code. Using multiple TDCs and averaging the measurements lead to a better voltage fluctuation coverage and measurement quality. However it also results in additional noise generated by the TDC sensors which in turn decreases the measurement quality. Therefore a trade-off has to be found with the number of TDCs and the resulting measurement quality. Previous work [8] investigating side-channels on a Zynq-7000 Processing System via TDC sensors found that the usage of 8 adjacent TDC sensors placed at the left-hand side of the FPGA produce the best measurement quality. We used this configuration as a baseline and performed further experiments in section 4.2.4, to evaluate the influence of the placement of the sensors on the covert channel quality.

3.3.2 Averaging and shift register: The first block of the decoding logic (visible in figure 5) is a block-averaging mechanism which can be seen as a low pass filter that is applied to the noisy TDCs' measurements. The approach consists in summing 1500 samples from the 8 TDCs. Using this approach rather than a more complex low pass filter is sufficient since the covert channel transmission frequency is much lower than the TDCs' sampling frequency. The averaging

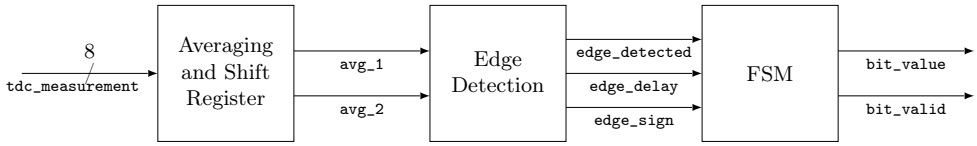


Fig. 5: Block diagram of the decoder logic with intermediate signal names

block is followed by a shift register of size 2, which keeps the current (`avg_1`) and previous block-averaged values (`avg_2`).

3.3.3 Edge detection: The signal edge detection is implemented as a gradient-based mechanism which compares the difference between two block-averaged values. If the absolute value of this difference is higher than a fixed threshold, the `edge_detected` signal is set high for one clock cycle. Falling edges are encoded as 0 whereas rising edges are encoded as 1 in the `edge_sign` signal. Furthermore, a counter is used to reflect the delay between two consecutive edges. According to the Manchester code specification, a signal can only show a constant level for either a half or a full bit-period. Therefore, this threshold is set to 3/4 of a bit-period. In practice, no clock is shared between the decoder logic and the transmitter code running on the CPU. Moreover, the non-determinism of the Linux operating system might cause slight variations in the bit-period. Consequently, we determined the threshold value empirically. If the delay between two detected edges exceeds this threshold, the `edge_delay` signal is set high for one clock cycle.

3.3.4 FSM: The final step of the decoder logic is an FSM that gets the decomposed signal and returns the decoded bit values. To determine whether an edge in the Manchester code actually encodes a bit, we require knowledge about the previous edges in the signal (cf. section 2.3). Hence, we use an FSM since it stores the information about the previous edges in the currently active state. Figure 6 visualizes the Mealy FSM with two input and two output bits. The first input represents the delay passed since the previous edge with zero meaning a half bit-period passed and a one signalizing an entire bit-period passed. The second input bit shows the edge sign where falling edges are represented by a zero and rising edges by a one. The first output bit finally stores the decoded bit value and the second output is set high whenever the decoded bit is valid. The FSM is triggered asynchronously whenever the upstream edge detection logic detects an edge.
The main function of the FSM is to decode the Manchester encoded signal. If an edge is detected and it does not encode a bit, one of the two *pre-** states is entered and the validation output is set low. This prevents subsequent logic blocks from accepting the current decoder output. Contrarily, if an edge is detected that encodes a bit the corresponding *valid-** state is entered and the validation output is set high. The distinction of whether an edge encodes a bit or not is

accomplished based on the currently active state. According to the Manchester code definition, edges in the data signal that encodes bit values must be aligned with the rising edge of the shared clock signal. Since our implementation only has a single transmission line and no shared clock signal we extract a virtual clock signal from the data signal. This virtual clock signal is stored in the first input bit which stores the delay passed since the previous edge was detected. If one of the *valid-** states is active it means the last detected edge did encode a bit and was consequently aligned with the rising edge of the virtual clock signal. If the next edge is detected after a half bit-period it will be aligned with the falling edge of the virtual clock signal. Hence, it does not encode a bit and the corresponding *pre-** state is entered. But in case the next edge is detected after an entire bit-period it will be aligned with a virtual rising clock edge again thereby encoding a bit value.
Moreover, the FSM has the function to resynchronize the receiver with the transmitter. If an edge is missed or an additional one is detected the decoder FSM might enter a false state and incorrectly decode subsequent edges. To cope with this issue, we use a property of the Manchester code that unambiguously shows if an edge decodes a bit or not. If a delay of an entire bit-period between two edges is detected, both edges must encode a bit according to the Manchester code. Consequently, whenever an edge after a delay of an entire bit-period is detected the FSM enters the corresponding *valid-** state, regardless of the currently active state.

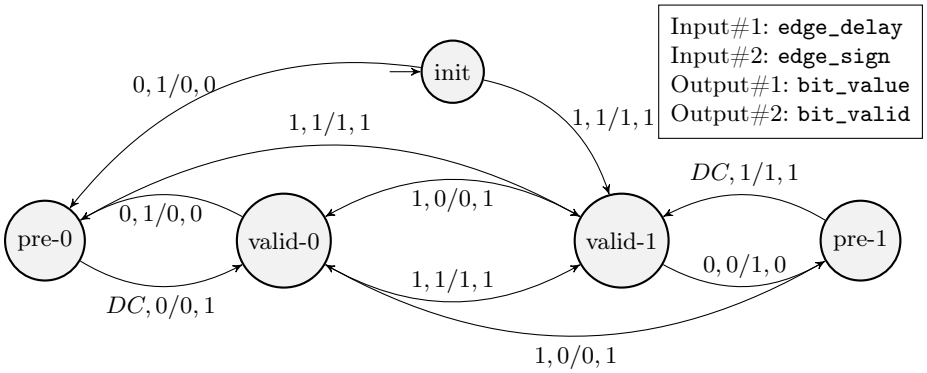


Fig. 6: FSM to determine which signal edges encode to an actual bit value

3.3.5 Decoder Control The current implementation uses control signals for configuring the decoder and starting and stopping the decoder logic. The configuration of the decoder consists in specifying the block size used for averaging, the thresholds for the detection of edges, and the delay between two consecutive edges. All these parameters are configurable via software by writing into

specific AXI-addressable registers. This enables flexibility in the decoder configuration without the necessity of regenerating a bitstream for a different decoder configuration.
In addition, the start and stop signals used for a transmission are also encoded into AXI-addressable registers. These registers are system-wide accessible, which violates the concept of covert channel presented in section 2.1. In future work, the start and stop signals used for the covert communication should be encoded in the PDN via a specific start and stop bit pattern.

4 Power Covert Channel Characterization

In this section, the performance and transmission quality of the covert channel are evaluated. Furthermore, we analyze the influence of the sensors placement on the covert channel characteristics and present the FPGA resource usage of the sensors and decoder logic.

4.1 Data Rate Limitations

The communication performance is represented by the achieved transmission rate. The bit-period of the covert communication is derived from the transmitter implementation. As shown in section 3.2 the minimal duration of a constant signal level is 30 µs. Using the Manchester code results in a bit-period of 60 µs since every bit is transmitted as a combination of a high and low signal level of equal duration. Consequently, this results in a data rate of 16.7 kbit/s.
The lower bound of 30 µs for a constant signal level is due to the presented behavior of the `nanosleep` function (cf. figure 3). It is important to note that this bound does not correspond to the physical limits of the used device. Implementing the transmission software as a bare-metal program instead of a Linux application results in an increased transmission rate of 47.1 kbit/s.

4.2 Transmission Quality

To determine the quality of the communication channel we examine the ratio of falsely detected bits i.e., bit error. This allows a fine granular analysis of the conditions under which errors are especially likely to occur. Additionally, we determine the word success rate, meaning the ratio of correctly transmitted words. Both metrics are calculated from a set of 10 000 word transmissions.

4.2.1 Bit Error vs. Word Size

Figure 7 shows the bit error for the four different word sizes from 8 to 64 bit. As expected, the smallest evaluated word size of 8 bit results in the lowest ratio of falsely transmitted bits of 2.3 %. Furthermore, figure 7 shows an almost linear increase of the bit error with increasing word sizes. This matches the results of Gnad et al. [5]. Their FPGA-to-FPGA covert channel also shows a nearly linear dependency between the bit error and the width of the transmitted word.

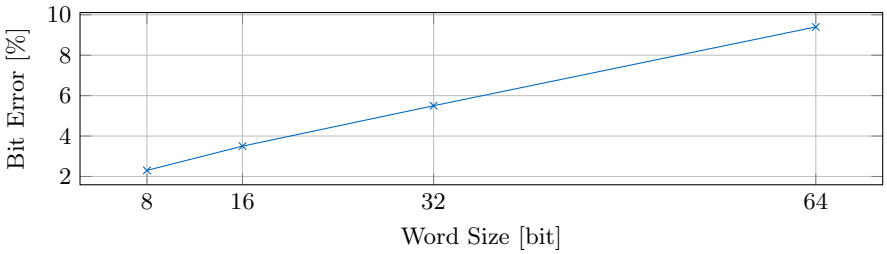


Fig. 7: Relative bit error in percent against different word sizes, calculated form a set of 10000 samples

4.2.2 Bit Error Distribution

As a metric to evaluate the communication quality, we measure the bit errors occurring in a set of 10 000 transmissions. For that purpose, we transmit 64-bit wide words in sequence and calculate the relative amount of falsely detected bits for every index of the words.

Figure 8 presents the bit error evaluation results. The bar plot depicted in blue shows the bit error distribution for a set of randomly generated words. Since every single word in this set is independent of every other word, it is possible to derive the general dependence of the bit error on the index and word size. The bit error distribution measured after transmitting random words in figure 8

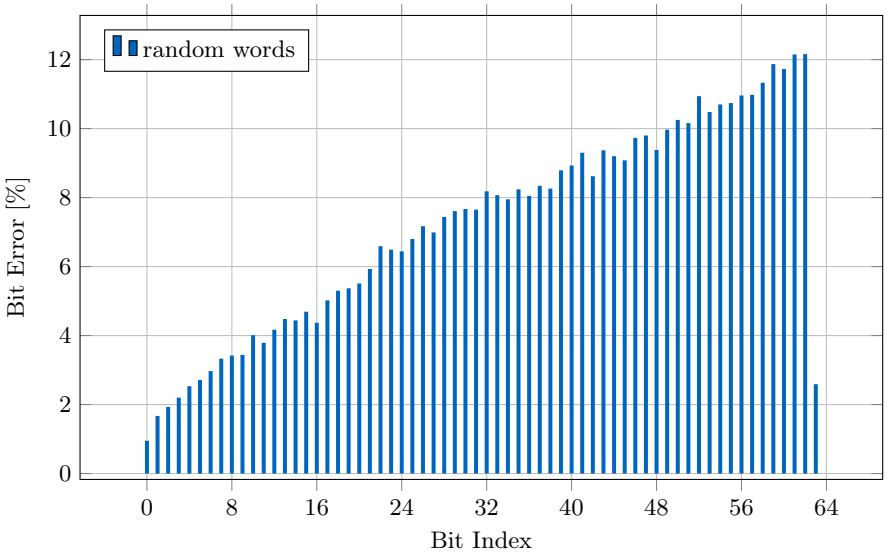


Fig. 8: Relative bit error in percent against the position of the respective bit in a 64-bit wide word

shows a linear increase with rising index. This effect can be explained by the occurrence of synchronization errors. The PDN covert channel uses only a single transmission line and does not have a shared clock signal. The Manchester code in combination with the implemented decoder FSM enables the communication without synchronized clocks on the transmitter and receiver sides. However, this approach is prone to detecting unintentional edges and missing intentional edges in the data signal. If such a detection error occurs, the FSM might enter a false state and is not able to decode the following edges correctly. Furthermore, even after a resynchronization of the FSM, the signal pattern is decoded correctly but the data might be aligned at an incorrect index. Consequently, bit errors at one specific position in the data word induce further errors at the subsequent indices. Summing the individual bit error at an index and the bit errors at upstream indices results in the linear increase shown in figure 8.

4.2.3 Word Success Rate

A further metric that represents the quality of the communication channel besides the bit error is the word success rate. This characteristic describes the relative number of successfully decoded words in comparison to the total number of transmitted words.
Figure 9 shows the word success rate against different word sizes. Transmitting byte-sized words results in the highest success rate of 94.5 %. It is visible that the success rate decreases almost linearly with increasing word sizes. The moderate gradient is surprising since two known effects contribute to a decrease in the word success rate. First, with a longer word size the probability that at least one bit error occurs increases. Secondly, the bit error distribution shows higher numbers of errors in wider words suggesting a decrease in signal quality.

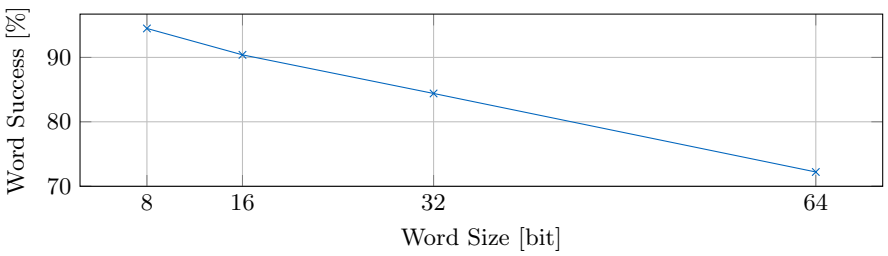
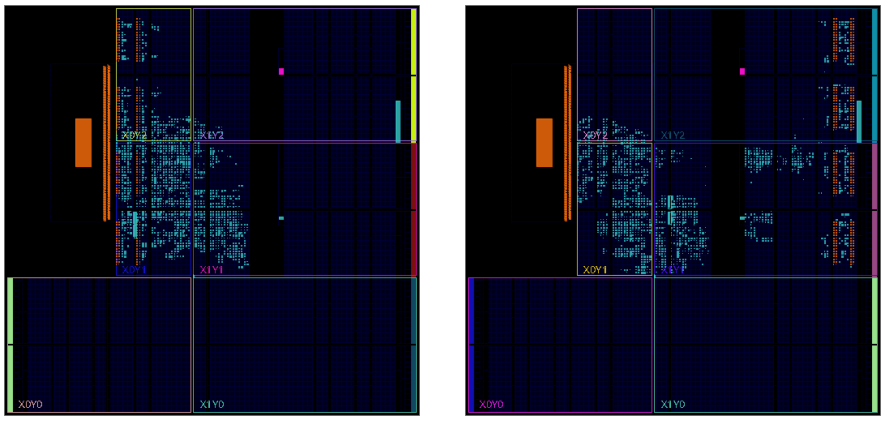


Fig. 9: Word success rate in percent against different word sizes, calculated form a set of 10000 samples

4.2.4 Influence of the Sensors Placement To determine whether the physical location of the TDC sensors inside the FPGA influences the communication quality, we chose two distinct placements on opposite sides of the FPGA (see

figure 10). In both positions, all 8 TDCs are placed near each other. For the first setup, the slices directly next to the CPU are used to implement the TDCs. This leads to a placement of the sensors on the left-hand side of the FPGA fabric. For the second location, the sensors are placed on the far-most right-hand side of the FPGA. The two positions are 80 slices apart horizontally.



(a) (b)

Fig. 10: Placement of the eight TDCs (orange) a) next to the CPU and b) far away from the CPU

Figure 11 shows the comparison of these two placements in terms of communication quality. Figure 11a depicts the signal waveforms measured by the TDCs. The upper blue curve corresponds to the TDCs' placement next to the CPU. The lower orange curve shows the waveform measured with the sensors placed on the opposite side of the FPGA. Comparing both measurements shows that the orange waveform is scaled down by 25% in comparison to the blue waveform. Consequently, the signal edge height stays constant relative to the signal level. This shows that in both positions the TDCs measure a significant voltage variation when the PDN is modulated by the CPU. The downscaled voltage measurements can be explained by a non-uniform PDN. Hence, the supply voltage varies slightly due to the design of the PDN across the SoC. Comparable results are presented in the work of Krautter et al. [15] who exhaustively analyzed the influence of the transmitter and receiver placement on the quality of an intra-chip side-channel attack. They found that the power distribution is not uniform within the chip, which can result in a different transmission quality, that is not necessarily influenced by the physical distance to the transmitter.
Since we use a gradient-based approach to detect edges in the data signal and the downscaling of the TDCs' measurements results in a reduced gradient value, a negative influence on the communication quality is expected. A comparison between the word success rate, i.e. the relative number of correctly transmitted

data words of the two different TDC placements is shown in figure 11b. It shows that the downscaling of the edge height due to the different sensor positions does not present a problem for small word sizes. In contrast, transmitting wider words result in a significantly decreased word success rate. Here, the high number of bits and therefore the increased error probability in combination with a more susceptible signal-to-noise ratio result in a lower communication quality.

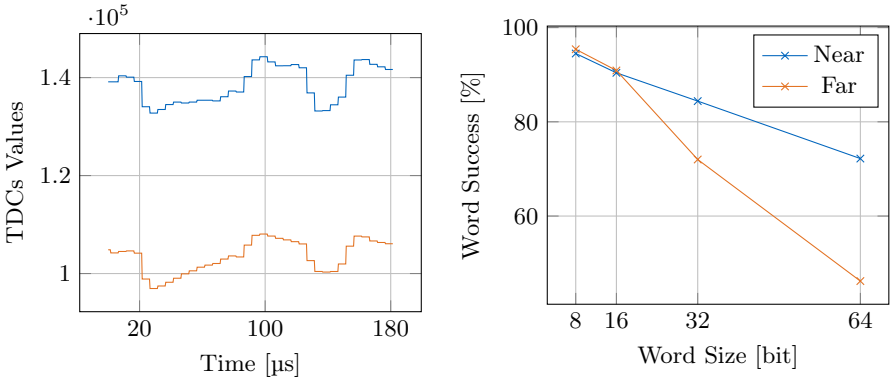


Fig. 11: Comparison of a) the signal waveforms and b) word success rate between the placement of the TDCs next to the CPU (blue) and far away (orange)

4.3 Resource Utilization

As shown in table 2, the receiver and decoder logic can be implemented with a small resource usage, using only 2.35% of the available FPGA LUTs.
On the CPU side, the transmitter is implemented with only one thread performing a sequence of divisions and `nanosleep` function calls. Moreover, the capability of achieving a good transmission at a high bandwidth without having to resend a message multiple times contributes to make the transmitter code stealthy on the CPU usage.

Type	***Amount***	***Utilization***
Slices	555	4.17 %
LUTs	857	2.35 %
Registers	1364	1.28 %

Table 2: Resource utilization caused by the receiver and decoder logic

5 Discussion and Future Work

The following section first presents a comparison between the CPU to FPGA covert channel implemented in this work and other state-of-the-art covert channels shown on FPGAs. In a second time, the usage of the implemented covert channel as an activation function for a trojan is discussed. Finally some considerations regarding noise and potential countermeasures are presented.

5.1 Comparison with Other Power Covert Channels Involving FPGAs

Temperature [12, 27] and power consumption [5] are the most promising transmission medium that can be used as covert channel on FPGA platforms since they can bypass FPGA isolation mechanisms. The temperature covert channel uses high and low temperature level to encode bits and has been shown to be practicable on standard FPGA platforms [12] upto FPGA platforms integrated in the cloud [27]. Its downside is mainly the achieved transmission speed, with which several minutes are required to transmit a 128 bit AES key [27]. A faster transmission medium relying on the FPGA power consumption is presented in the work of Gnad et al. [5]. Their implementation uses ROs in a custom logic circuit to modulate the supply voltage. A TDC sensor, programmed into the same FPGA fabric, observes the PDN. Since they also use the Pynq-Z1 as their evaluation platform, we can use their results to classify our covert channel. In comparison to our implementation, their covert channel achieves data rates up to 8 Mbit/s. They use the same transmission medium on the same hardware and an equivalent receiver circuit. Therefore, we can derive that the transmitter software executed on the CPU is the main performance limitation. This supports the observation presented in section 4.1, showing an increased transmission rate when executing the software as a bare-metal program instead of a Linux application. Moreover, Gnad et al. are able to generate three distinct voltage levels whereas our covert channel uses rising and falling level edges between two voltage levels. Consequently, using a custom logic circuit to stress the PDN results in fine granular control over the PDN in terms of timing and level modulation.
A different approach towards power covert channels was taken by Giechaskiel et al. [4]. Instead of implementing both the transmitter and receiver on the same chip and using the shared PDN as the transmission medium, they have used a computer PSU. The transmitter and receiver are implemented using discrete devices which are either placed directly on the motherboard or connected via PCIe acceleration cards. This setup results in a more complex transmitter and receiver design. While we are able to modulate the supply voltage using only a single core of an embedded CPU, they have required multiple threads of a desktop-class CPU. Moreover, the receiver that is implemented on a discrete FPGA requires additional circuitry to measure deliberate voltage variations. They have used additional ROs, stressing the voltage regulators to make the supply voltage more vulnerable to high CPU loads. A receiver of this complexity is not required for our covert channel. As shown in section 3.3, voltage variations

are directly measurable using TDC sensors with a simple block averaging scheme to filter high-frequency noise.
In conclusion, the threat of a power covert channel heavily depends on the transmitter implementation and the nature of the transmission medium. The shared PDN of a single chip is especially vulnerable to covert channel communication. In comparison to a common PSU, exploiting the shared PDN by implementing the transmitter and receiver on the same chip shows significant improvements in the achievable transmission rate. Furthermore, it allows a simplified transmitter and receiver design.

5.2 Activation of a Hardware Trojan via the Covert Channel

Hardware trojans consist of malicious circuits hidden within a benign design. In the context of FPGA-SoCs, they can typically be contained in IP cores obtained from third parties. The insertion of HTs in FPGA-SoCs is also facilitated due to the relaxed trust assumptions on the FPGA reconfiguration interfaces made on Xilinx FPGA-SoCs [28]. Xilinx considers the PCAP and the ICAP as trusted in the context of secure boot. Therefore, it is possible to load un-authenticated and un-encrypted bitstram after a secure boot process due to these relaxed trust assumptions [28]. After its insertion, a hardware trojan should remain discrete and only activated under specific conditions, which are not reproducible during normal operation. The activation signal should also be communicated via an indirect communication channel. In this work, we have the capability to encode a chosen bitstream in the PDN via the bit to instruction mapping presented in section 3.2. To ensure that the transmitted bitstream cannot be reproduced during normal conditions and still be transmitted reliably, the activation bitstream should be large enough, transmittable in a short duration, and have a good word success rate. With the analysis presented in sections 4.2.1 and 4.2.3, we think that a trigger signal of 16 bits can be a good trade-off for ensuring those requirements. In section 5.3, the influence of noise which may degrade the activation signal transmission quality is discussed together with techniques that can be investigated in future work for improving the transmission quality in presence of noise.

5.3 Influence of Noise and Countermeasures

Besides the analysis in terms of performance and quality, further topics are still worth investigating. Currently, the used FPGA-SoC is operated in an ideal state for the covert channel implementation. Gnad et al. [5] showed that a PDN covert channel can be implemented with noise sources located within the FPGA. In future work, we should verify if noise sources generated by logic inside the FPGA can disturb the reception of the message encoded in the PDN by the CPU. In addition to noise on the receiver side, noise on the transmitter software should also be considered. In the current evaluation, no major application is running on the Linux operating system during the experiments. One transmitter thread is executed using only one of the two available cores of the integrated ARM Cortex-A9

and is not running in parallel or being preempted by another thread. A promising strategy to deal with thread preemption could be the multi-threading of the transmitter signal, as presented in [4]. While the results presented in [4] haven't explicitly considered preemption of the transmitter code, they showed an increase in the transmission quality by considering multiple threads running *stress*, which is the PDN stressor the authors used for encoding bits in their covert channel implementation. Future work should investigate if running multiple transmission threads increase the transmission quality at the price of a decrease in the implementation stealthiness and if it enables the run of parallel victim workload during the covert transmission.
The implementation of countermeasures in the context of a PDN covert channel remains challenging. In contrast to side-channel attacks, where the implementation to be protected is clearly defined and can be masked with adequate techniques [21, 9], equalizing a hidden PDN transmitter on a CPU or within the FPGA logic is more difficult. One approach could be to use a bitstream analyzer scheme which prevents the insertion of power sensors within the FPGA logic [16]. However, it was shown that power sensing circuits that are harder to detect can still be used despite this approach [25].

6 Conclusion

In this work, we have shown that the PDN is a vulnerable resource that can be used for implementing a covert channel between a CPU and an FPGA on an FPGA-SoC platform. This communication channel is particularly interesting for the activation of hardware trojans in FPGA-SoCs. By using simple sleep function calls and integer divisions, we were able to modulate the PDN usage in a stealthy way. The hidden message encoded in the PDN can then be decoded within the FPGA using TDC sensors and a decoder logic. Overall, the presented covert channel achieves a transmission rate of up to 16.7 kbit/s and a bit error rate of 2.3%, which is a significant improvement in comparison to other CPU to FPGA covert channels. Future research should evaluate the robustness of the covert channel to noise sources running in parallel to the transmitter software and investigate possible countermeasures.

References

1. Amazon ec2 f1 instances, `https://aws.amazon.com/ec2/instance-types/f1/`
2. Instance family in alibabacloud, `https://www.alibabacloud.com/help/en/elastic-compute-service/latest/instance-family`
3. The xilinx isolation design flow for fault-tolerant systems, `https://www.xilinx.com/content/dam/xilinx/support/documents/white\_papers\\/wp412\_IDF\_for\_Fault\_Tolerant\_Sys.pdf`

4. Giechaskiel, I., Rasmussen, K.B., Szefer, J.: C3apsule: Cross-fpga covert-channel attacks through power supply unit leakage. In: IEEE Symposium on Security and Privacy. pp. 1728–1741. IEEE (2020)
5. Gnad, D.R.E., Nguyen, C.D.K., Gillani, S.H., Tahoori, M.B.: Voltage-based covert channels using fpgas. ACM Trans. Des. Autom. Electron. Syst. **26**(6) (Jun 2021). https://doi.org/10.1145/3460229, https://doi.org/10.1145/3460229
6. Gnad, D.R.E., Oboril, F., Kiamehr, S., Tahoori, M.B.: An experimental evaluation and analysis of transient voltage fluctuations in fpgas. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **26**(10), 1817–1830 (2018). https://doi.org/10.1109/TVLSI.2018.2848460
7. Gravellier, J., Dutertre, J.M., Teglia, Y., Loubet-Moundi, P.: High-speed ring oscillator based sensors for remote side-channel attacks on fpgas. In: 2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig). pp. 1–8 (2019). https://doi.org/10.1109/ReConFig48160.2019.8994789
8. Gravellier, J., DUTERTRE, J.M., Teglia, Y., Loubet-Moundi, P., Francis, O.: Remote Side-Channel Attacks on Heterogeneous SoC. In: Smart Card Research and Advanced Applications, 18th International Conference, CARDIS 2019. Prague, Czech Republic (Nov 2019), https://hal.archives-ouvertes.fr/hal-02380092
9. Gross, H., Mangard, S., Korak, T.: Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In: Proceedings of the 2016 ACM Workshop on Theory of Implementation Security. p. 3. TIS '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2996366.2996426, https://doi.org/10.1145/2996366.2996426
10. Gross, M., Jacob, N., Zankl, A., Sigl, G.: Breaking trustzone memory isolation and secure boot through malicious hardware on a modern fpga-soc. Journal of Cryptographic Engineering (Sep 2021). https://doi.org/10.1007/s13389-021-00273-8
11. Gross, M., Krautter, J., Gnad, D., Gruber, M., Sigl, G., Tahoori, M.: Fpganeedle: Precise remote fault attacks from fpga to cpu. In: Proceedings of the 28th Asia and South Pacific Design Automation Conference. p. 358–364. ASPDAC '23, Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3566097.3568352, https://doi.org/10.1145/3566097.3568352
12. Iakymchuk, T., Nikodem, M., Kępa, K.: Temperature-based covert channel in fpga systems. In: 6th International Workshop on Reconfigurable Communication-Centric Systems-on-Chip (ReCoSoC). pp. 1–7 (2011). https://doi.org/10.1109/ReCoSoC.2011.5981510
13. Jacob, N., Heyszl, J., Zankl, A., Rolfes, C., Sigl, G.: How to break secure boot on fpga socs through malicious hardware. In: Cryptographic Hardware and Embedded Systems – CHES 2017. Lecture Notes in Computer Science, vol. 10529, pp. 425–442. Springer (2017). https://doi.org/10.1007/978-3-319-66787-4_21
14. Kocher, P., Horn, J., Fogh, A., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M., Yarom, Y.: Spectre attacks: Exploiting speculative execution. In: 2019 IEEE Symposium on Security and Privacy (SP). pp. 1–19 (2019). https://doi.org/10.1109/SP.2019.00002
15. Krautter, J., Gnad, D., Tahoori, M.: Cpamap: On the complexity of secure fpga virtualization, multi-tenancy, and physical design **2020**, 121–146 (Jun 2020). https://doi.org/10.13154/tches.v2020.i3.121-146, https://tches.iacr.org/index.php/TCHES/article/view/8585

16. Krautter, J., Gnad, D.R.E., Tahoori, M.B.: Mitigating electrical-level attacks towards secure multi-tenant fpgas in the cloud. ACM Trans. Reconfigurable Technol. Syst. **12**(3) (aug 2019). https://doi.org/10.1145/3328222, https://doi.org/10.1145/3328222
17. Krautter, J., Gnad, D.R.E., Tahoori, M.B.: Fpgahammer: Remote voltage fault attacks on shared fpgas, suitable for DFA on AES. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(3), 44–68 (2018)
18. La, T., Pham, K., Powell, J., Koch, D.: Denial-of-service on fpga-based cloud infrastructures — attack and defense. IACR Transactions on Cryptographic Hardware and Embedded Systems **2021**(3), 441–464 (Jul 2021). https://doi.org/10.46586/tches.v2021.i3.441-464, https://tches.iacr.org/index.php/TCHES/article/view/8982
19. Mahmoud, D.G., Hussein, S., Lenders, V., Stojilović, M.: Fpga-to-cpu undervolting attacks. In: 2022 Design, Automation Test in Europe Conference Exhibition (DATE). pp. 999–1004 (2022). https://doi.org/10.23919/DATE54114.2022.9774663
20. Maurice, C., Neumann, C., Heen, O., Francillon, A.: C5: Cross-cores cache covert channel. In: Proceedings of the 12th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9148. p. 46–64. DIMVA 2015, Springer-Verlag, Berlin, Heidelberg (2015). https://doi.org/10.1007/978-3-319-20550-2_3, https://doi.org/10.1007/978-3-319-20550-2_3
21. Nikova, S., Rechberger, C., Rijmen, V.: Threshold implementations against side-channel attacks and glitches. In: Ning, P., Qing, S., Li, N. (eds.) Information and Communications Security. pp. 529–545. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
22. Provelengios, G., Ramesh, C., Patil, S.B., Eguro, K., Tessier, R., Holcomb, D.: Characterization of long wire data leakage in deep submicron fpgas. In: Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. p. 292–297. FPGA '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3289602.3293923, https://doi.org/10.1145/3289602.3293923
23. Schellenberg, F., Gnad, D.R., Moradi, A., Tahoori, M.B.: Remote inter-chip power analysis side-channel attacks at board-level. In: 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). pp. 1–7 (2018). https://doi.org/10.1145/3240765.3240841
24. Stallings, W.: Data and computer communications (2007), https://memberfiles.freewebs.com/00/88/103568800/documents/\\Data.And.Computer.Communications.8e.WilliamStallings.pdf
25. Sugawara, T., Sakiyama, K., Nashimoto, S., Suzuki, D., Nagatsuka, T.: Oscillator without a combinatorial loop and its threat to fpga in data centre. Electronics Letters **55**(11), 640–642 (2019). https://doi.org/https://doi.org/10.1049/el.2019.0163, https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/el.2019.0163
26. Sun, J., Bittner, R., Eguro, K.: Fpga side-channel receivers. In: Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays. p. 267–276. FPGA '11, Association for Computing Machinery, New York, NY, USA (2011). https://doi.org/10.1145/1950413.1950462, https://doi-org.eaccess.ub.tum.de/10.1145/1950413.1950462
27. Tian, S., Szefer, J.: Temporal thermal covert channels in cloud fpgas. In: Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. p. 298–303. FPGA '19, Association for Computing Machinery,

New York, NY, USA (2019). https://doi.org/10.1145/3289602.3293920, https://doi.org/10.1145/3289602.3293920
28. Xilinx: Developing Tamper-Resistant Designs with Zynq UltraScale+ Devices (August 2018), xAPP1323 (v1.1)
29. Xilinx: libdfx - linux user space solution for fpga programming. https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/1959854475/libdfx+-+Linux+User+Space+Solution+for+FPGA+Programming (2022)
30. Xilinx: Solution zynq pl programming with fpga manager. https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841645/Solution+Zynq+PL+Programming+With+FPGA+Manager (2022)
31. Zhao, M., Suh, G.E.: Fpga-based remote power side-channel attacks. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 229–244 (2018). https://doi.org/10.1109/SP.2018.00049
32. Ziener, D., Baueregger, F., Teich, J.: Using the power side channel of fpgas for communication. In: 2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines. pp. 237–244 (2010). https://doi.org/10.1109/FCCM.2010.43