Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Robin Rolf Fröhlich

Differential Power Analysis of MAC-KECCAK on a Field Programmable Gate Array

Fakultät Technik und Informatik

Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science

Department of Information and Electrical Engineering

Robin Rolf Fröhlich

Differential Power Analysis of MAC-KECCAK on a Field Programmable Gate Array

Bachelor Thesis based on the examination and study regulations for
the Bachelor of Engineering degree program
Informations- und Elektrotechnik
at the Department of Information and Electrical Engineering
of the Faculty of Engineering and Computer Science
of the University of Applied Sciences Hamburg

Supervising examiner: Prof. Dr. Heike Neumann
Second examiner: Prof. Dr.-Ing. Robert Fitz

Day of delivery: February 29th, 2016

Robin Rolf Fröhlich

Title of the Bachelor Thesis

Differential Power Analysis of MAC-KECCAK on a Field Programmable Gate Array

Keywords

FPGA, Side-Channel Analysis, differential power analysis, MAC-KECCAK, hash functions, cryptography, serial communication interface, PicoScope

Abstract

One of the newest hash functions to generate message authentication codes is MAC-KECCAK. It has been analyzed for side channel leakages, however previous works used hardware platforms dedicated for side channel analysis. In this thesis, an attack framework is developed that features an unprotected reference implementation of MAC-KECCAK running on a general purpose FPGA board. For this purpose, both software and hardware have been designed and integrated into the attack framework. The goal of this thesis is to create the attack framework. Further it is attempted to attack the underlying KECCAK implementation.

Robin Rolf Fröhlich

Thema der Bachelorarbeit

Differentielle Stromprofilanalyse des MAC-KECCAK auf einem Field Programmable Gate Array

Stichworte

FPGA, Seitenkanalanalyse, differentielle Stromprofilanalyse, MAC-KECCAK, Hash-Funktionen, Kryptographie, Serielle Kommunikationsschnittstelle, PicoScope

Kurzzusammenfassung

Eine der neuesten Hash-Funktionen, die genutzt werden kann, um Nachrichten-Authentifikations-Codes zu berechnen, ist MAC-KECCAK. Sie wurde bereits auf Seitenkanallecks hin untersucht, allerdings geschah dies auf Hardwareplattformen, die speziell für Seitenkanalanalysen ausgelegt sind. In dieser Thesis soll ein Angriffsaufbau realisiert werden, welcher das ungeschützte Referenzdesign des MAC-KECCAK angreift, welche auf einer Standard-FPGA Plattform implementiert ist. Hierfür wurden Soft- und Hardware entwickelt und in den Angriffsaufbau eingebettet. Das Ziel der Thesis ist hierbei die Entwicklung des Angriffsaufbaus. Ein Angriff auf die KECCAK Implementierung wird versucht.

Content

Chapter 1 - Introduction

Communication needs to be authentic. With today's computerized and networked world, it is even more important to have secure communication: to know who you are communicating with if you are talking to someone in person is easy. But, with unsecured computerized communication like the internet, one cannot easily differentiate between his communication partner and an adversary faking the communication partner's identity. Electronic communication therefore requires measures to be secure in the aspect of authenticity.

Cryptography has a solution to this problem. It can be used to cipher information to keep it secret, but also to authorize information. A special kind of functions provide excellent use for that purpose. These are the so called one-way functions. These one-way functions are relatively easy to perform in one way, but reversing the function is difficult. Hash functions are one-way functions. They take a relatively large binary input and generate a more or less pseudo random and relatively short output value from it. As explained later, hash functions can be used together with a secret key to authenticate messages in form of message authentication codes (MAC) [1].

Functions or other algorithm are implemented and computed on power dependent chips. These chips, as explained later, may leak information about the function through their power supply. Leaks can contain information about the operations computed. Therefore, it is possible to retrieve information about the computed operation by analyzing the power consumption. Such information can be critical, like the secret key used in MAC generation. This analysis of unintended information channels is called side-channel analysis (SCA). Trying to recover the key is considered an attack. A family of these attacks is differential power analysis (DPA) [2].

Modern hash functions, like KECCAK or its MAC generation variant MAC-KECCAK, have been analyzed and attacked in both software and hardware implementations. Most of these implementations have been performed on hardware platforms and test labs that have been specially designed for the needs of side-channel analysis. Examples are the SASEBO [3] or SAKURA [4] platforms. Unlike these examples, the task of this thesis is to build a test framework with an inexpensive general purpose field programmable gate array (FPGA) evaluation platform and inexpensive, easy acquirable equipment to proof that these attacks are applicable in a more realistic scenario. To do this, both hard- and software have to be created.

Chapter 2 - Background

Cryptography is a viable part of modern live. To have secure one-way functions hash algorithms have been developed. These hash functions are used to ensure data integrity, authentication and digital signatures. The algorithms are run either in software or dedicated hardware, just like any other algorithm. But, just like any other computer algorithm, they need some sort of microchip that does need a power supply and may leak information about the algorithm or the data involved. These leaks can be found on physical channels, called side-channels, as they have never been intended to be used by the designers. Examples for this are power consumption or electromagnetic radiation.

This chapter gives background information about cryptographic hash functions, how KECCAK and MAC-KECCAK function, and how side-channel analysis in general works. It also introduces the attack method, in this case differential power analysis (DPA). For better understandable examples three personas are used, Alice and Bob as communication partners, and Eve as the adversary.

2.1 About Cryptographic Hash Functions

Most of today's cryptographic hash functions work by taking a large data input to compute a small unique output that is dependent on every single bit inserted. If one bit flips, usually the output changes significantly. This property can be used to check data integrity. An example:
Some data of Alice needs to be correctly transferred to Bob. Alice runs her data through a hash function to calculate a hash value, also called checksum. After she transferred the data and the checksum to Bob, he computes the hash of the transferred data and compares it against the received checksum. If the data was corrupted, for example by interference, the comparison will fail and Bob cannot use the data.

This principle can be used for checks of data integrity, but also for message authentication. In that case, a secret key is negotiated between Alice and Bob. This key is added to the message and run through a hash-function to compute a MAC. Since only the communication partners Alice and Bob know the key, an adversary, like Eve, that wants to fake a message by Alice, cannot fake the MAC and thereby the fake message can be detected by Bob.

2.2 About KECCAK and MAC-KECCAK

Today, there are plenty of available hash functions. To get a worldwide standard, the US National Institute of Standards and Technology (NIST) has issued competitions, making the winners the standard cryptographic hash algorithms used around the world. The winner of the third and most recent competition, the Secure Hash Algorithm - 3 (SHA-3) competition, is formerly known as KECCAK [5]. The information about KECCAK is taken from its reference [6].

KECCAK is a sponge algorithm, first absorbing the input data in multiple cycles and later squeezing the output data in an arbitrary number of cycles. Data is internally stored in a three dimensional bit pattern shown in Figure 2-1. This pattern is called *state* and is described by $S(x, y, z)$, with x and y being 5 bit and z being 64 bit long in the SHA-3 configuration. A single bit is described by $s[x][y][z]$, with $x, y \in \{0, \dots, 4\}; z \in \{0, \dots, 63\}$. This results in 1600 bit per state. 1024 of these bit, called rate r, are used as input with the other 576 bit used as capacity c.

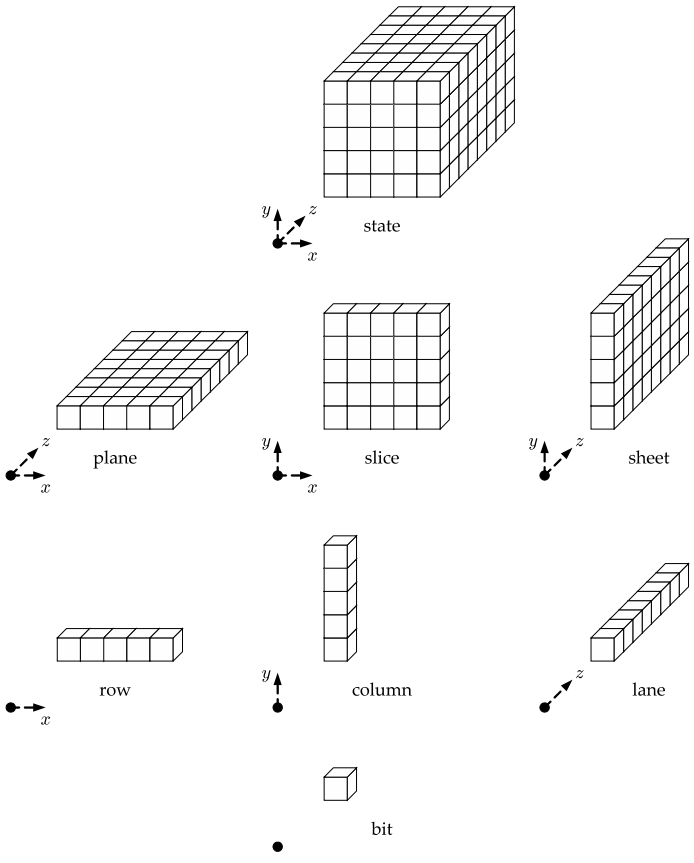

Figure 2-1. Naming conventions for parts of the KECCAK - f state [7]

A cycle consists of 24 rounds, with each round having 5 different round functions named θ, ρ, π, χ, and ι. The round functions are applied as followed:

$$Output = \iota \circ \chi \circ \pi \circ \rho \circ \theta\,(Input) \tag{2-1}$$

The round functions are briefly described below:

- θ:
  The θ-function is a binary XOR operation between a bit and two of its neighboring columns (see Figure 2-2). It can be parted into two phases, phase one being the parity plane generation of every column and phase two the XORing of each bit with two neighboring column parity bits. Both steps can be described as followed:

$$\theta_1 \colon parity[x][z] = \theta_{plane} = (\bigoplus_{i=0}^{4} s[x][i][z]) \tag{2-2}$$

$$\theta_2 \colon s[x][y][z] = s[x][y][z] \oplus parity[x-1][z] \oplus parity[x+1][z+1] \tag{2-3}$$

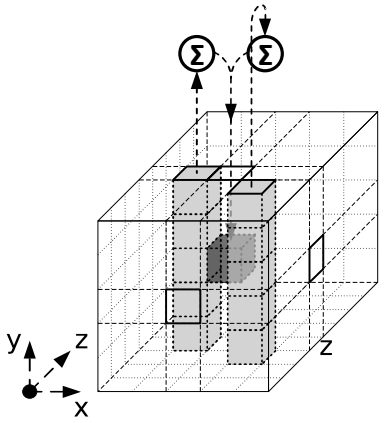

Figure 2-2. The θ- Operation [8]

- ρ and π:
  Both functions are simple linear permutations of the bits over the state.
- χ:
  The χ-function is a non-linear combination of AND, XOR and NOT operations. Each bit is XORed with the result of the AND of an inversed neighbor bit and another neighbor bit, described as followed:

$$\chi \colon s[x][y][z] = s[x][y][z] \oplus \left(\overline{s[x+1][y][z]} \cdot s[x+2][y][z]\right) \tag{2-4}$$

- ι:
  The ι–function applies a round constant with XOR.

The sponge construction, as shown in Figure 2-3, consists of two phases, the absorbing and the squeezing phase. Prior to all operations the state bits are reset to 0. The input message is then divided into blocks of integer multiples of r. If the message does not exactly match an integer multiple of r, padding is appended. Padding has the style 10*1 with as many zeros in the middle as required to reach an integer multiple of r.

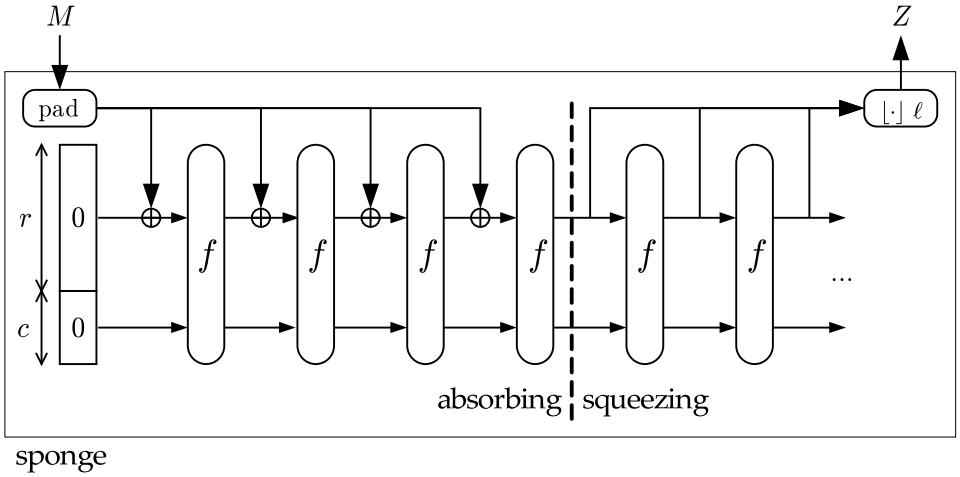

Figure 2-3. The sponge construction [9]

In the absorbing phase the first message block is XORed into the state. Afterwards, a full set of 24 rounds is applied to the state. Then, the next input block is XORed to the state followed by another 24 rounds of hashing. This is continued until all input blocks have been included. Afterwards, in the squeezing phase, the current state is put out in a similar manner but without XORing. Not all bit of the state are returned as output. For SHA-3, the output sizes are fixed at 224, 256, 384, or 512 bit to replace older SHA-1 or SHA-2 easily [10]. In any case, the first n bits of the state are returned. If the output size n is larger than the rate r, only r bit are put out, and another cycle of KECCAK is applied. This is repeated until all n bit are put out. Therefore, depending on the output size, a different number of squeezing rounds is applied. This makes it possible to use the same implementation for various output lengths.

For MAC generation, the message is simply prepended with a key of any desired length before being split into blocks. This is called MAC-KECCAK. Since not the entire state is represented by the output, a nested MAC construction like in keyed-hash message authentication codes (HMAC) [1] is not necessary [11]. The construction is described as followed:

$$MAC(M, K) = H(K||M) \tag{2-5}$$

2.3 About Side-Channel Analysis

The power consumption of any chip changes with the operations it is performing. The consumption may also change by different data involved in the operation. By theory, it should thereby be possible to gather information about the current operation or of a system by observing its power consumption. This was never intended by the designers of the hardware or software. Therefore, this information is gathered on side-channels. Cryptographic hard- and software are also prone to side-channel attacks. Since most cryptographic include a secret key but the algorithm is publicly known, key recovery is the target of most attacks.

Side-channel analysis can be done on different channels or methods. For power analysis, there are two major methods: simple power analysis (SPA) and differential power analysis (DPA). In a SPA, an adversary may detect time profiles and patterns by looking at a single or an averaged number of captured power traces. Some operations may use more power than others and feature a specific pattern in the power trace. Just by examining the trace, an adversary may gather information about the operation or the data used. A famous example is the RSA [12] using the square and multiply [13].

Differential power analysis is different compared to SPA. On modern semiconductors, an operation changing the value of a bit drains more power than the bit keeping its value. Mathematically speaking, the Hamming distance of an operation changes the power drain. Therefore, a bit change can be detected as a small spike in the power consumption. In a complex system, this small spike is usually masked by noise. A power model calculates the theoretic power consumption of the algorithm or system by combination of specific bit changes at a specific time. Now many power traces can be compared to or correlated against said power model. High correlation will show where the power model was correct [14]. Most power models use the Hamming distance (HD) of specific algorithm steps or their Hamming weight (HW), which is the HD compared to all bit set to 0.

In order to use DPA to gather key bits, key assumptions need to be made. Also, parts of the input and/or output of the algorithm need to be known to the adversary. Together with the assumed key, a theoretical power consumption can then be calculated. This theoretical consumption is than correlated against the captured power traces with the correct time alignment. The best assumption will show the best correlation at the expected time [14].

Other forms of correlation power analysis only differ from DPA in their used power models. Therefore, all correlation attacks are part of the same family of attacks [15].

Chapter 3 - Previous Work

KECCAK has been already been analyzed by various authors in the aspect of side-channel leakage. One of the earliest works has been published by Taha et al. [11]. In their paper "Side-Channel Analysis of MAC-Keccak" a software implementation of MAC-KECCAK running on a 32 bit MicroBlaze™ [16] soft-core processor implemented on a Spartan-3e FPGA is attacked. Also, the key-length's impact on the attack difficulty and possible attack points are introduced and discussed. These attack points include the θ and the χ step of the first round. As this attack is performed on a software implementation, it is not directly applicable on a hardware implementation.

In "Power Analysis Attack on Hardware Implementation of MAC-Keccak on FPGAs" Lou et al. [17], the unmasked reference hardware implementation of MAC-KECCAK [18] is attacked. A SASEBO-GII [3] featuring a Xilinx Virtex-5 FPGA is used for implementation. Four power models are introduced, all of them focusing on the θ step of the first round. A fixed key length of 320 bit is used, partially based on the results of Taha et al. [11].

In another work written by Tran [19], the unmasked reference hardware implementation of MAC-KECCAK using a fixed 320 bit key length is attacked. But unlike Lou et al.'s work [17], simulated power traces are used instead of physically captured power traces of actual hardware.

This chapter introduces and discusses the power leakage models, the selected key-length, and the chosen hardware platform used in the applied attack of this thesis in contrast to previous works.

3.1 Power Leakage Models

A DPA is only as good as its underlying power model. An ideal power model delivers best correlation with the fewest number of traces to recover the required information. In their previous work, Lou at al. [17] discovered four power models that can be used to recover the secret key used in MAC-KECCAK. All of these models focus on the first round because the key bits are directly used. The operations and important time points are briefly described below (cf. [17], p. 4, see Figure 3-1):

- The input is XORed into the state register after padding. Since the state register is reset to 0 at the start of a new hash computation, this step can be ignored.
- The parity plane θ_{plane} is computed in θ_1, this takes t_1 time.
- At t_1, the output of θ_1 changes from 0 to θ_{plane}.
- θ_2 is computed, this takes t_2 time.
- At t_2, the output of θ_1 is still 0. Therefore, the output of θ_2 changes from 0 to P.
- At $t_1 + t_2$, the output of θ_2 changes from P to θ_{out} in reaction to the change of the output of θ_1 from 0 to θ_{plane}.

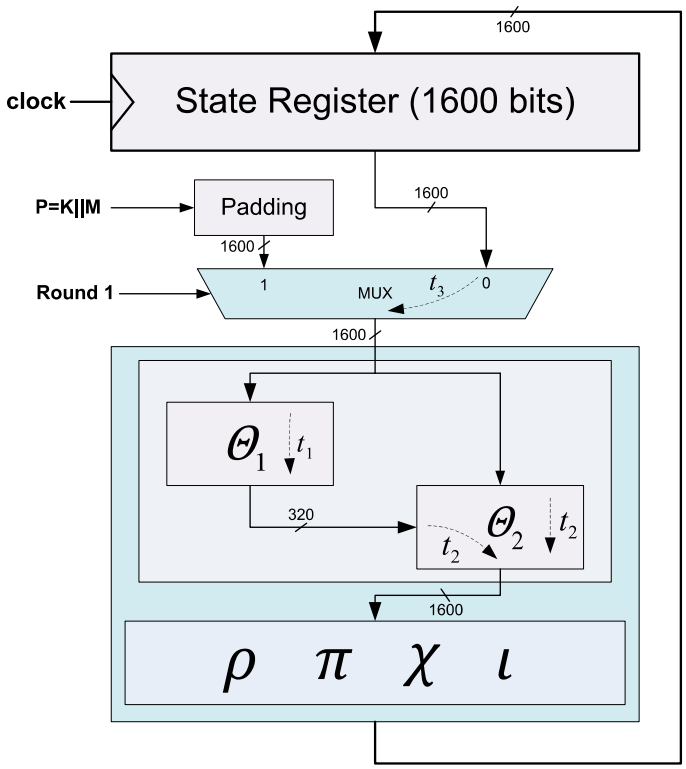

Figure 3-1. Structure of Keccak hardware implementation [17], p. 5

The models are described as follows ([17] , pp. 4-6):

Model I: "At t_1, [the output of θ_1] changes from 0 to θ_{plane} and thus the HD is HW$\left(\theta_{plane}\right)$. There should exist a time point on the power trace that has good correlation between its power value and HW$\left(\theta_{plane}\right)$."

Model II: "At $t_1 + t_2$, [the output of θ_2] changes from P to θ_{out} and thus the HD is HD(P, θ_{out}). There should exist a time point on the power trace that has good correlation between its power value and HD(P, θ_{out})."

Model III: "The state register output changes from 0 to R_1. There exists a time point in the second clock cycle that is correlated with $\mathrm{HW}(\theta_{out})$, (actually HW$(R_1)$)."

Model IV: "The output of the multiplexer changes from P to R_1. There exists a time point in the second clock cycle that is correlated with HD(P, θ_{out}), (actually HW(R_1))."

Lou et al. have discovered that **Model III** has the best correlation. However, it is not sufficient to recover all 320 key bits. On the other hand, is possible to recover all key bits with **Model I**, but it requires more traces due to lower correlation compared to **Model III**.Using **Model I**, the adversary is able to recover any key byte at the cost of the number of required traces. (To achieve better correlation than a single bit would have, eight bit sets are used in **Model I** by Lou et al. This results in retrieving key bytes instead of single key bits.) Thus, for a proof of concept it is sufficient to recover one key byte using **Model I**.

The intermediate values for the first key byte are calculated as followed
(cf. [17], formula (11)):

$$HW\left(\theta_{plane}(x, 0{:}\,7)\right) = \sum_{Z=0}^{7}\left(\bigoplus_{y=0}^{4} S(x, y, z)\right) \text{ with } x = 0 \tag{3-1}$$

The correlation model is therefore calculated as followed ([17] , formula (12)):

$$correlation\left(\sum_{Z=0}^{7}\left(\bigoplus_{y=0}^{4} S(x, y, z)\right), power\right) \tag{3-2}$$

As the goal of this thesis is to get a proof of concept, a single key byte will be recovered using **Model I**. Despite the low correlation this model is sufficient to recover all key bits at the cost of computation time and number of required traces.

3.2 Key-Length

Key length has a significant impact on attack difficulty. A longer key is more difficult to attack, however message throughput decreases with a larger key length. Taha et al. [11] discovered the dependency of key length and attack difficulty regarding MAC-KECCAK. The following descriptions are based on their work.

In software implementations it is possible to recover more than one unknown bit per θ step. This is due to the way the parity plane generation is implemented. As displayed in Figure 3-2, the targeted XOR operation is staged into four steps, providing three possible attack points. With these multiple attack points it is possible to recover multiple bits. Therefore, in software implementations a key longer than one plane (320 bit) can be recovered by SCA.

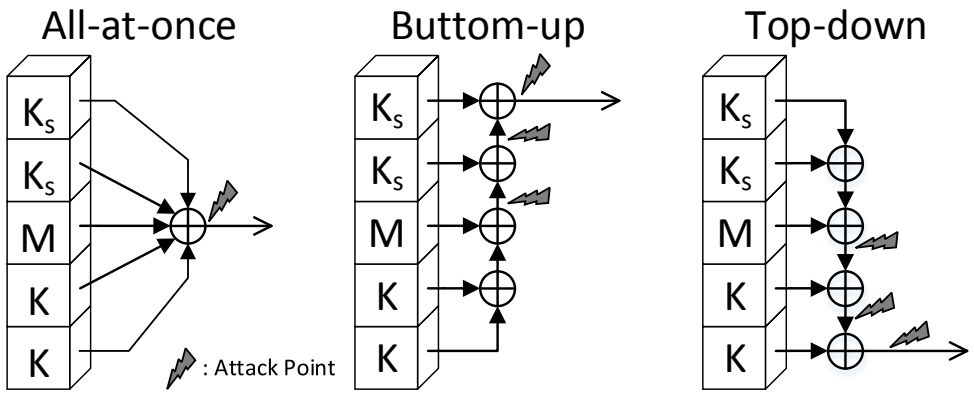

Figure 3-2. Column XOR of θ operation [11], p. 128

Parity plane generation in hardware is implemented directly using a five input XOR. This provides only one attack point. Following that, the value of multiple unknown bits cannot be recovered. A key length longer than 320 bit would fill up more than one plane resulting in two or more unknown bits per XOR operation. With the key length set to 320 bit, the key bits fill up one plane resulting in only one unknown key bit per XOR operation which can be recovered. Thus, the key length is chosen to be 320 bit, as a smaller key would compromise security and a larger key would not to be attackable with the previously described power model.

3.3 Used Platforms

Previous work used SCA-dedicated hardware platforms, like Lou et al. [17] have done using a SASEBO-GII [3]. This platform features a dedicated power measurement connector enabling accurate power trace capturing. However, this platform is export restricted and can only be obtained on request. Also, it does not provide a realistic attack scenario. All of this is problematic if applicability is tested.

Another approach has been selected by Tran [19], who has used simulated power traces rather than an actual hardware implementation. This is a new approach, but is even further away from a real life attack scenario. The implementation and capture times are reduced at the cost of simulation time and complexity. Nevertheless, this approach might be appropriate in research and development of attack counter-measures. Also, it does give new ideas and perspectives on SCA and its weaknesses.

Chapter 4 - Task

In this thesis an unprotected MAC-KECCAK implementation shall be implemented onto a general purpose FPGA evaluation board to develop an attack framework capable of acquiring power traces and later processing that is required for differential power analysis. Measurement equipment shall be standard and easily acquirable. For this purpose, a PC operated oscilloscope shall be used for trace capture. The goal of developing the attack framework is to see if the theoretically proven attacks on SCA-optimized platforms, such as the SASEBO-GII or simulated hardware, can be applied to a more realistic test framework.

For hardware implementation the unprotected reference implementation of KECCAK is used. It needs to be supplied with data for processing. Therefore, the hardware implementation is required to have some kind of communication interface. The RS232 serial communication interface is used for that purpose. An external core power supply is needed since the platform is run at high clock frequencies, this should ease control as well as measurement of the power consumption. The internal oscillator is used as the clock source to keep the framework as simple as possible and to avoid further modifications of the FPGA board. Clock modifications like divisions have therefore to be done inside the FPGA itself.

This test framework should be able to capture power traces of MAC-Keccak with a specified key together with the corresponding random input messages automatically. Therefore, a software has to be developed that operates the oscilloscope and generates and exchanges messages with the FPGA board.

Chapter 5 - Hardware Architecture

One of the tasks is to implement KECCAK in hardware. Since MAC-KECCAK only varies from KECCAK in its input message, the implementation is designed to calculate the hash value from the input unprocessed, answering the output directly without any other control possibilities for the sake of simplicity. This shifts any key or padding related tasks to the message generation.

A modular approach is chosen to have simpler state machines compared to an all-in-one implementation approach. As most of the used modules are by third parties, an all-at-once approach would be more difficult and therefore more likely prone to errors.

Means of data input and output supply are required by the design. For this purpose, the serial communication interface RS232 is used as it is easily implementable, widespread, it is included in most general purpose FPGA evaluation boards, and supported by most personal computers either directly or via emulating adapters.

This design results in an external hash computation chip that is connected with an RS232 interface.

All modules are implemented in VHDL using Vivado by Xlinix. To avoid optimization and replacement of unconnected modules, the *dont_touch* attribute is introduced. Other synthesis and design implementation settings remain standard. The VHDL sources can be found in Appendix A.

In this chapter, the hardware concept is introduced, followed by the description of the architectural top level. At last, the included state machines are described.

5.1 Concept

To achieve a modular architecture, the functionality is divided into four parts:

- *clock management*,
- *control*,
- *serial communications*,
- and *hash computation*.

Clock management is required as the oscillator frequencies offered by the used hardware platform might not include an integer multiple of one of the commonly used *serial communication* baud rates. This is due to the *serial communication interface* requiring 16-times oversampling compared to the selected baud rate. Also, the clock frequency can be lowered for reduced power trace bandwidth, easing power trace capture.

The *control* block is be responsible for initiating the *serial communication* and the *hash computation* blocks upon reset. It also starts either receiving an input or sending an output by the communication block or start the hash computation. It features a converter element that parses between *hash* and *serial communication* message format and synchronizes the input reset signal.

The *serial communication* block receives input messages and sends output messages. The message length is hardcoded into blocks of 128 byte for input and 32 byte for output, adapted to the *hash function*'s input and output length. To see if the device is reset properly, a one byte reset message is sent.

The *hash computation* block features the KECCAK implementation. Upon the start signal, it computes a hash value from its input to its output. Afterwards, it resets the Keccak implementation and waits for a new input block. The input block is the same size as rate r. It is 1024 bit long and divided into 16x64 bit blocks. The output block is 4x64 bit long. It also drives trigger pins that indicate the start and end of *hash computation*.

Upon reset, a reset signal is sent. Afterwards, the *serial communication block* starts listening for a message. After the message block is received, the input is parsed into the *hash function*'s input block format and the *hash computation* is started. After the *hash computation*, the hash output is parsed back into output bytes and sent back via the serial communication block.

5.2 Architecture Overview

The designed hardware architecture is displayed in Figure 5-1. It features a top level *clock management* (CLK_MAIN) with a clock division component (CLK_DIV) and the MAIN component block with underlying features as described in section 5.1. Clock management has to be at top level because of synthesis restrictions. For this purpose the Clocking Wizard intellectual property (IP) core by Xlinix [20] is used.

In MAIN level, the *control* block is implemented as a state machine (CONTROL_stm). The reset signal synchronization however is implemented in its own process, RES_SYNC. As some of the integrated components use a low active reset, the synchronized reset signal NRES is also designed as low active. The message block conversion is done inside the CONVERTER component. *Serial communication* is handled by the RS232_stm component with RX and TX as communication lines. *Hash computation* is handled by the SHA3_stm component. To increase power consumption this component is doubled with a dummy component instance. However, the dummy component's outputs remain unconnected. The start and stop triggers are respectively represented by TRIG_BIT and STOP_BIT.

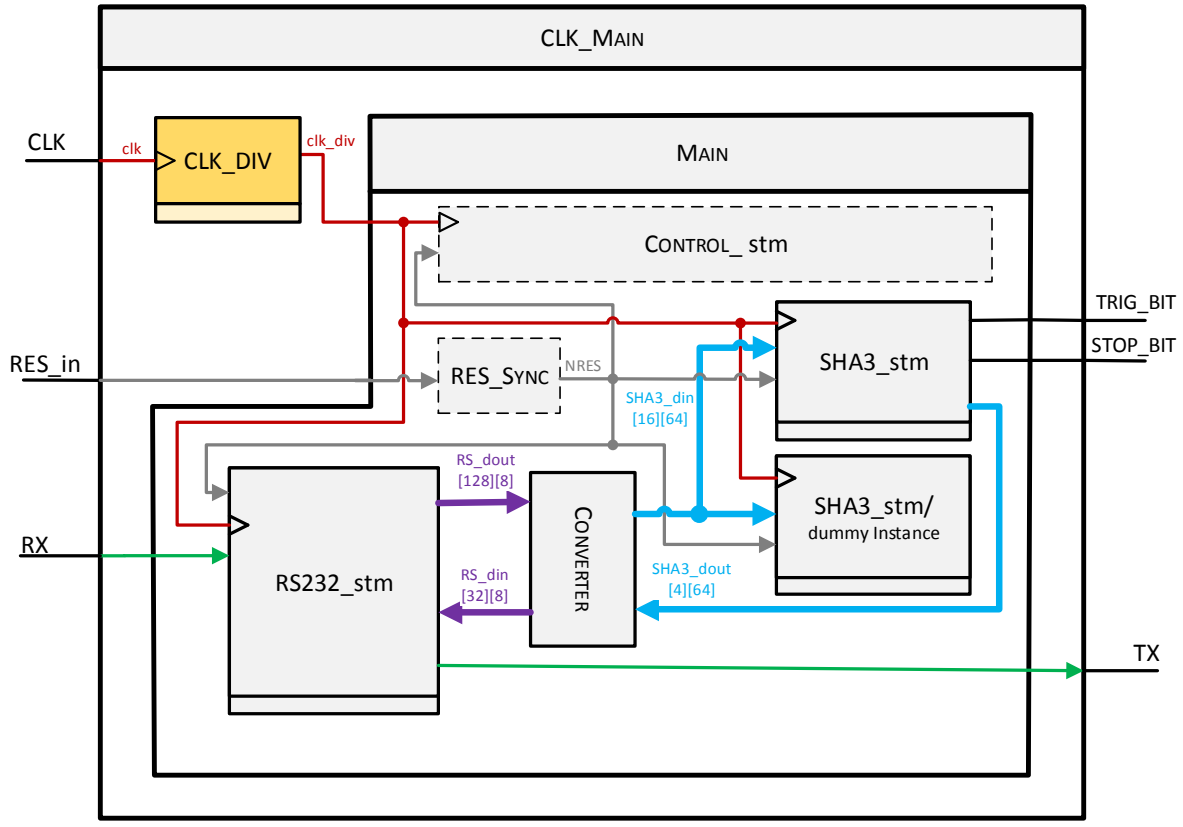

Figure 5-1. Simplified, top level overview of the implemented hardware design. Most internal signals are omitted for better overview.

5.3 Control

As already pointed out in 5.1 and 5.2, the control block is responsible for intercomponent communication. It features a state machine, see Figure 5-2, that controls start signals for *reception* (start_recv), *hashing* (start_kec), and *transmission* (start_trans) and reacts on corresponding termination signals (done_recv, done_kec, and done_trans) by the components. Additionally, intermediate states between each phase allow for *block format conversion* (conv_r2k and conv_k2r) by the CONVERTER component or *register reset* (INIT).

A cycle starts with the *input reception* phase (green), followed by the *hash computation* phase (red). At last, the output is transmitted in the *output transmission* phase (blue). Each phase starts by setting the corresponding *start* signal and terminated by the component setting the *done* signal.

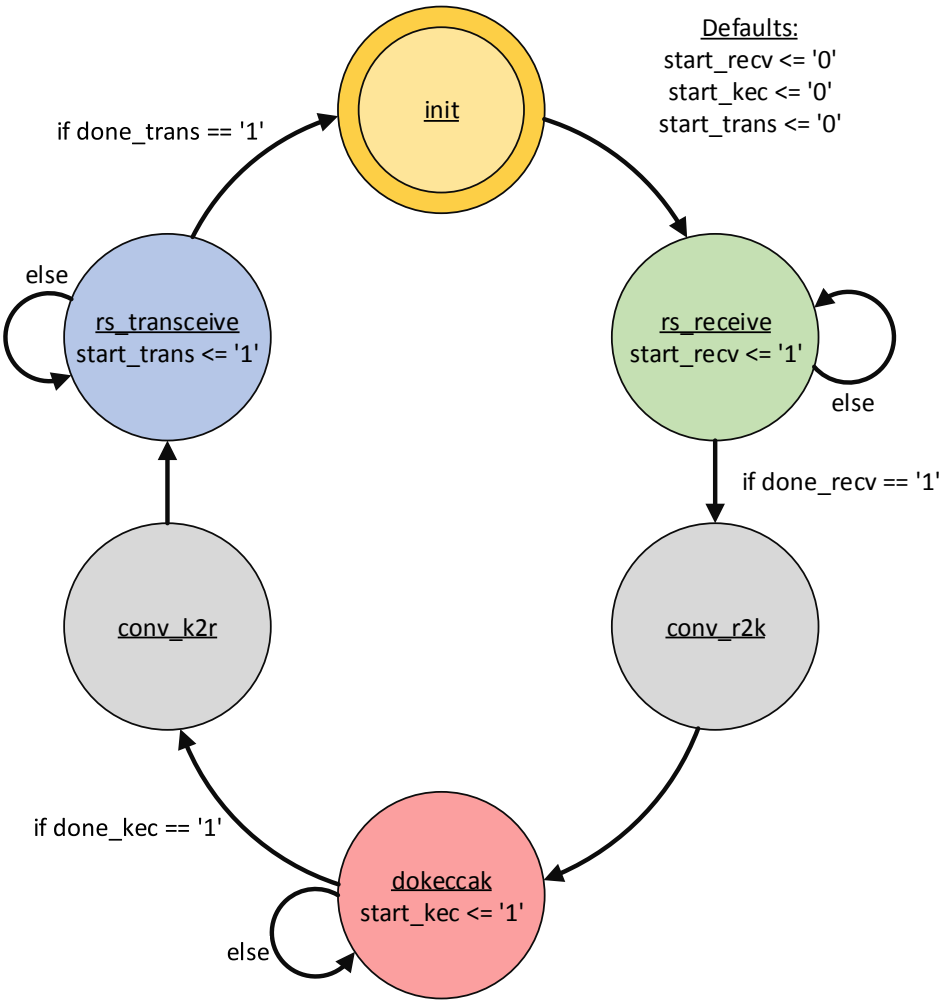

Figure 5-2. Control state machine diagram. Green state is *input reception* (rs_receive), red state is *hash computation* (DOKECCAK), and blue state is *output transmission* phase (rs_transceive). Grey states are intermediate states that allow for *block format conversion* (conv_r2k and conv_k2r) inside the CONVERTER component. The yellow state allows for *register reset* (INIT)

5.4 Serial Communication Interface

As seen in Figure 5-1, the *serial communication* block is implemented in component RS232_stm. This component features a state machine and a universal asynchronous receiver transmitter (UART) component by Bennett [21], carrying out the serial communication. The state machine is based on Figure 5-3, displaying its state diagram.

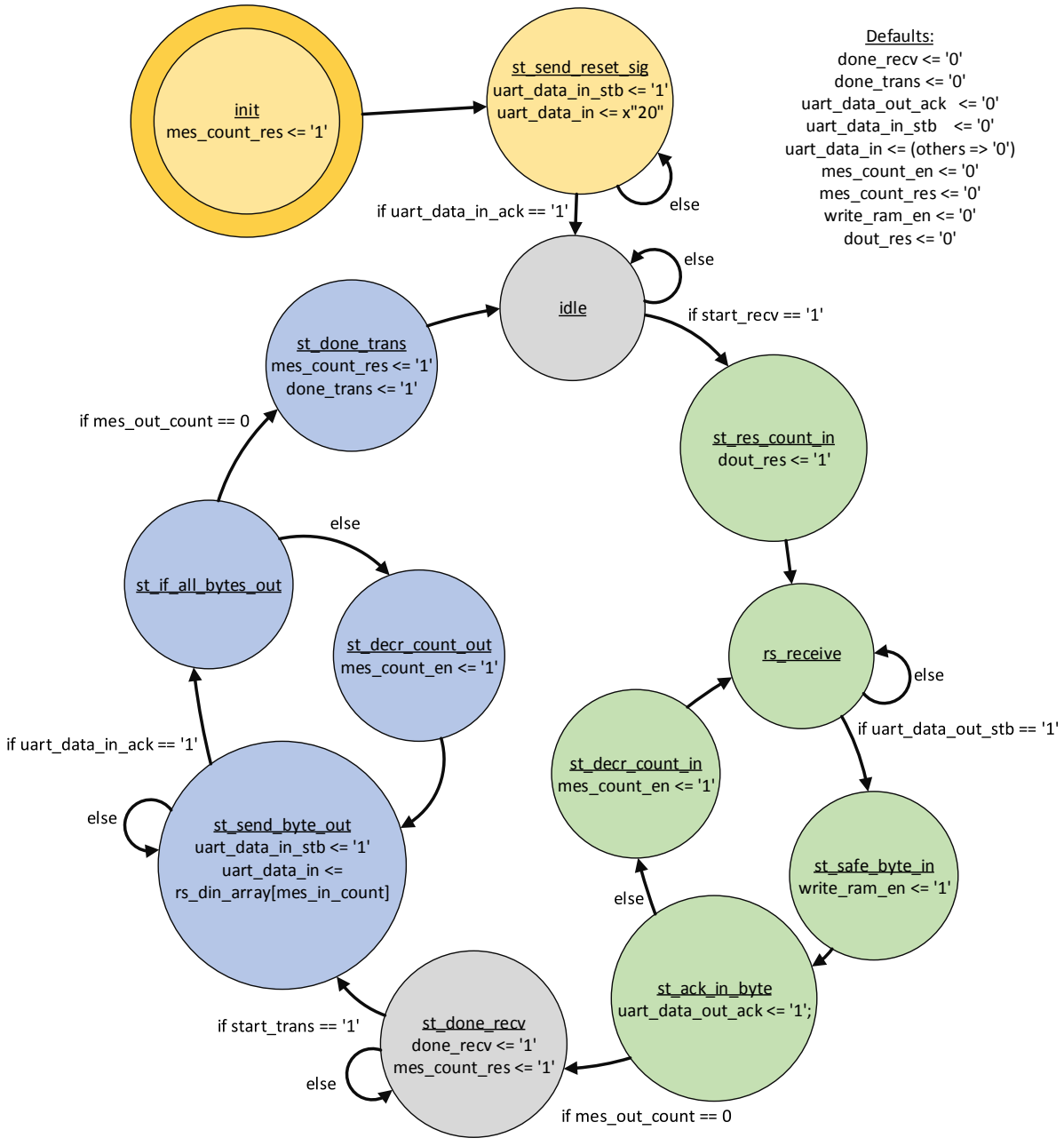

Figure 5-3. RS232_stm state diagram. Yellow states are part of the reset routine, green states indicate the *input reception* phase, blue states indicate the *output transmission* phase. Grey states display *wait* on signal phases.

The state machine has 5 phases:

- *reset routine* (yellow),
- *wait on start reception signal* (grey),
- *input reception* (green),
- *wait on start transmission signal* (grey),
- and *output transmission* (blue).

In the *reset routine*, a reset signal byte is transmitted. The reset signal has a value of 02_{16}, representing the ASCII character for a whitespace. Afterwards, the state machine continues with the *wait on start reception signal* phase.

When the start receiving signal (start_recv) is set, the *input reception* phase is entered and the input message register is reset (dout_res). After a full byte has been received, acknowledged to the UART modules, and saved; the input counter (mes_out_count) is decremented (mes_count_en). If the input counter reaches zero, the reception phase is completed. Otherwise another byte will be received. Upon completion, the *wait on start transmission signal* phase is entered, where the reception completed bit (done_rs) is set.

After the start transmission signal (start_trans) is set, the state machine will enter the *output transmission* phase. In this phase, the current output byte is transmitted. If a byte has been fully transmitted, the transmission counter (mes_count_out) is decremented (mes_count_en). If it reaches zero, all bytes have been transmitted. The state machine will then set the transmission completed bit (done_trans) and start a new cycle by entering the *wait on start reception* phase. Otherwise the next byte will be transmitted.

Furthermore, the message counters for both input (128) and output (32) are reset to their corresponding value prior to every *wait on start reception signal* and during the *wait on start reception* phase.

5.5 KECCAK Implementation

The KECCAK implementation is based on the testbench *'tb_keccak.vhd,'* found inside the reference KECCAK implementation "High Speed Core" [18]. As Tran also used this implementation, he provides a detailed description in his thesis [19]. The modified state machine (SHA3_stm, see Figure 5-1) that feeds an underlying KECCAK component is represented by its state machine diagram (see Figure 5-4).

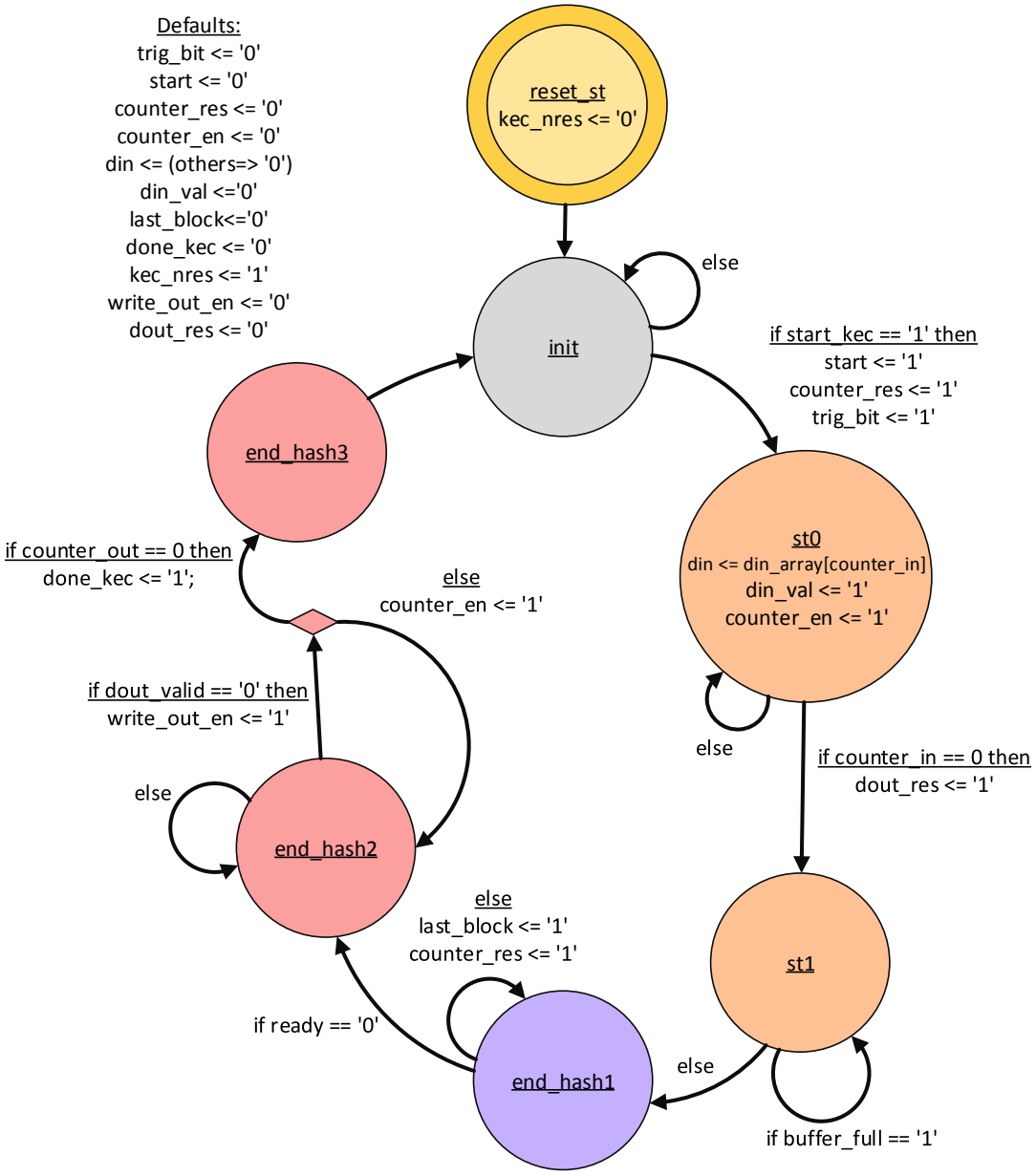

Figure 5-4. SHA3-stm state diagram. Orange states indicate the *absorbing* phase of KECCAK, red states the *squeezing* phase, and the lavender state the *hash computation* phase (end_hash1). In the grey state, the start signal is awaited (init).

The state machine iterates through four phases and the reset routine (yellow). These phases are *waiting* on the start signal (grey), *absorbing* the input into the Keccak buffer (orange), *computing* the hash (lavender), and *squeezing* the output (red). In the reset routine, the KECCAK component is reset before starting to *wait* on the start signal.

After the hash computation is started by setting the start signal (start_kec), the *absorbing* phase begins. In this phase, the input block is loaded into the KECCAK buffer. At start, the KECCAK component is signaled to start loading the input into the Keccak buffer (start). Furthermore, the external trigger signal is set (trig_bit), and the index counters are reset (counter_res). To load the input blocks (din_array) into the Keccak buffer input (din), the index of the current input block (counter_in) is decremented by enabling the index counter (counter_en). When the counter reaches zero the output register is reset (dout_res). The Keccak buffer will then indicate that all blocks have been loaded (¬buffer_full).

The *hash* is then *computed* by the KECCAK component. As the input length is fixed to a single rate r, respectively 1024 bit, the KECCAK component is signaled that this is the last input block (last_block). Also, the index counter is reset (counter_res). Once the Keccak component indicates the *hash computation* is done (¬ready), the *squeezing* phase begins.

In the *squeezing* phase, the state machine will wait on the Keccak component to signal its data output is valid (¬dout_valid). Once this happens, the output (dout) is stored into the output blocks (dout_array, write_out_en). The current output block is indicated by the output index counter (counter_out). Once all output blocks have been stored, the state machine indicates the concluded hash computation (done_kec, which directly routed as the competition bit stop_bit) and starts waiting on the next start signal. Otherwise, the output index counter is decremented (counter_en).

Chapter 6 - Software Implementation and Attack Framework

To attack MAC-Keccak running on an external FPGA board that is designed as described in Chapter 5 - Hardware Architecture, several tasks have to be fulfilled: first, data input and output have to be exchanged using a serial communication interface. Sending an input will start the hash computation. Second, the power consumption of the FPGA board during the hash computation has to be captured by an oscilloscope. Third, the hypothetical power consumption for each input has to be calculated based on the chosen power model. This has to be done for each piece of key that is tried to be recovered. And last, the captured power traces have to be correlated with the hypothetical power consumption of each guessed key piece.

As the transmission of input data to the external computation chip will start hash computation, it is necessary to synchronize this with power trace capture. The easiest way to synchronize these tasks is to implement them into the same software. This software should feature multiple threads and should be using events for synchronization purposes.

To increase the signal to noise ratio (SNR) of the power traces, input messages are hashed multiple times. By averaging the corresponding power traces, the environment noise is reduced, resulting in a higher SNR for the averaged power trace.

The key used for all generated MACs is (displayed as hexadecimal values):

$$(A0\ A1\ A2\ A3\ A4\ A5\ A6\ A7\ A8\ A9\ AA\ AB\ AC\ AD\ AE\ AF$$
$$20\ 21\ 22\ 23\ 24\ 25\ 26\ 27\ 28\ 29\ 2A\ 2B\ 2C\ 2D\ 2E\ 2F$$
$$C0\ C1\ C2\ C3\ C4\ C5\ C6\ C7)_{16}$$

In this chapter, the test framework is introduced by naming the used equipment and software, detailing the performed modifications and written software operation.

6.1 Used Equipment and Software

The attack framework consists of a modified Basys3 general purpose FPGA evaluation board by Digilent featuring an Artix7 FPGA by Xilinx with a laboratory power supply, a personal computer running Microsoft Windows 7 and Visual Studio, as well as a PicoScope® 5444B by Pico Technology. As probes for trace capture and triggering, the included TA131 probes are used in 10x setting. Correlation traces are computed in Octave by using parts of the sample DPA script provided by Oswald [22], see Appendix C.

For trace capture and serial communication, a C++ program is used. For oscilloscope control the PicoSDK [23] and the underlying PicoScope application programming interface (API) is used. For serial communication, the serial communications routine [24] by Microsoft are used. This routine is backed up by the Windows software development kit (SDK). All C/C++ sources and the Visual Studio solution can be found in Appendix B. Modifiable compile time settings for this software can be found in the “settings.h” file.

Some of the KECCAK C-Sources, taken from [25], have been used as a starting point for power model computation. These are computed offline after trace capture has been completed based on the chosen power model. This allows for other power models to be applied if necessary.

The Basys3 features only one 100 MHz oscillator. This frequency is too high for the PicoScope 5444B and the TA131 probes, as the PicoScope 5444B is able to capture two channels in an eight bit resolution with a sample period of 2 ns. In order to allow the RS232 implementation to use a baud rate of 115200 baud per second and to ease trace capture, the clock manager of the FPGA implementation is configured to divide the 100 MHz clock into an 18.432 MHz clock signal.

6.2 Framework Overview

The attack framework is displayed in Figure 6-1.

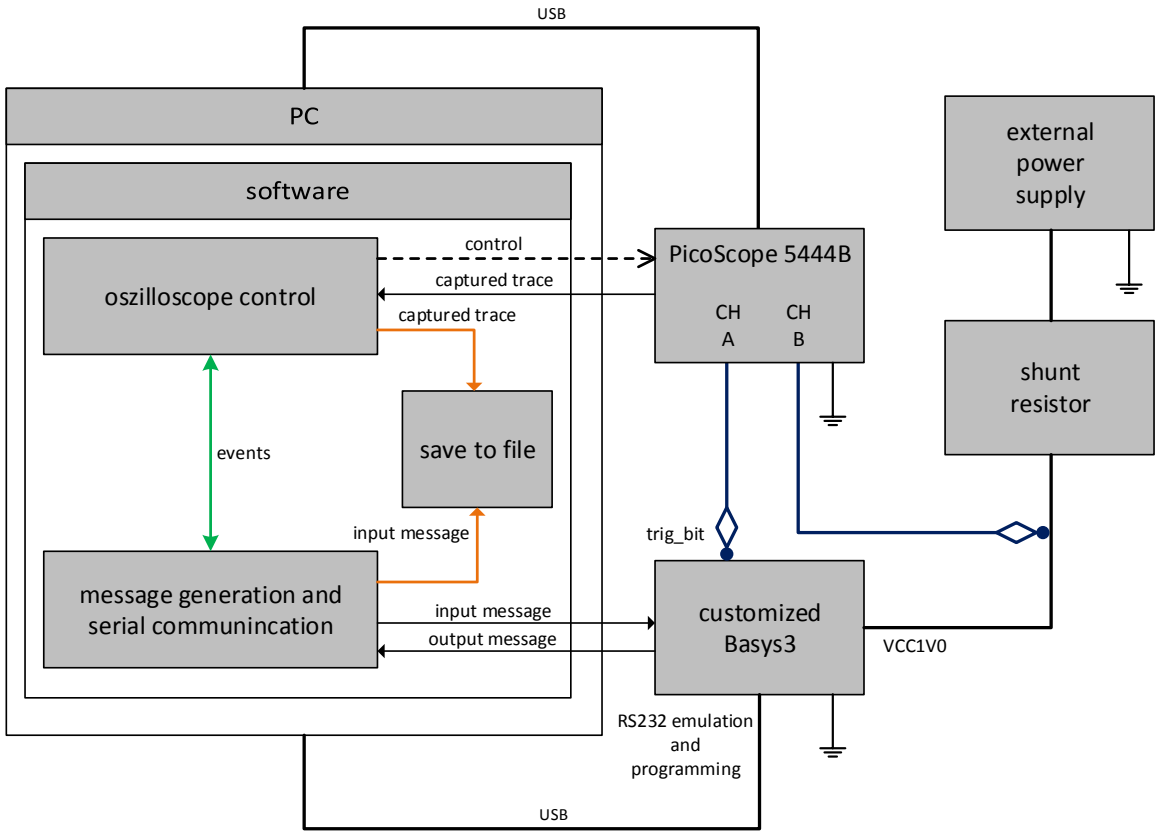

Figure 6-1. Attack Framework overview. The software for oscilloscope control and message generation and exchange is running on a PC. Both the PicoScope and the customized Basys3 are connected to the PC via USB. The Basys3 core voltage (VCC1V0) is provided by a laboratory power supply with a shunt resistor in line. The PicoScope triggers on CH A measuring the trig_bit output pin of the Basys3. On CH B the Basys3 core supply voltage is measured with an -1 V offset.

The PC is connected to both the PicoScope and the Basys3 via USB, as the Basys3 features a serial communication interface driver that emulates an RS232 connection via the USB link. The Basys3 is modified and features a custom power supply. More details on this can be found in 6.3. The software performing both the trace capture and the serial communication with the Basys3 board is detailed in 6.4.

6.3 Basys3 Modifications

The used Basys3 board needs to be modified in order to capture the included Artix7 power consumption. The reference [26] and modified board schematic can be found in Appendix C. A simplified version of the power trace capture schematic is displayed in

Figure 6-2. The Basys3 voltage supply is set to USB. The applied modifications will disable the power switch.

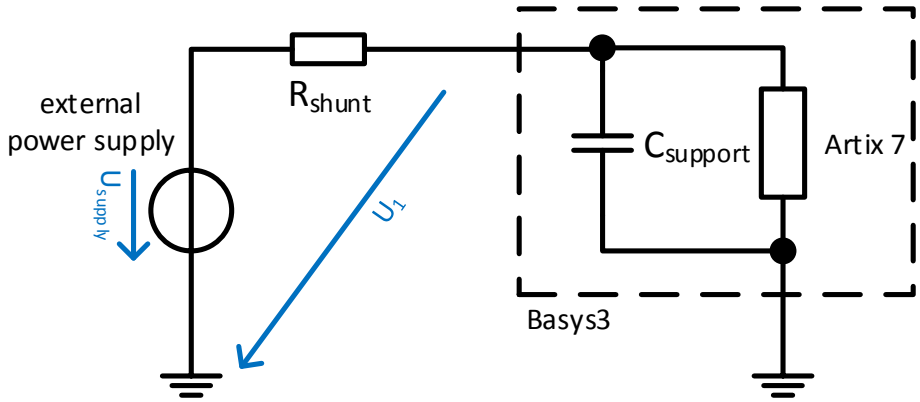

Figure 6-2. Simplified power trace capture schematic. Most of the C_{supply} capacitors are removed. R_{shunt} is chosen to be 500 Ω. U_{supply} is adjusted so that $U_1 = 1$ V. For power trace capture, U_1 is captured with a -1 V offset.

To be able to insert a custom power supply, the integrated voltage regulator has to be disconnected from the FPGA. To achieve this, the *L1* inductor is removed. Also, as capacitors dampen the high frequency power consumption, some of the supply capacitors are removed (*C113*, *C114*, *C115*, and *C53*). Not all of the supply capacitors on the core voltage supply (VCC1V0) can be removed; otherwise, the Basys3 may operate unstable.

Instead of the removed capacitors C113 and C114 the external power supply is inserted with a shunt resistor in line (cf.

Figure 6-2). The supply voltage U_{supply} is set to have 1 V after the shunt resistor, so that $U_1 = 1$ V.The voltage U_1 is captured with a -1 V offset for power traces.

6.4 Software

As previously discussed, the trace capture and serial communication with the hardware is processed by the same software. This software, written in C++, is object based and event driven. It features two objects, one for oscilloscope control and trace capture (PicoScope), the other for serial communication with the hardware (SerialIO). As both of these tasks need to be synchronized, control events are introduced. The program flowchart detailing the thread synchronization is displayed in Figure 6-3.

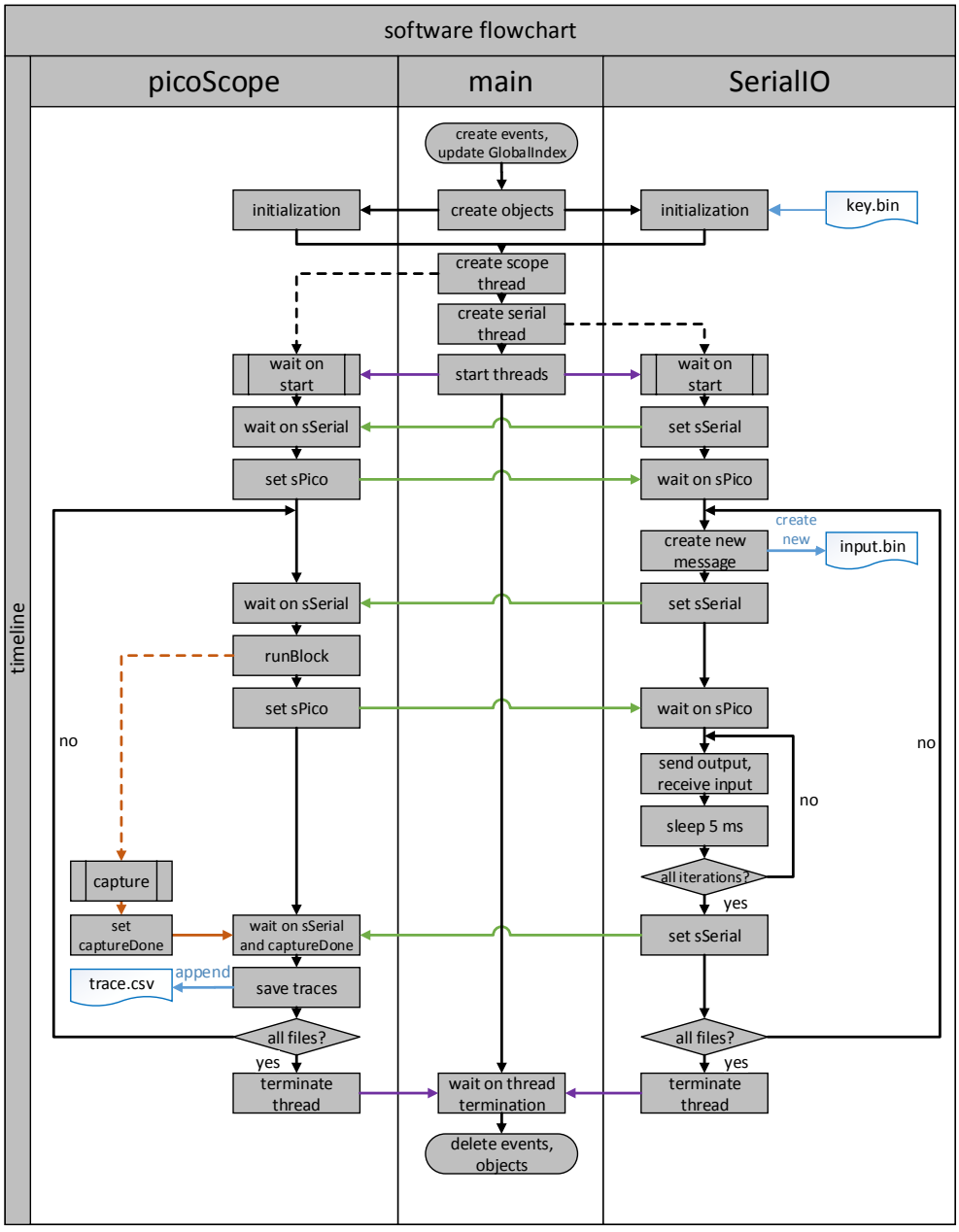

Figure 6-3. Software flowchart. The picoScope and SerialIO objects feature their own threads that are synchronized with events, displayed by the green arrows. Blue arrows indicate file access, lavender thread control related events, orange capture related events as well as program flow, and black program flow.

Upon program start all required objects and events are created. During the object creation, all required settings are adjusted, including the secret key that is loaded from the "key.bin" file. Afterwards, the oscilloscope (scope) thread is created, followed by the serial communication thread. The threads are then started by the main thread. The main thread will then wait upon worker thread termination.

After the threads have been started, they synchronize initially. After this initial synchronization, a new message is created in the serial thread. This message is prepended with the secret key forming the output buffer. This buffer is then saved into an indexed "input.bin" file for later processing. When the file has been saved, the serial thread signals its readiness to the oscilloscope thread to indicate readiness for message exchange.

The oscilloscope thread arms the PicoScope by calling *startBlock*, starting a capture block, after being signaled that the serial thread is ready for message exchange. Capture block completion is signaled by a callback function that sets the captureDone event (orange). This callback function is called by the PicoScope. The oscilloscope thread waits on both the captureDone event and the serial thread (sSerial) event.

Message exchange consists of sending the 128 byte long hash input and receiving the 32 byte hash output in blocking mode. After the output has been received, the thread sleeps for 5 ms to give the PicoScope time to rearm itself. This cycle is repeated `numIterationsPerFile` (settings.h, Appendix B) times to increase SNR. After all iterations the serial thread signals the oscilloscope thread.

After completion of the capture block and the signal from the serial thread, the oscilloscope thread loads the trace data from the PicoScope and appends it to the "traces.csv" file as a new line.

This repeats `numFiles` (settings.h, Appendix B) times as a cycle. A cycle will restart at new message generation for the serial thread and waiting on the serial signal for the oscilloscope thread. After all cycles have been computed, both worker threads shutdown and the main thread clears all resources.

Chapter 7 - Results and Analysis

Before correlating the power traces, a SPA may provide a good starting point for further analysis. Therefore, a single power trace will be analyzed before correlating the hypothetical power consumption using **Model I**.

As the internal clock frequency of the FPGA implementation is 18.432 MHz, and the sample period is 2 ns, a single clock cycle takes approximately 27 samples. The trigger is set just before starting to read in the input block which takes 16 clock cycles to be read. Therefore, the first round of KECCAK computation should approximately be located at sample 461, the second round at sample 488, counted from the trigger.

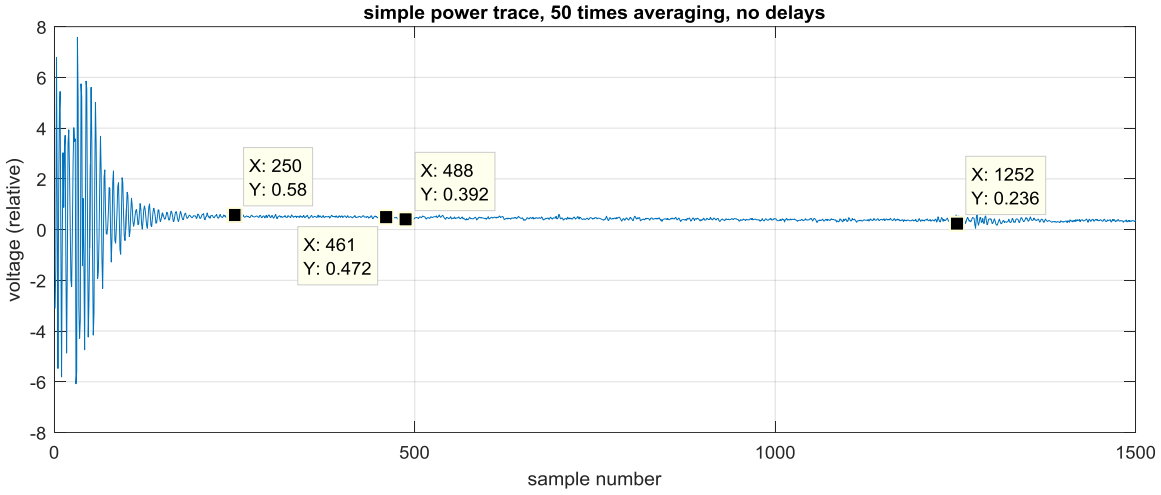

Figure 7-1. Single power trace without any trigger delays, using 50 times averaging. After 250 samples, the interference introduced by the trigger signal has worn off. The unconnected stop trigger signal can also be seen around sample 1250.

As seen in Figure 7-1, the trigger signals, located at samples 0-250 (trig_bit) and 1250-1500 (stop_bit) are influencing the measurement. This could be due to the shared ground potential of both the PicoScope and the Basys3. As stop_bit is unconnected in the test framework, as it has been used only for debugging purposes, it has far lower influence compared to trig_bit.

As there is enough time between the trigger interference and the first round of KECCAK (around sample 434), the actual capture will be delayed by 250 samples for following measurements. Sample numbers will therefore reduced by 250. Also, the last 250 samples are dropped as they do not contain any useful information.

The same power trace is displayed in Figure 7-2 with the above modifications. As the trigger is offset by 250 samples, the first round should now be located around sample 211, the second around sample 238.

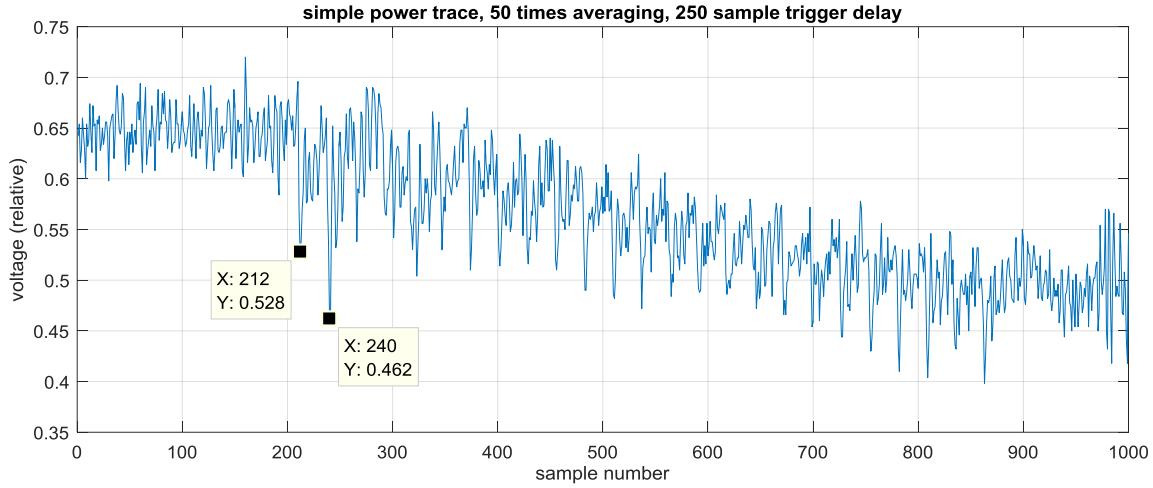

Figure 7-2. Zoomed single power trace with trigger delayed by 250 samples, using 50 times averaging. KECCAK rounds can be identified: the first and second round of KECCAK feature larger spikes, located close to the expected locations, followed by others.

The expected sample numbers for round one and two of KECCAK computation are correct. The actual peaks are very close with Sample 212 for round one, and Sample 240 for round two. For a more detailed analysis and to further reduce noise, 500 different traces are averaged in Figure 7-3.

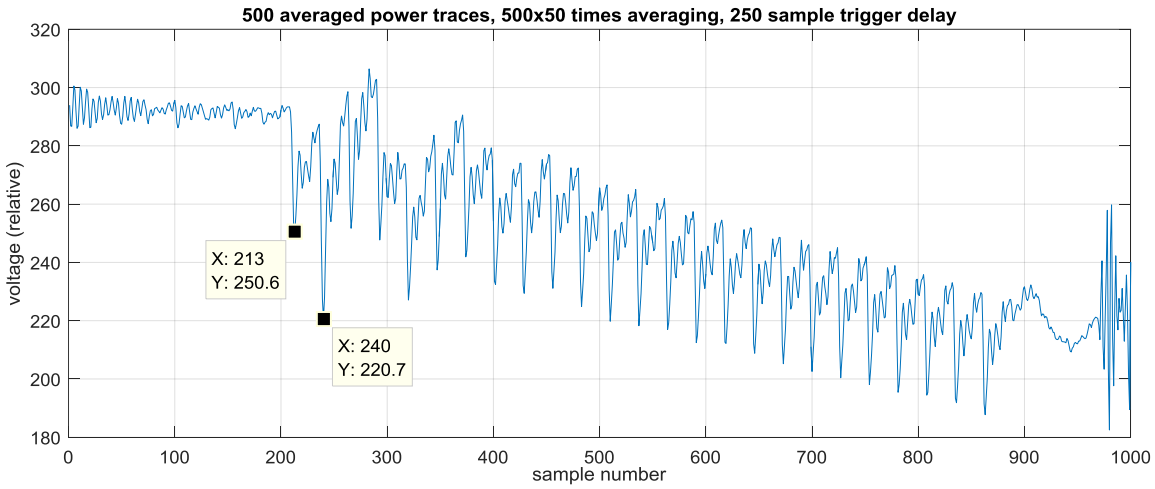

Figure 7-3. Zoomed, averaged power traces. Traces have been averaged by summing. KECCAK rounds can be identified comparatively better than in a single power trace.

As expected, the clock period is approximately 27 samples. The first round of KECCAK's round computation is located 213 samples, close to the expected 211 samples. The supply voltage drops off for later rounds.

As Lou et al. have shown in [17], using power **Model I**, there should already be enough correlation to drastically reduce the number of key byte candidates with approximately 200,000 traces on a SCA specialized FPGA board. Correlation peaks should show up at the first clock cycle of Keccak computation. To proof this on the developed test framework, 300,000 messages are hashed 50 times each, resulting in 300,000 traces with 50 times averaging.

As the used key starts with an $A0_{16}$ (160_{10}) byte, it is expected as the right key byte guess, and therefore to have the highest correlation. In Figure 7-4, the correlation traces for the guessed key bytes BE_{16}-$A3_{16}$ (157_{10}-162_{10}) are displayed. There are no spikes in the correlation traces. However, each key byte guess does have a different mean correlation. The five highest and the five lowest mean correlation values are displayed in Table 7-1. The detailed table specifying the mean correlation values for all guessed key bytes can be found in Appendix D.

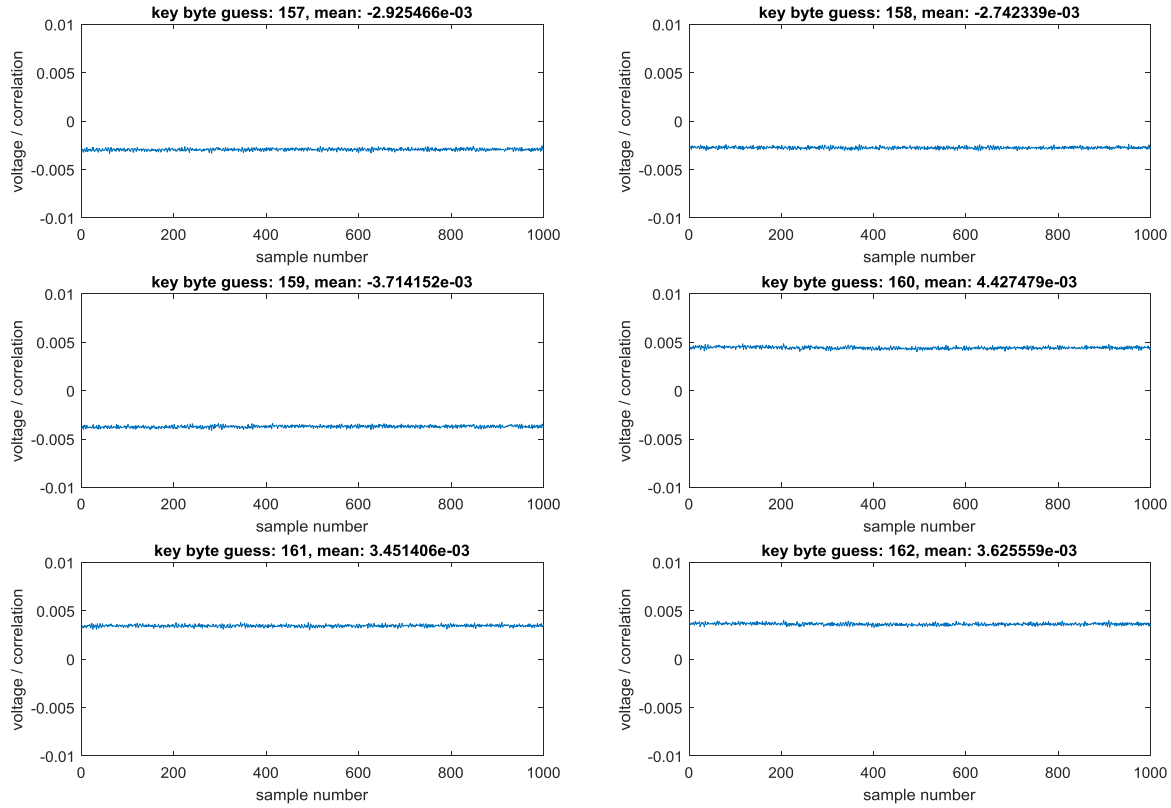

Figure 7-4. Correlation traces for key byte guesses 158-162. The correct value is 160.

Table 7-1. Top five mean correlation values, first key byte.

Rank	1st	2nd	3rd	4th	5th
Largest	**224**	**160**	**228**	**164**	**226**
	4,65E-03	4,43E-03	4,31E-03	4,09E-03	3,86E-03
Smallest	**31**	**95**	**27**	**91**	**29**
	-4,65E-03	-4,43E-03	-4,31E-03	-4,09E-03	-3,86E-03

The correct key byte (160) and its binary complement (95) feature the top two correlation values. It does not stand out but it is clearly amongst the best values. Other top values differ in only one or two bit.

To test whether other key bytes can also be identified using the mean correlation, the same technique is applied to the second key byte. The top five correlation results are displayed in Table 7-2. Just like the first byte, no correlation peaks can be identified in the correlation traces. These are omitted for this reason.

Table 7-2. Top five mean correlation values, second key byte.

Rank	1st	2nd	3rd	4th	5^{th}
Largest	**196**	**192**	**224**	**228**	**212**
	3,30E-03	3,30E-03	2,70E-03	2,69E-03	2,66E-03
Smallest	**59**	**63**	**31**	**27**	**43**
	-3,30E-03	-3,30E-03	-2,70E-03	-2,69E-03	-2,66E-03

The second byte in the key is 161. It does not show up in the top five of the mean correlation values. Its mean correlation is very low (-7,25E-05) compared to the top results.

Chapter 8 - Conclusions and Future Work

The task of this thesis was to develop the attack framework to test if the attacks, carried out by previous works, could also be applied on a general purpose FPGA board rather than SCA-designed boards. It is possible to capture power traces with clearly visible patterns. However, based on the results, it is not possible to retrieve a key byte of MAC-KECCAK using power **Model I** by capturing 300,000 traces with the developed attack framework. The first key byte featured the top two average correlation, however this does not apply to the second byte. Therefore, the average correlation does not indicate a correctly guessed key byte. This concludes with the fact that the key cannot fully be recovered with 300,000 traces if they have only 50 times averaging.

Compared to Lou et al.'s and Tran's work, the correlation traces do not feature peaks at the first or second round of KECCAK. This could be due to the lower SNR provided by the test framework compared to the dedicated power measurement connector of the SASEBO-GII or simulated power traces. Also, the supply voltage drops of over the course of KECCAK computation. This is different compared to the traces of Lou et al. ([17], Fig.4(a), p. 6).

With higher averaged traces however it might be possible to increase correlation by decreasing the noise. This may eventually lead to visible peaks in the correlation traces which would make distinct key guesses possible. The effect of averaging can be seen by comparing Figure 7-2 with Figure 7-3. With higher averaging, the environment noise could drastically be reduced. It made the averaged power trace's (Figure 7-3) patterns clearly stand out compared to those of the noise obscured single power trace (Figure 7-2).

Although the developed attack framework is more realistic than those of previous attacks, it is still theoretical. The adversary has to physically change the platform by removing capacitors and introducing a custom power supply. He also has to capture the power traces with the correct time alignment which is difficult without any trigger signals or the reference clock. If the adversary does not have access to the implementation itself, it unlikely that he will have that information. However, it might still be possible for an adversary to overcome these limitations.

For future work, more traces would have to be captured with higher averaging in order to successfully attack MAC-KECCAK on the developed test framework. The higher averaging is the key task, as this will be increasing the SNR eventually leading to higher correlation. Other modifications to the test framework are not required. It is operating stable and trace capture does not require many system resources.

I. List of Abbreviations

API	application programming interface
ASCII	American Standard Code for Information Interchange
DPA	differential power analysis
FPGA	field programmable gate array
HD	Hamming distance
HW	Hamming weight
MAC	message authentication code
NIST	US National Institute of Standards and Technology
PC	personal computer
PicoSDK	software development kit for PicoScopes by Pico Technology
SCA	side-channel analysis
SDK	software development kit
SHA-3	Secure Hash Algorithm 3
SNR	signal to noise ratio
SPA	simple power analysis
UART	universal asynchronous receiver transmitter
VHDL	Very High Speed Integrated Circuit Hardware Description Language
XOR	$F(a,b) = (a \wedge \neg b) \vee (\neg a \wedge b) = a \,\mathrm{XOR}\, b = a \oplus b$

II. List of Figures and Tables

III. Bibliography

[1] M. Bellare, R. Canetti and H. Krawczyk, ""Keying Hash Functions for Message Authentification", CRYPTO, ser. Lecture Notes in Computer Science," in *Advances in Cryptology, vol. 1109*, Berlin / Heidelberg, Springer, 1996, pp. 1-15.

[2] P. Kocher, J. Jaffe and B. Jun, "Differential Power Analysis," in *Advances in Cryptology — CRYPTO' 99*, Santa Barbara, California, USA, Springer, 1999, pp. 388-397.

[3] Morita Tech / AIST, "Side-channel Attack Standard Evaluation Board (SASEBO)," Morita Tech / AIST, 01 04 2012. [Online]. Available: http://satoh.cs.uec.ac.jp/SASEBO/en/index.html. [Accessed 01 02 2016].

[4] Satoh Lab./UEC, "SAKURA Project," Satoh Lab./UEC, 01 01 2016. [Online]. Available: http://satoh.cs.uec.ac.jp/SAKURA/index.html. [Accessed 01 02 2016].

[5] NIST, "NIST Selects Winner of Secure Hash Algorithm (SHA-3) Competition," 02 10 2012. [Online]. Available: http://csrc.nist.gov/groups/ST/hash/sha-3/sha-3_selection_announcement.pdf. [Accessed 15 01 2015].

[6] G. Bertoni, J. Deamen, M. Peeters and G. Van Assche, "The KECCAK reference," 14 01 2011. [Online]. Available: http://keccak.noekeon.org/Keccak-reference-3.0.pdf. [Accessed 15 01 2016].

[7] G. Bertoni, J. Deamen, M. Peeters and G. Van Assche, "Keccak - Pieces of State," [Online]. Available: http://keccak.noekeon.org/Keccak-f-PiecesOfState.pdf. [Accessed 25 01 2016].

[8] G. Bertoni, J. Deamen, M. Peeters and G. Van Assche, "Keccak - Theta," [Online]. Available: http://keccak.noekeon.org/Keccak-f-Theta.pdf. [Accessed 25 01 2016].

[9] G. Bertoni, J. Deamen, M. Peeters and G. Van Assche, "The sponge construction," 02 05 2012. [Online]. Available: http://sponge.noekeon.org/. [Accessed 25 01 2016].

[10] NIST, "Announcing Draft Federal Information Processing Standard (FIPS) 202, SHA-3 Standard," 28 05 2014. [Online]. Available: https://www.federalregister.gov/articles/2014/05/28/2014-12336/announcing-draft-federal-information-processing-standard-fips-202-sha-3-standard-permutation-based. [Accessed 2016 01 2015].

[11] M. Taha and P. Schaumont, "Side-Channel Analysis of MAC-Keccak," in *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, Austin, TX, USA, 2013.

[12] R. L. Rivest, A. Shamir and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM,* no. 21, Issue 2, Feb. 1978, pp. 120-126, 02 02 1978.

[13] F. Amiel, B. Feix, M. Tunstall, C. Whelan and W. P. Marnane, "Distinguishing Multiplications from Squaring Operations," in *Selected Areas in Cryptography*, Berlin Heidelberg, Springer, 2009, pp. 346-360.

[14] L. Chen and G. Gong, Communication System Security, Boca Raton, FL 33487-2742, USA: CRC Press - Taylor & Francis Group, 2012.

[15] "Univariate Side Channel Attacks and Leakage Modeling," *Journal of Cryptographic Engineering,* no. 1, Issue 2, Aug. 2011, pp. 123-144, 03 11 2011.

[16] Xilinx, "MicroBlaze," Xilinx, [Online]. Available: http://www.wiki.xilinx.com/MicroBlaze. [Accessed 13 02 2016].

[17] P. Lou, Y. Fei, X. Fang, A. A. Ding, M. Leeser and D. R. Kaeli, "Power Analysis Attack on Hardware Implementation of MAC-Keccak on FPGAs," IEEE, Cancun, 2014.

[18] G. Bertoni, J. Deamen, M. Peeters und G. Van Assche, „Keccak - Hardware implementation in VHDL - High Speed Core,“ [Online]. Available: http://keccak.noekeon.org/KeccakVHDL-3.1.zip. [Zugriff am 13 02 2016].

[19] X. D. Tran, *Power Analysis Attacks on Keccak,* Rochester, NY: Rochester Institute of Technology, 2015.

[20] Xlinix, "Clocking Wizard," Xlinix, 18 11 2015. [Online]. Available: http://www.xilinx.com/products/intellectual-property/clocking_wizard.html#overview. [Accessed 20 02 2016].

[21] P. Bennett, "A VHDL UART for communicating over a serial link with an FPGA," 18 10 2011. [Online]. Available: https://github.com/pabennett/uart. [Accessed 12 01 2016].

[22] E. Oswald, "DPA book," University of Bristol, 2006. [Online]. Available: www.dpabook.org. [Accessed 15 01 2016].

[23] Pico Technology, "PicoSDK 10.6.11," 03 09 2015. [Online]. Available: https://www.picotech.com/downloads/_lightbox/pico-software-development-kit-64-bit-beta. [Accessed 28 01 2016].

[24] Microsoft, "Serial Communications MSDN," 11 12 1995. [Online]. Available: https://msdn.microsoft.com/en-us/library/ff802693.aspx. [Accessed 29 01 2016].

[25] "KeccakCodePackage - Main code repository / GitHub," 06 01 2016. [Online]. Available: https://github.com/gvanas/KeccakCodePackage. [Accessed 02 02 2016].

[26] Digilent, "Basys3 FPGA Board Schematic," 2014. [Online]. Available: https://reference.digilentinc.com/_media/basys3:basys3_sch.pdf. [Accessed 06 02 2016].

Statutory Declaration

I declare on oath that I completed this work on my own and that information which has been directly or indirectly taken from other sources or the internet has been noted as such. Neither this, nor a similar work, has been published or presented to an examination committee.

Eigenständigkeitserklärung

Ich versichere, die vorliegende Arbeit selbständig ohne fremde Hilfe verfasst und keine anderen Quellen und Hilfsmittel als die angegebenen benutzt zu haben. Die aus anderen Werken oder dem Internet wörtlich entnommenen Stellen oder dem Sinn nach entlehnten Passagen sind durch Quellenangaben eindeutig kenntlich gemacht. Die vorliegende Arbeit wurde keinem anderen Prüfungsgremium vorgelegt.

Hamburg, 2016-02-29,

_____