Karlsruher Institut für Technologie

Voltage-based Covert Channel Communication between logically separated IP Cores in FPGAs

Bachelor Thesis of

Cong Dang Khoa Nguyen

at the Department of Informatics
Institute of Computer Engineering
Chair of Dependable Nano Computing

Reviewer:	Prof. Dr.-Ing. Mehdi Tahoori
Advisor:	M.Sc. Dennis Gnad

Tag der Anmeldung:	01. April 2018
Tag der Abgabe:	31. August 2018

Erklärung

Ich versichere hiermit wahrheitsgemäß die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt zu haben. Alle benutzten Hilfsmittel sind vollständig angegeben, und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichtem oder unveröffentlichtem Schrifttum entnommen sind, habe ich als solche gekennzeichnet.

Karlsruhe, den 31. August 2018

[illegible]

Cong Dang Khoa Nguyen

Abstract

Field Programmable Gate Arrays (FPGAs) are increasingly used in systems like Systems-on-Chip (SoCs) or cloud computing. In these systems users are getting remote access to programmable blocks of the FPGA, which can be customized to their needs. Nowadays, one FPGA is assigned to one user, but in the near future there will be multiple users on one FPGA, which creates new security vulnerabilities at an electrical level. A successful side-channel attack was proven by using malicious IP-cores or a loaded software accelerator containing a Time-to-Digital Sensor (TDC-Sensor), which can measure the voltage fluctuations. These information and a correlation analysis is already enough to successfully get the secret keys of an AES implementation. There is also a successful fault attack, in which several Ring Oscillators(ROs) are used to crash FPGAs inside System-on-Chip (SoC) systems. Before multi-user FPGAs are being used frequently, as many security vulnerabilities as possible must be found. In this thesis the possibility to implement a voltage-based Covert Channel (CC) between logically separated IP-Cores in FPGAs is explored. To do so a concept is presented, where the two components from the successful attacks are used as sender (ROs) and receiver (TDC-Sensor) of the CC. The ROs are turned on and off in a specific way which depends on the dataword being sent, to create a specific waveform of the voltage fluctuations in the "Power Distribution Network" (PDN) so that the TDC-Sensors can read it and then, can be interpreted into data. Finally, the working transmission is tested in reliability.

Zusammenfassung

Field Programmable Gate Arrays (FPGAs) werden zunehmend in Systemen wie System-on-Chip (SoCs) oder Cloud Computing eingesetzt. In diesen Systemen erhalten Nutzer Zugriff auf programmierbare Bausteine des FPGAs aus der Ferne, die an ihre Bedürfnisse angepasst werden können. Heutzutage wird ein FPGA einem Nutzer zugewiesen, aber in naher Zukunft wird es mehrere Nutzer auf einem FPGA geben, wodurch neue Sicherheitlücken auf elektrischer Ebene entstehen. Ein erfolgreicher Seitenkanal-Angriff wurde durch den Einsatz bösartiger IP-Cores oder eines geladenen Software-Beschleunigers, der Time-to-Digital-Sensoren (TDC-Sensoren) enthält, der die Spannungsschwankungen messen kann, nachgewiesen. Diese Informationen und eine Korrelationsanalyse reichen bereits aus, um die geheimen Schlüssel einer AES-Implementierung erfolgreich herauszulesen. Es gibt auch einen erfolgreichen Fehlerangriff, bei dem mehrere Ringoszillatoren (ROs) verwendet weren, um FPGAs in System-on-Chip-Systemen(SoC) zum Absturz zu bringen. Bevor FPGAs immer häufiger von mehreren Nutzern gleichzeitig genutzt werden, sollten so viele Sicherheitslücken wie möglich gefunden werden. In dieser Arbeit wird die Möglichkeit untersucht, einen spannungsbasierten verdeckten Kommunikationskanal (CC) zwischen logisch getrennten IP-Cores in FPGAs zu implementieren. Dazu wird ein Konzept vorgestellt, bei dem die beiden Komponenten der erfolgreichen Attacken als Sender (ROs) und Empfänger (TDC-Sensor) des CCs verwendet werden. Die ROs werden auf eine bestimmte Art und Weise ein- und ausgeschaltet, die von dem gesendeten Datenwort abhängt, um eine spezifische Wellenform der Spannungsschwankungen im "Power Distribution Network"(PDN) zu erzeugen, so dass die TDC-Sensoren es lesen und als Daten interpretiert werden können. Schließlich wird die Übertragung durch den CC auf Zuverlässigkeit getestet.

Contents

1. Introduction

Field Programmable Gate Arrays (FPGAs) are increasingly used in systems like Systems-on-Chip (SoCs) or cloud computing. In these systems users are getting remote access to programmable blocks of the FPGA, which can be customized to their needs. Nowadays, one FPGA is assigned to one user, but in the near future there will be multiple users on one FPGA, which causes security issues. There are "Intellectual Property Cores" (IP-Cores), which are already programmed blocks by different developers and have different functions. Since anyone can create an IP-core, it is possible that some of those are created for a malicious purpose, hence they are not trustworthy. So theoretically it is possible for IP-Cores to contain hardware-trojans. If a user installs the malicious IP-Cores, hardware-trojans can infiltrate its systems only through software (i.e. FPGA bitstreams), which is a security problem.
It has already been shown that isolation primitives on the logical level can increase security between IP-Cores. However, the state of the art isolation mechanism *Moats and Drawbridges* [1] was already proven in [2] and [3] to not have proper security against attacks through the electrical level, even when only standard FPGA-logic are used.
The voltage drop-based fault attack [3] uses several ROs to crash the FPGA. By turning ROs on and off voltage fluctuations in the PDN are caused. So by turning all ROs on and off at a specific frequency which produces huge voltage fluctuations that the PDN can not handle and thus the FPGA crashes.
The side-channel attack [2] uses TDC-Sensors (TDCs), which can measure the voltage fluctuations of the PDN, caused by processes and runtime variations. These information and a correlation analysis is already enough to successfully get the secret keys of an AES implementation. The TDCs can be implemented by using malicious IP-Cores or a loaded software accelerator.

1.1. Motivation

With the voltage drop-based fault attack and the side channel attack, it is already proven, that security threats in a single FPGA accessed by multiple users exist. With this thesis the goal is to show another possible security threat by using the already working components of those two successful attacks to create a Covert Channel between two logical isolated FPGA-blocks to exchange information between these systems. Ring Oscillators (ROs) can be used as data senders, while TDCs act as data receivers.
To achieve this goal, the first thing to do was to experiment with different activation sequences of the Ring Oscillators, to shape different voltage fluctuations of the PDN and then use the best shape for the transmission. After that, the stability and the speed of the communication in the covert channel is tested.

1.2. Overview

The content of this thesis is split into six chapters. The first chapter contains the introduction to the thesis. The second chapter contains the background information and the related work about relevant components and tools, such as the FPGA-board, Hardware Description Language (HDL), Power Distribution Network, Ring Oscillators and TDC-Sensor. Followed by the basics of an IP-Core and a Covert Channel and some more details on an existing temperature-based Covert Channel (CC). After that the construction of both sender and receiver are shown and explained.
The third chapter explains the theoretical methods of how to successfully design an electrical Covert Channel in two different blocks in the FPGA.
The third chapter shows the construction of the design followed by different experiments to test how the changes of different RO activation influences the changes in the voltage of the PDN. While doing these experiments the design has been implemented piece by piece. After the Covert Channel was implemented stress tests were run. These tests run 100000 transmissions in a row and write the result in a csvfile. In the fourth chapter the results of the tests are shown and discussed. These were then being evaluated. The last chapter contains the conclusion.

2. Background and Related Work

2.1. FPGA

FPGA (Field Programmable Gate Array) is a semiconductor device which is widely used in electronic circuits. Different than other integrated circuits e.g. Application Specific Integrated Circuits (ASICs), which are manufactured for a specific design task and can not be reprogrammed, FPGAs contain programmable logic blocks and interconnection circuits, which can be reprogrammed after manufacturing. So FPGAs contain circuit which do not have any functions yet, unless they are being designed. To create a design we use a Hardware Description Language (HDL) like Verilog and VHDL. Then the HDL is synthesized into a bit file using a bitgen to configure the FPGA. The configuration is stored in the FPGA-internal configuration and must be configured every time power is supplied [4].

2.2. HDL

A Hardware Description Language (HDL) is a type of computer language used to describe the behavior and structure of electronic systems like ASICs, FPGAs and other digital circuits. HDL enables a formal description of electronic circuits which can be analyzed and simulated. The HDL description can also be synthesized into a netlist and then be placed and routed to create an integrated circuit. HDL looks like a programming language with an important difference, that HDL is inherently parallel, and requires the designer to manually control the sequence in which computations are performed, i.e. in which clock cycle. The two most used HDLs are Very High Speed Integrated Circuit Hardware Description Language (VHDL) and Verilog [5].

2.3. Power Distribution Network

A Power Distribution Network (PDN) is responsible for the power supply of modern ICs. A PDN has resistive, capacitive and inductive components at board level and on-chip. Through different operating conditions the power consumption changes, which causes the voltage in the network to be unstable. Since the power consumption changes, the supply current and voltage will change, too, which can be measured depending on the power. ($P = V \cdot I$) Differences in power causes voltage fluctuations in the PDN. These fluctuations affects circuit delay, which can be measured to reveal power consumption information about a running workloard. This information can be used e.g. to recover secret keys [2].

2.4. Ring Oscillator

Ring Oscillators (ROs) are cascaded combinations of delay stages which are connected in a close loop chain. In [3] it was shown, that ROs, implemented in an FPGA, can change the voltage of the PDN, and induce a crash of an FPGA. In this work we actively use the ROs to influence the voltage of the PDN and shape them into waveforms, which the TDC-Sensor can interpret. To find out, how to best influence the voltage, many experiments were run, in which different kind of RO-Activation were tested while the voltage of the PDN was being observed.

2.5. IP Cores

An intellectual property core (IP core) is a predefined block of logic or data. It can be used multiply for FPGAs or ASICs and can be portable across technology generations. Examples for IP cores are Universal Asynchronous Receiver/Transmitter (UART), central processing unit (CPU), Ethernet controllers and PCI interfaces. There are three different kind of IP cores: hard cores, firm cores and soft cores. Soft cores are the most flexible and exist as a netlist or HDL code. Hard cores are fixed and unchangeable IP designs. The performances in speed, surface needed and power consumption is optimized, but these cores are less portable and flexible. The firm cores a mixture of both hard and soft cores [6] [7].
Modern System on Chip (SoC) design often integrate a lot of IP-cores into a single chip.

Also multi tenant FPGAs integrate IP cores from multiple users [8]. Anyone can create an IP-core, so it is possible that some of those are created for a malicious purpose, e.g. a IP-core can contain TDC-Sensors, which can read the voltage values from a PDN.

2.6. Covert Channel

Covert channels are a way to secretly exfiltrate information from a system, and thus fall in the category of computer attacks. They allow hidden communication by using an existing medium to convey data in small parts, which is virtually undetectable. They add a small amount of data to a data stream without affecting the main body. A package might contain as little as 1-2 bits of payload data.

There are different types of CCs.

Covert storage channel
CC that use shared resources, which uses the attribute of the shared resource.

Cover timing channel
CC, which uses the temporal/ordering relationship among accesses to a shared resource, e.g. it uses a real-time clock to measure intervals between accesses. Relative ordering of events are covert timing channels, too.

Temperature covert channel
CC, which uses the temperature to communicate. It creates heat patterns, which spread through the system, so it can be received at a different position.

Electrical covert channel
CC, which uses voltage to communicate. It creates specific voltage fluctuations in the PDN, which can be interpreted by decoder.

Noiseless CC
CCs that only the sender and the receiver have access to. In noiseless CCs the receiver gets the information only from the sender and in noisy CCs the sender's information is mixed with meaningless information from others entities. That's why it additionally requires a protocol to minimize these interference from other entities.

Noisy CC

CCs that other besides the sender and the receiver have access to.

The key properties which all CCs have in common are the existence (does a channel where information can be transmitted exist?) and the bandwidth (how fast can information be sent) of a CC [9].

2.7. Covert Channels in FPGAs

In FPGAs so far, temperature-based covert channels have been shown [10]. This channel enables circuits located in distant areas to exchange information. The basic idea of a thermal based CC is to insert a thermal transmitter in the secure IP-core, isolated via *Moats and Drawbridges* [1]. The thermal transmitter encodes secret data (encryption key) by creating heat patterns correlated to the transmitted data. The heat is able to spread through the silicon die and the chip package. So the external receivers can get these covert transmission in those heat form. As seen in Fig. 2.1 primitives which were used for the construction of a temperature-based covert channel:
Transmitter module: This module consists of the data encoder and the heater. The data encoder encodes the data into the heater control data which is used to control the operation of the heater. Thus, it provides timing control of the transmitter. The heater consists of ROs and shift registers (SR).
Receiver module: The components of this module are a RO, a counter and a small control logic. This module is used to detect the thermal transmission. The RO cycles are recorded by a 16-bit counter. The counter represents the frequency of the RO, which determine the temperature. The counter is decoded in two steps. In step 1 the decoder decides on the transmitted bit and in step 2 the received data stream is decoded. The receiver also has a last block to decode error detection.
In this work, electrical instead of thermal covert channels will be used, with the main similarity, that they use a CC to communicate between secure IP-cores in FPGAs, which are isolated via moats. They also use ROs for the transmitter module. While their communication is thermal based, this work does the same, but is voltage based. It is also most similar to the covert timing channels, because it hides information in voltage instead of timing variations.

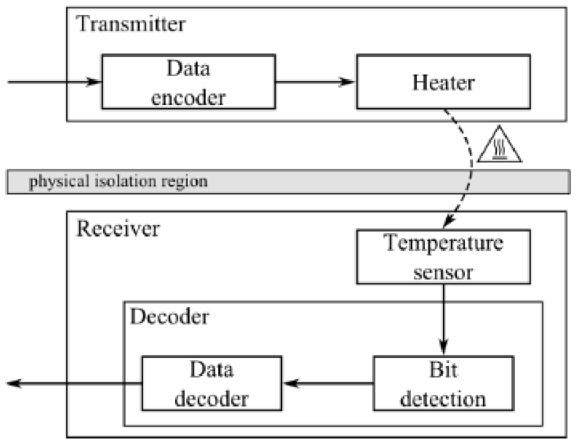


Figure 2.1.: Thermal Channel Communicator [10]

2.8. Ring Oscillators for Fault Attack

The basic idea of the fault attack in [3] is to try crashing FPGAs by generating voltage drops temporarily with huge current difference. This is achieved by activating many ROs and let them toggle. The system gets unstable and crashes within a few milliseconds. For their implementation only a small part of the FPGA was used (8-35 percent). This procedure was tested on many FPGA Boards. Every tested FPGA crashed after these voltage drops, some of them crashed permanently until they were power-cycled. Possible reason for the crash are the resonance of the voltage regulator, faults inside the JTAG module or loss of the configuration in the SRAM.
As shown in Fig. 2.2, a Xilinx LUT5 FPGA component is configured as an inverter and their input is connected their output in a loop. This configuration forms a RO that can oscillate at a very high frequency. To toggle the RO on and off, there is an AND in the LUT with an additional input "enable" signal [3]. Standard FPGA tools were used to implement this RO. In this work the same ROs are being used for the sender module. The construction is shown in 3.3.

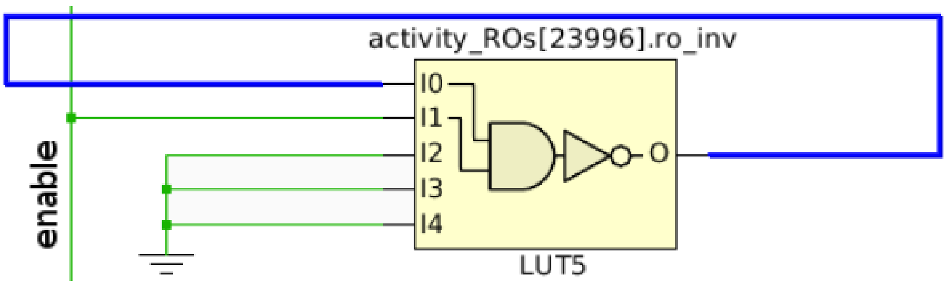


Figure 2.2.: Single ring oscillator [3], implemented using a Xilinx LUT5 FPGA component

2.9. TDC-Sensors for Side-Channel Attack

The basic idea of the power-based side-channel attack in [2] is to measure the voltage fluctuations, temporary caused by processes and runtime variations, with voltage variation sensors. Then the voltage traces were used to correlate with a hypothesis of the secret keys. The more voltage traces there is, the higher the chances to deduce the key. This experiment was carried out at a Sakura-G board (FPGA). The voltage variation sensor was first placed nearby the attacked core and then in a position which was further away of that core. After sending a random plain text to the AES core, the correlation between the cipher text from the core with the voltage data resulted in a successful key recovery attack in both differently positioned sensors. As shown in Fig. 2.3, a custom sensor based on the concept of measuring propagation time of signal with Time-to-Digital Converters (TDCs) is used to sense variation in supply voltage.
The sensor uses a delay line, where a clock signal goes through a chain of buffers. Since the delay of these buffers depends on the supply voltage, we can just monitor the buffers. By adding latches between these buffers the delay line can be tapped. The latches has the same clock as input as the buffers. When other circuits on the same PDN becomes active, it will cause a voltage drop which slows down the buffers of the delay line. Thus, the numeric value of the TDC register will fall [2].
In this work the same TDC-Sensors are used for the receiver module. The construction is shown in 3.4.

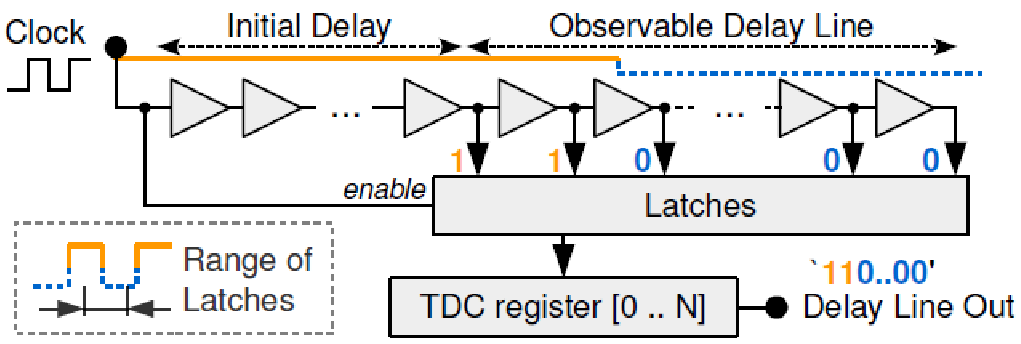


Figure 2.3.: TDC-Sensor [11], to sense voltage fluctuations through variations in the speed of the buffers

3. Methods for Voltage-based Covert Channels in FPGAs

3.1. Overview

The idea of creating a voltage-based Covert Channel is to use the voltage behavior of the PDN as the medium, which can be read and influenced by circuit designs in different logically separated IP-cores. In this work the ROs in [3], which act as sender, are used to shape the voltage into a specific form like a signal code. The TDC-Sensors in [2], which act as receiver, are used to read the voltage values of the PDN which can then be interpreted by a decoder.
Before a full data transmission is possible, some preliminary experiments are required. Firstly, toggling the amount of designed ROs need to cause voltage fluctuations. Secondly, the voltage fluctuations have to be large enough to be distinguishable from noise. Finally, a good way of shaping the voltage fluctuations has to be found, so it is similar to a form of signal codes. Overall, the following software and hardware are used to achieve the goal in this work:
The FPGA part of the circuit board called PYNQ-Z1 [12] is used to implement the CC design. To create the design the software Vivado 2017.1 from Xilinx is used. It uses a synthesis tool to create a netlist with the given code in HDL. Then it calculates an optimal way to place the netlist in the given board. After that, a bitstream file is generated, which can be used to program the given board. The design in this work is written in VHDL.

3.2. Signal Coding

This section mentions preferences for the signal coding briefly. The actual signal coding which is used, is described in section 4. Because the behavior of the voltage fluctuations for different kind of RO toggling is unknown, to find proper Signal Coding, it has to be tested and monitored first. The ideal way of Signal Coding would be something like line codes, e.g. the "Non Return to Zero" (NRZ) line code [13] or something close to it, because it is already in use and is working. So given a dataword with 8-bits, for all one bits the RO should use a specific activation sequence to cause the voltage to be high and for all zero bits the voltage should be low. Hence, two different kind of RO toggling has to be found and then combined as a sequence.

3.3. Construction of Sender

This section describes the construction of the sender briefly. This will be described in more details along with the experiments in section 4. For the construction of the sender the ROs in [3] is used. In total, 2504 ROs are being used to influence the voltage of the PDN. These are separated into 8 groups, that can be controlled independently. By that, the strength of a voltage drop can be controlled in 8 levels. The process of the ROs activation is being handled by a state machine. The state machine iterates through each bit of the dataword and decide how to activate the ROs. When the machine goes through the dataword, it will be in a finished state and will deactivate the ROs. It also has two waiting states, one makes sure that the transmission of each bit is done first, before transmitting the next bit. The other waiting state is used to adjust the placement of the RO activation 4.1, so that the maxima and minima would be at the same distance to each other.

3.4. Construction of Receiver

This section describes the construction of the receiver briefly. This will be described in more details along with the experiments in section 4. For the construction of the receiver the TDC-Sensors in [2] are used. This also includes the calibration of these sensors, to notice differences in the voltage of the PDN and the TDC-decode to decode it to binary values that can be further processed in the system. In this work six TDC-Sensors are being used with each 63 latches. Boundaries are identified in an experimental way, to find reasonable upper and lower bounds that can be generated from the ROs. These values are than being interpreted by a decoder. It decides which bit is being sent. The decided bit then is written to a transmitted word array.

3.5. Combining Sender and Receiver

The sender and the receiver are placed in two different and logically separated IP-cores, which uses the PDN as a communication medium. While the sender was sending the data-word of a specific length by shaping the voltage in the PDN in a given form, the receiver was trying to extract the dataword out of the read voltage fluctuations of the PDN. The order of the transmission is as follows. At first a dataword is given for the transmission. Then for each bit of the dataword the ROs are activated in a given order depending on the bit. While that is happening, the decoder is getting the read voltage data from the TDC-Sensor. It interprets the data by having upper and lower boundaries and only recognizes values in the upper bound as 1 bit and values in the lower boundaries as 0 bit. After the transmission is done and the dataword is received and written in the transmitted word array, the ROs will stop.

4. Experiments and Implementation

4.1. Overview

Fig. 4.1 shows the complete floorplan of the implemented design in the PYNQ-Z1 board. In this figure there are three areas. The upper area contains the logic of the TDC-Sensors. The lower area contains the logic of the ROs. In the middle area there are logic which handles the process of the transmission. This contains the state machine which handles the activation of the ROs, the logic for the timeframe, the boundaries, the dataword which is being sent and the transmitted word.

Fig. 4.2 shows the display of the the Integrated Logic Analyzer (ILA) [14]. We use the *mark_debug* VHDL attribute that allows us to monitor the marked signals in ILA. On the left side of Fig. 4.2, all selected signals are displayed, together with their value at the current marker position(288). In this case there are signals for the timeframe and the boundaries displayed as *timeframe_start[7:0]*, *timeframe_end[7:0]*, *LOWER_BOUND[7:0]* and *UPPER_BOUND[7:0]*. The signal *active_part_reg[7:0]* shows if the ROs are activated or not. The signal tdc_s shows the voltage changes. To see the changes in the voltage, the TDC-Sensor has to be calibrated first. After the calibration, the value of the *tdc_s[5:0]* is typically between 36 and 38. There are cases where the value is way under or over the usual value,but which has to be corrected in the calibration mechanism and is not part of this thesis. If ROs are being activated the value of the tdc_s will change. At the top of the waveform is a timescale. This shows how many clock pulses have passed already. In this experiment a 125 MHZ clock is used. This means, that 8 nanoseconds will pass after one clock pulse. Under that indication the values of the signals are being displayed for every clock pulse which is displayed in an analog waveform.

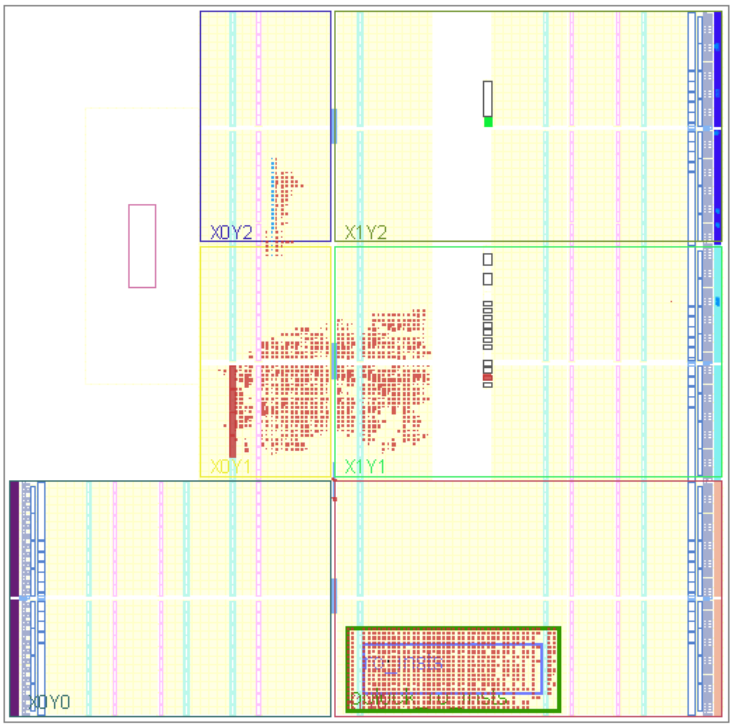
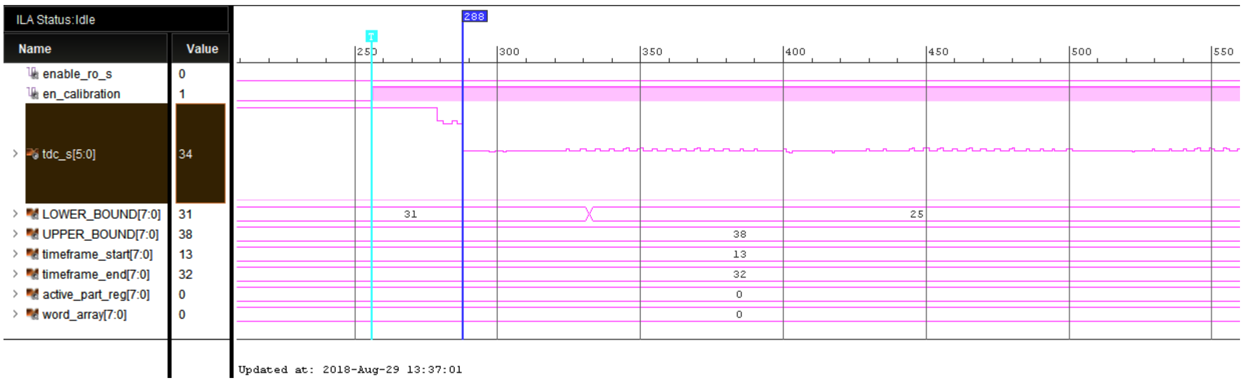


Figure 4.1.: Floorplan of the design implemented in the Zynq ZC7020 on the PYNQ-Z1 Board



ILA Status: Idle

Name	Value
enable_ro_s	0
en_calibration	1
tdc_s[5:0]	34
LOWER_BOUND[7:0]	31
UPPER_BOUND[7:0]	38
timeframe_start[7:0]	13
timeframe_end[7:0]	32
active_part_reg[7:0]	0
word_array[7:0]	0

Updated at: 2018-Aug-29 13:37:01

Figure 4.2.: A Display of HW-ILA showing the initial calibration of the TDC-Sensors. It can be seen that tdc_s[5:0] is an average idle value of about 34 after the calibration is finished at sample 288

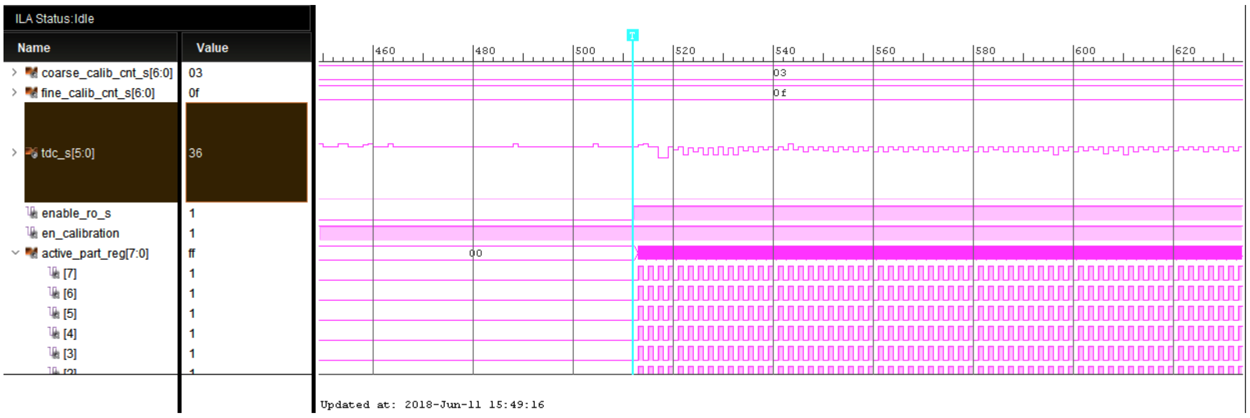


Figure 4.3.: This ILA display shows that the tdc_s[5:0] always falls a bit before it goes back to its previous value, while the ROs are turned on and off

4.2. Shaping Voltage in PDN

At first, it has to be found out, how exactly the activation of the ROs influences the voltage behavior in the PDN. Different possibilities to activate the ROs were being tested. The numbers of ROs being used for shaping the voltage can be diversified, too. But since the main goal of this work was to get any working design, a fixed number of 2504 ROs was determined at the beginning and kept fixed to this value throughout all of the following experiments. It was not in the scope of this work to further evaluate the differences in transmission between different numbers of ROs, and is thus supsect to future work.
The intuitive thing to check first is to toggle all the ROs at the same time and speed and see if there is already a change in the voltage in the PDN, observed through the TDCs. As you can see in Fig. 4.3 the voltage fluctuations are similar to a wave. The waveform seen in *tdc_s[5:0]* drops every time the *active_part_reg[7:0]* are activating, but with a small delay **Assumption-1**. Most of the voltage fluctuations though, probably have not a high enough amplitude for the decoder to interpret with little or not errors.

To check, if **Assumption-1** was true, the time where the ROs stay activated while toggling, are increased like in Fig. 4.4. This shows, that after activating the ROs the voltage clearly falls after a delay of a few clock pulses.

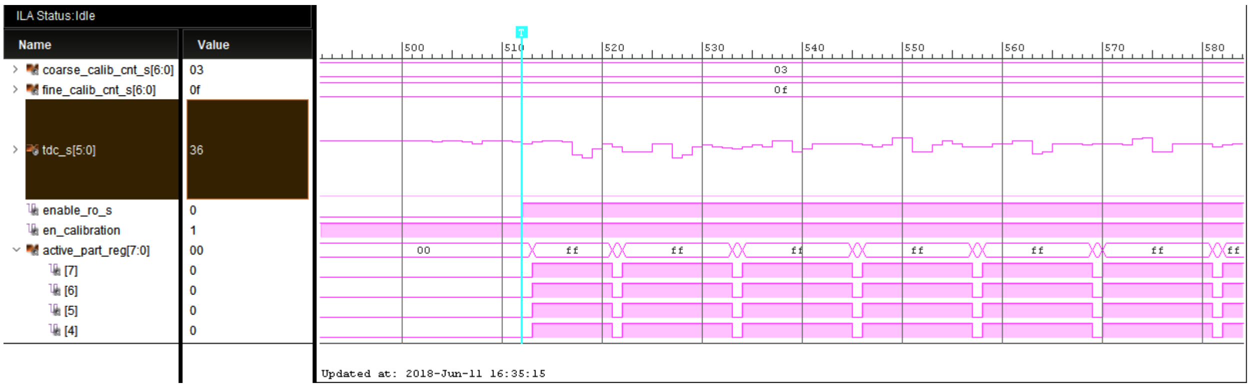


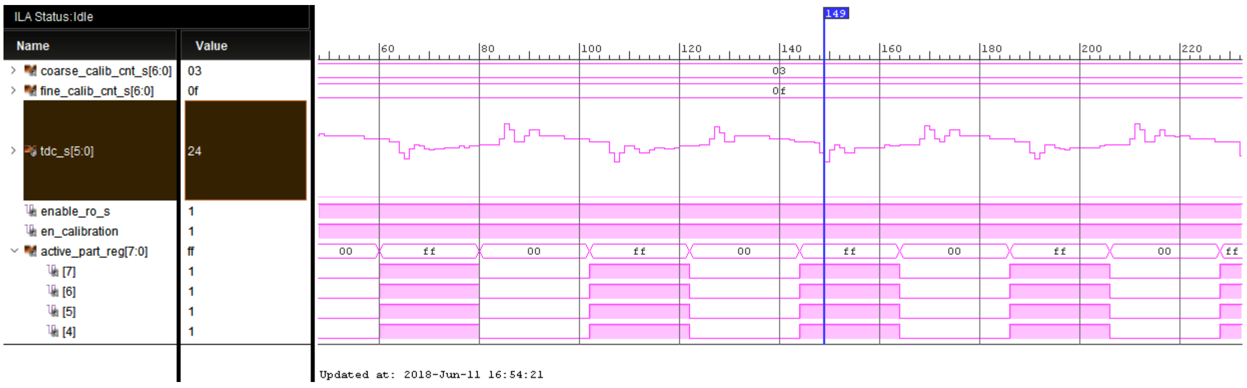
Figure 4.4.: In this ILA display the ROs are being turned on longer than being turned off. It shows that the value of the tdc_s[5:0] always falls with a delay after the ROs are being turned on

To determine the time until the impact from ROs is not visible in the voltage anymore Fig. 4.5 shows what happens if the ROs are deactivated for a longer time. This figure shows, that after deactivating the ROs the voltage rises for a short period. Based on Fig. 4.5 and Fig. 4.4 theoretically, the decoder can interpret the rise of *tdc_s[5:0]* as a 1 bit and the fall of voltage as a 0 bit. However, using only this mechanism makes it still not possible to separate these two kind of voltage fluctuations, because the voltage only rises after the ROs are being turned from on to off, which always leaves a falling voltage before the rising voltage. So voltage fluctuations for datawords like "1001" where two following zeros or more can not be achieved yet.

In Fig. 4.4 the time the ROs are on is longer than the time they are off, resulting in less and lower voltage peaks (i.e. high values of *tdc_s[5:0]*. To achieve higher voltage peaks and no negative voltage peaks, the idea is to reverse the process, hence less time for ROs to stay on and more time to stay off. However, as shown in Fig. 4.6, there is still a negative voltage peak before the voltage peak, which is not yet sufficient for clear distinction between 0 or 1.

4.2.1. Method-1 for Sending 1-Bits

There is a way to represent zero bits shown in Fig. 4.4, but a way to represent ones still needs to be found. To find it, another possibility to activate the ROs was implemented shown in

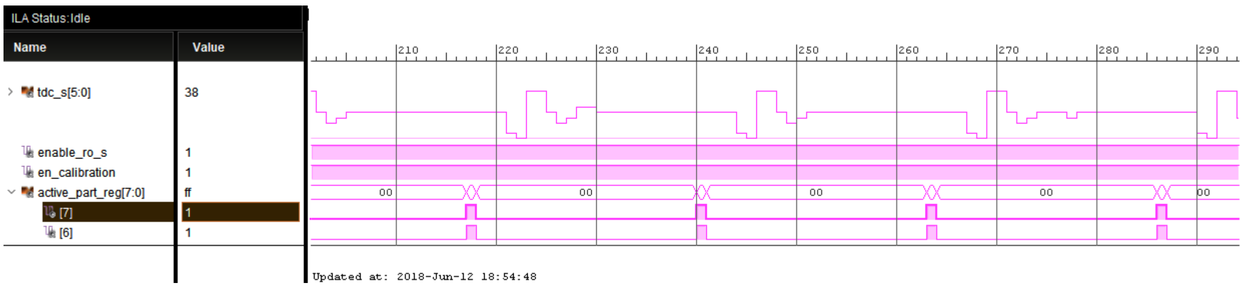


ILA Status:Idle

Name	Value
coarse_calib_cnt_s[6:0]	03
fine_calib_cnt_s[6:0]	0f
tdc_s[5:0]	24
enable_ro_s	1
en_calibration	1
active_part_reg[7:0]	ff
[7]	1
[6]	1
[5]	1
[4]	1

Updated at: 2018-Jun-11 16:54:21

Figure 4.5.: In this ILA display the ROs stay activated and deactivated for a longer period. It shows that the value of the tdc_s[5:0] falls with a delay after the ROs are being activated and rises with a delay after the ROs are being deactivated. It also shows that the value of the minima from the tdc_s[5:0] are 24 or close



ILA Status:Idle

Name	Value
tdc_s[5:0]	38
enable_ro_s	1
en_calibration	1
active_part_reg[7:0]	ff
[7]	1
[6]	1

Updated at: 2018-Jun-12 18:54:48

Figure 4.6.: In this ILA display the time where the ROs stay deactivated are longer than the time these are activated. After the ROs are activated for a short time the value of the tdc_s[5:0] falls, but then rises higher than the normal value

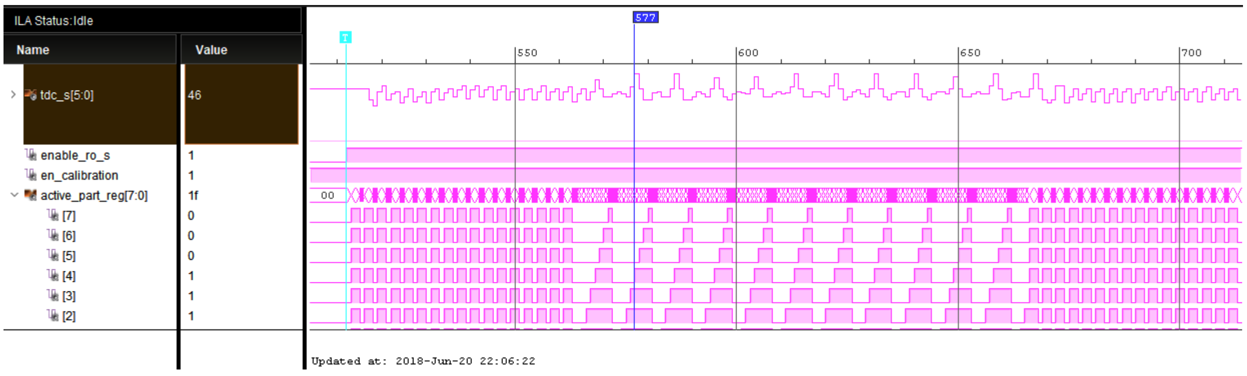


Figure 4.7.: In this ILA display two methods to activate the ROs are combined. First the ROs are being activated longer than being deactivated. This causes the value of tdc_s[5:0] to fall under the normal value. Second the ROs are not being activated all at once, but being activated one after another. This causes the tdc_s[5:0] value to rise above the normal value.

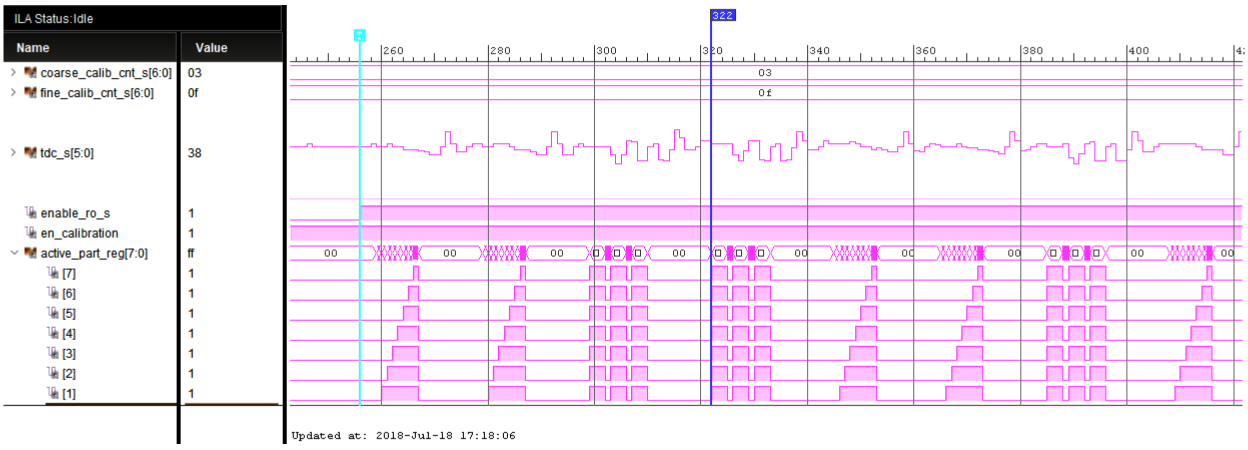
Fig. 4.7. In the middle part of the waveform from *active_part_reg[7:0]* the ROs are activated subsequently before they are all being turned off again, which is referred to as **Method-1**. This method causes the voltage to rise and form a maximum for each iteration. This can be used to represent the one bit.

4.2.2. Method-2 for Sending 0-Bits

There is also another method called **Method-2** in Fig. 4.7 to activate the ROs, which happens before and after **Method-1** in the waveform from the signal *active_part_reg[7:0]*. This method represents the zero bits and was used instead of the RO activation in Fig. 4.4. The difference compared to the RO activation in Fig. 4.4 is, that **Method-2** has a lower time duration where the ROs stay activated, but still forms a minimum, which means, that it needs less clock pulses to form minimums.

4.2.3. Test Transmission by combining Method-1 and Method-2

After finding suitable RO activation sequences for the transmission, a test for a dataword transmission has been run. As shown in Fig. 4.8, for the transmission the ROs were activated



ILA Status:Idle

Name	Value
coarse_calib_cnt_s[6:0]	03
fine_calib_cnt_s[6:0]	0f
tdc_s[5:0]	38
enable_ro_s	1
en_calibration	1
active_part_reg[7:0]	ff
[7]	1
[6]	1
[5]	1
[4]	1
[3]	1
[2]	1
[1]	1

Figure 4.8.: In this ILA display the ROs are being activated depending on the dataword. For 1 bits the ROs are being activated one after another and for 0 bits the ROs are being activated longer than being deactivated.The tdc_s[5:0] value shows that after activating the ROs gradually, it rises once. After activating the ROs longer than being dactivated, the value falls, but is rising once after the activation

subsequently (**Method-1**) once for each "1" bit and for each 0 bit, the ROs were turned on and off three times like **Method-2**. Also a high enough waiting time between each bit has been added so that both methods can not influence each other and the waveform is also much clearer. As one can see in the activation sequence of the ROs the test dataword should be "11001101". For "1" bits the decoder only needs to detect the maximum, but for the '0' bit the decoder has to detect subsequently three minima followed by one maximum. This method should already work, but to simplify the decoder in the receiver module, another method to represent the 0 bit has to be found.

4.2.4. Reverse Method-1 for Sending 0-Bits

Since **Method-1** causes a maximum, the reverse **Method-1**, hence activating the ROs all at once and then deactivating them gradually, could cause a minimum. As shown in Fig. 4.9 the reverse method is causing the value of *tdc_s[5:0]* to fall, forming exactly one minimum after one iteration. This can be used to represent the '0' bits.

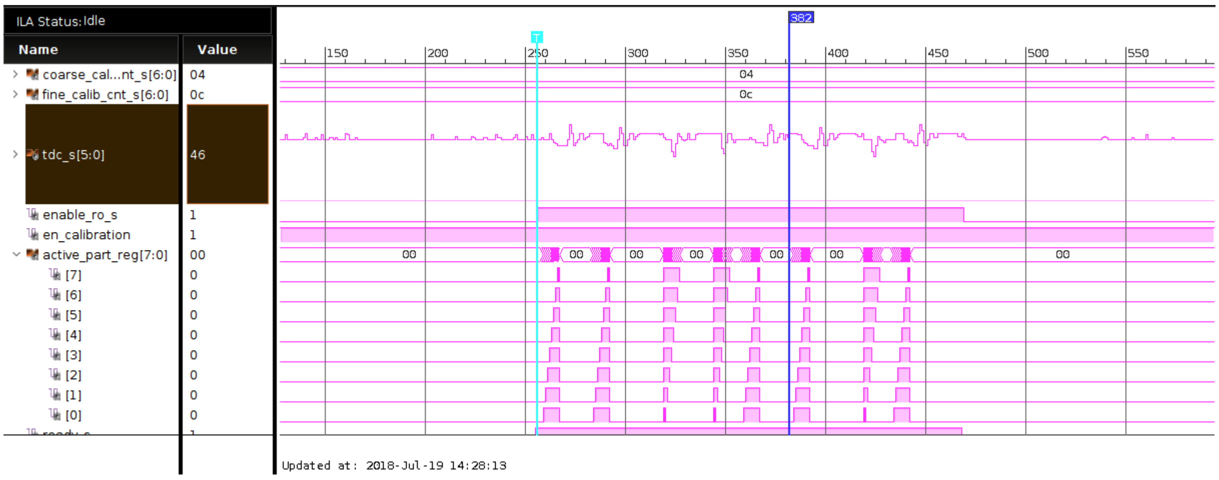


Figure 4.9.: This ILA display shows the final solution of the RO activation.By activating the ROs gradually, the value of the tdc_s[5:0] rises. By activating the ROs all at once and then deactivating them gradually, the value of the tdc_s[5:0] falls.

4.2.5. Combined Sending and Receiving

The final methods which are used for all the transmissions are **Method-1** for 1 bits and the reverse **Method-1** for "0" bits. As the receiver the TDC-Sensor only has to read the values of the voltage. Then upper and lower boundaries has to be determined. Then the decoder only picks the voltage values which are higher than the upper boundary or lower than the lower boundary, which should be the maxima and minima. Values higher than the upper boundary are then interpreted as "1" bits and values lower than the lower boundary as 0 bits. Fig. 4.9 also shows that this method still has many flaws. There can be two values in a row, which can be higher or lower than the boundaries, so the decoder identifies two bits instead of one. To prevent that from happening, after the decoder identifies a bit, it needs to wait and skip some clock pulses first, before being active again. However, skipping only one clock pulse after identifying a bit showed to be sufficient in our experiments, since the problem did not occur again. In the next section, a timeframe has been implemented, which also fixes the problem.

4.3. Higher Error Tolerance by Skipping Bits outside Timeframes

In this section a timeframe has been implemented and optimal boundaries has been found to minimize transmission erros. One problem is that after many transmissions, the decoder could fail to identify some bits, because the maximum or minimum were not over or under the determined boundaries. In that case a measure has been made to detect them. After each transmitted bit, a *timeframe* is defined, in which either a '0' or '1' is expected in. If that is not the case, no bit can be identified. Depending on the speed of our sender logic, we chose the default timeframe minimum and maximum values to be 13 and 37 respectively, which means that the bit is to be expected between the 13th and the 37th clock pulse. To check in which clock pulse the transmission occurs, a counter was implemented. After the second bit the size of the timeframe is changing dynamically depending on when last bit was found and the counter is reset to 0 again. The next timeframe minimum is calculated by replacing the value with half of the time the decoder needs to find the next bit. The next timeframe maximum is calculated by replacing the value with one and a half of the time the decoder needs to find the next bit.
If no bit is found within the timeframe, the decoder still decides on a bit to be received and the counter is reset to 0 again. To decrease the amount of transmission errors, each minima and maxima have to be the same distance between each other or as close as possible, so that the timeframe does not change that much. To achieve that, the waiting time between the RO activation has been adjusted. Also the timeframe is always being reset back to the default after every transmission.
Also boundaries were implemented so that the decoder only interpret the values over and under the boundaries. To do so a average *tdc_s[5:0]* value is determined first, the boundaries are then determined based on that. To calculate the average *tdc_s[5:0]* 32 values of the *tdc_s[5:0]* are used after the calibration of the TDC-Sensor is finished. Then a fixed value is added to the average tdc_s as upper boundary and a fixed value is subtracted from the average *tdc_s[5:0]* as the lower boundary. To find the best possible boundary for a stable transmission, different boundaries were determined experimentally. In this case the upper boundary should be 4 and the lower boundary should be 8. Hence, given an average *tdc_-s[5:0]* of 36 the *LOWER_BOUND[7:0]* is 28 and the *UPPER_BOUND[7:0]* 40.

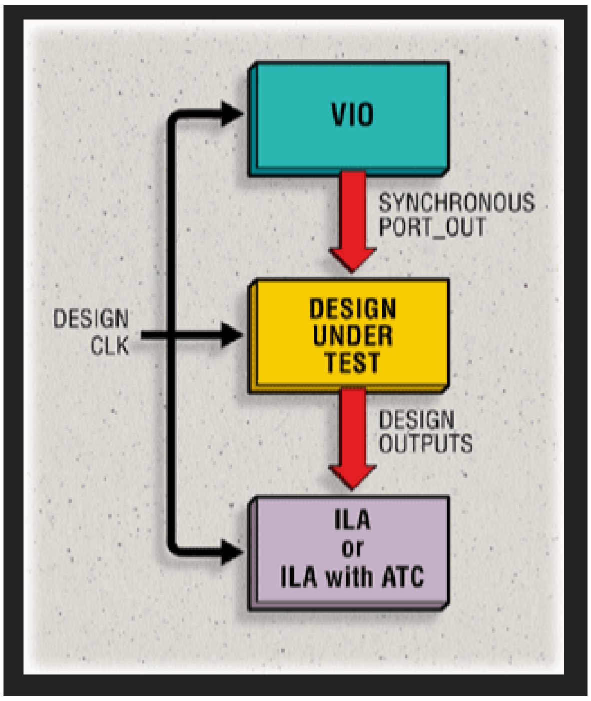


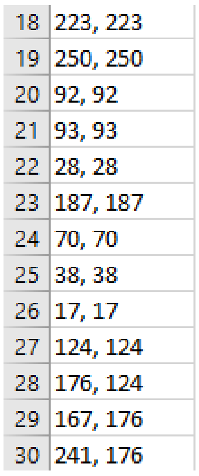
Figure 4.10.: This figure shows how the VIO [15] works together with the Design Under Test (DUT). The VIO puts test signals in the DUT, then the resulting signals are being displayed on ILA, the VIO then gets the resulting signals from the ILA

4.4. Using VIO and TCL-Script for Testing

The found and optimized RO activation sequence, the implemented timeframe and boundaries are used to test and observe the actual reliability of the transmission. To do so, a the Xilinx Vivado Virutal Input/Output (VIO) has been used. Fig 4.10 shows how the VIO works together with the Design Under Test (DUT). The VIO puts test signals in the DUT, then the resulting signals are being displayed on ILA, the VIO then gets the resulting signals from the ILA. One of the output called *operation_mode* functions as a trigger to turn the transmission on and off. The other output called *dataword_s* is the dataword which is being transmitted.

The ILA shows what is happening in the design under test. In addition, the inputs of the VIO is being updated after the transmission is done. One of the input called transmitted word, which shows the transmitted word. The other input is called *is_result_correct*, which shows if the transmitted word matches the dataword. To change the values of the VIO, a tcl script has been used. This tcl script first configures the hardware device and then the properties in the hw ILA. Then the process called *launch_and_display_vio_ila* is being executed. In this process the output datawords of the VIO is being randomly changed, then after the hw ila is displayed the transmission is being turned on. After the transmission is done the inputs of the VIO has been updated and the transmission is being turned off again. Then the results are being written in a csv.file. This is repeated a fixed time. In the first test a 8-bit dataword is being transmitted. This transmission is only repeating for a few times, but has already caused a strange error. The transmitted word which does not match the sent dataword, but matches the dataword in the previous transmission or the one before as seen in Fig. 4.11. To find out, why this error happens, the actual VHDL code is checked for any timing errors or logical errors and then it is fixed. Then, a few transmission is run, but the same error occurs. So it might be the VIO part, where the inputs did not update fast enough. So the script just writes the old not updated inputs. To check that, after every updated outputs, a long waiting time of 1000ms has been added. It has to be made sure, that this is the problem. When doing that, this kind of error occurs much less. So to prevent this kind of error, the script is on hold until the inputs of VIO have been updated. This is achieved by always checking if the transmitted word still matches the previous transmitted word. If it does not match, the script will continue and if it does match the transmitted word is going to be compared with the datawords. If both match the script will continue. This case prevents the program being stuck when the random number generator (RNG) generates two consecutive same numbers. So the only case where the program could still get stuck is, when the transmitted word matches the previous transmitted word, but do not match with the dataword. Since this case did not happen in our experiments, the problem was not addressed yet and may be investigated in future work.

After this VIO synchronization error is fixed, tests for the CC design are run. In these tests the transmission were being repeated 100000 times to test the reliability of the CC and see how many errors occur. Different sizes of the dataword (8bit and 16bit) are transmitted. Also there is a difference in the bit to replace in case no bit is found within the timeframe [5.1]. Then, a few adjustment in the boundaries and timeframe has been made to see if the reliability of the CC has been improved. Also 32-bit dataword transmissions has been included [5.2].



18	223, 223
19	250, 250
20	92, 92
21	93, 93
22	28, 28
23	187, 187
24	70, 70
25	38, 38
26	17, 17
27	124, 124
28	176, 124
29	167, 176
30	241, 176

Figure 4.11.: Screenshot of a part in the csvfile of the 8-bit transmission. It shows the odd error.

5. Results

5.1. Transmission with Timeframe and Boundaries

The tests are run with the tcl script, the python script are used to create a plot with the csvfile and save it in a pdf file. This plot shows how many errors were made and on which position the error happened. For every test one plot is created.
The first plot is made with a 16bit dataword. If within the timeframe no bit is found, 1 bit is added, later on it will be called *1-bit timeframe*. In this test 42 transmission errors out of 100000 transmission occurs. Each transmission error has only one wrong bit. As one can see in Fig. 5.1 most of the errors happens in the last positions. These errors are either caused by the maxima or minima not being high or low enough to be identified, or by maxima and minima which are caused unintentionally by the RO sequence. Since in this case if the bit is not found, 1 bit will be added to the position of the transmitted word, the first type of error probably happens because a minima is not found.

The plot in Fig 5.2 is made with a similar test like in Fig. 5.1. The only difference is, that the 0 bit is added if no bit is found, later on it will be called *0-bit timeframe*. This test only has 2 transmission errors out of 100000 transmissions. But it has 7 total bits which are wrong. So multiple errors in one transmitted word can occur. The plot 5.2 shows, that errors occur in the last positions of the transmitted word again. But a lot of errors also occur in the first positions of the transmitted word.

The plot in Fig. 5.3 is made with 8-bit transmissions with the *1-bit timeframe* and default boundaries. The plot in Fig. 5.4 is also made with 8-bit transmissions but with the *0-bit timeframe*. The test in Fig. 5.3 has 31 transmission errors, but with 94 erroneous bits in total. The test in Fig. 5.4 has 23 transmission errors and each transmitted word has only one wrong bit again. If these two plots are compared there are more wrong bits at the start in the test when using *0-bit timeframe* and for the other one there are more wrong bits after the first

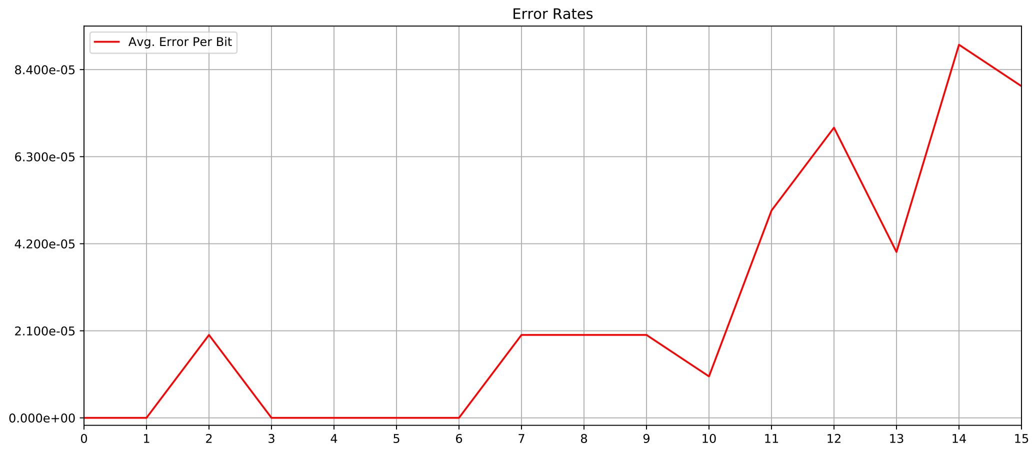


Figure 5.1.: This plot is made by the result of 100000 16bit-transmissions using the 1-bit timeframe and default boundaries. It has 42 wrong datawords, each with 1 erroneous bit.

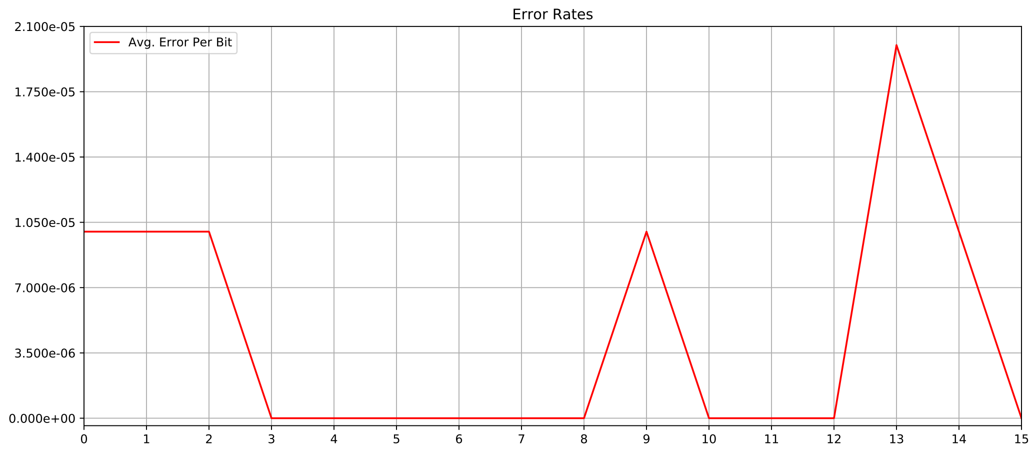


Figure 5.2.: This plot is made by the result of 100000 16bit-transmissions using the 0-bit timeframe and default boundaries. It has 2 wrong datawords with 7 erroneous bits.

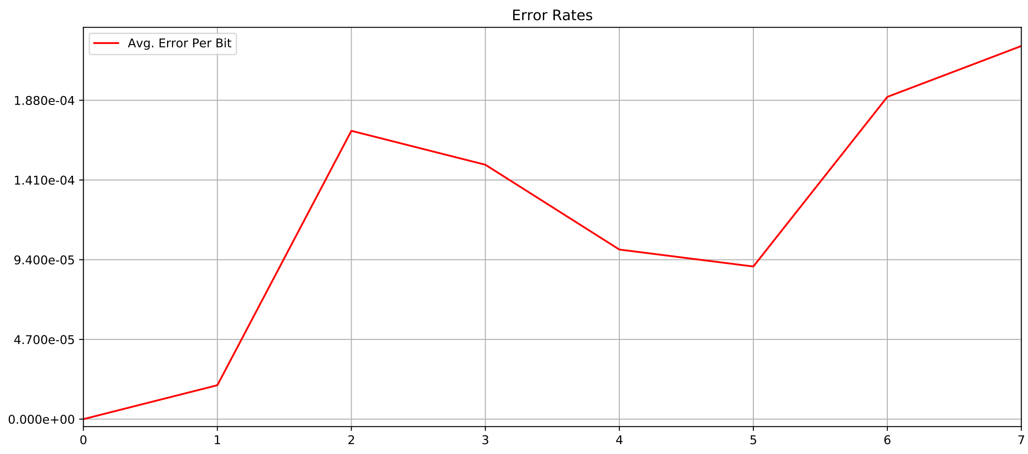


Figure 5.3.: This plot is made by the result of 100000 8bit-transmissions using the 1-bit timeframe and default boundaries. It has 31 wrong datawords with 94 erroneous bits.

positions of the dataword. But overall the test with the *1-bit timeframe* in Fig. 5.3 has 71 more wrong bits than the test with the *0-bit timeframe*.

The tests show, that the implemented covert channel is not completely error-free. These errors are either caused by the maxima or minima not being high or low enough to be identified, or by random maxima and minima, which are caused by unintentional noise in the system.

5.2. Transmission with adjusted Timeframe and Boundaries

In this section, the transmission has been improved with adjusted timeframe and boundaries. The timeframe has been adjusted in a way, so that it does not grow to arbitrary lengths in case multiple bits are missing and stays near the default timeframe. This is achieved as follows: Instead of adding half of the time to the full time of when the bit is found, only a small and fixed value is added to the full time. The fixed value is 7 and the default timeframe were also changed from 13 and 37 to 13 and 32. If no bit is found within the timeframe the counter is

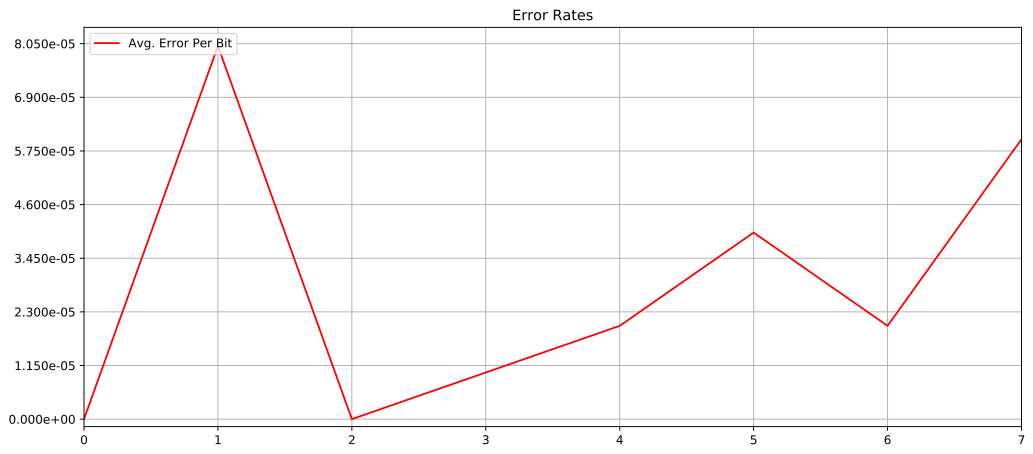


Figure 5.4.: This plot is made by the result of 100000 8bit-transmissions using the 0-bit timeframe and default boundaries. It has 23 wrong datawords, each with 1 erroneous bit.

not reset to 0, but divided by 4. This causes the timeframe to be close to the default value, too. Also the boundaries are adjusted depending on which bit is being replaced in case no bit is found within the timeframe. In the 1 bit case the upper boundary is five and the lower boundary is still the same. In the other case the upper boundary stays the same and the lower boundary is nine. By doing that, wrongly identified minima and maxima can be left out by the decoder this time. But increasing the boundary may cause the actual maxima or minima not to be recognized either. But since it will be replaced with the right bit anyway, if no bit is found, and the timeframe has been adjusted, so that it stays close to the default range, this problem is negligible.
Then six tests with each 100000 are run again with these few adjustments. Four tests are run with 8bit and 16bit datawords like the first tests in section 5.1. Then the transmission with 32bit dataword are being tested, too. The first two tests are 8-bit dataword transmissions using the adjusted *1-bit timeframe*, *0-bit timeframe* and boundaries. As one can see in Fig. 5.5 and in 5.6, no error are found. So in comparison to the previous tests, the adjustments in the design improved the stability of the transmission.

Then, tests are run for the 16-bit dataword transmissions. In Fig. 5.7 is the plot of the result from the transmissions with the *1-bit timeframe* and adjusted boundaries. Out of 100000

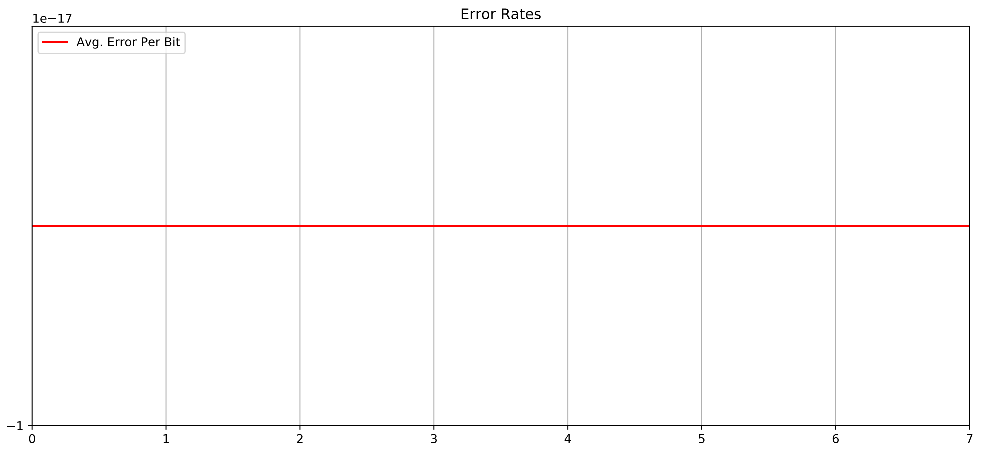


Figure 5.5.: This plot is made by the result of 100000 8bit-transmissions using the adjusted 1-bit timeframe and boundaries. It has no errors.

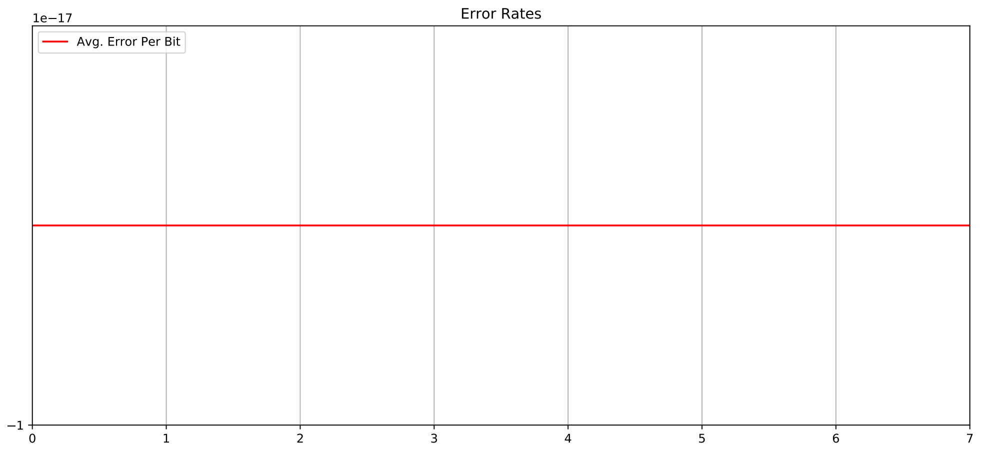


Figure 5.6.: This plot is made by the result of 100000 8bit-transmissions using the adjusted 0-bit timeframe and boundaries. It has no errors.

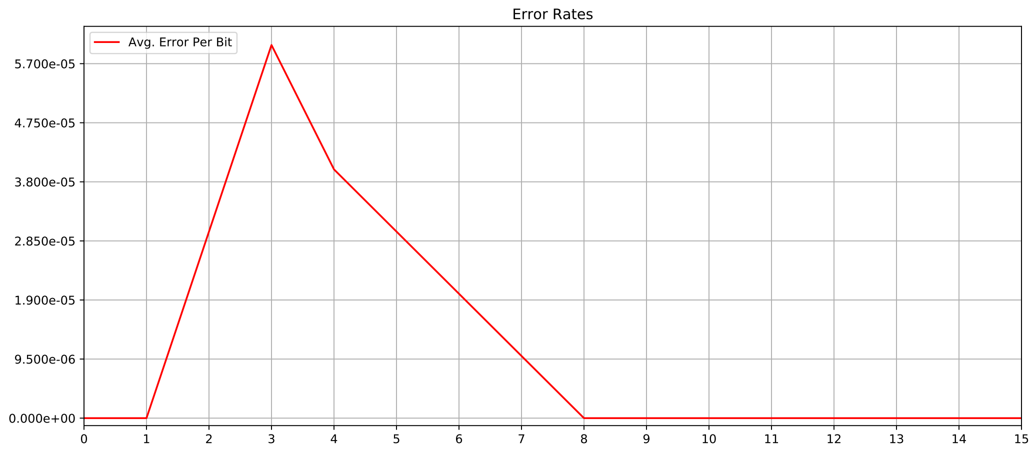


Figure 5.7.: This plot is made by the result of 100000 16bit-transmissions using the adjusted 1-bit timeframe and boundaries. It has 19 wrong datawords, each with 1 erroneous bit.

transmissions there are 19 wrong datawords. This plot shows that most of the errors happens between the second and seventh bit. In Fig. 5.8 is the plot with the *0-bit timeframe*. 2 incorrect transmissions happens here. Each incorrect transmitted word has 1 wrong bit. The related plot shows that these errors happened at 14th and 15th position of the dataword.

Then, tests are also run for the 32-bit dataword transmissions. Fig. 5.9 shows the plot with the *0-bit timeframe* and adjusted boundaries. Only one incorrect transmitted word occurrs. This dataword though, has 18 erroneous bits. As the plot shows, the errors happens nearly everywhere and not in a specific pattern.

Fig 5.10 shows the plot with the *1-bit-timeframe* and adjusted boundaries. In this chain of transmissions, 101 transmitted words does not match the dataword. And there are a total of 103 erroneous bits. Hence, most the transmitted word has a 1-bit error. This plot also shows, that most of the errors happens at the last half of the transmitted word. This suggests that on transmission with longer words, the sender and receiver might get out of synchronization. The bits from position 0 to 15 are even flawless.

The test with the *1-bit-timeframe* has much more errors, which is probably caused by not recognizing the sent bit. A bad timeframe or boundary can be the cause. Hence, the

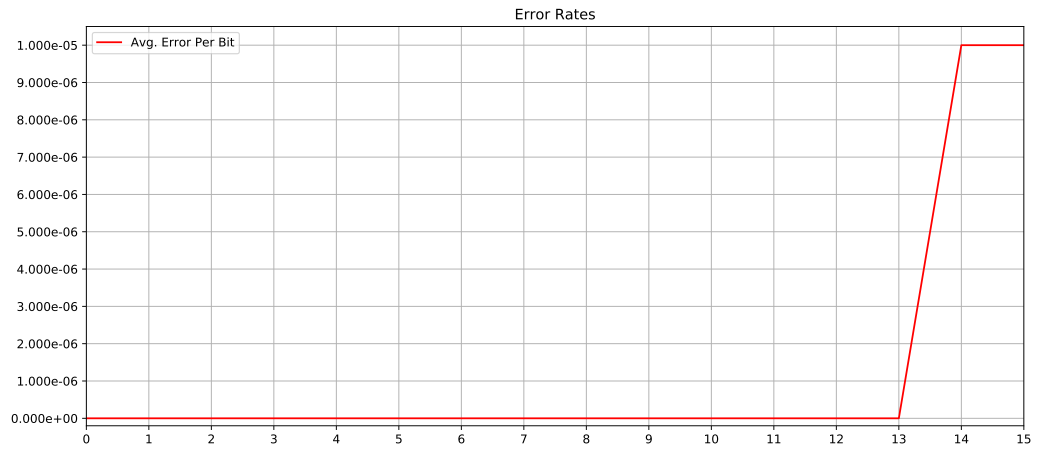


Figure 5.8.: This plot is made by the result of 100000 16bit-transmissions using the adjusted 0-bit timeframe and boundaries. It has 2 wrong datawords, each with 1 erroneous bit.

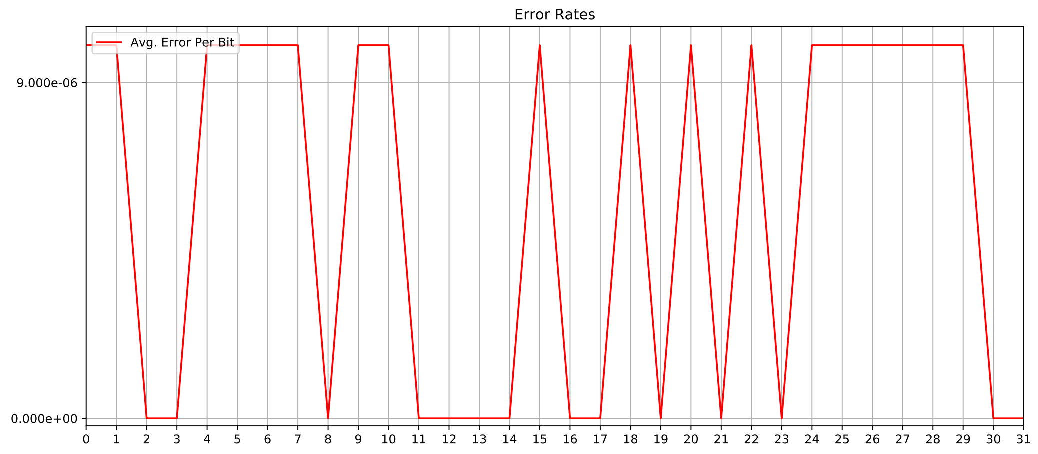


Figure 5.9.: This plot is made by the result of 100000 32bit-transmissions using the adjusted 0-bit timeframe and boundaries. It has 1 wrong dataword and 18 erroneous bits.

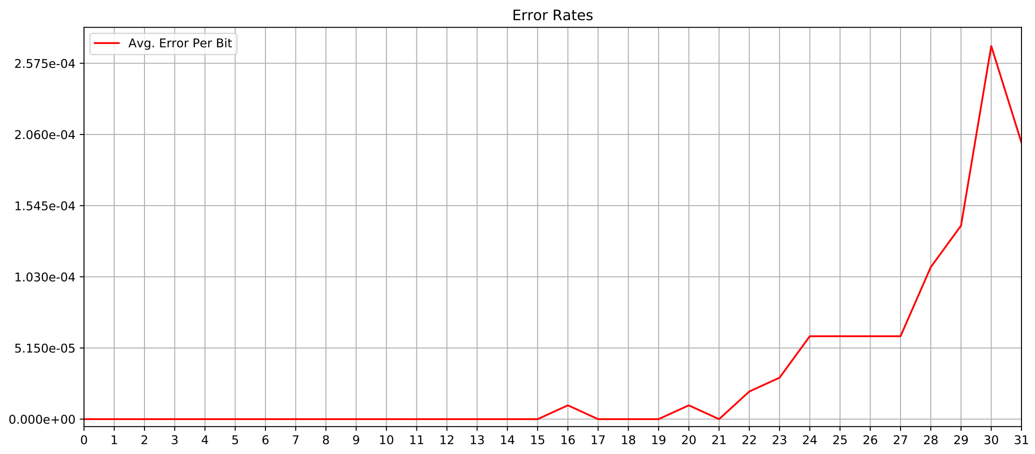


Figure 5.10.: This plot is made by the result of 100000 32bit-transmissions using the adjusted 1-bit timeframe and boundaries. It has 101 wrong datawords and 103 erroneous bits.

timeframe are checked by running some transmission again and observe the behavior of the timeframe. The timeframe is always close to the default value. So the only problem lies in the boundary. If the boundaries of both the tests as seen in Fig. 5.11 and Fig. 5.12 are being compared, the *UPPER_BOUND[7:0]* is 43 and *LOWER_BOUND[7:0]* is 30 for one and the other is 40 and 27. So there is a difference of 3 between both tests. This can be a factor, why the *1-bit timeframe* test has significantly more errors. The boundaries of the 16 bit and 8 bit transmission are checked, too. All of them have the *UPPER_BOUND[7:0]* 42 and the *LOWER_BOUND[7:0]* 29. Since those tests do not have a lot of errors, but have close boundaries to both the 32-bit test with *0-bit-timeframe* and *1-bit-timeframe*, more experiments in the boundary area has to be made. Also in Fig. 5.12 at the tdc_s wave there is sometimes a really low minimum before the maximum. So it also can be assumed, that the decoder could detect a 0 bit first and because of the timeframe, skip the 1 bit which should be identified. This error may be fixed, if the decoder additionally checks within the next 3 to 4 pulses after detecting a "0" bit, whether there is a maximum or not.

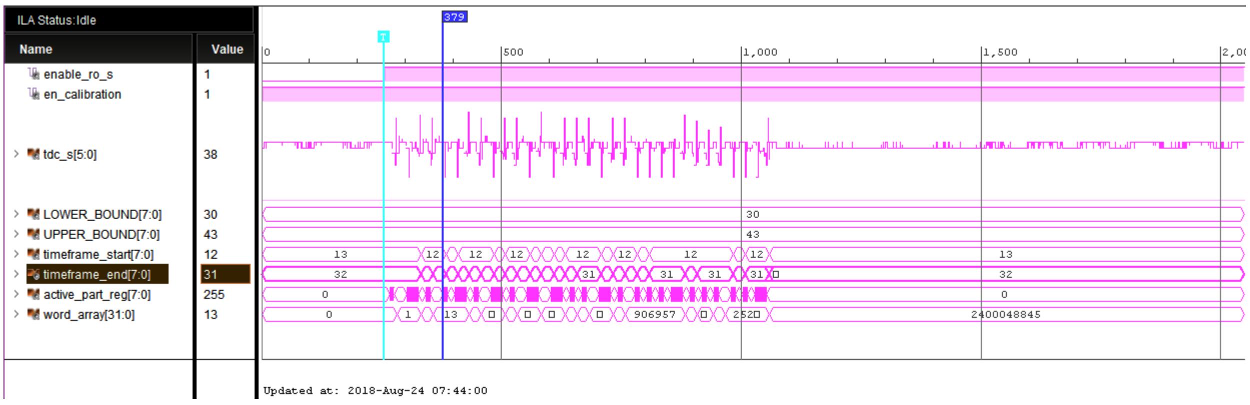


Figure 5.11.: ILA Display of a 32bit transmission with a 1bit timeframe

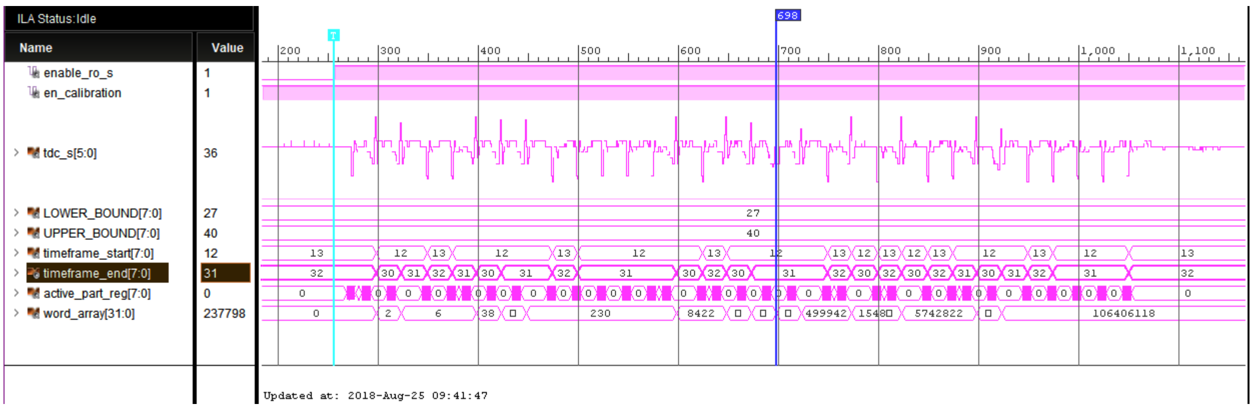


Figure 5.12.: ILA Display of a 32bit transmission with a 0bit timeframe

5.3. Discussion

In this section, the results from section 5 are being discussed. None of the tests with the default timeframes and boundaries are error-free. The tests with the *1-bit timeframe* have more errors than the tests with the *0-bit timeframe*. This means, that more 1 bits are being recognized than 0 bits. The tests also show that the 16-bit transmissions with the *0-bit timeframe* has the least errors. This is surprising, because more bits are being transmitted here than in the tests with 8-bit transmissions. Also the 8-bit transmissions should be more synchronized than the 16-bit transmissions, because of its smaller transmission length. However, this might just be a coincidence. So more tests has to be made. The tests with the adjusted timeframes and boundaries show better results. The 8-bit dataword transmissions are completely error-free. In the 16-bit dataword tranmissions errors occur. But in comparison to the tests with the default timeframe and boundaries, these tests have less error. Also the 32-bit dataword transmissions with the *0-bit timeframe* has even slightly less errors than the 16-bit dataword transmissions with the *0-bit timeframe*. This could be a coincidence, too. The test with the 32-bit dataword transmissions with the *1-bit timeframe* has a lot more errors compared to the 16-bit dataword transmission with the *1-bit timeframe*. Most of the errors occur in the later positions of the dataword though. So there might be synchronization errors from the sender and receiver. But overall,tests with the *0-bit timeframe* show better results than the tests with *1-bit timeframe*. The adjusted boundaries and timeframe are also slightly better than the default.

5.4. Future work

After doing the needed experiments to adjust the boundaries further and maybe improve the decoder for more transmission stability, an Error Correction Code (ECC) could be implemented for the transmission. After that, the tranmission could be tested under an other interference source, to see if this approach of the transmission still would work. Then, the CC could be implemented between two different cores. For instance, a CC between a FPGA and an ARM-core would be interesting.

6. Conclusion

6.1. Conclusion

So as FPGAs are being increasingly used in systems like Systems-on-Chip (SoCs) or Clouds and there will be multiple users on one FPGA, which causes many security issues, this work shows an other security issue, by successfully implementing a voltage-based covert channel between logically separated IP-cores in FPGAs. To do so, in section 2 relevant researches are made, to get a basic understanding of this topic. Then a concept for the voltage-based covert channel in FPGAs has been made in section 3. This concept shows, that theoretically you need a specific signal coding, a sender and a receiver to build this covert channel and how you combine them. In section 4 experiments are made, to understand how to implement the actual covert channel. By experimenting with different methods to activate the ROs two methods have been found and picked to successfully transmit datawords via the voltage of the PDN. Lower and upper boundaries have been implemented for decoding purposes. Furthermore, they have been adjusted to improve the reliability of the transmission. A timeframe has been implemented in the receiver also to improve the reliability of the transmission. In section 5 the transmission of the implemented covert channel are tested. Testing the transmission with 8-bit, 16-bit and 32-bit datawords with *1-bit timeframe* and *0-bit timeframe* produced different results. For all transmissions, except for 8-bit, errors occurred. This means, that the implemented covert channel is still in its raw state and can still be further improved. However, this thesis proves, that a voltage-based covert channel between logically separated IP-cores in FPGAs is possible and is implemented successfully.

A. Appendix

A.1. VHDL Code

```
 -- RO state machine, which determines
 -- how to activate the ROs
RROstate_proc:process(clk) is
begin
  if rising_edge(clk) then
  case current_state is

    when IDLE =>
         if (switch_s = '1') then
             -- used to adjust the placement
             -- of the first activating RO-instance
             if (start_counter /= wait_time) then
                 start_counter <= start_counter + 1;
             else
                 start_counter <= 0;
                 current_state <= READ;
             end if;
         else
             current_state <= IDLE;
         end if;

   when INIT =>
         if is_init = '1' then
             current_state <= READ;
         else
             current_state <= WAIT_INIT;
         end if;

   when WAIT_INIT =>
```

```
            -- this is used to adjust the placement
            -- of the activated RO-instances
           if last_bit = '1' then
              if last_bit /= (dataword(word_counter)) then
                wait_max_counter_init <= 22;
              else
                 wait_max_counter_init <= 12;
              end if;
              elsif last_bit = '0' then
              if last_bit /= (dataword(word_counter)) then
                wait_max_counter_init <= 2;
              else
                    wait_max_counter_init <= 12;
              end if;
              else
                 wait_max_counter_init <= 12;
              end if;
          init_counter <= init_counter +1;
          if (init_counter = wait_max_counter_init) then
            init_counter <= 0;
            is_init <= '1';
            current_state <= INIT;
          else
            current_state <= WAIT_INIT;
              end if;

      when READ =>
           if (word_counter /= dataword'length) then
               if dataword(word_counter) = '1' then
                   bit_state <= b"11";
                   word_counter <= word_counter + 1;
                   current_state <= WAIT_READ_1;
               else
                   bit_state <= b"00";
                   word_counter <= word_counter + 1;
                   current_state <= WAIT_READ_0;
               end if;
           else
               word_counter <= 0;
               current_state <= FINISH;
           end if;
```

```
    when WAIT_READ_0 =>
        read_counter <= read_counter +1;
        if (read_counter = wait_max_counter_0) then
            read_counter <= 0;
            bit_state <= b"10";
            last_bit <= '0';
            is_init <= '0';
            current_state <= INIT;
        else
            current_state <= WAIT_READ_0;
        end if;

    when WAIT_READ_1 =>
        read_counter <= read_counter +1;
        if (read_counter = wait_max_counter_1) then
            read_counter <= 0;
            bit_state <= b"10";
            last_bit <= '1';
            is_init <= '0';
            current_state <= INIT;
        else
            current_state <= WAIT_READ_1;
        end if;

    when FINISH =>
        is_init <= '0';
        bit_state <= b"10";
        if switch_s = '0' then
        current_state <= IDLE;
        end if;
 end case;
 end if;
end process;
-- this process decides which bit
-- is going to be sent with the RO activation
bitstate_proc: process(clk) is
begin
if rising_edge(clk) then
case bit_state is
```

```
    when "00" =>
        bit_sent <= "00";


    when "11" =>
        bit_sent <= "11";

    when others =>
        bit_sent <= "10";
end case;
end if;
end process;
```

Listing A.1: A state machine to handle the activation process of the ROs

```
-- basic counter and blinking leds to show the FPGA
-- is still operating at all
-- also enables/disables the ROs depending on the switch
alternating : process (clk)
    begin
      if rising_edge(clk) then
         if operation_mode(0) = '1' then
          blink_reg <= not blink_reg;
               if calib_done_s = '1' then
                 enable_ro_s <= '1';
               end if;
         else
          enable_ro_s <= '0';
          blink_reg <= blink_reg;
         end if;
       end if;
   end process;

-- Enabling the calibration
enable_operation : process(clk)
    begin
        if rising_edge(clk) then
            btn_reg <= btn;
            if (btn_reg(0) = '1') then
                rst_calibration <= '0';
                en_calibration <= '1';

            end if;

        end if;
     end process;
```

Listing A.2: Process to activate or deactivate the ROs

```
-- process to read voltage value and only interpret
-- the values within the timeframe
-- and over/under the fixed boundaries
tdc_value_proc: process(clk) is
begin
if rising_edge(clk) then
  if(enable_ro_s = '1') and (operation_mode(0) = '1') then
    if (current_position /= (dataword_s'length)) then
if (current_position < 1) then
   is_result_correct <= b"0";
   is_word_written <= '0';
 end if;

if (current_position > 0) then
   counter <= counter + 1;
else
   counter <= 0;
end if;

if to_integer(unsigned(tdc_s)) > UPPER_BOUND or
to_integer(unsigned(tdc_s)) < LOWER_BOUND  then

  if counter < timeframe_start
  and (current_position > 0) then
  -- ignore this event

  else
    -- update timeframe
    if (current_position > 0) then
       timeframe_start <= (counter+1)/2;
       timeframe_end <= counter + timeframe_end_range;
    end if;

    if to_integer(unsigned(tdc_s)) > UPPER_BOUND then
      word_array(current_position) <= '1';
    else
      word_array(current_position) <= '0';
    end if;
    current_position <= current_position + 1;
    counter <= 0;
 end if;

elsif counter > timeframe_end then
  word_array(current_position) <= '0';
     if to_integer(unsigned(tdc_s)) > UPPER_BOUND then
        word_array(current_position + 1) <= '1';
        current_position <= current_position + 2;
        counter <= 0;
     else
```

```
                word_array(current_position + 1) <= '0';
                current_position <= current_position + 2;
                counter <= 0;
            end if;
                current_position <= current_position + 1;
                counter <= (counter/4);
        end if;

    else -- transmission finished
        if (is_word_written = '0') then
            transmitted_word <= word_array;
            if (word_array = dataword_s) then
                is_result_correct <= b"1";
            else
                is_result_correct <= b"0";
            end if;
                is_word_written <= '1';
        end if;
        --reset timeframe after every transmission
        timeframe_start <= timeframe_start_default;
        timeframe_end <= timeframe_end_default;
        counter <= 0;
      end if;
  else
      current_position <= 0;
      word_array <= (others => '0');
      counter <= 0;
  end if;
end if;
end process;
```

Listing A.3: This process reads the voltage values of the PDN and interpret it

```
-- determines the value of the boundaries
-- depending on the average voltage value
bound_proc: process(clk) is
 variable tdc_average : integer range 0 to 255 := 0;
    begin
    if rising_edge(clk) then
      if (calib_done_s = '1') then
        if (tdc_counter = tdc_max) then
            tdc_average := to_integer
```

```
                    (shift_right(to_unsigned(tdc_sum,11),5));
                    LOWER_BOUND <= tdc_average - tdc_lower_range;
                    UPPER_BOUND <= tdc_average + tdc_upper_range;
                    tdc_counter <= tdc_max+1;
                    tdc_sum <= 0;
                elsif (tdc_counter < tdc_max) then
                    tdc_counter <= tdc_counter + 1;
                    tdc_sum <= tdc_sum + to_integer(unsigned(tdc_s));
                end if;
            end if;
        end if;
```

Listing A.4: This process determines the boundaries for the decoder

```
--handling the RO activation
state_proc: process(clk) is
begin
    if rising_edge(clk) then
        case bit_sent is
            when "11" =>
                    if active_part_reg < b"1111_1111"
                and enable_ro_s = '1' then
                    active_part_reg <= active_part_reg(6 downto 0)
                     & '1';
                else
                    active_part_reg <= (others => '0');
                end if;
            when "00" =>
                if active_part_reg > b"0000_0000"
                and enable_ro_s = '1' then
                    active_part_reg <= active_part_reg(6 downto 0)
                    & '0';
                else
                    active_part_reg <= (others => '1');
                end if;
            when others =>
                    active_part_reg <= (others => '0');
             end case;

    end if;
end process;
```

Listing A.5: This process activates the ROs in a way depending on the state of the state machine

A.2. TCL-Script

```
global transmitted_word_current
set repetitions 100002
set filestring "_unknown_"

set devicestr Digilent/003017A6DE6CA
set tracesdir "./traces/"

set basedir "C:/Users/Khoa/minimum_example_tdcs_ros"

# The section to program the device via Vivado, adjust some options
# and interact with the hw_ila was left out, since it is not
#  relevant and can be done with the Vivado interface, too.
after 10

proc launch_and_display_vio_ila {} {
        global init
        global transmitted_word_current
        set is_value_changed 0
# set the dataword of VIO with a random number
        set_property OUTPUT_VALUE_RADIX
        unsigned [get_hw_probes dataword_s]
        set dataword [expr { wide(rand()*(2**8)) }]
        set_property OUTPUT_VALUE $dataword [get_hw_probes dataword_s]

        commit_hw_vio [get_hw_vios hw_vio_1]

#       Running one transmission
        run_hw_ila hw_ila_1
    set_property OUTPUT_VALUE_RADIX binary
     [get_hw_probes operation_mode]
    set_property OUTPUT_VALUE 1 [get_hw_probes operation_mode]
    commit_hw_vio [get_hw_vios hw_vio_1]
    wait_on_hw_ila hw_ila_1


    commit_hw_vio [get_hw_vios hw_vio_1]
        foreach probe [get_hw_probes -of [get_hw_ilas]] {
            set_property DISPLAY_RADIX unsigned $probe
                set_property DISPLAY_AS_ENUM false $probe}
 {CELL_NAME=~"u_ila_0"}]
# Stop transmission
    set_property OUTPUT_VALUE 0
     [get_hw_probes operation_mode]

    commit_hw_vio [get_hw_vios hw_vio_1]
    if {$init == 0} {
         set transmitted_word_current - 1
         set init 1
    }
 # waits until the inputs of VIO are changed
 while {$is_value_changed == 0} {
        set transmitted_word_next
        [get_property INPUT_VALUE [get_hw_probes transmitted_word]]
        if { $transmitted_word_current != $transmitted_word_next} {
           set transmitted_word_current $transmitted_word_next
           set is_value_changed 1

        } else {
           if {$transmitted_word_next ==
                  [get_property OUTPUT_VALUE
                  [get_hw_probes dataword_s]]} {
                  set transmitted_word_current $transmitted_word_next
                  set is_value_changed 1
                 }
        }
}
        current_hw_ila_data [upload_hw_ila_data hw_ila_1]
        display_hw_ila_data [current_hw_ila_data]

}

#Running transmissions repeatingly and write the results in a csvfile
```

```
for {set i 0} {$i < $repetitions} {incr i} {

                global init
                if {$i < 1} {

                set init 0
        }
                launch_and_display_vio_ila
 [current_hw_ila_data]

                if {$i > 1} {
 is_result_correct]]"}


                        set fid [open ${basedir}/traces/Result.csv a]

                        puts $fid "[get_property OUTPUT_VALUE
                        [get_hw_probes dataword_s]],
                         [get_property INPUT_VALUE
                         [get_hw_probes transmitted_word]]"
                        close $fid

                }

}
```

Listing A.6: This script is used to run many transmission in a row

A.3. Python-Script

```
from matplotlib import pyplot as plt
import numpy as np
import sys

# bits per word sent
BITS = 8

def get_bit(word, bit):
    return ((word >> bit)&0x1)

statfile = r'C:\Users\Khoa\Desktop\Bachelortheorieteil\
Testergebnisse\test\100000-8-bit-1bit-newtimeframe-0errorword.csv'
#statfile = sys.argv[1]
pdffile = statfile+'.pdf'

rows = np.genfromtxt(statfile,delimiter=',',dtype='int64') #.transpose()
# Retrieve the number of samples/rows:

errorbits = np.zeros(rows.shape[0],dtype='int64')
error_bit_array = np.zeros((rows.shape[0],BITS),dtype='int64')

print(error_bit_array.shape)

all_worderrors=0
all_biterrors=0
for i in range(0,len(rows)):

    sent = (rows[i][0])
    recv = (rows[i][1])

    errors = sent ^ recv
    if (errors != 0):
        all_worderrors += 1

    badbits = 0
    # for each bit in the word
    for b in range(0,BITS):
        # isolate one bit 0/1 and add it to the bit error count
        errorbits[i] += get_bit(errors,b)
        error_bit_array[i][b] = get_bit(errors,b)
    all_biterrors += errorbits[i]

    print("Sent: "+str(sent)+", Recv: "+str(recv), end='');
    print(", Bit Errors: "+str(errorbits[i]));

# Average bit errors per all words
avg_errorbits = np.average(errorbits)
avg_error_per_bit = np.average(error_bit_array,axis=0)

# Not that interesting stats, they depend on how much data we had
print("Total erroneous words: "+str(all_worderrors));
print("Total bits erroneous: "+str(all_biterrors));
print("Total bits sent: "+str(len(rows)));
# More interesting stats are the ratio/fractions of errors,
# or in which bits errors happen more often, etc.
print("Average erroneous bits per transmitted word:
 "+str(round(avg_errorbits,4)));
# Average error rate per specific bit of the word
# (to find out if maybe the first or last bit is more susceptible)
print("Average error per bit: "+str(avg_error_per_bit))

#Plotting the results
ep=0.00000000000000001
ylim=(-np.max(avg_error_per_bit)*0.02-ep,
np.max(avg_error_per_bit)*1.05+ep)

fig = plt.figure()
ax = fig.gca()
ax.set_xticks(range(0, BITS, 1))

ax.set_yticks(np.arange(ylim[0],ylim[1], avg_errorbits*0.05))
plt.grid()
plt.plot(avg_error_per_bit,color='red',label="Avg. Error Per Bit")
plt.xlim(0,BITS-1)
plt.ylim(ylim)

plt.title("Error Rates")
plt.legend(loc='upper left')

# Save to file:
plt.savefig(pdffile)
```

Listing A.7: This script is used to evaluate the results of the transmissions

Bibliography

[1] T. Huffmire, B. Brotherton, G. Wang, T. Sherwood, R. Kastner, T. Levin, T. Nguyen, and C. Irvine, "Moats and drawbridges: An isolation primitive for reconfigurable hardware based systems," in *2007 IEEE Symposium on Security and Privacy (SP.* IEEE, 2007, pp. 281–295.

[2] F. Schellenberg, D. R. Gnad, A. Moradi, and M. B. Tahoori, "An inside job: Remote power analysis attacks on fpgas," in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2018*, 2018.

[3] D. R. Gnad, F. Oboril, and M. B. Tahoori, "Voltage drop-based fault attacks on fpgas using valid bitstreams," in *Field Programmable Logic and Applications (FPL), 2017 27th International Conference on.* IEEE, 2017, pp. 1–7.

[4] "Basic fpga architecture and its applications." [Online]. Available: https://www.edgefx.in/fpga-architecture-applications

[5] Wikipedia contributors, "Hardware description language — Wikipedia, the free encyclopedia," 2018, [Online; accessed 31-August-2018]. [Online]. Available: https://en.wikipedia.org/wiki/Hardware_description_language

[6] "Ip-cores." [Online]. Available: http://www.informatik.uni-ulm.de/ni/Lehre/SS03/ProSemFPGA/IP-Cores.pdf

[7] "Ip core (intellectual property core)." [Online]. Available: https://whatis.techtarget.com/definition/IP-core-intellectual-property-core

[8] S. A. Fahmy, K. Vipin, and S. Shreejith, "Virtualized fpga accelerators for efficient cloud computing," in *Cloud Computing Technology and Science (CloudCom), 2015 IEEE 7th International Conference on.* IEEE, 2015, pp. 430–435.

[9] Z. Wang and R. B. Lee, "Covert and side channels due to processor architecture," in *Computer Security Applications Conference, 2006. ACSAC'06. 22nd Annual.* IEEE, 2006, pp. 473–482.

[10] T. Iakymchuk, M. Nikodem, and K. Kępa, "Temperature-based covert channel in fpga systems," in *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2011 6th International Workshop on*. IEEE, 2011, pp. 1–7.

[11] D. R. Gnad, F. Oboril, S. Kiamehr, and M. B. Tahoori, "Analysis of transient voltage fluctuations in fpgas," in *Field-Programmable Technology (FPT), 2016 International Conference on*. IEEE, 2016, pp. 12–19.

[12] "Pynq-z1 python productivity for zynq." [Online]. Available: https://store.digilentinc.com/pynq-z1-python-productivity-for-zynq

[13] Wikipedia contributors, "Non-return-to-zero — Wikipedia, the free encyclopedia," 2018, [Online; accessed 31-August-2018]. [Online]. Available: https://en.wikipedia.org/wiki/Non-return-to-zero

[14] "Integrated logic analyzer (ila)." [Online]. Available: https://www.xilinx.com/products/intellectual-property/ila.html

[15] "On-chip design verification with xilinx fpgas." [Online]. Available: https://www.eetimes.com/document.asp?doc_id=1275831

List of Figures