CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY

ASSIGNMENT OF BACHELOR’S THESIS

Title:	Side channel analysis of cryptographic algorithms implementations
Student:	Bc. Lukáš Mazur
Supervisor:	Dr.-Ing. Martin Novotný
Study Programme:	Informatics
Study Branch:	Computer engineering
Department:	Department of Digital Design
Validity:	Until the end of summer semester 2016/17

Instructions

1. Get familiar with methods of side-channel analysis of cryptographic algorithms implementations. Focus mainly on Differential Power Analysis (DPA).
2. Learn how to use DPA for implementations in SmartCards.
3. Consequently, explore viability of DPA to FPGA implementations of cryptographic algorithms.
4. Based on the consultation with a supervisor, select suitable variants of implementations.
5. Explore resistance of these variants against DPA.

References

Will be provided by the supervisor.

L.S.

doc. Ing. Hana Kubátová, CSc.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague December 6, 2015

Czech Technical University in Prague

Faculty of Information Technology

Department of Digital Design

Bachelor's thesis

Side channel analysis of cryptographic algorithms implementations

Bc. Lukáš Mazur

Supervisor: Dr.-Ing. Martin Novotný

8th January 2017

Acknowledgements

At this point I want to thank to my supervisor Dr.-Ing. Martin Novotný for all his comments and remarks during writing the thesis, and for the help with measurement. I want to thank to Ing. Jiří Buček for the help with measurement. I also want to thank to Priv.-Doz. Amir Moradi for consultations on proper oscilloscope setup. I want to thank to Jan Severyn for valuable collaboration while working on this thesis.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 8th January 2017 . . . .

Citation of this thesis

Mazur, Lukáš. *Side channel analysis of cryptographic algorithms implementations.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2017.

Abstrakt

V bakalářské práci byly prozkoumány možnosti rozdílové odběrové analýzy (DPA) na programovatelná hradlová pole (FPGA). V rámci práce byl modifikován program pro měření proudové spotřeby, vytvořeny skripty pro provádění DPA a vytvořeny různé implementace algoritmu AES na FPGA. Vyvinuté skripty a programy pro DPA byly ověřeny proti implementaci AES na čipové kartě. Poté, co tyto programy úspěšně prolomily implementaci na čipové kartě, přistoupili jsme k aplikaci DPA proti implementaci AES na desce s FPGA. DPA proti FPGA byla provedena v šesti různých konfiguracích. Tyto konfigurace se lišily v implementaci AES na FPGA, v konfiguraci desky, v nastavení osciloskopu a v metodě útoku. Byly nalezeny konfigurace, které byly úspěšně prolomeny. Bylo zjištěno, že nastavení osciloskopu a měřícího prostředí má významný dopad na proveditelnost DPA na FPGA. Implementace je méně důležitá pro úspěch útoku. Nejdůležitejší aspekt implementace je hodinová frekvence. Také bylo zjištěno, že použití různých zdrojů proudu a odebrání kondenzátorů na FPGA desce má významný dopad na proveditelnost DPA.

Klíčová slova Rozdílová odběrová analýza, útoky postranními kanály, Advanced Encryption Standard, FPGA, čipová karta, korelační analýza

Abstract

We explored the possibilities of the Differential Power Analysis (DPA) on the Field Programmable Gate Array (FPGA). We have modified the application for measuring a power consumption, created scripts for performing DPA, and created different implementations of AES algorithm for FPGA. Developed scripts and applications for DPA were verified against AES implementation for smart cards. Once those applications successfully broke the implementation for smart cards, we continued with the application of DPA against AES implementation for an FPGA board. DPA against FPGA was performed in six different configurations. Those configurations differed in AES implementation for FPGA, in board configuration, in oscilloscope setup, and in method of the attack. We found variants that could be successfully broken. We found out that an oscilloscope and measuring environment setups has major impact on the feasibility of the DPA on FPGA. The implementation is less important for the success of the attack. The most important aspect of the implementation was the clock frequency. We have also found out that using different power sources and removing capacitors on the FPGA board have significant impact on the feasibility of the DPA.

Keywords Differential Power Analysis, Side Channel Attacks, Advanced Encryption Standard, FPGA, Smart Card, Correlation Analysis

Contents

List of Figures

List of Tables

Introduction

Cryptography is a major part of computer security and data security in general. Substantial effort is being put into developing more secure and faster cipher algorithms, and into breaking them. Many governments have their own standards and requirements for cryptographic algorithms usually based on mathematical findings and theories that make them secure. But even the most mathematically secure algorithms can be broken in a matter of hours or minutes by side channel attacks. Side channel attacks do not exploit weaknesses in the mathematical model behind the cryptographic algorithm, but the implementation of the algorithm is targeted instead. Even a very secure cipher algorithm implementation can be broken if implemented incorrectly.

The volume of digital data in the world is growing rapidly and the demand for its encryption is growing as well. Cryptography has its use not only in protecting government secrets but also in protecting people's lives. Cryptography makes it impossible[1] for an intruder to intercept a remote communication in mission-critical systems such as planes or automated urban metro subway systems. By researching possible attacks, the researches can find weak spots in encryption systems and find effective ways of securing them.

Traditionally, research was mostly done in the ciphers themselves, and not theirs implementations. This has changed since Paul Kocher et al. presented their article in 1999 [18]. They showed that it is possible to get the secret key from cryptographic device by looking at its power consumption. An intensive research in the field of Differential Power Analysis (DPA) has begun and it is sill ongoing. Different attacks and countermeasures against those attacks have been introduced.

The aim of this thesis is to get familiar with side channel attacks and especially with the DPA. The DPA will be applied to the smart card to properly learn the method of performing it. After that the viability and resistance of different implementations of cryptographic system in Field Programmable

[1] At least we hope it is impossible

Gate Array (FPGA) will be explored.

The structure of this thesis is as follows: In the first chapter we offer a brief introduction into the topic. In the second chapter we analyse several variants of the implementation of the cryptographic system, we discuss the ways of implementation of the side channel attack, and we make a selection of suitable FPGA board with respect to the feasibility of the attack. In the third chapter we describe the design of different implementations of cryptographic systems for smart card and FPGA, the implementation of scripts used for the attack, and modification of the application for measurement. We performed both the verification and validation testing. The tests and their results are described in the fourth chapter. We present results of different attacks on different configurations of the cryptographic system in the fifth chapter. In the sixth chapter we propose a few topics that could be researched in the future. In last chapter we make conclusions from our findings.

Chapter 1

Background

This chapter briefly introduces the reader into the topic. After the consultation with thesis supervisor we have selected Advanced Encryption Standard (AES) as a cryptographic system that will be used in this thesis. The chapter begins by a discussion of the AES selection process and short description of the AES algorithm in section 1.1. The side channel attacks are explained in section 1.2.

1.1 Advanced Encryption Standard

The AES predecessor, Data Encryption Standard (DES), was often criticized because of its short key being only 56 bits long. DES was broken by brute-force attack for the first time in 1997 by a cluster of thousands personal computers [16]. It became apparent that a new encryption standard was needed. In the same year, National Institute of Standards and Technology (NIST) announced that DES successor would be called AES [24]. There was a call for submissions of new algorithms a few months later [25]. The algorithm was required to have a block size of 128 bits, and to support 128, 192 and 256 bit keys. The cipher Rijndael won the competition and has been standardized as Federal Information Processing Standards (FIPS) 197 [26] in 2001.

The AES processes the block in rounds. The number of rounds is 10, 12, or 14 for 128, 192, or 256 bit key respectively. The algorithm performs 4 transformations in each round:

- SubBytes—non-linear byte substitution according to a predefined table
- ShiftRows—exchange of positions of some bytes
- MixColumns—combination of multiple bytes, this transformation is skipped in the last round
- AddRoundKey—xoring of round key and cipher state

A different round key is used for each round. Round keys are derived from the original key by the mechanism called Key Schedule or Key Expansion. A detailed explanation of the cipher is provided in [28] or [17].

A dedicated hardware for breaking DES by a brute force attack, COPACOBANA, can break the cipher in 9 days on average [20]. Later, a new device, RIVYERA, was created. It can break DES in 1 day on average [31]. Breaking AES with 128 bit key by a brute-force attack on a device with the same computational power as computational power of RIVYERA would take approximately $13 * 10^{18}$ years. The brute force attack against AES is not feasible with state-of-the-art hardware.

Related key attacks against AES algorithm are known. The complexity of breaking AES192 and AES256 can be reduced to 2^{176} and $2^{99.5}$ respectively [12]. The 9 round version of AES256 can be broken in time 2^{39}, and 10 round version in time 2^{45} [11].

1.2 Side Channel Attacks

According to [28], the cryptanalysis can be divided into 3 types:

- Classical cryptanalysis—it involves attacks that exploit the internal structure of the cipher or brute force attacks. It tries to find some weak spot in mathematical model behind the cipher, in its properties, or in its design.
- Social engineering attacks—they usually involve humans, and the aim of those techniques is to get the secret key directly from humans. Members of this group are techniques such as bribing, blackmailing, phishing, or tricking.
- Implementation attacks—side channel attacks can be used to reveal the key from information leaks caused by the implementation. In this thesis we focus on one type of such attack.

Attacks which belong to implementation attacks include power analysis, magnetic field analysis, timing analysis, or temperature analysis.

In this thesis we focus on side channel attacks, namely the method called Differential Power Analysis (DPA). As has been said in [30] *"The efficiency of DPA attacks is much greater than the efficiency of differential or linear cryptanalysis"*. Below we bring brief introduction into this method.

1.2.1 Differential Power Analysis

The detailed explanation of DPA can be seen in [21] or [19]. We present a brief description here. The power consumption depends on the data being processed

[15]. The key can be obtained by analysis of the power traces captured during the encryption.

During the attack we are using the device and capturing the power traces. We are sending different plaintexts and storing the traces of the power consumption.

The key recovery process starts by creating a key hypothesis. We estimate the consumption for different values of each byte of the key (0–255). Those estimates are then correlated with obtained traces and the key hypothesis with highest correlation coefficient is considered to be a correct key.

Different models can be used during the creation of the key hypothesis. There are two orthogonal parts of the model which can be combined, the consumption model, and the round (just in case of block ciphers). The combination produces four different possible attack variants.

Consumption model

During the creation of a key hypothesis we try to estimate changes in the power consumption. Two models are commonly used: Hamming weight and Hamming distance.

Hamming weight By Hamming weight we can measure a current state of the system (register). It gives us the number of ones in a register. Hamming weight is a static model in its nature. It measures only the current state and not the changes that preceded the current state.

Hamming distance Hamming distance gives the difference between two values. If those two values represent different time points, the distance can actually measure the change in time of one value. Hamming distance is useful in cases when a small change in register (for instance the change of just one bit) can cause a chain of changes in the following combinational logic. Each of these changes has an impact on the overall consumption. By using distance we can more closely approximate the changes and their power consumption in the model.

Round (in a block cipher)

Block ciphers are typically composed of multiple rounds. There is usually a sequence of different types of operations in each round. Typically those types of operations include permutation, substitution, diffusion, and key addition. When applying DPA on a block cipher, we focus on the inner state of cipher in suitable moment either in first round or last round. If we are attacking the first round, we estimate the cipher state according to the plaintext and we guess first round key. If we are attacking the last round we estimate the cipher state according to the ciphertext and we guess last round key.

Substitution, which is a non-linear operation, is required to be present between the inner state and plaintext/ciphertext. The addition of round key needs to be performed before the non-linear operation, and between the inner state and the plaintext/the ciphertext. Permutation and other linear operations can be present between the key addition and substitution. The diagram of both methods (for AES) is shown in Figure 1.1.

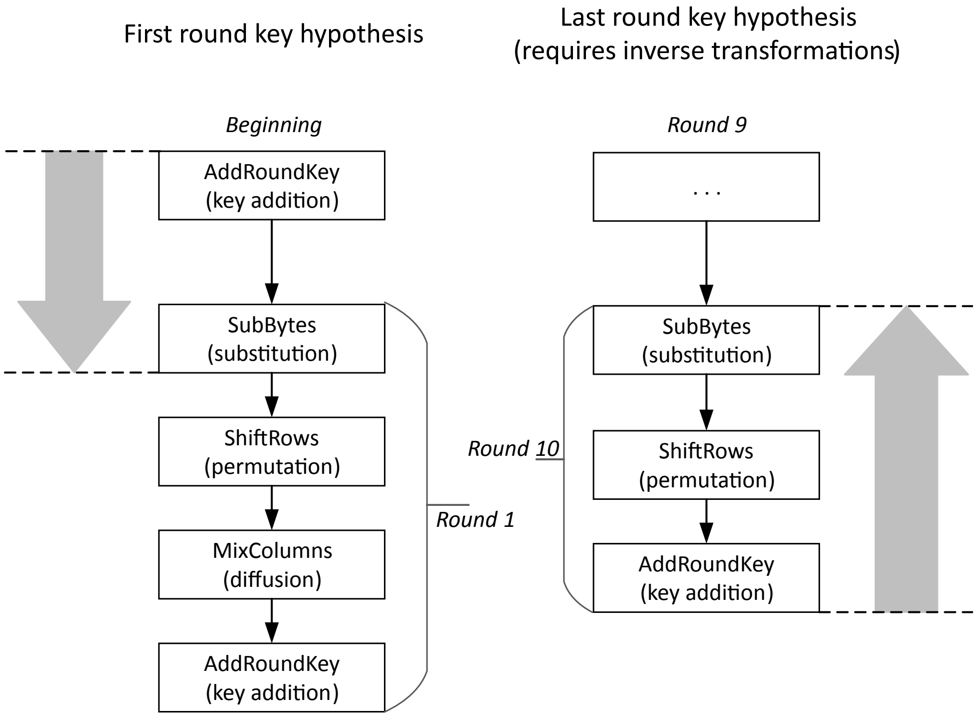

Figure 1.1: Possible methods of DPA against AES128

Combination of different consumption models and rounds leads to 4 possible attacks:

- Hamming weight and first round
- Hamming weight and last round
- Hamming distance and first round
- Hamming distance and last round

Chapter 2

Analysis

This chapter contains the analysis of the structure of the implementation in section 2.1. Then it discusses different methods of the attack and summarizes the methods which will be used (see section 2.2). The chapter ends by section 2.3 which contains the analysis and selection of the FPGA board used for the implementation.

2.1 Encryption Algorithm

The DPA can be used against various encryption algorithms because it does not target the underlying mathematical model of the cipher but it targets its implementation instead. With regard to the previous statement we were free to choose whatever cipher we wanted. As has been state above, after consultation with thesis supervisor, we decided to use the AES encryption algorithm with 128 bit key, as it is widely used and secure algorithm. Many resources on AES are available. Because of that it was fairly easy to get familiar with inner function of AES.

Different approaches to the structure of the implementation exist. We were considering three options. Expected impact of these options on the feasibility of DPA is described below. The scheme of all three variants can be seen in Figure 2.1. Detailed description of different designs is in [29]. Below we introduce three design approaches we considered for designing AES. Possible options are summarized in Table 2.1.

Combinational logic All the rounds would be performed in one clock cycle in this design. DPA usually targets just one round, but this design would perform all the rounds and transformations in one clock cycle. We suppose that it could introduce a lot of noise to the captured traces.

Iterative design Just one round of one block would be encrypted per clock cycle. Encryption of one block of plaintext takes 10 clock cycles plus

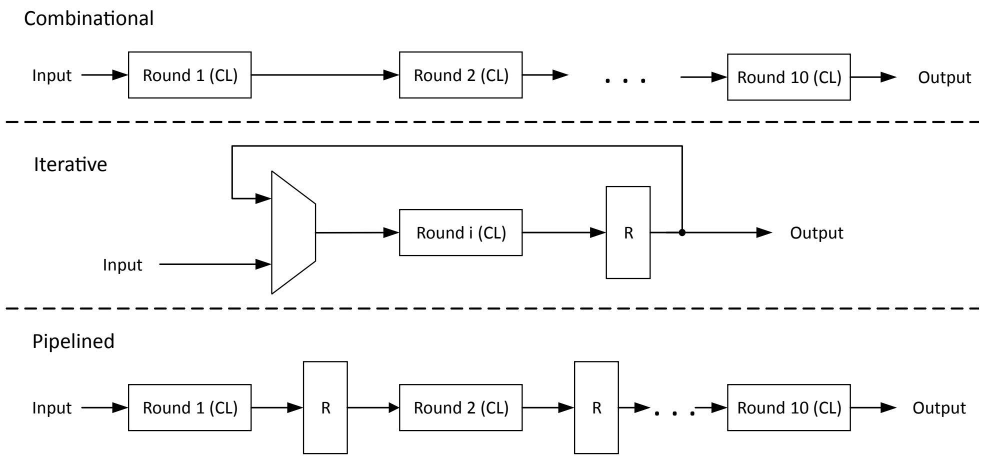

Figure 2.1: Different implementation variants of AES128 (CL—combinatorial logic, R—register)

overhead. It gives us the opportunity to measure the consumption more precisely without the noise caused by other rounds.

Pipelined architecture The encryption is separated into multi-stage pipeline. One round of multiple blocks[2] is being processed in one clock cycle. We expect this option to be more noisy than previous one because data from different blocks are influencing the consumption.

	Per clock cycle	
Structure	Rounds of each block	Blocks
Combinational	10	1
Iterative	1	1
Pipelined	1	10

Table 2.1: Comparison of different implementation variants

We have decided to use the iterative implementation because we expect it to be easier to break than the others. Only one round and one encryption are performed at a time. Other versions process more data at one time and thus we expect them to be harder to break because there is more information involved in the encryption in one clock cycle. Moreover we have decided to use iterative variant because it consumes less chip area than other variants.[3]

[2] Number of processed blocks depends on number of stages of pipeline

[3] Lately, we found out that using iterative variant was clever decision because iterative variant uses around 30% of chip area. Other variants would be too large to fit in the selected chip.

2.2 Methods of the Attack

We discuss the suitability of different methods of the attacks in this section. A summary of attacks which will be performed is placed at the end of the section.

2.2.1 Consumption Model

Two consumption models are commonly used (see section 1.2.1). We expect each model to be effective against different kind of cryptographic devices.

We assume the Hamming weight to be more effective for smart cards (which is actually a single chip computer). Our assumption is based on following hypothesis. Changes in power consumption occurs when the data on the internal bus inside the processor changes. At this time the bus needs to charge or discharge depending on the data. Thus the consumption is linked to the current state of the bus. Hamming weight should be relevant model in this case because we are guessing the content of data on the internal bus. The fact that Hamming weight gives better results than Hamming distance on smart cards was confirmed by Tillich and Herbst [33].

We will use both Hamming weight and Hamming distance for FPGA, but we expect that Hamming distance will be more efficient because the number of zeroes and ones in register is not relevant to the overall consumption. Changes of values are more relevant in the case of FPGA. By Hamming distance we can estimate the number of changes that happened in one clock cycle.

2.2.2 Round

Possible rounds that can be a subject of the attack are described in section 1.2.1. We decided to mount the attack both against the first round, and against the last round. The description of the creation of the key hypothesis is below.

First round Creation of key hypothesis is simpler for the first round. It requires just 2 steps.

1. AddRoundKey
2. SubBytes

The key hypothesis is finished at this point.

Last round In the case of the last round the transformations are applied in the reverse order. The process of creating key hypothesis begins with the ciphertext (the final value of the state register) and goes backwards through the last round. This attack requires using the inverses of all the transformations (except the AddRoundKey).

1. AddRoundKey[4]
2. Inverse ShiftRows
3. Inverse SubBytes

The order of SubBytes and ShiftRows is not important. ShiftRows just changes position of a byte. I could perform the SubBytes before ShiftRows in a DPA if it would lead to easier implementation.

2.2.3 Intended Attacks

We have decided to use Hamming weight against the first round in case of smart card. This is a proven method and our primary aim was to use smart cards to get familiar with the DPA. We have made a decision to test both Hamming weight against the first round, and Hamming distance against the last round on FPGA. We want to start with Hamming weight and first round because this method will be already developed for the smart card. If unsuccessful, we will switch to the Hamming distance and last round because we expect better result for this DPA parameters. The summary of intended attacks is provided in Table 2.2.

	First round	Last round
Hamming weight	Smart card, FPGA	
Hamming distance		FPGA

Table 2.2: Overview of intended attacks

2.3 FPGA Board

This section describes the analysis of possible FPGA boards. At first we identified a set of requirements that the board has to satisfy:

- Availability of some communication channel (we prefer the serial line as a communication mean)
- Ability to measure the change of the current
- Availability in the university hardware lab
- Possibility to connect to our own power supply

[4] The inverse transformation for AddRoundKey is not needed because it is same as non-inversed one

2.3.1 Suitable boards

We were considering two boards, Digilent Basys 2 and Digilent Spartan 3E Starter Board. We found out that the Spartan 3E Starter Board satisfies all the feature requirements. There was no need to evaluate other boards because it was a very good match.

Digilent Basys 2

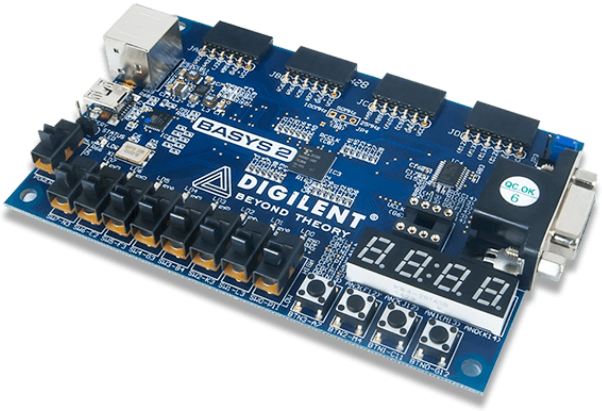

Figure 2.2: Digilent Basys 2, source: Digilent Inc.[5]

This is an entry level board by Digilent. It has a Spartan 3E-100 core and a wide range of peripherals such as VGA port or PS/2 connector. Unfortunately it is missing the serial port. Moreover the FPGA core is not large enough for the design. We verified this fact later during the development. If we targeted the core in Basys 2 in the development environment, the synthesis failed because the core was not big enough. But, we didn't know this fact at the time when we were selecting the board. The technical specification can be found in the reference manual [8].

Spartan 3E Starter Board

This is a Spartan 3E development board by Digilent. It features the Spartan 3E-500 core. It has 5 times more logic gates in comparison to the core placed on Basys 2. According to the data sheet [6] it has all the required features.

[5] Available online at https://reference.digilentinc.com/_media/reference/programmable-logic/basys-2/basys2-0.png [cit. 2016-11-21]

[6] Available online at http://cdn6.bigcommerce.com/s-7gavg/products/110/images/3604/S3E_top_600__14188.1449786819.1280.1280.png [cit. 2016-11-21]

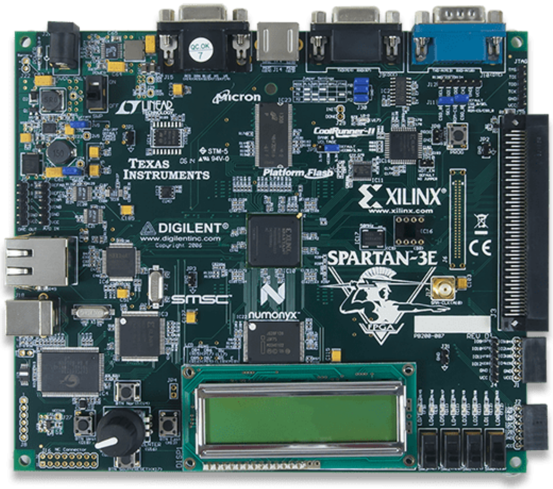

Figure 2.3: Spartan 3E Starter Board, source: Digilent Inc.[6]

We identified a spot on the board suitable for placing the oscilloscope probe. It is the jumper JP7 located on the 1.2V power line between voltage regulator and the core. We have also identified spots for connecting our own power supply bypassing the original one. We could use the JP7 jumper (the same as for measurement) for connecting 1.2V source and JP6 jumper for connecting the 2.5V supply (see Schematic Sheet 5 and Schematic Sheet 9 in User Guide [6]). The 3.3V source could be connected to any peripheral VCC pin and GND pin.

Chapter 3

Implementation

This chapter describes all the designs and other applications that we made or modified as a part of this thesis.

We have modified the existing application SC Power Measurement which is the application used for communication with the oscilloscope and automatic measurement. It was originally created by Ing. Jiri Bucek and Ing. Petr Vyleta for teaching DPA in the course MI-BHW [4] that is taught at Faculty of Information Technology (FIT) of Czech Technical University (CTU) (section 3.1).

We have created following applications and designs:

- The application used for the key recovery. It is used for computing the correct key and obtaining correlation matrices (section 3.2).

- The AES algorithm. We have developed 6 different AES128 implementations. These implementations differ in programming language they are written in and in their inner structure. The resistance against DPA of these implementations was tested. Following implementations were made:

  – AES for smart card (section 3.3)

  – AES for FPGA (section 3.4), with following modifications:

    ∗ Lower clock rate (section 3.5)
    ∗ Register placed after the subbytes operation (section 3.6)
    ∗ Each byte substitution is written into register in different clock cycle (section 3.7)
    ∗ The key is not harcoded but there is a register for key and it is transmitted over serial line (section 3.8)

3.1 SC Power Measurement

We used the SC Power Measurement from the course MI-BHW [4] for performing the measurements. It is written in C++ programming language. It communicates with Agilent oscilloscope via VISA drivers. It generates random plaintexts, sends them to the device under attack, and waits for the response. It downloads power traces that were captured during the encryption by the oscilloscope. It is also capable of sending the same data repeatedly which is useful for initial setting of measurement setup and environment.

It outputs 4 files containing data used for encryption and power traces.

traces.bin Binary file containing power traces. Each sample has size 1 byte.

plaintext.txt Text file that contains the plaintexts. Each line has one block of plaintext. The bytes of plaintext are written hexadecimally and separated by space.

ciphertext.txt File containing the ciphertexts. It has the same format as *plaintext.txt*.

traceLength.txt Text file containing the number of samples per trace.

There is no information about the number of traces but it can be computed from the size of the file *traces.bin* and the number of samples per trace, or by counting the number of plaintexts/ciphertexts.

The application was originally developed to be used with smart cards. Together with Jan Severyn who is also performing a DPA in his bachelor thesis we added a support for FPGAs. Moreover there is a bug in VISA drivers causing the SC Power Measurement to crash randomly. We performed a few changes in the application to make it more resistant against the failure. This is a list of all changes that we made:

- We added support for serial communication. The parameters of the communication such as baud rate, number of parity bits or port are hard-coded in file *sources/main.cpp*. It uses 115,200 bauds, 1 start bit, 1 stop bit and no parity bits. It uses following Windows application programming interface (API) functions:
  - *CreateFile(...)*
  - *WriteFile(...)*
  - *ReadFile(...)*
- In the original application, all the power traces were saved at the end of measurement. If the application crashed during measurement, no traces were kept at all. We changed the way of storing traces. Each trace is saved after each single measurement and not at the end. This preserves the obtained traces even in case of crash.

- The random generator that generates the plaintexts was originally seeded by zero. We changed the seed to the current timestamp. This change allowed us to see different plaintexts in each run of the application.

- We added various logs to the standard output. This is useful for tracking the progress during measurement.

- We added the possibility to prepend each plaintext by value 0xFF. This value at the beginning is required by the implementation with the register for key which can be found in section 3.8. The SC Power Measurement exists in two versions. One version prepends the byte at the beginning and the other does not.

3.2 Key Recovery

We have created a script for obtaining the correct key in Wolfram Mathematica. It uses files output from SC Power Measurement. Loading traces was optimized to use as low memory as possible. The loading procedure was taken from the webpage of the course MI-BHW [4]. But even with this optimization it run out of memory on machine with 8GB of RAM if I attempted to perform the analysis with 100,000 traces and thousands of samples per trace. As it can handle 1,000–2,000 samples per trace for 100,000 traces at most, the visual inspection of captured data is necessary. Then we can use a subset of samples that is located around the clock cycle which the operation was performed in. There are ways of optimizing the computation (see section 6).

We have created two versions of the key recovery script. They differ in the round that is being attacked and in the consumption model that is used. They use methods proposed in subsection 2.2.3. It works on a byte basis and the computation needs to be repeated for each byte of the key.

The script at first creates a key hypothesis. Key hypothesis is a matrix with dimensions (number of traces, 256). Number 256 represents all possible values that can be held by one byte of key[7]. First version attacks the first round and uses Hamming weight. Pseudocode of first version can be seen in Listing 3.1. Second version attacks the last round and uses Hamming distance. Pseudocode of second version can be seen in Listing 3.2.

The script then loads traces into the matrix with dimensions (number of traces, number of samples). Those two matrices are correlated producing a resulting matrix with dimensions (number of samples, 256). The plot of correct key hypothesis correlation coefficient is different from others. Thus, visual inspection is necessary to find the key. The number of column subtracted by one (indexes in Mathematica start from 1) is the value of correct key. The number of row with highest value corresponds to the sample which the

[7] Provided one byte has 8 bits

Listing 3.1: First round Hamming weight key hypothesis pseducode

```
for (i = 0; i < number_of_traces; i++)
    for (j = 0; j < 256; j++)
        tmp = bit_xor
            (plaintexts[number_of_byte, i], j)
        tmp = sbox(tmp)
        tmp = to_binary(tmp)
        key_hypothesis[i,j] = hamming_weight(tmp)
```

Listing 3.2: Last round Hamming distance key hypothesis pseducode

```
for (i = 0; i < number_of_traces; i++)
    for (j = 0; j < 256; j++)
        tmp = bit_xor(
            ciphertexts[number_of_byte, i], j)
        before_sbox = inverse_sbox(tmp)
        position = inverse_shift_rows(number_of_byte)
        key_hypothesis[i,j] = hamming_distance(
            before_sbox,
            ciphertexts[number_of_byte, position])
```

operation took place in. Depending on the settings of the oscilloscope there are usually more samples per one clock cycle of AES design.

3.3 AES for Smart Card

This is the first AES implementation that we made. It is written in C programming language and it is intended for smart card. The primary aim of this implementation was to get familiar with AES and DPA. The DPA against smart cards was heavily researched and there are many available resources on it.

The smart card we were using is controlled by Atmel AVR microcontroller unit (MCU). It is an 8 bit MCU, all operations in the implementation are byte oriented. It runs the AES in a for loop where each iteration is one round. All the transformations are performed per byte. The S-Box is implemented as a look-up table in program memory. Whole implementation is located in one source file with one exported function. Its signature can be seen in listing 3.3. For better readability the implementation is separated into multiple functions but those functions are not exported. The arguments of function in Listing 3.3 are:

Listing 3.3: AES C implementation interface

```
void aes128_block_encrypt(
        const unsigned char * plaintext,
        unsigned char * ciphertext,
        const unsigned char * key)
```

plaintext The address of a buffer with plaintext. Its expected size is 16 bytes.

ciphertext The address of a buffer which is used for storing ciphertext. Its expected size is 16 bytes.

key The address of a key. Because the function encrypts by 128 bit AES, the size of a key is 16 bytes.

The source file is a part of simple operating system for smart cards called BHW SOSSE. This system is used in the course MI-BHW [4] for teaching DPA against smart cards.[8] The communication with this system is via application protocol data unit (APDU).[9] The system supports 5 commands. Four of them invoke example encryption and decryption algorithms in C and assembler. The fifth actually does nothing. It is up to the students of MI-BHW course to add a genuine implementation of AES here. We added our AES implementation here and we used this system for performing a DPA on a smart card.

3.4 AES for FPGA

This is the first VHSIC Hardware Description Language (VHDL) implementation of AES that we made. The other VHDL implementations are modifications of this one. We subsequently modified this implementation to make it easier to break. Summary of all FPGA implementations is provided in Table 3.1.

The encryption in each of AES designs takes a different number of clock cycles. The operation was subsequently divided among more and more clock cycles. There is also a preparation before the encryption which takes one clock cycle. The summary of clock cycles needed for the encryption is shown in Table 3.2.

The top level entity is called *AES128* and its interface is shown in listing 3.4. The following list presents a brief description of the *AES128* ports.

RS232_RXD A serial line receive signal connected to RXD pin (R7) of serial line connector on the board.

[8] It was originally created by Matthias Bruestle at Ruhr-University Bochum

[9] A protocol for communication between a smart card and a reader

Variant	Frequency (kHz)	Trigger before start	S-Box register	S-Box per clock cycle	Key register
AES	50,000	✗	✗	✗	✗
AES_CLK	1,562.5	✓	✗	✗	✗
AES_SUBBYTES	1,562.5	✓	✓	✗	✗
AES_BYTES	1,562.5	✓	✓	✓	✗
AES_KEY	1,562.5	✓	✓	✓	✓

Table 3.1: Overview of AES FPGA implementations

Variant	Initial preparation	One round	Total
AES	1	1	11
AES_CLK	1	1	11
AES_SUBBYTES	1	2	21
AES_BYTES	1	17	171
AES_KEY	1	17	171

Table 3.2: Overview of clock cycles needed for encryption in each variant (Total equals Initial preparation plus 10 times One round)

Listing 3.4: FPGA implementation top level entity interface

```
entity AES128 is
  port (
      RS232_RXD   : in std_logic;
      RST         : in std_logic;
      CLK         : in std_logic;
      RS232_TXD   : out std_logic;
      ENCRYPTING  : out std_logic
  );
end entity;
```

RST A reset connected to the South Button (FPGA pin K17). During testing I have found out that the reset button needs to be pushed down after loading the bitstream to the FPGA chip for the design to correctly work. If the reset button is not pushed down the board returns incorrect results.

CLK A clock signal connected to the oscillator with frequency 50 MHz (pin C9).

RS232_TXD A serial line transmit signal connected to TXD pin (M14) of serial line connector on the board.

ENCRYPTING A trigger signal connected to the pin IO9 (FPGA pin D7). In the initial design the trigger is in the state of logic one during the encryption. The logic driving the trigger was changed in subsequent designs. Trigger was put high for 16 cycles and then low for another 16 cycle before the actual encryption begun. Trigger signal helps to properly locate the encryption on the oscilloscope during measurement. We have found out that without trigger it is nearly impossible to locate the time frame of the encryption.

All the functionality is encapsulated by this entity. It basically listens to serial interface and waits till enough bytes, which is 16 in this case, were received. After that, it starts the encryption and waits until the encryption is done. Any data received over serial line during the encryption are discarded. Once the encryption is completed, the encrypted block is sent back. Then it waits for another block.

The block diagram of top level entity *AES128* can be seen in Figure 3.1. It consists of 5 entities:

AES128_CONTROLLER The finite state machine (FSM) that controls receiving, encrypting and sending data.

AES128_BLOCK This entity performs the encryption.

RS232 A module that sends and receives data over serial line.

CIPHERTEXT_REG Register for storing ciphertext.

PLAINTEXT_REG Register for storing plaintext.

The *AES128_BLOCK* entity expects a 128 bit wide vector containing the key. The constant key *0x00 0x11 0x22 0x33 0x44 0x55 0x66 0x77 0x88 0x99 0xAA 0xBB 0xCC 0xDD 0xEE 0xFF* is being put to this vector. It is hardcoded in the design and cannot be changed in runtime.

Below we bring description of all 5 blocks composing the top level entity *AES128*.

3.4.1 AES128_CONTROLLER

The controller is realized as a Mealy FSM. It follows a typical structure with 3 processes. The first process is a combinatorial logic of transitions, second process is a combinatorial logic of outputs and third process is a sequential logic storing current state. The FSM state diagram can be seen in Figure 3.2

There is another auxiliary process. It serves as a counter of received bytes from the serial line before the actual encryption takes place, and transmitted bytes over serial line when the encryption is done. If the signal *INIT_COUNTER* is put high the counter value is set to 16. The counter

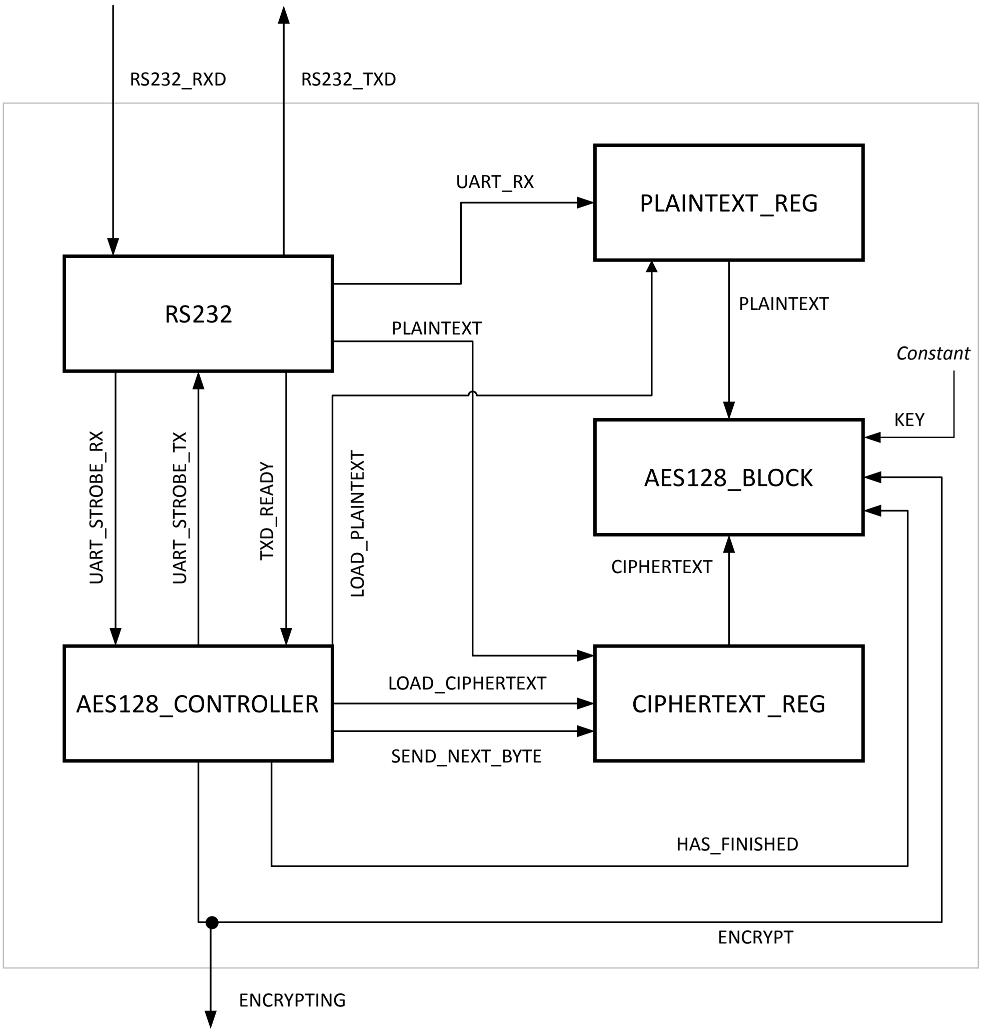

Figure 3.1: AES128 block diagram (each name of signal inside the entity ends by _INNER, this string was removed from the diagram because of simplicity)

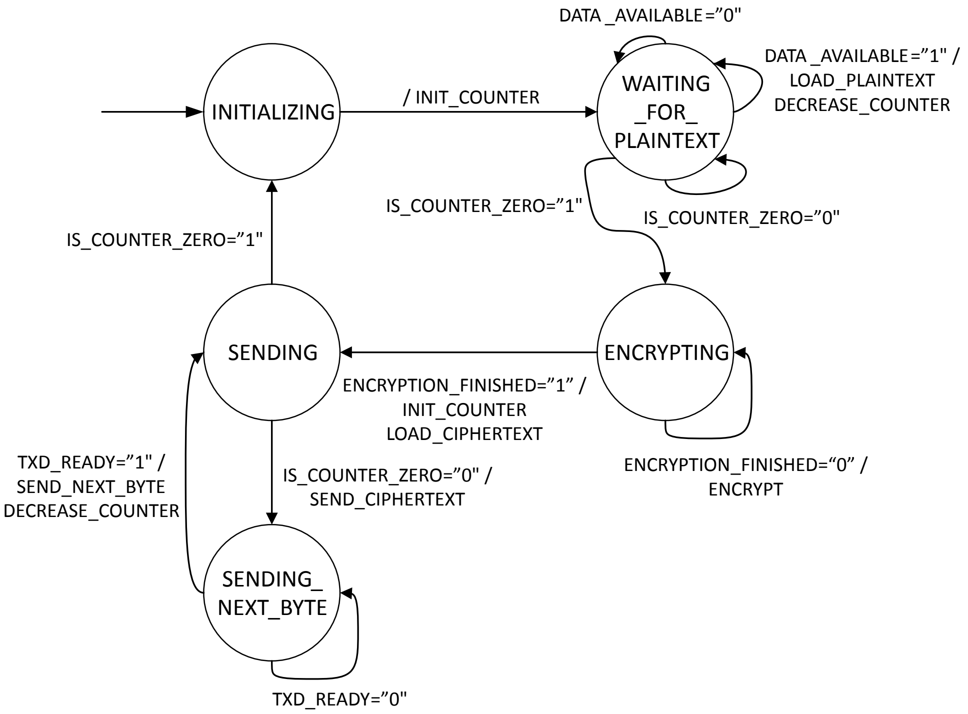

Figure 3.2: AES128_CONTROLLER state diagram

decrements by one if signal *DECREASE_COUNTER* is in the state of logic one. If the counter value is zero the signal *IS_COUNTER_ZERO* is pulled up.

The input of the controller are state signals from data paths. State transitions in the finite state machine are driven by those state signals. State signals are as follows:

DATA_AVAILABLE Active if new byte was received over serial line.

TXD_READY Active if *RS232* entity (see 3.4.2) is able to sent another byte.

ENCRYPTION_FINISHED High if the encryption has finished.

The output of the controller are following control signals:

LOAD_PLAINTEXT Stores next byte of plaintext in *PLAINTEXT_REG* (see 3.4.3).

LOAD_CIPHERTEXT Stores the output of *AES128_BLOCK* (see 3.4.4) in *CIPHERTEXT_REG* (see 3.4.3).

SEND_CIPHERTEXT Emits a signal to the *RS232* converter (see 3.4.2) to transmit another byte.

SEND_NEXT_BYTE Performs a shift by one byte to the left in *CIPHERTEXT_REG* (see 3.4.3).

ENCRYPT Starts the encryption.

3.4.2 RS232

This is the entity used for communication over serial line. At first we used an implementation that we had created together with Jan Severyn as a part of an assignment in the course BI-PNO [1]. However due to issues with correct communication in our own implementation we switched to the implementation provided by the supervisor. It can be downloaded from webpage of course MI-BHW [4].

In comparison with original implementation we changed the baud rate from 9,600 bauds to 115,200 bauds. The entity is capable of sending and receiving data over serial line. The RS232 unit takes the data from *TXD_DATA* if *TXD_STROBE* is active and transmits them to the counter party. Once the transmission is done, and RS232 converter is able to transfer another byte, it pulls the signal *TXD_READY* high. The entity waits for data received by serial connection. When it receives complete byte it makes the signal *RXD_STROBE* active and the received data are present in vector *RXD_DATA*. The data are valid as long as *RXD_STROBE* is in the state of logic one.

3.4.3 CIPHERTEXT_REG and PLAINTEXT_REG

Shift registers for storing ciphertext or plaintext respectively. Both are simple and similar to each other. Because of their simplicity they were not realized as standalone entities but just as processes in the top level entity.

CIPHERTEXT_REG It stores the ciphertext from *AES128_BLOCK* (see 3.4.4) if the signal *LOAD_CIPHERTEXT_INNER* is active. The left shift is performed if the signal *SEND_NEXT_BYTE_INNER* is "1". The most left byte is discarded and the register is padded by one byte with value 0 on the right. The highest 8 bits of the output are connected to the input of *RS232* (see 3.4.2)

PLAINTEXT_REG The only purpose of this register is to store received bytes until enough bytes were received. The encryption starts once 16 bytes were received.

3.4.4 AES128_BLOCK

This is the entity that performs the encryption by AES128 cipher. It encrypts one block at a time. The interface of this entity is shown in Listing 3.5. If the signal *ENCRYPT* is put high the entity stores the *PLAINTEXT* and *KEY* into internal registers and the encryption begins. Once the encryption finishes

Listing 3.5: AES128_BLOCK interface

```
entity AES128_BLOCK is
  port (
    PLAINTEXT    : in std_logic_vector (127 downto 0);
    KEY          : in std_logic_vector (127 downto 0);
    ENCRYPT      : in std_logic;
    RESET        : in std_logic;
    CLK          : in std_logic;
    CIPHERTEXT   : out std_logic_vector (127 downto 0);
    HAS_FINISHED : out std_logic
  );
end entity AES128_BLOCK
```

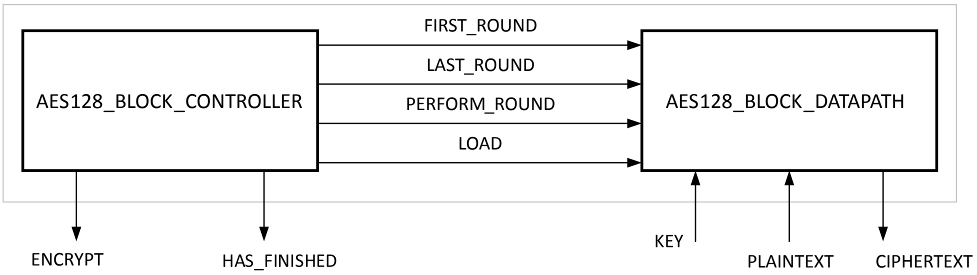

Figure 3.3: AES128_BLOCK block diagram (each name of signal inside the entity ends by _INNER, this string was removed from the diagram because of simplicity)

the signal *HAS_FINISHED* is active. If it is active the data in the vector *CIPHERTEXT* are valid. It is active only for one clock cycle. Encryption of new block can start when the encryption of previous block is done.

We have decided to use the implementation with iterative design. Our original idea was to test different implementations but eventually we sticked just with this one and we have not tested the pipelined one for instance. We have found out during measurement that the determinant for success is mostly the oscilloscope setup and not the implementation itself.

The entity performs one round per one clock cycle. At the end of the round the data are stored to the register. Next round starts in next clock cycle. The encryption takes 11 clock cycles in total. Ten round accounts for 10 cycles and one more cycle is used at the beginning of the encryption to store original key and plaintext. This sums up to 11 clock cycles in total.

AES128_BLOCK is separated to two entities: data paths and the controller (see Figure 3.3 for overall structure).

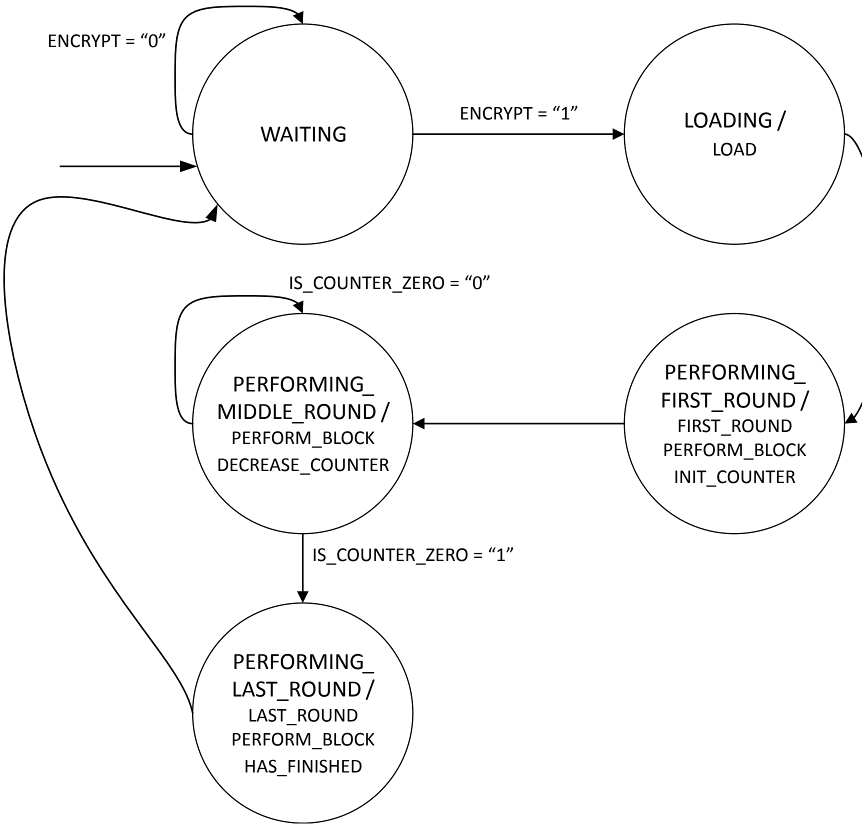

Figure 3.4: AES128_BLOCK_CONTROLLER state diagram

AES128_BLOCK_CONTROLLER

The controller of the *AES128_BLOCK* entity is realized as a Moore FSM. As in the case of the top level entity (see subsection 3.4.1), the controller consists of 3 processes. It waits till the signal *ENCRYPT* has a value of ”1” and then it starts the encryption. It makes active signals *FIRST_ROUND* and *LAST_ROUND* according to the round that is being processed. It holds the *PERFORM_ROUND* high for the duration of the encryption. When the encryption is over it sets the *HAS_FINISHED* to ”1”. The ciphertext is valid as long as the signal *HAS_FINISHED* has value ”1”. The controller also contains an internal counter for counting rounds. The diagram of the FSM can be seen in Figure 3.4.

AES128_BLOCK_DATAPATH

This is the entity which the actual encryption takes place in. The block diagram is shown in Figure 3.5. The entity has a register which is used for storing the current cipher state. The current cipher state is written to the register when the *PERFORM_ROUND* signal is active. If the signal *LOAD* is active the plaintext is loaded to the register instead of the cipher state.

Round keys are not precomputed at the beginning but each successive round key is computed in applicable round. There are two registers used for the round key computation. The registers are *ROUND_KEY_REG* and *RCON_REG*. The *ROUND_KEY_REG* register stores the current round key. If the signal *PERFORM_ROUND* is high a new round key is stored. If the signal *LOAD* is active the original key is loaded to the register. The *RCON_REG* is used for storing the round constant. It computes and stores new round constant if the signal *PERFORM_ROUND* is active. If the signal *LOAD* is active the value *0x01* is loaded to the register.

All the registers do not write anything if both *PERFORM_ROUND* and *LOAD* are not active. If the signal *FIRST_ROUND* is active, the plaintext is xored with the key. If the signal *LAST_ROUND* is high, the operation MixColumns is omitted.

The transformations in AES are defined in terms of operations in $GF(2^8)$. In the design we created functions used for computations in $GF(2^8)$. Those functions are located in file *AES_COMMON.vhd*. The summary of the functions is below:

Addition Addition of two values in $GF(2^8)$ is a bit xor of those values.

Multiplication by 2 When multiplying by 2 the value is shifted to the left by one. If there is an overflow the result of shifting is xored with the value *0x1B*.

Multiplication by 3 The value is multiplied by 2 at first and then the original value is added to the result of the multiplication. It is based on the identity $3a = 2a + a$.

The AES consists of 4 transformations as described in section 1.1. The list below describes realization of each of the blocks in the design.

AddRoundKey This is simply realized as a *xor* in VHDL language. The vector with the key and the vector with the current cipher state are xored.

SubBytes Subbytes are realized as a *case* statement. The substitution of values is performed on a byte basis. We have found out during synthesis that this operation takes the most FPGA chip area. We were experimenting with different realizations but all ended up with similar or worse results.

ShiftRows We change the order of bits. It is implemented as a vector assignment with different indexes.

MixColumns This operation is based on matrix multiplication in $GF(2^8)$. For the multiplication we use functions described earlier in this section.

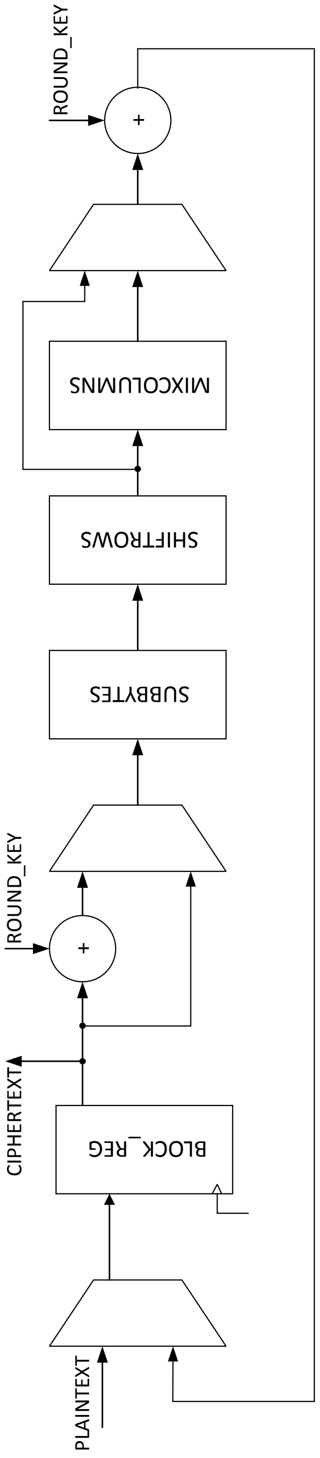

Figure 3.5: AES128_BLOCK_DATAPATH block diagram (the logic responsible for key schedule was omitted because of better readability)

Listing 3.6: Variant AES_CLK: top level entity interface

```
entity AES128 is
  port (
      RS232_RXD  : in  std_logic;
      RST        : in  std_logic;
      CLK        : in  std_logic;
      RS232_TXD  : out std_logic;
      TRIGGER    : out std_logic
  );
end entity;
```

3.5 AES_CLK

We decided that a few modifications are needed. This section and following ones contain the description of the modifications. Each modification has all the features that the previous one plus the features described in particular section. Based on suggestions of thesis supervisor we decided to lower the clock rate and change the logic behind the trigger.

The previous implementation (section 3.4) was clocked on 50 MHz. In modification *AES_CLK*, we divided the clock rate by a factor of 32 which resulted to 1,562.5 MHz. It was the lowest clock rate the universal asynchronous receiver/transmitter (UART) module was able to operate on (with 115,200 baud). We achieved the required behavior by adding a prescaler to the top level entity. Prescaler is a 5 bit counter that increases by one with every rising edge of the clock signal. The highest bit of prescaler is connected to the clock inputs of other entities.

The second change that we made regards the trigger. The trigger was originally active during whole encryption. Because of possible interferences that could lower the quality of signal we changed the trigger so as the trigger is set active for 16 cycles then there is a delay 16 cycles long and then the encryption begins. The trigger reuses the counter used for counting bytes. This change involved change of the *AES128_CONTROLLER* (see subsection 3.4.1), and change of one signal in the top level entity. Changes are shown in Figure 3.6, and Figure 3.7 respectively. The output port *ENCRYPTING* of the top level entity was renamed to *TRIGGER* because it has more accurate meaning (see Listing 3.6).

3.6 AES_SUBBYTES

We were not able to get power traces without the noise and break the implementation at this point. As was revealed later (see chapter 5), it was mostly

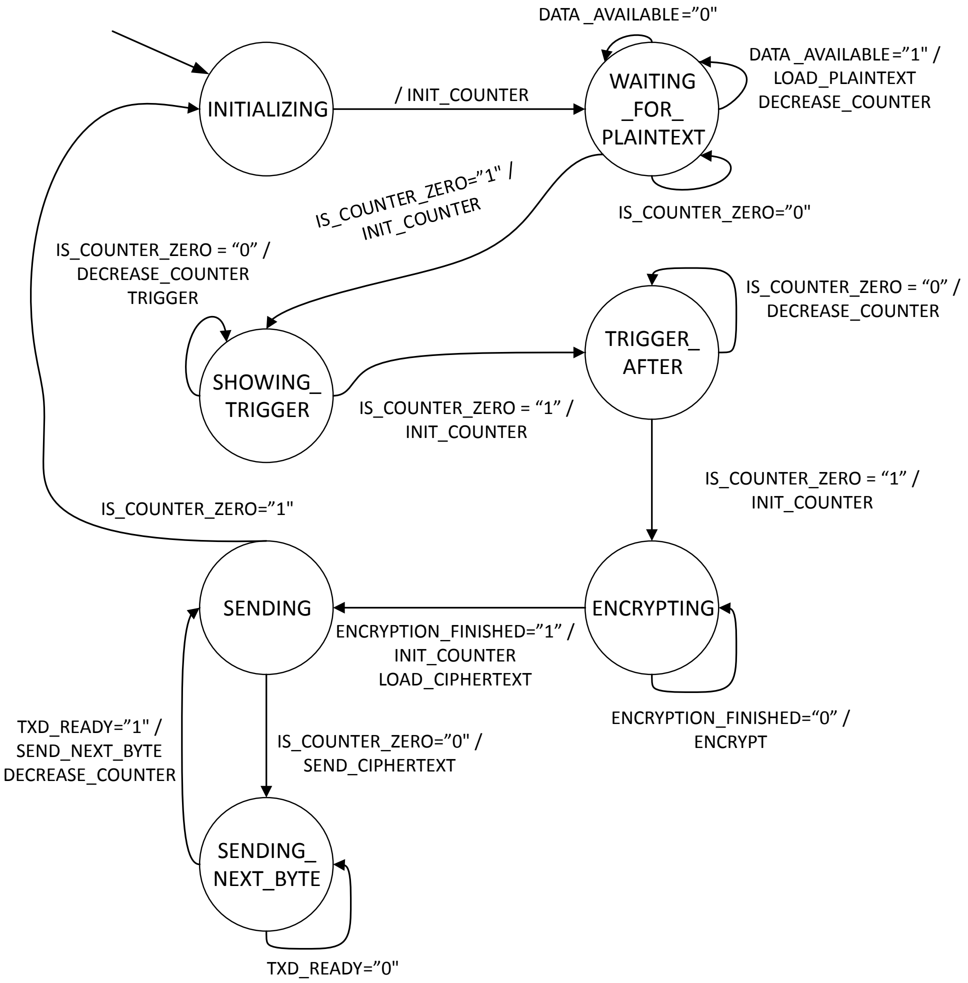

Figure 3.6: Variant AES_CLK: AES128_CONTROLLER state diagram

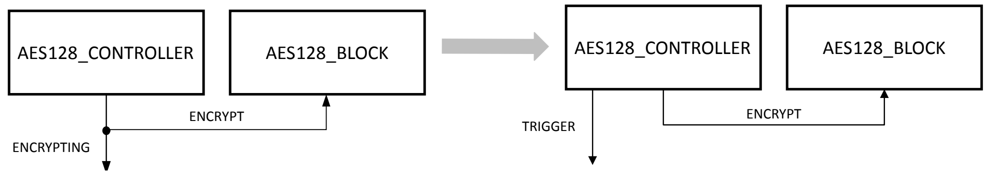

Figure 3.7: Variant AES_CLK: AES128 block diagram change

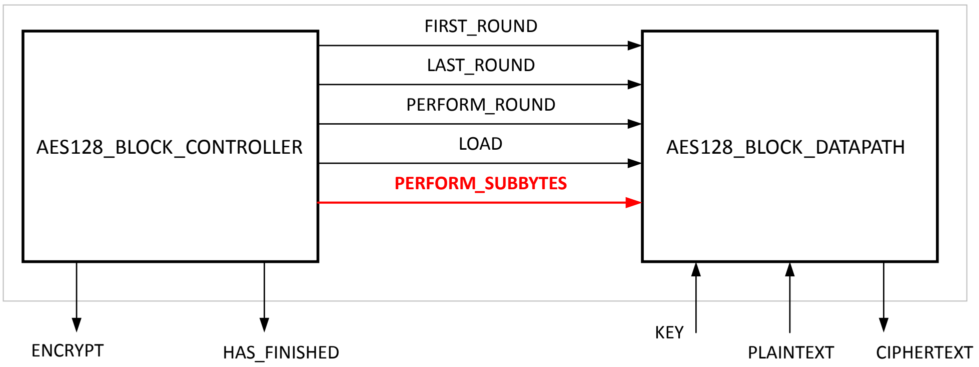

Figure 3.8: Variant AES_SUBBYTES: AES128_BLOCK block diagram (modifications in red, each name of signal inside the entity ends by _INNER, this string was removed from the diagram because of simplicity)

caused by improper measurement setup. However, we were not originally familiar with this fact, and so we made further modifications to the design to make the implementation easier to break.

The first idea was to separate the S-box from the rest of the computation. We added the register after the S-box operation. This modification divided one round into two clock cycles. The S-box result is written into S-box register in first clock cycle and the result of all other operations is written in the second clock cycle into cipher state register. A new signal *PERFORM_SUBBYTES* was added between the data paths and the controller (see Figure 3.8). The result of S-box is written to the S-box register if signal *PERFORM_SUBBYTES* is high (see Figure 3.9).

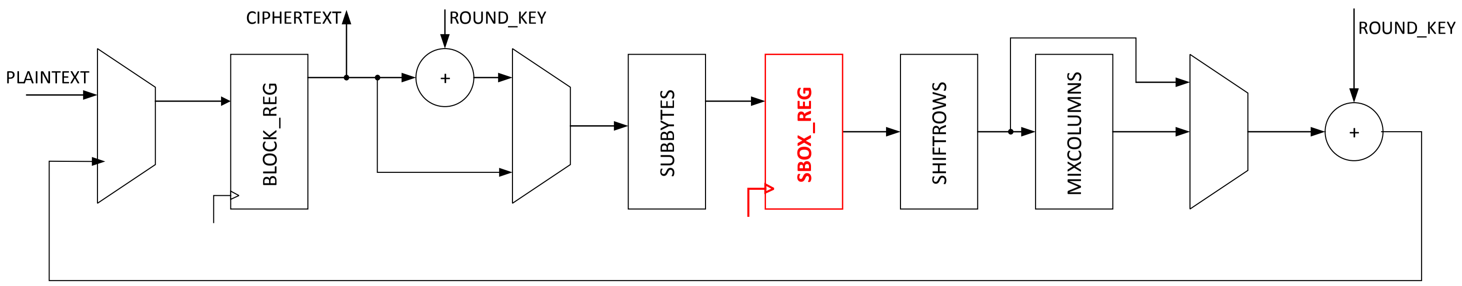

Figure 3.9: Variant AES_SUBBYTES: AES128_BLOCK_DATAPATH block diagram (modifications in red, the logic responsible for key schedule was omitted because of better readability)

Beside adding a register we also modified the controller. The modification reflects the change in data path and emits a signal that writes the data to the S-box register. At this point we decided to rewrite the controller from Moore FSM to Mealy FSM. By rewriting we lowered the number of states and the FSM has became more maintainable. The FSM is shown in Figure 3.10. The controller contains signal *COUNTER_VALUE* with type *natural* which is used as a counter of rounds. Value 9 is loaded to the *COUNTER_VALUE* if signal *INIT_COUNTER* is active. If signal *DECREASE_COUNTER* is active value of the counter is decremented by 1.

In this variant, one round takes 2 clock cycles. The data are written to *BLOCK_REG* register in first clock cycle, and to *SBOX_REG* in second clock cycle.

3.7 AES_BYTES

Writing to the S-box register was divided into multiple clock cycles in this variant. Each byte of the SubBytes operation is written into the S-box register in a different clock cycle. One round takes $16 + 1 = 17$ clock cycles (Writing to *BLOCK_REG* takes 1 clock cycles, subsequent writing of bytes to *SBOX_REG* takes 16 clock cycles.). Out motivation was to divide the operation we are targeting into particular cycles (DPA reveals just one byte of a key at one moment).

A new control signal *SUBBYTE* was added between controller and data paths (see Figure 3.11). The signal has type *natural* with range from 0 to 15. It contains the number of byte that should be written to the S-box register.

The controller was modified to support this behavior. There is a new signal *BYTE_COUNTER_VALUE* which has the same type as a signal *SUBBYTE*. The value is loaded to the byte counter if signal *INIT_BYTE_COUNTER* is "1". If signal *DECREASE_BYTE_COUNTER* is active the value of byte counter is decreased by one. The signal *BYTE_COUNTER_VALUE* is connected to the signal *SUBBYTE* in *AES128_BLOCK*. The state diagram is nearly the same as the one in previous variant except two modifications:

- Signal *INIT_BYTE_COUNTER* is active in states:
    - *PERFORMING_ROUND*
    - *WAITING* if *ENCRYPT* is high
- Signal *DECREASE_BYTE_COUNTER* is active in states:
    - *PERFORMING_SUBBYTES*

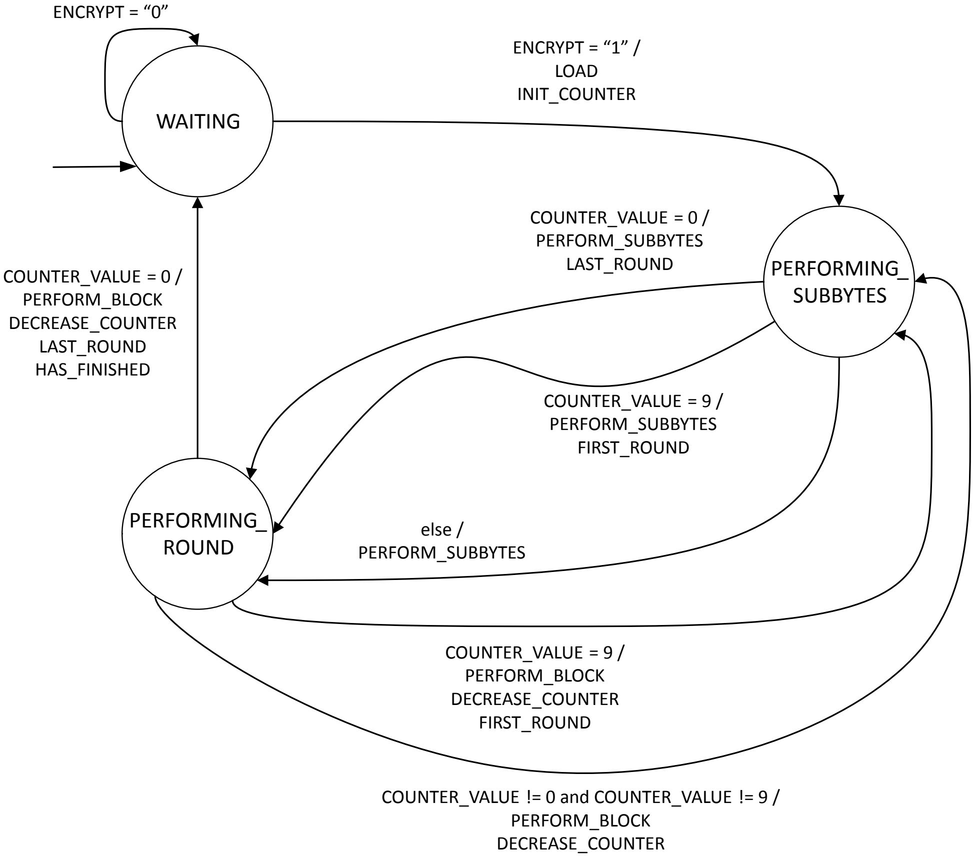

Figure 3.10: Variant AES_SUBBYTES: AES128_BLOCK_CONTROLLER state diagram

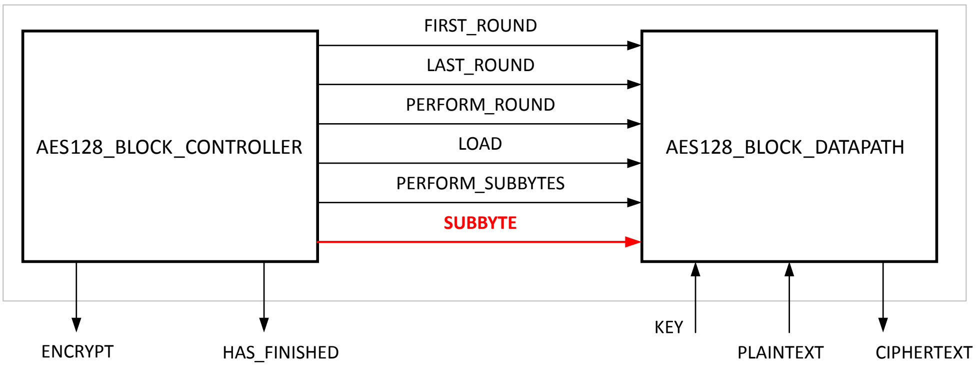

Figure 3.11: Variant AES_BYTES: AES128_BLOCK block diagram (modifications in red, each name of signal inside the entity ends by _INNER, this string was removed from the diagram because of simplicity)

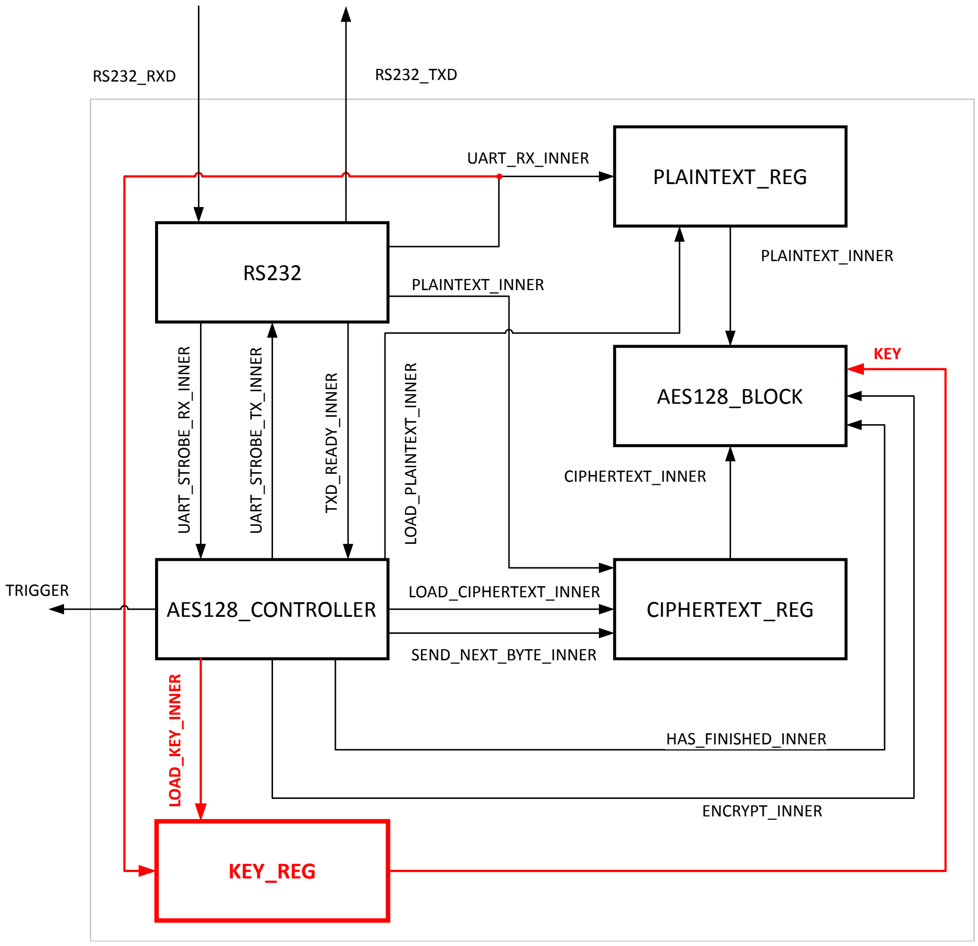

Figure 3.12: Variant AES_KEY: AES128 block diagram (modifications in red)

3.8 AES_KEY

We were worried that the hard-coded key could be optimized away by the synthesis tool. The key was hard-coded in the top level entity in previous implementations. We added the register for key to the top level entity (see Figure 3.12), and modified the controller to support receiving the key over serial line.

The format of the communication between the board and its counterpart has changed. Instead of sending 16 bytes of plaintext there are 17 bytes. The first byte can be seen as a control byte. Depending on the value of the first byte the controller decides what state transition will be performed. If the byte has the value 0x00 the controller takes the following 16 bytes as a key and stores them to the key register. No response is sent back in this case. If the first byte has the value 0xFF the controller takes the following 16 bytes as

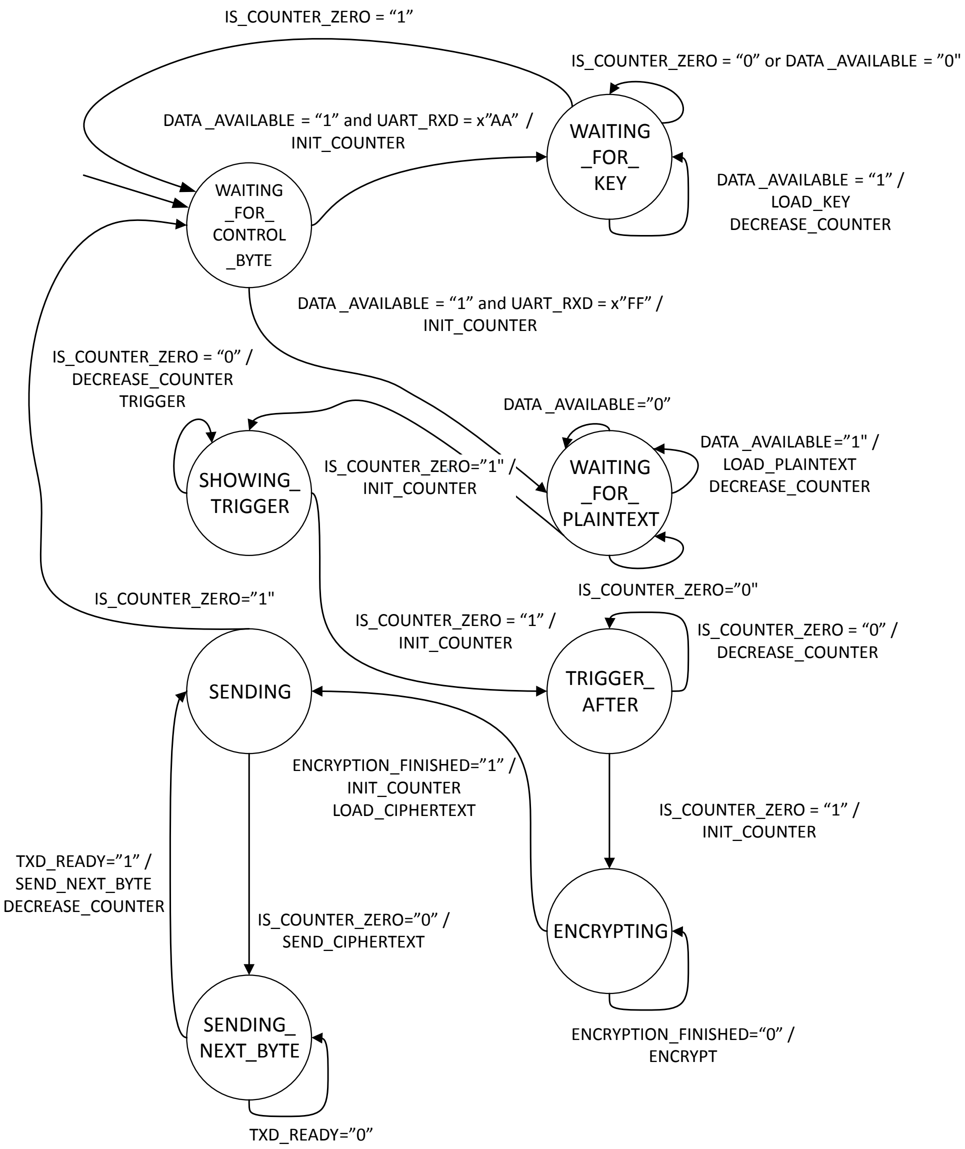

Figure 3.13: Variant AES_KEY: AES128_CONTROLLER state diagram

a plaintext and encrypts them. After encryption the ciphertext is sent back. The key can be changed repeatedly and the encryption always uses current key. This state diagram of the controller can be seen in Figure 3.13.

Application SC Power Measurement (see section 3.1) has been updated accordingly and it sends 17 bytes instead of 16. The first byte is always 0xFF and the key must be set outside the SC Power Measurement before the measurement begins.

Chapter 4

Testing

This chapter describes the testing that we performed. The next paragraph presents an overview of different kinds of tests. The section 4.1 contains a description of verification and validation tools which we created to help us to test hardware designs. Testing strategies and results of applications and designs that we described in chapter 3 are presented in the section 4.2 and section 4.3.

The tests that we performed can be divided into two groups.

Verification tests Testing in simulator or development environment. This testing is usually conducted on a unit basis. Each unit of design is covered by the tests separately. The simulator sends some arbitrary input and checks the output for correctness.

Validation tests The design is deployed to the device and then tested. The whole design is tested at once under circumstances that can occur during deployment. Tests can be performed both manually or automatically.

We used both verification and validation testing for the entity *AES128_BLOCK*. We tested all the applications, scripts, and designs by validation tests (see Table 4.1).

Unit	Verification testing	Validation testing
SC Power Measurement	✗	✓
Scripts for performing DPA	✗	✓
AES for smart card	✗	✓
AES128_BLOCK entity (FPGA)	✓	✓
FPGA design	✗	✓

Table 4.1: Overview of performed tests

4.1 Testing Tools

As a part of this thesis we created different tools used for testing. The aim of those tools is to help us with testing and make the testing more automatic. We have created two testing applications. One is for *AES128_BLOCK* entity verification and the other one is for the whole FPGA design validation. Both are written in C# programming language.

AESGenerator It generates random data and random keys. The application takes one argument from the command line. It is a number telling how many blocks and keys should be generated. The generated data and keys have size 16 bytes so they reflect block size and key size for 128 bit AES. The application then encrypts those data by a generated random key. Each block of data is encrypted by a different key. Generated data, encrypted data, and keys are stored in text files one block per line. The values are bytes in hexadecimal format separated by spaces. Those files can be read by the testbench used in FPGA designs.

FPGATester It is an application that can communicate with FPGA. It generates random plaintext and sends it to the FPGA. Then it waits for the response. It encrypts the generated plaintext and compares it with the received ciphertext. The encryption key is hard-coded and must be the same as the key used in FPGA design that is tested. If the ciphertext is not received in specified time interval or the ciphertext is incorrect it stops testing. Otherwise it runs indefinitely until the application is closed. The test result can be seen in console. We used this application for the FPGA designs validation.

4.2 Verification

We performed the verification of the entity *AES128_BLOCK* from FPGA designs. We created an entity *AES128_BLOCK_TB* that is a testbench intended to be used in the simulator. The testbench reads its input from 3 files (plaintexts, ciphertexts, and keys). Each block is on separate line in hexadecimal text format. Those files can be generated by *AESGenerator* (see section 4.1). It uses the data from input and sends them to the entity. It waits for the result and compares it with expected result.

We performed verification testing of entity *AES128_BLOCK* in each design. We performed a behavioral simulation in Xilinx ISE development environment. It was a pre-synthesis simulation (i.e. there was no timing information). We used 256 different blocks and keys. All tests were successful.

4.3 Validation

We performed the validation of all applications. scripts, and designs that we created. We did an automatic validation of FPGA design by the *FPGATester* (see section 4.1). We performed a manual validation of the rest.

SC Power Measurement We performed a measurement on FPGA with this application. We then plotted the first trace and visually inspected the plot. Plotted traces seemed correct. We also recovered correct key from captured traces.

Key Recovery using DPA There are sample data in the same format as the output of SC Power Measurement that are available at the CTU website [14]. The key used for obtaining sample date is known.[10] We used those data and we verified that the key we received from Key Recovery application is the same as the key from the sample data.

AES for smart card We manually sent a plaintext to the card and received the ciphertext. We encrypted the plaintext using a third party tool [5] and compared the ciphertext received from smart card with ciphertext from this third party tool. We verified that both ciphertexts are same. We repeated this test a few times. There is a very low probability that we would get the correct ciphertext in all the attempts and the AES implementation would be working incorrectly. Based on previous statement we can conclude that the implementation is returning correct results.

AES for FPGA We used the *FPGATester* (see section 4.1) for FPGA AES implementation validation. We let the tool run for a few tens of minutes and verified that all results were correct. This statistical sample was large enough to state that the implementations are working correctly. We performed this test for each FPGA implementation.

[10] The key is 0x00112233445566778899AABBCCDDEEFF.

Chapter **5**

Mounting the DPA Attack on AES Implementations

We performed various measurements with different devices (smart card and FPGA), different implementations (see chapter 3), different methods of the attack (see subsection 2.2.3) and different modifications of the FPGA board. At first we performed the DPA against a smart card (results are described in section 5.1). Then, we mounted the attack against FPGA (results are discussed in section 5.2).

Correlation matrices are stored on the attached DVD, and can be found in a directory *measurements*. The format of correlation matrix is (256, *number_of_samples*) where each row is one key hypothesis. The directory *measurements* contains subdirectories reflecting each measurement. The name of the subdirectory is same as the name used for the measurement in this chapter. A correlation matrix was exported from Wolfram Mathematica for each measurement, and it can be imported back to the Wolfram Mathematica (see Listing 5.1). We also included captured traces for some measurements. It was not possible to include traces from all the measurements because the size of the file with traces is usually several gigabytes.

Each measurement contains a plot of traces and a plot of correlation coefficients of all key hypothesis. The plot of traces contains only the first trace

Listing 5.1: Wolfram Mathematica import and export of correlation matrices

```
(* Command used for export *)
(* cor is a matrix with correlation coefficients *)
Export["correlation_matrix.mx", cor, "Table"];

(* Command used for import *)
cor = Import["correlation_matrix.mx", "Table"];
```

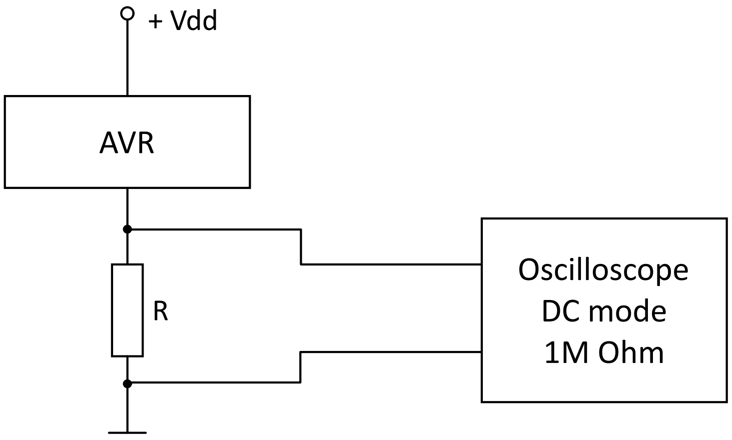

Figure 5.1: Measurement setup for smart card

in all the measurements, and the plot of correlation matrix contains first byte of a key.

5.1 Smart Card

At first, we decided to mount the attack against a smart card. Our primary aim was to verify that we are able to successfully perform the attack. We used a special adapter for measuring smart card power consumption used in the course MI-BHW [4]. It has a resistor placed on the GND line and pins that can be used for connecting the oscilloscope (see Figure 5.1). We used Agilent[11] DSOX3012A [9] oscilloscope. We performed 150 measurements, used Hamming weight, and mount the attack against the first round. Summarization can be found in Table 5.1 We received a smart card with a secret key from our supervisor. It was possible to recognize each round in captured traces (see Figure 5.2) and the attack was successful. We were able to reveal the secret key. We encrypted randomly chosen plaintext with the obtained key and compared it with respective ciphertext as a confirmation that the key is correct. The correlation matrix on the attached CD contains correlation coefficients for the last byte of the key—correct key hypothesis is 0x79.

Attack	Hamming weight and first round
Implementation	Smart card
Samples per trace	550,000

Table 5.1: Measurement on smart card

[11] The company is now called Keysight

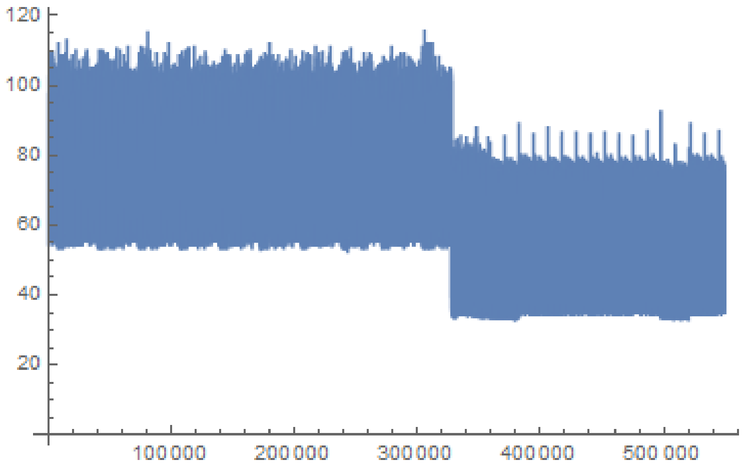

Figure 5.2: Power trace. Configuration section 5.1 (smart card)

5.2 FPGA

Once we successfully broke the smart card implementation we moved to the FPGA implementation. We performed all the measurements with Jan Severyn who is also working on DPA in his bachelor thesis [32]. We replaced a jumper JP7 on the board by 1 Ohm resistor. We performed an initial measurement of the implementation without any simplification (see section 3.4). We did not obtained any usable result. Then we realized that we short-circuited the FPGA via oscilloscope and decided to perform following changes:

- Use a differential probe Hameg HZO41 [2] to prevent a short-circuit.
- Lower clock rate in the design.
- Use the oscilloscope Agilent MSO7104A [7] because it has higher sampling frequency (1 GHz). Later, we found out that the lower the number of samples the better because the computation is faster.

The measurement setup is shown in Figure 5.3.

Subsequently we removed capacitors C158–C175 on the 1.2 V power line to the core as proposed by Velegalati and Yalla [34] who were exploring the possibilities of the DPA on Spartan-3E board.

Wa faced issue with memory demands of the DPA scripts during the computations. We decided to use as few samples as possible to lower the memory consumption. We visually examined captured traces before each computation and tried to visually locate the subset of samples where the operation took place in. Also we were able to run the analysis with tens of thousands of traces at most because of high memory consumption.

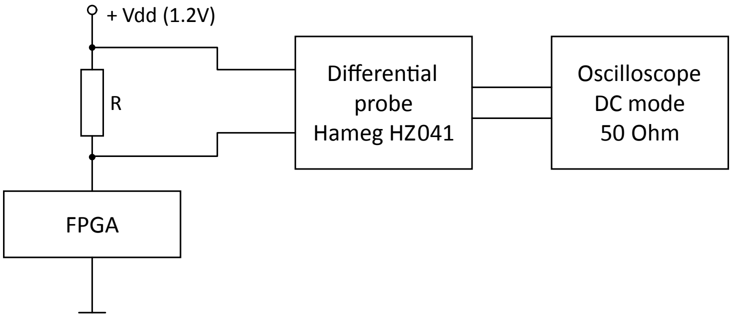

Figure 5.3: Measurement setup for FPGA with differential probe

Even though we used 115,200 bauds for serial communication, the measurement was considerably slow. Collecting 100,000 power traces took approximately 2 hours. We found out that communication is the slowest part of the measurement process. The communication takes a few milliseconds whereas the encryption just a few microseconds. We propose an improvement in chapter 6.

We used key (0x00 0x11 0x22 0x33 0x44 0x55 0x66 0x77 0x88 0x99 0xAA 0xBB 0xCC 0xDD 0xEE 0xFF) in all the measurements. Correct key hypothesis for the first byte of a key is 0x00 in the case of an attack to the first round and 0x36 in the case of an attack to the last round (0x36 is the first byte of round key for 10^{th} round).

Summary of all performed measurements of FPGA implementations is provided in Table 5.2

FPGA design	Decoupling capacitors	Power supply	Measurement setup	Description in subsection
AES_CLK	✓	switched	DP	5.2.1
AES_SUBBYTES	✓	acc	DP	5.2.2
AES_KEY	✗	acc	DP	5.2.3
AES_CLK	✗	acc	Preamp	5.2.4
AES_CLK	✗	switched	Preamp	5.2.5
AES_CLK	✓	acc	Preamp	5.2.6

Table 5.2: Summary of performed measurements (Decoupling capacitors: ✓= present, ✗= removed, Power supply: switched = switched-mode power supply, acc = accumulators, Measurement setup: DP = differential probe, Preamp = preamplifier)

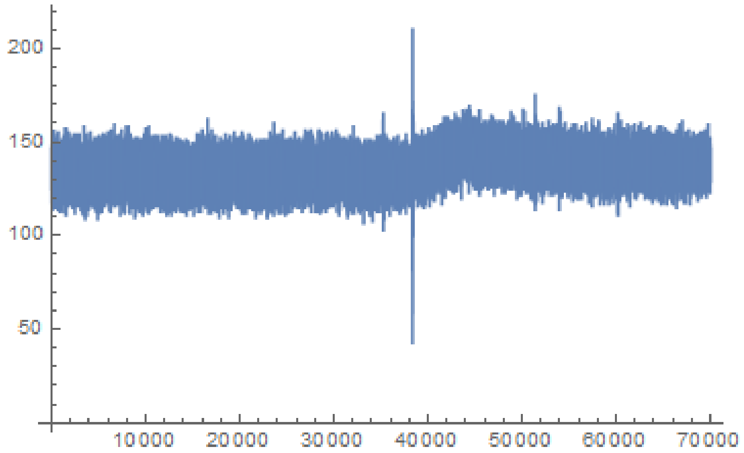

Figure 5.4: Power trace. Configuration: subsection 5.2.1 (AES_CLK, capacitors present, switched-mode power supply, differential probe)

5.2.1 AES_CLK, differential probe

This is the first measurement that we performed (except the short-circuited one). We used the *AES_CLK* implementation and board with present decoupling capacitors and powered by switched-mode power supply. Summary of the setup is in Table 5.3. Captured traces are shown in Figure 5.4. The quality of the samples is not very good. Moreover there was a low frequency wave which we attributed to the switched-mode power supply. We estimate that the operation took place between samples 38,000 and 44,000. We can observe a voltage growth during given period. We tried Hamming weight and first round as an attack, but the attack was unsuccessful (see Figure 5.5).

Environment	Differential probe
Board setup	No modification
Attack	Hamming weight and first round
FPGA design	AES_CLK
Samples per trace	200,000

Table 5.3: Measurement AES_CLK, differential probe

5.2.2 AES_SUBBYTES, accumulators, differential probe

We decided to power the FPGA from accumulators. We connected the accumulators to the spots on the board we identified in subsection 2.3.1. The 1.2 V line was connected to the jumper JP7, the 2.5 V was connected to the jumper JP6, and the 3.3 V line can be connected to any peripheral Vcc pin. The accumulators were connected through stabilizers giving us required voltages (see

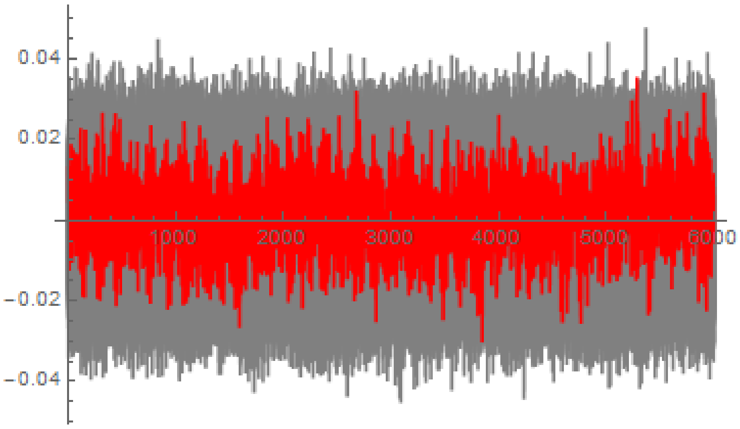

Figure 5.5: Traces of correlation coefficients (correct key in red), samples 38,000–44,000. Configuration: subsection 5.2.1 (AES_CLK, capacitors present, switched-mode power supply, differential probe)

Figure 5.6). At first we connected accumulators to the stabilizers and then to the board. We started with the lowest voltage first and then continued to the highest voltage, always connecting GND before Vcc. Summary of the setup is in Table 5.4.

We also decided to use the implementation with register after the SubBytes operation. We wanted to isolate the S-box to obtain better results.

Environment	Differential probe
Board setup	Accumulators
Attack	Hamming weight and first round
FPGA design	AES_SUBBYTES
Samples per trace	2,000

Table 5.4: Measurement AES_SUBYTES, accumulators, differential probe

We decided to take 1,000 power traces. The number 1,000 is not probably high enough to successfully get the key but we wanted to put focus on reducing the noise, and getting better traces first. Visually the traces were better than the traces in previous measurement, but still far away from the traces we got in subsequent measurements. Captured traces can be seen in Figure 5.7. We tried Hamming weight and first round attack, but it was unsuccessful (see Figure 5.8). We decided to continue with modifications, and try to get less noisy signal.

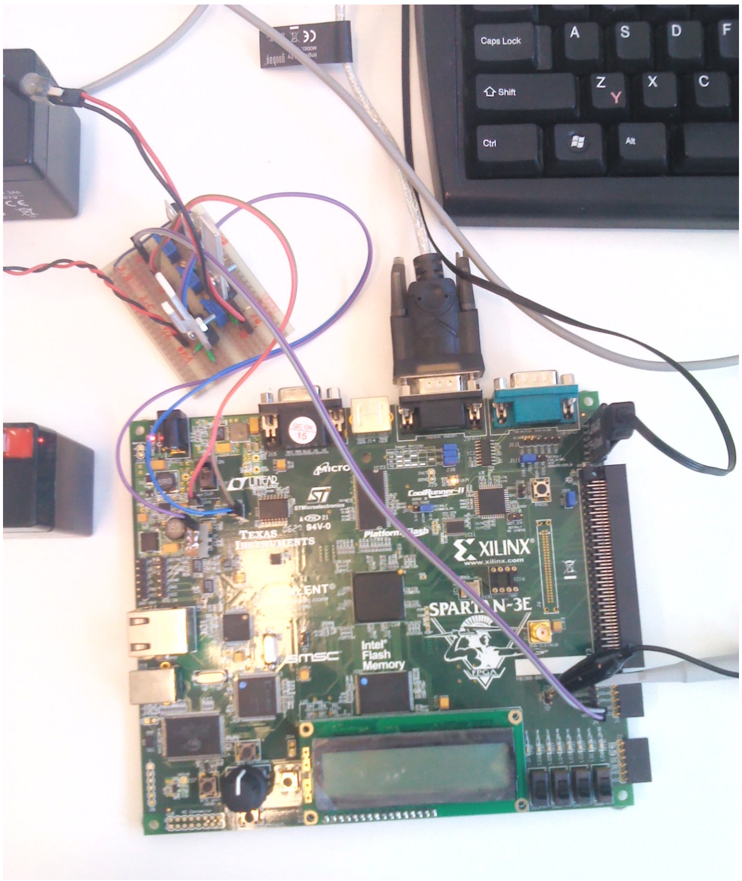

Figure 5.6: FPGA board powered by accumulators, which are connected through stabilizers

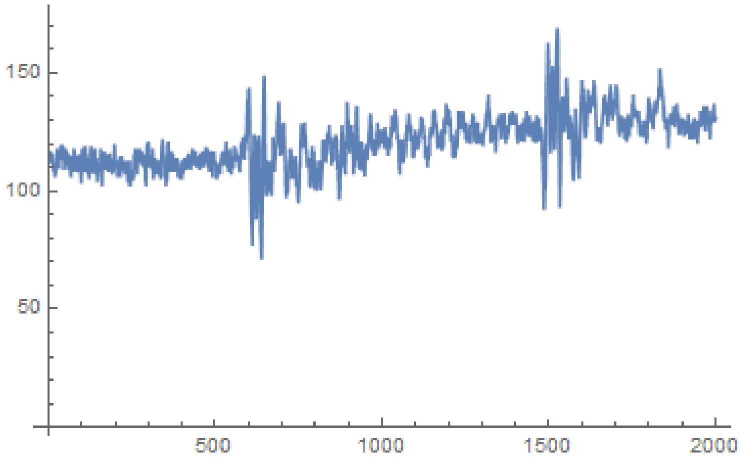

Figure 5.7: Power trace. Configuration: subsection 5.2.2 (AES_SUBBYTES, capacitors present, accumulators, differential probe)

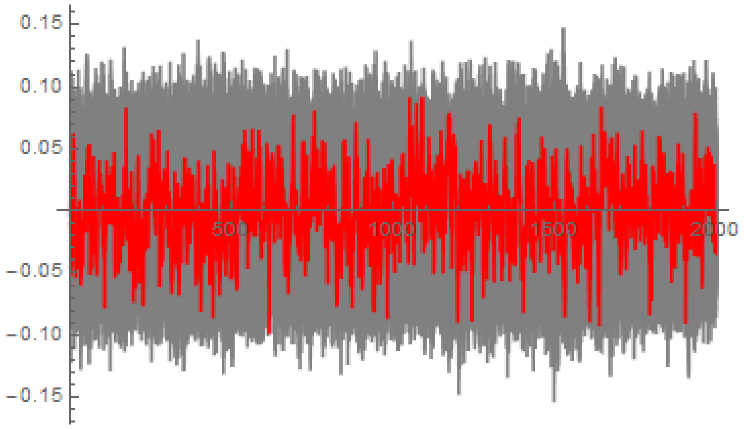

Figure 5.8: Traces of correlation coefficients (correct key in red). Configuration: subsection 5.2.2 (AES_SUBBYTES, capacitors present, accumulators, differential probe)

5.2.3 AES_KEY, no capacitors, accumulators, differential probe

We decided to further modify the design and use an *AES_KEY* implementation (see section 3.8). To further enable viability of a DPA attack we decided to remove decoupling capacitors from the FPGA board. We captured 10,000 power traces with 200,000 samples per trace. Samples 50,000–100,000 can be seen in Figure 5.9. The plot contains only 50,000 samples because it is less dense and better readable than plot with 200,000 samples. Summary of the setup is in Table 5.5. At this point we changed the resistor from 1 Ohm to 10 Ohms and performed the same measurement. The first trace from the second measurement is shown in Figure 5.10. Traces captured with 10 Ohm resistor seem clearer than traces captured with 1 Ohm resistor. We run the analysis using Hamming distance and last round on the data obtained during measurement with 10 Ohm resistor. Because it was hard to identify the cycle where the last round took place in we run the computation on samples 100,000–200,000. We expect the last round took place in this time frame. As in the previous cases, the attack was unsuccessful in this case as well. The

Environment	Differential probe
Board setup	Removed capacitors, accumulators
Attack	Hamming distance and last round
FPGA design	AES_KEY
Samples per trace	200,000

Table 5.5: Measurement AES_SUBYTES, accumulators, differential probe

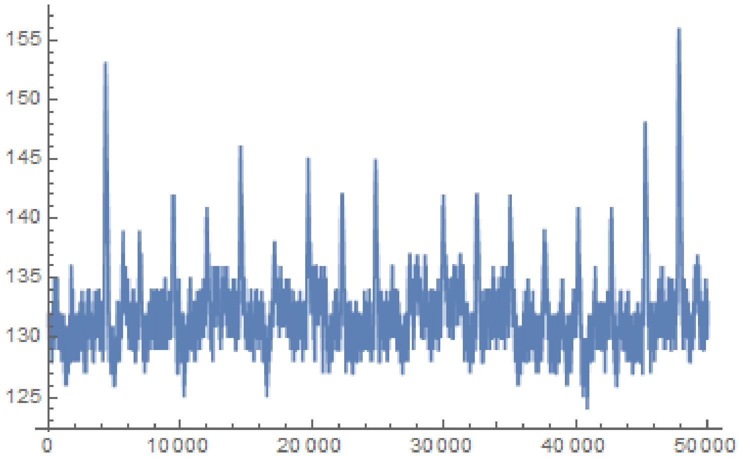

Figure 5.9: Power trace, samples 50,000–100,000. Configuration: subsection 5.2.3 (AES_KEY, capacitors removed, accumulators, differential probe, 1 Ohm resistor)

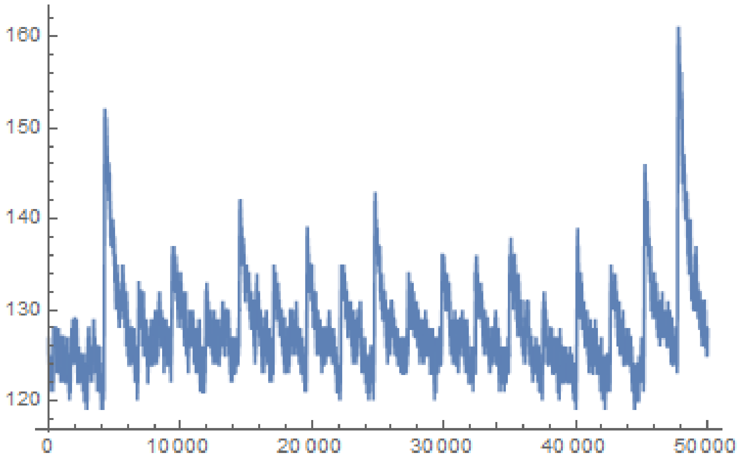

Figure 5.10: Power trace, samples 50,000–100,000. Configuration: subsection 5.2.3 (AES_KEY, capacitors removed, accumulators, differential probe, 10 Ohm resistor)

attack revealed value 0x64, but correct value is 0x36. We were not able to plot the correlation coefficients of key hypothesis because the plotting always crashed because of low memory. Also, we were not able to put the correlation matrix on the attached CD because the correlation matrix was to large.

5.2.4 AES_CLK, no capacitors, accumulators, SMA connector, AC preamplifier

Based on the consultation with Priv.-Doz. Dr. Amir Moradi[12] we decided to change our environment as follows:

- Use the coaxial cable with SMA connector and AC preamplifier PA 303 BNC by Langer EMV-Technik (with gain 30 dB) [3] instead of differential probe

- Turn on bandwidth limit on the oscilloscope

The differential probe we were using before had an attenuation 10:1 (attenuation 20 dB). We replaced the differential probe by the AC preamplifier with gain 30 dB (gain approximately 31.6:1) to obtain better traces. The measurement setup is shown in Figure 5.11.

We observed during computations with Hamming distance that the correct key yields a negative correlation. This is because the higher the power consumption, the higher the voltage drop is (i.e. the change of voltage is negative). The voltage drop can be computed as $u_{drop} = 1.2V - i * R$, where R is a resistor used for measurement, and i is the current flowing through R (see Figure 5.12).

Dr. Moradi also told us that we did not need so many samples, and 10,000–20,000 samples should be sufficient. We should also eliminate all the interferences during the measurement such as mobile phones.

We added a register for key in the last implementation (section 3.8) because we were worried that the key could be optimized out. We concluded that this was not the case and it wouldn't be optimized out, and we decided to use the AES_CLK (see section 3.5) implementation. We used the following oscilloscope settings: DC mode, 50 Ohm input impedance, and bandwidth limit turned on. The exported oscilloscope settings is shown in Listing 5.2 (we used channel 1 for measuring the consumption, and channel 2 for trigger). Summary of the setup is in Table 5.6.

From then on we were performing all the measurements with Martin Mašek who is investigating the resistance of different designs against DPA in his bachelor thesis.

We performed 100,000 measurements with 20,000 samples per trace. We concluded after visual inspection that the traces are much clearer now. It is possible to clearly see rising edge of each clock cycle (see Figure 5.13). If we were sending the same data repeatedly we were obtaining very similar traces.

[12] Member of Embedded Security Group, Horst Görtz Institute for IT-Security, Faculty of Electr. Eng. & Information Technology, Ruhr-Universitaet Bochum

[13] Photo was taken by Martin Mašek.

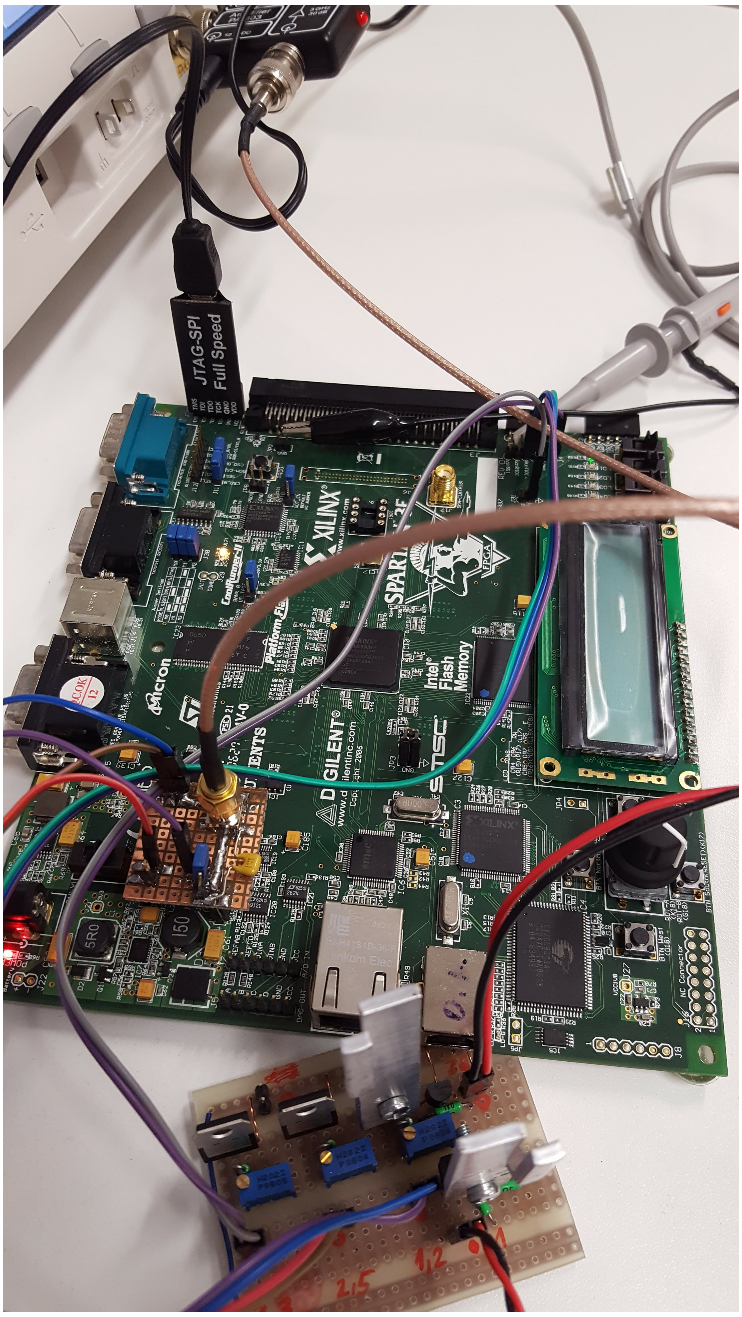

Figure 5.11: FPGA board powered by accumulators, which are connected through stabilizers. The board is connected to the oscilloscope through preamplifier.[13]

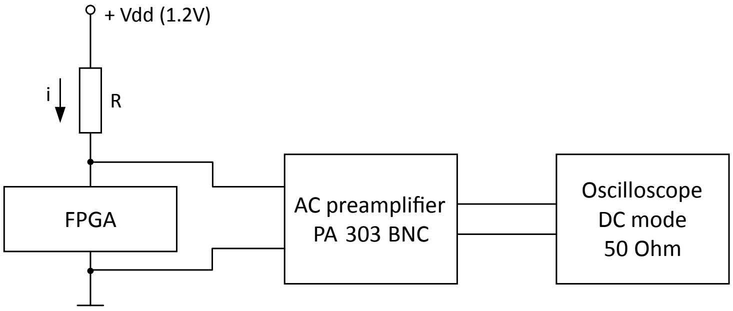

Figure 5.12: Measurement setup for FPGA with AC preamplifier

Listing 5.2: Oscilloscope settings, channel 1 was connected to the source of traces, channel 2 was connected to the trigger. Configuration: subsection 5.2.4 (AES_CLK, capacitors removed, accumulators, AC preamplifier)

```
Anlg Ch State   Units/Div    Position   Coupling
BW Limit   Invert
Ch 1:      On       360mV/    -576.00mV     DC
On         Off
Ch 2:      On       2.00V/     7.17400V     DC
Off        Off

Anlg Ch Impedance      Probe
Ch 1:      50 Ohm    1.00 : 1
Ch 2:      1M Ohm    10.0 : 1

Trigger   Mode    Coupling   Noise Rej  HF Reject   Holdoff
Edge   Normal      DC          Off         Off         60ns

Trigger Source    Slope    Level
Edge    Ch 2    Rising   +1.32V

Time   Zoom   Time Ref   Main s/div    Delay
Normal   Off    Center     1.000us/    26.92us

Acquisition  Realtime   Vectors   Inf Persist
Normal         On          On          Off
```

Environment	AC preamplifier
Board setup	Removed capacitors, accumulators
Attack	Hamming distance and last round
FPGA design	AES_CLK
Samples per trace	20,000

Table 5.6: Measurement AES_CLK, no capacitors, accumulators, AC preamplifier

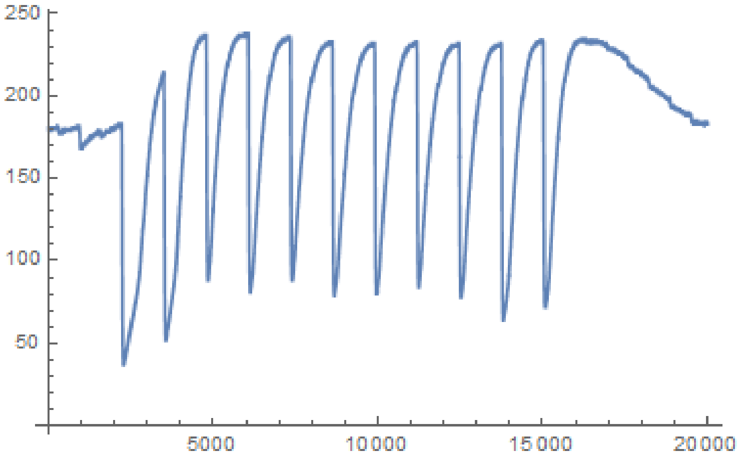

Figure 5.13: Power trace. Configuration: subsection 5.2.4 (AES_CLK, capacitors removed, accumulators, AC preamplifier)

We were able to recover correct key. The number of power traces needed for successful key recovery was 5,000. We randomly chose a few bytes of the key and all were successfully recovered. The correlation coefficient of correct key hypothesis was two times lower than correlation coefficients of other key hypotheses and its value was −0.12 (see Figure 5.14). The correlation coefficient has lowest value in sample 15,045 which is the sample where the last round starts. We concluded from this measurement that the oscilloscope and environment setup plays a key role in breaking the implementation. We used the implementation without all the simplifications we made (lower clock frequency was the only simplification we used), and we were still able to get correct key.

We observed that using more traces changes the value of the correlation coefficient of correct key hypothesis just slightly, but decreases variance of correlation coefficients of incorrect key hypotheses. We run the computation for different number of power traces and for sample 15,045. From the result, we can conclude that 1^{st} byte of key could be successfully revealed with 2,000–3,000 power traces (see Figure 5.15).

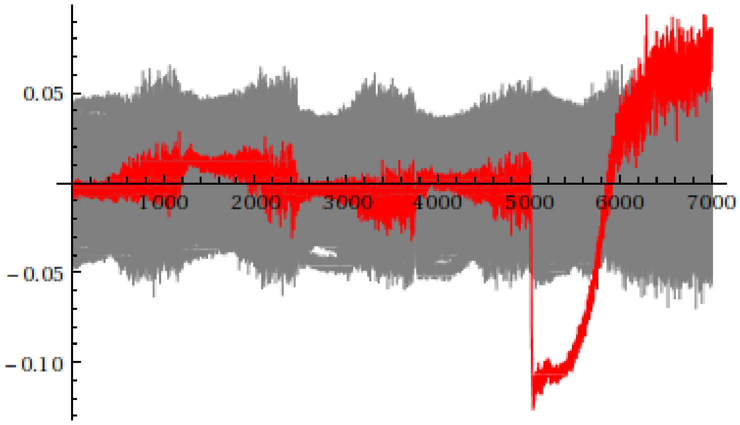

Figure 5.14: Traces of correlation coefficients (correct key in red), samples 10,000–17,000, 5,000 traces. Configuration: subsection 5.2.4 (AES_CLK, capacitors removed, accumulators, AC preamplifier)

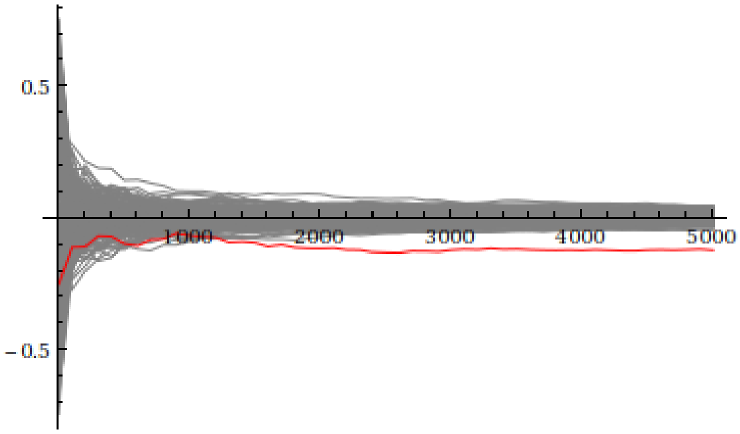

Figure 5.15: Traces of correlation coefficients for different number of power traces (correct key in red), sample 15,045. Configuration: subsection 5.2.4 (AES_CLK, capacitors removed, accumulators, AC preamplifier)

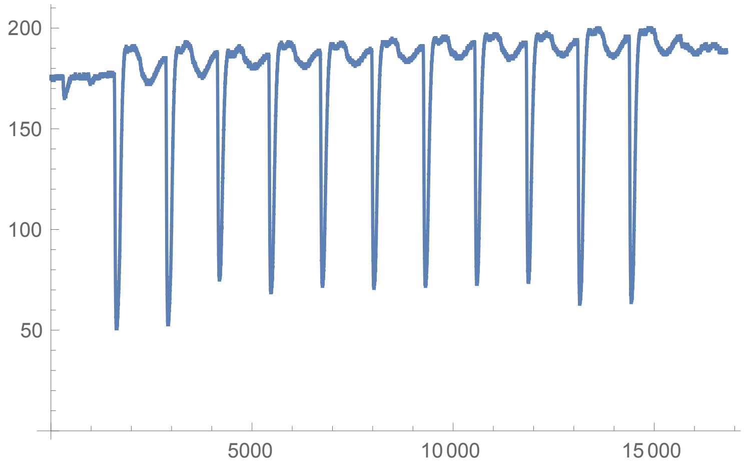

Figure 5.16: Power trace. Configuration: subsection 5.2.5 (AES_CLK, capacitors removed, switched-mode power supply, AC preamplifier)

5.2.5 AES_CLK, no capacitors, AC preamplifier

Up to this point we were modifying the setup to be able to successfully perform the DPA. Once we were able to successfully break the implementation we took a backward route. We started making the configuration to more similar to the real world one. At first we decided to power up the FPGA again from standard switched-mode power supply rather than from accumulators. The obtained traces were different even for the same plaintext. There was also a low frequency wave in the traces. The trace is shown in Figure 5.16. Summary of the setup is in Table 5.7.

Environment	AC preamplifier
Board setup	Removed capacitors
Attack	Hamming distance and last round
FPGA design	AES_CLK
Samples per trace	16,800

Table 5.7: Measurement AES_CLK, no capacitors, AC preamplifier

Oscilloscope settings were the same as in subsection 5.2.4. We performed 100,000 measurement with 16,800 samples per trace. We were able to reveal correct key with 30,000 traces (see Figure 5.17). We were able to successfully get other (a few randomly selected) bytes of a key, and all produced similar plot of correlation coefficients (see Figure 5.18 for 7^{th} byte). The correlation coefficient was 2.5–3.5 times more different than other correlation coefficients. The correlation coefficient for the first byte of a key was -0.05. We tried the computation with 5,000 traces, which was enough for the measurement with

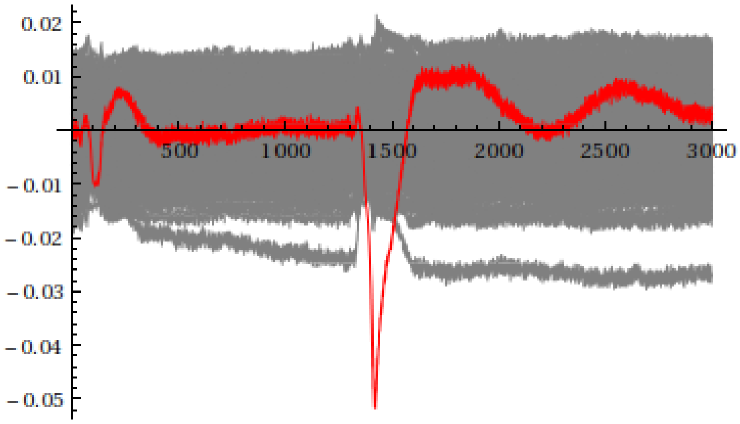

Figure 5.17: Traces of correlation coefficients (correct key in red), samples 13,000–16,000, 30,000 traces, 1^{st} byte. Configuration: subsection 5.2.5 (AES_CLK, capacitors removed, switched-mode power supply, AC preamplifier)

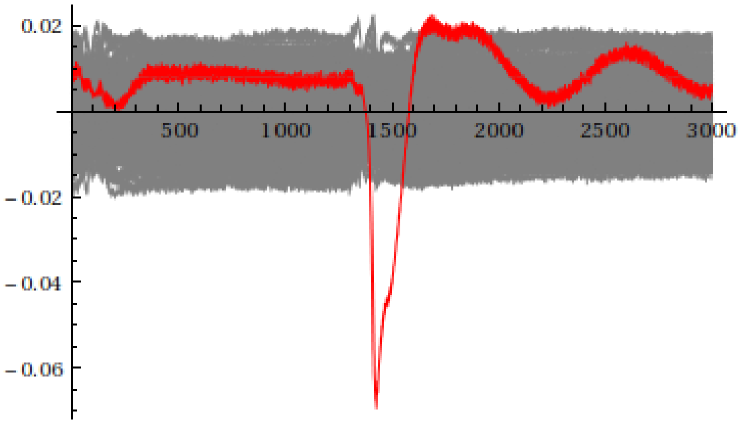

Figure 5.18: Traces of correlation coefficients (correct key in red), samples 13,000 - 16,000, 30,000 traces, 7^{th} byte. Configuration: subsection 5.2.5 (AES_CLK, capacitors removed, switched-mode power supply, AC preamplifier)

accumulators, but we were not able to get correct key in this measurement (see Figure 5.19). This can lead to a conclusion that using accumulators instead of switch-mode power supply lowers the resistance against the DPA.

5.2.6 AES_CLK, accumulators, AC preamplifier

We measured a configuration with decoupling capacitors being removed, and powered by switched-mode power supply in previous measurement. We de-

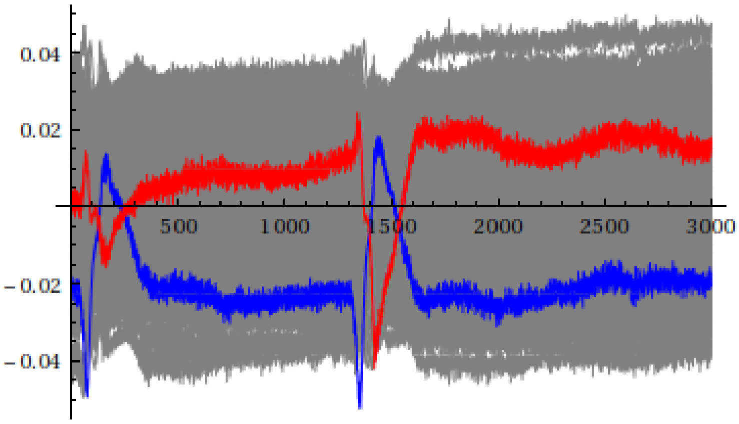

Figure 5.19: Traces of correlation coefficients (correct key in red, key hypothesis with lowest correlation coefficient in blue), samples 13,000–16,000, 5,000 traces. Configuration: subsection 5.2.5 (AES_CLK, capacitors removed, switched-mode power supply, AC preamplifier)

Environment	AC preamplifier
Board setup	Accumulators
Attack	Hamming distance and last round
FPGA design	AES_CLK
Samples per trace	16,400

Table 5.8: Measurement AES_CLK, accumulators, AC preamplifier

cided to also measure the other combination (board with decoupling capacitors being present, and powered by accumulators) and compare the results with previous two measurements.

Measurement setup is in Table 5.8. We used the same oscilloscope settings and implementation (AES_CLK) as in subsection 5.2.4, and performed 93,715 measurements with 16,400 samples per trace. Visually the traces are not as clean as the traces captured during measurement without capacitors. The traces can be seen in Figure 5.20. We were able to get the correct key with 30,000 traces (see Figure 5.21). The correlation coefficient has value −0.03, and the margin among correct key correlation coefficient and incorrect key hypotheses correlation coefficients is very small (smaller than margin in previous two measurements). We were not able to recover the correct key with 5,000 traces, and the correlation coefficient looks like a white noise in this case (see Figure 5.22). We can conclude that the figures are worse than figures in previous two measurements, and that using capacitors increases the resistance against the DPA. Correlation coefficient traces of correct key hypothesis were different from others in setup with decoupling capacitors being removed, and there was also a huge spike around the sample where the operation took place

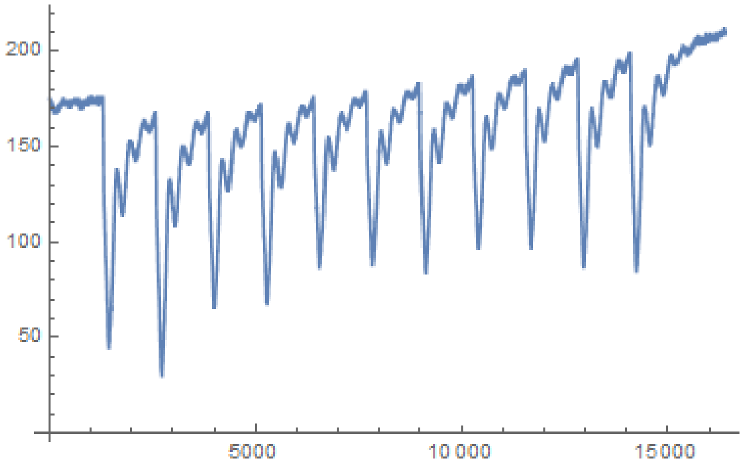

Figure 5.20: Power trace. Configuration: subsection 5.2.6 (AES_CLK, capacitors present, accumulators, AC preamplifier)

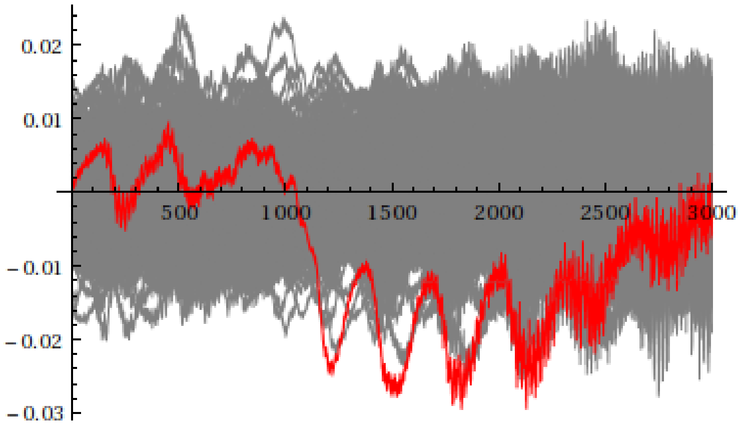

Figure 5.21: Traces of correlation coefficients (correct key in red), samples 13,000–16,000, 30,000 traces. Configuration: subsection 5.2.6 (AES_CLK, capacitors present, accumulators, AC preamplifier)

in. This spike was not present in correlation coefficient traces in setup with decoupling capacitors being present.

5.3 Summary

We performed the DPA with different implementations, board modifications, oscilloscope setups, and different methods of the attack. We discovered that the successful attack is mostly determined by the oscilloscope setup, the board setup, and the methods of the attack. The implementation has low impact

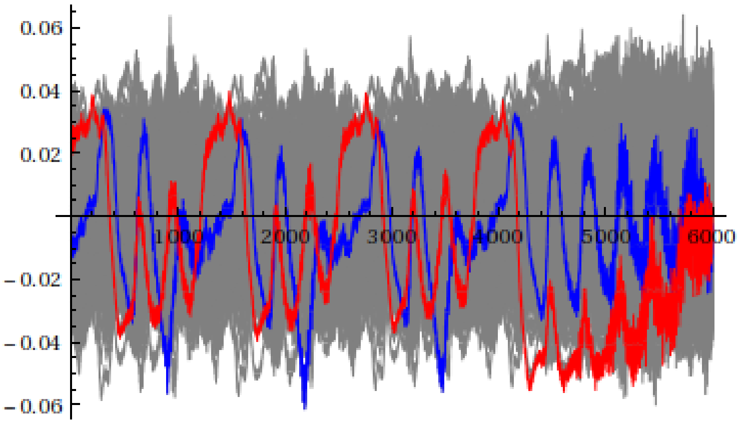

Figure 5.22: Traces of correlation coefficients (correct key in red, key hypothesis with lowest correlation coefficient in blue), samples 10,000–16,000, 5,000 traces. Configuration: subsection 5.2.6 (AES_CLK, capacitors present, accumulators, AC preamplifier)

on the success of the DPA. The most important aspect of the implementation is the clock frequency. We had better success with Hamming distance and attack to the last round than Hamming weight and attack to the first round. We also had better success with AC preamplifier (with gain 30 dB, i.e. about 31.6:1)instead of differential probe, and the bandwidth limit turned on. It is not necessary to use high sampling frequency (10,000–20,000 samples per trace is enough) because it slows down the computation and increases memory consumption of the DPA scripts. We also found out that using resistor with higher resistance gives us clearer traces.

With proper oscilloscope setup (described in subsection 5.2.4) we were able to break the implementation without any simplification (except lower clock frequency). We broke three different board setups (see Table 5.9). The board with removed capacitors powered by accumulators was the easiest to break, and we needed only 5,000 traces to find the correct key. For breaking the board powered by switched-mode power supply, and with removed capacitors we needed 30,000 traces for successful breaking. We needed 30,000 traces for successful breaking of the board with capacitors powered by accumulators as well. The margin between correct key correlation coefficient, and incorrect key

Setup	Necessary #traces
Accumulators and removed capacitors	5,000
Removed capacitors	30,000
Accumulators	30,000

Table 5.9: DPA summary

hypotheses correlation coefficients was higher in the case of a board without capacitors powered by switched-mode power supply than in the case of a board with capacitors powered by accumulators. Both removing capacitors, and using accumulators lowers the resistance against DPA. It seems from the results that removing capacitors lowers the resistance more than using accumulators.

Chapter 6

Future Work

In this chapter we propose ideas for future work and research that could be conducted. Those ideas mostly regard continuation in research of DPA against FPGA and ways how to make the key recovery and measurement process more efficient.

Impact of fault-tolerant designs to the DPA resistance

Fault-tolerance means a resistance against threats from physical and natural factors. Fault-tolerant systems should be resistant against those factors but they don't need to be necessarily resistant against intentional human attack. Basic fault-tolerance can be achieved by using some kind of redundancy such as TMR or Duplex.

Because fault-tolerant systems usually use redundant items, their security[14] can be lowered. This can have impact on a resistance against intentional attacks such as side channel attacks. Some investigation in this field has already been done [22]. The authors provided a summary of the topic but did not performed any actual comparison of the resistance of different designs. Future work could involve the comparison of the resistance in terms of number of measurements.

Efficient computation of correlation coefficient

The application for recovering the key, which we used (see section 3.2), has very high memory consumption and the computation is slow. Different techniques of efficient computation of the correlation coefficient exist [13]. The key recovery application could be changed to employ some of these techniques. This change would allow us to use much more traces.

[14] Security means the resistance against intentional attacks

Faster measurement

We have found out that performing 100,000 measurements takes approximately 2 hours with 115,200 baud serial line. We have also found out that the serial communication with FPGA is the slowest part of the measurement process.

It is possible to partially eliminate the communication. Both the measurement application and the encryption device would use a pseudo-random number generator (PRNG) or a linear-feedback shift register (LFSR). The seed for PRNG or the initial value of the LFSR would be generated before the measurement begins by one party and sent to the other party. Instead of sending each plaintext to the cryptographic device, the device could get next plaintext from PRNG or LFSR. In setup we used during measurements, the device sends all the ciphertexts back. It is necessary to have plaintext or ciphertext for each trace to successfully break the implementation [27]. But we can get all the ciphertexts even if the device won't send them back. Because the initial seed is known to both parties we can generate exactly same plaintexts and encrypt them. The cryptographic device can send back just each x^{th} trace as a confirmation that it is functioning properly. This modification would considerably decrease the serial line traffic and increase the speed of measurement.

Getting more traces

State-of-the-art research uses hundreds of millions of traces [23, 10]. The maximum number of traces we got was 100,000. We were limited mostly by inefficient key recovery application that was not constant in memory complexity with respect to the number of traces, and the slowness of measurement process. We proposed ways how to make the key recovery and memory consumption more efficient in previous sections of this chapter. With these changes we could be able to capture and use in our computations more traces. Then we could attempt to break the implementations that we weren't able to break with 100,000 traces.

Conclusion

We explored the possibilities of the application of the DPA to the FPGA implementation of cryptographic system. We were attacking the implementations of AES encryption algorithm on a smart card and a Spartan 3E Starter Board.

We modified the application SC Power Measurement used for measuring the power consumption at FIT CTU, and we added a support for serial communication (the application was originally used only for attacks on smart cards). We created scripts for performing the DPA. We created two scripts, one using Hamming weight and attacking the first round of the encryption algorithm, and the other one using Hamming distance and attacking the last round of the encryption algorithm. The scripts did not have a constant memory complexity with respect to the number of traces which made impossible to do a computations with thousands of traces.

We created AES implementations for smart card, and various AES implementations for FPGA. We decided to use the iterative design for the FPGA. Then, we modified the design several times to make it easier to break. We lowered frequency, divided the operation into multiple clock cycles and experimented with replacing the hard-coded key by a dedicated key register. All these designs were properly tested.

We performed DPA attacks against created implementations. We experimented with different board modifications (power either by switched-mode power supply or accumulators; with decoupling capacitors either present or removed), and with different oscilloscope settings. We found out that the oscilloscope setup had substantial impact on our ability to break the implementation. The FPGA design has lower impact on the ability to break the implementation. We were able to successfully break the design without nearly all the modifications we made. The attack was not successful with a differential probe, but it was successful with an AC preamplifier (with gain 30 dB), and an oscilloscope bandwidth limit turned on. The clock rate of the design in successful attack was 1,562.5 kHz. We were also observing better traces with

10 Ohm resistor than with 1 Ohm resistor.

We used two different methods of the attack. We used the Hamming weight and attacked the first round, and we used the Hamming distance and attacked the last round. The Hamming weight and first round attack was successful against a smart card. The Hamming distance and last round attack was successful against an FPGA chip.

Having proper oscilloscope settings we were able to break an FPGA implementation with different modifications of the board. We observed that the correlation coefficient is negative for the correct key hypothesis. We attributed this observation to the fact the we measure a voltage drop instead of a current. We needed 5,000 traces to break the implementation on a board with removed capacitors and powered by accumulators. We needed 30,000 traces to break the implementation on a board with removed capacitors and power by a switched-mode power supply. We were not able to break this implementation with 5,000 traces. We observed that the correlation coefficients of incorrect key hypotheses tends to have lower variance with growing number of traces. We needed 30,000 traces to break the implementation on a board with present capacitors and powered by accumulators. The margin between correlation coefficient of correct key and correlation coefficients of incorrect key hypotheses was smaller than the margin on a board with removed capacitors powered by a switched-mode power supply.

Based on the findings we can conclude that both removing capacitors and replacing a switched-mode power supply by accumulators decrease a resistance against the DPA. Removing capacitors seems to decrease the resistance more than using accumulators instead of switched-mode power supply.

We were facing the issues with high memory consumption during the DPA computations, and the issues with very slow measurements. We proposed solutions to both of these issues. The slow measurement was mostly caused by the communication between the board and the computer which we were performing measurement on. We proposed a way of partial elimination of the communication between the board and the computer.

Bibliography

[1] *BI-PNO course support material*. [cit. 2016-12-05].
URL https://edux.fit.cvut.cz/courses/BI-PNO/

[2] Hameg probes datasheet. [cit. 2017-01-07].
URL http://www.farnell.com/datasheets/1806004.pdf

[3] Langer EMV-Technik PA 303 BNC preamplifier. [cit. 2017-01-08].
URL https://www.langer-emv.de/en/product/preamplifier/37/pa-303-bnc-set-preamplifier-100-khz-up-to-3-ghz/519

[4] *MI-BHW course support material*. [cit. 2016-12-04].
URL https://edux.fit.cvut.cz/courses/MI-BHW/

[5] *Online Domain Tools*. [cit. 2016-09-15].
URL http://aes.online-domain-tools.com/

[6] Spartan-3E Starter Kit Board User Guide, 2009. [cit. 2016-11-21].
URL https://reference.digilentinc.com/_media/s3e:s3estarter_ug.pdf

[7] Agilent Technologies InfiniiVision 7000A Series Oscilloscopes. USA, 2012. [cit. 2016-12-29].
URL http://literature.cdn.keysight.com/litweb/pdf/5989-7736EN.pdf?id=1373609

[8] Basys 2 FPGA Board Reference Manaual, 2016. [cit. 2016-11-21].
URL https://reference.digilentinc.com/_media/basys2:basys2_rm.pdf

[9] Keysight Technologies InfiniiVision 3000 X-Series Oscilloscopes. USA, 2016. [cit. 2016-12-29].
URL http://literature.cdn.keysight.com/litweb/pdf/5990-6619EN.pdf?id=2002858

[10] Bilgin, Begül, Gierlichs, Benedikt, Nikova, Svetla, Nikov, Ventzislav, and Rijmen, Vincent. *Higher-Order Threshold Implementations.* In Palash Sarkar and Tetsu Iwata, editors, Advances in Cryptology – ASIACRYPT 2014, pp. 326–343. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. ISBN 978-3-662-45608-8. doi:10.1007/978-3-662-45608-8_18.

[11] Biryukov, Alex, Dunkelman, Orr, Keller, Nathan, Khovratovich, Dmitry, and Shamir, Adi. *Key Recovery Attacks of Practical Complexity on AES Variants With Up To 10 Rounds.* Cryptology ePrint Archive, Report 2009/374, 2009. [cit. 2016-10-08].
URL `http://eprint.iacr.org/2009/374`

[12] Biryukov, Alex and Khovratovich, Dmitry. *Related-key Cryptanalysis of the Full AES-192 and AES-256.* Cryptology ePrint Archive, Report 2009/317, 2009. [cit. 2016-10-08].
URL `http://eprint.iacr.org/2009/317`

[13] Bottinelli, Paul and Bos, Joppe W. *Computational Aspects of Correlation Power Analysis.* Cryptology ePrint Archive, Report 2015/260, 2015. [cit. 2016-12-04].
URL `http://eprint.iacr.org/2015/260`

[14] Buček, Jiří, Novotný, Martin, and Štěpánek, Filip. *Practical Session: Differential Power Analysis for Beginners*, 2014. [cit. 2016-12-02].
URL `https://rozvoj.fit.cvut.cz/Main/Lisbon`

[15] Caddy, Tom. *Differential Power Analysis.* In Henk C. A. van Tilborg, editor, Encyclopedia of Cryptography and Security, pp. 152–154. Springer US, 2005. ISBN 978-0-387-23483-0.

[16] Curtin, Matt. Brute force : Cracking the data encryption standard. New York: Springer/Copernicus Books, 2005. ISBN 0-387-20109-2.

[17] Daemen, Joan and Rijmen, Vincent. The design of Rijndael:AES - The Advanced Encryption Standard. Berlin Heidelberg: Springer-Verlag, 2002. ISBN 978-3-662-04722-4.

[18] Kocher, Paul, Jaffe, Joshua, and Jun, Benjamin. *Differential Power Analysis.* In Michael Wiener, editor, Advances in Cryptology – CRYPTO '99, pp. 388–397. Springer Berlin Heidelberg. ISBN 978-3-540-48405-9.

[19] Kocher, Paul, Jaffe, Joshua, Jun, Benjamin, and Rohatgi, Pankaj. *Introduction to differential power analysis.* 1:5–27. ISSN 2190-8516.

[20] Kumar, Sandeep, Paar, Christof, Pelzl, Jan, Pfeiffer, Gerd, and Schimmler, Manfred. *Breaking Ciphers with COPACOBANA – A Cost-Optimized Parallel Code Breaker.* In Mitsuru Matsui Louis Goubin, editor, Cryptographic Hardware and Embedded Systems – CHES 2006. Springer-Verlag, 2006. ISBN 3-540-46559-6.

[21] Mangard, Stefan, Oswald, Elisabeth, and Popp, Thomas. Power Analysis Attacks: Revealing the Secrets of Smart Cards. New York: Springer US, 2007. ISBN 0-387-38162-7.

[22] Miškovský, Vojtěch, Kubátová, Hana, and Novotný, Martin. *Influence of fault-tolerant design methods on differential power analysis resistance of AES cipher: Methodics and challenges.* In 2016 5th Mediterranean Conference on Embedded Computing (MECO), pp. 14–17. Institute of Electrical and Electronics Engineers, 2016. ISBN 978-1-5090-2221-2. doi:10.1109/MECO.2016.7525685.

[23] Moradi, Amir, Poschmann, Axel, Ling, San, Paar, Christof, and Wang, Huaxiong. *Pushing the Limits: A Very Compact and a Threshold Implementation of AES.* In Kenneth G. Paterson, editor, Advances in Cryptology – EUROCRYPT 2011, pp. 69–88. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. ISBN 978-3-642-20465-4. doi:10.1007/978-3-642-20465-4_6.

[24] National Institute of Standards and Technology. Announcing development of a Federal Information Processing Standard for Advanced Encryption Standard [online], 1997. [cit. 2016-10-08].
URL `http://csrc.nist.gov/archive/aes/pre-round1/aes_9701.txt`

[25] National Institute of Standards and Technology. Announcing request for candidate algorithm nominations for the Advanced Encryption Standard (AES) [online], 1997. [cit. 2016-10-08].
URL `http://csrc.nist.gov/archive/aes/pre-round1/aes_9709.htm`

[26] National Institute of Standards and Technology. FIPS PUB 197: Advanced Encryption Standard (AES). Gaithersburg: National Institute of Standards and Technology, 2001.

[27] Paar, Christof. Implementation of Cryptographic Schemes 1. Ruhr University Bochum, 2015.

[28] Paar, Christof and Pelzl, Jan. Understanding Cryptogtaphy. Berlin Heidelberg: Springer-Verlag, 2010, 2nd corrected printing ed. ISBN 978-3-642-04101-3.

[29] Patterson, David A. and Hennessy, John L. Computer Organization and Design. Oxford: Elsevier/Morgan Kaufmann, 2013, 5th ed. ISBN 978-0-12-407726-3.

[30] Prouff, Emmanuel. *DPA Attacks and S-Boxes.* In Henri Gilbert and Helena Handschuh, editors, Fast Software Encryption, pp. 424–441. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-31669-5.

[31] SciEngines. *Break DES in less than a single day [online]*, 2008. [cit. 2016-10-08].
URL `http://www.sciengines.com/company/news-a-events/74-des-in-1-day.html`

[32] Severyn, Jan. Útoky postranními kanály na implementace kryptografických algoritmů. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016. Bakalářská práce.

[33] Tillich, Stefan and Herbst, Christoph. *Attacking State-of-the-Art Software Countermeasures—A Case Study for AES.* In Elisabeth Oswald and Pankaj Rohatgi, editors, Cryptographic Hardware and Embedded Systems – CHES 2008, pp. 228–243. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-85053-3.

[34] Velegalati, Rajesh and Yalla, Panasayya S V V K. *Differential Power Analysis Attack on FPGA Implementation of AES.* [cit. 2016-12-29].
URL `https://cryptography.gmu.edu/team/download.php?docid=2082`

Appendix A

Acronyms

AES Advanced Encryption Standard.

APDU application protocol data unit.

API application programming interface.

CTU Czech Technical University.

DES Data Encryption Standard.

DPA Differential Power Analysis.

FIPS Federal Information Processing Standards.

FIT Faculty of Information Technology.

FPGA Field Programmable Gate Array.

FSM finite state machine.

LFSR linear-feedback shift register.

MCU microcontroller unit.

NIST National Institute of Standards and Technology.

PRNG pseudo-random number generator.

UART universal asynchronous receiver/transmitter.

VHDL VHSIC Hardware Description Language.

APPENDIX **B**

Contents of Enclosed DVDs

DVD 1

- `bitstreams` … the directory with AES FPGA bitstreams
- `measurements` … the directory with data obtained during measurement
  - `aes_clk_accumulators_preamplifier`
  - `aes_key_no_capacitors_accumulators_differential_probe`
  - `aes_subbytes_accumulators_differential_probe`
- `src` … source codes
  - `aes_fpga` … source codes of AES FPGA designs
  - `aes_smart_card` … source codes of AES smart card implementation
  - `key_recovery` … DPA scripts
  - `sc_power_measurement` … source codes of SC Power Measurement
    - `16_bytes` … variant sending 16 bytes of plaintext
    - `17_bytes_with_control_byte` … variant prepending data with 0xFF
  - `testing_tools` … source codes of testing tools
- `text` … the thesis text directory
  - `src` … the thesis source codes
  - `thesis.pdf` … the thesis text in PDF format
- `readme.txt` … the file with DVD contents description

DVD 2

- `measurements` … the directory with data obtained during measurement
  - `aes_clk_differential_probe`
  - `aes_clk_no_capacitors_accumulators_preamplifier`
  - `aes_clk_no_capacitors_preamplifier`
- `readme.txt` … the file with DVD contents description