Universiteit Leiden

How can the testing setup for the PROACT SoC be optimized?

Nick Stijger (s2722836)

Supervisors:
Prof.dr. N. Mentens & Mr. A. Sajadi

Bachelor Thesis– Informatica

Leiden Institute of Advanced Computer Science (LIACS)
liacs.leidenuniv.nl

August 25, 2026

Abstract

The PROACT research project aims to improve the physical security of IoT devices against side-channel and fault injection attacks. Developing new prototypes requires repetitive, time-consuming power-trace acquisition. This thesis presents a test automation toolkit that optimizes the experiment workflow for the PROACT FPGA development board, reducing the time needed to program the controlling software to an already configured FPGA by more than 18x (6:18.66 $\rightarrow$ 0:20.92) and reducing the firmware size to 0.035% of the original size (87.5MB $\rightarrow$ 30.5KB). Using this improved toolkit, 60,000 power traces from three different AES implementations are collected: two hardware AES cores (LUT S-Box and composite field) and one software implementation running on a 32-bit RISC-V co-processor. The 20,000 traces for each implementation consist of 5000 samples, a random plaintext and a trigger count, which is used to trim the traces and exclude core idle time after encryption is completed. These traces are then used to perform a CPA on the AES implementations. The full 128-bit AES key is successfully recovered across all target implementations with the usage of the S-Box Output and Last Round State Difference leakage models withs MTDs of 5800, 7100 and 4000 respectively. This demonstrates the effectiveness of the developed toolkit and the vulnerability of the AES cores to side-channel attacks.

Acknowledgements

This bachelor thesis was conducted as part of the PROACT project at the Leiden Institute of Advanced Computer Science (LIACS).

I would like to express my gratitude to my co-supervisor, PhD candidate Mr. A. Sajadi, for his invaluable guidance, extraordinary support and patience throughout the research and writing process. His expertise and insights have been crucial in shaping the direction of this thesis. Without his extensive time investment and dedication, this work would not have been possible.

I am also grateful to my main supervisor, Prof.dr. N. Mentens, for providing me with the flexibility and understanding needed to manage my schedule and commit to this project.

Finally, I would like to thank my family and friends for their unwavering support and encouragement throughout my academic journey. Their belief in my abilities has been a constant source of motivation.

Contents

List of Figures

List of Tables

List of Abbreviations

AEAD Authenticated Encryption with Associated Data. 10, 14

AES Advanced Encryption Standard. 2, 3, 6, 10–13, 18, 19, 21, 22, 27, 29, 30

API Application Programming Interface. 14

ASIC Application-Specific Integrated Circuit. 5, 29, 30

CLI Command Line Interface. 1, 2, 13, 16, 29

CPA Correlation Power Analysis. 1–4, 7, 13, 17–19, 29, 30

DPA Differential Power Analysis. 3, 7

FPGA Field Programmable Gate Array. 1, 3, 5, 10, 13, 14, 16

GUI Graphical User Interface. 1, 2, 13, 16, 19, 21, 29, 30

HD Hamming Distance. 6

HW Hamming Weight. 6, 18

IoT Internet of Things. 1, 3

LUT Look-Up Table. 18

MDS Maximum Distance Separable. 8

MTD Measurements to disclosure. 7, 22, 27, 30

NIST National Institute of Standards and Technology. 10

PoC Proof of Concept. 16, 19, 21

PROACT Physical Attack Resistance of Cryptographic Algorithms and Circuits with Reduced Time to Market. 1–3, 5, 10–16, 29, 30

S-Box Substitution Box. 7, 18, 22

SAD Sum of Absolute Differences. 18, 19

SCA Side-Channel Analysis. 6, 30

SNR Signal-to-Noise Ratio. 5

SoC System on Chip. 1–3, 5, 10–14, 29, 30

SPI Serial Peripheral Interface. 1, 12, 13, 16

UART Universal Asynchronous Receiver-Transmitter. 12, 13, 16

1 Introduction

Every day, more and more data collecting devices are connected to the internet, which is called the Internet of Things (IoT). These are devices, such as smart watches and smart home devices that collect very sensitive data, like personal, health and business information. With more than 19.8 billion connected devices in 2025 and an expected rise to 40.6 billion by 2034 [13], the need for security keeps growing. Meanwhile, the cost of these devices is kept competitive, which means that security is often not a priority.

These devices are often physically accessible to an attacker, which makes them vulnerable to physical attacks in addition to digital attacks. Physical attacks such as fault injection and side-channel analysis can be used to extract sensitive data from these devices, including cryptographic keys that can be used to decrypt previously collected data or to inject malicious data into the system.

To combat these physical attacks, the Physical Attack Resistance of Cryptographic Algorithms and Circuits with Reduced Time to Market (PROACT) [1] investigates how IoT devices can be protected better against these physical attacks. This project aims to develop new algorithms and silicon chips, which are more resistant against these physical attacks. In order to create such new algorithms and chips, existing chips must be analyzed for vulnerabilities, which requires substantial data to be collected from these chips. For this reason, the PROACT project developed a toolkit consisting of both hardware and software, which can be used to collect the required data.

Unfortunately, the existing toolkit makes the process to collect the necessary amount of data very time consuming. In order to setup an experiment, a Field Programmable Gate Array (FPGA) board needs to be flashed with the PROACT System on Chip (SoC) via a Jupyter notebook every time the board is reconnected to the power supply. After this, the main core of the PROACT SoC needs to be programmed over a Serial Peripheral Interface (SPI) with the controlling software specific to the experiment. This binary consists of 87.5 MB of data and takes more than 6 minutes to program (see Table 5). This results in a very time consuming process, which consists of mostly waiting for the programming to finish, during which no power traces can be collected. Furthermore, in order to effectively analyze the vulnerabilities of a chip, multiple rounds of experiments must be performed, wherein the setup needs to be repeated for every experiment. Not only are these rounds time consuming, they are also prone to human error due to the different manual steps needed to setup the experiments.

In this thesis, we present a new toolkit that improves the efficiency of the PROACT testing setup. This toolkit consists of a Graphical User Interface (GUI), Command Line Interface (CLI) and a C library used as a backend on the FPGA. The GUI and CLI can automatically connect to the FPGA and program both the PROACT SoC and the controlling software. The C library can be used to set up multiple experiments, either sequentially in a single run or across multiple runs, without reprogramming the controlling software. This combination eliminates multiple rounds of manual programming, improving the efficiency of the PROACT testing setup and reducing the chance of human error.

This toolkit is used to collect power traces from the different AES implementations on the PROACT SoC. These traces are then analyzed using Correlation Power Analysis (CPA) to recover the key.

1.1 Contributions

The contributions of this thesis are: (1) a command-driven firmware and C-library for the PROACT SoC that removes the need for flashing the controlling software between every experiment; (2) a new GUI and CLI that can be used to program the developed firmware and the PROACT SoC in a single workflow, enabling the setup of multiple experiments without manual intervention; (3) a CPA evaluation of three different Advanced Encryption Standard (AES) implementations on the PROACT SoC using two different leakage models; and (4) the first reported measurements of AES emulation on the PROACT SoC using the software RISC-V core.

1.2 Research questions

- How can the PROACT testing setup be improved to efficiently collect power traces from the PROACT SoC?
- How can the experiment setup be simplified, such that the manual steps needed to setup an experiment are reduced and more fault-tolerant?
- Does the new PROACT testing setup allow for efficient collection of power traces from the different AES implementations on the PROACT SoC?

1.3 Thesis overview

Section 2 gives an overview of the related work, which is done in the PROACT project and by other researchers in the field of side-channel analysis. In Section 3 a detailed background is given on the needed knowledge to understand the rest of this thesis. Section 4 explains the methodology used to answer the research questions. In Section 5 the results of the experiments are presented and analyzed. Finally, in Section 6 a conclusion is drawn and future work is discussed.

2 Related work

This thesis is part of the PROACT project [1], which is a research project that aims to improve the physical security of IoT devices, while minimizing the time to market. The PROACT project is a collaboration between the Leiden Institute of Advanced Computer Science (LIACS), the Radboud University, Nijmegen and Riscure, Delft and is funded by the Dutch Research Council (NWO).

The research and development work in the PROACT project is divided across four work packages, each having their own specific goals. The first work package (WP1), is to create a feedback loop from the other work packages to guide the design and implementation of cryptographic algorithms. The goal of the second work package (WP2) is to create the PROACT SoC, of which the first version in this thesis is implemented on a FPGA board.

The main goal of the third work package (WP3) is to build an automated AI-driven simulation framework that predicts side-channel leakage and fault injection vulnerabilities in cryptographic circuits before the chip is physically manufactured. The main goal of the fourth work package (WP4) is to evaluate, validate and test the physical security of the developed cryptographic algorithms and circuits. This thesis contributes to the fourth work package, by improving the efficiency of trace collection by developing a new toolkit that allows efficient collection of traces.

As part of the research for the PROACT project, different countermeasures against side-channel attacks on AES implementations on a RISC-V Ibex core were evaluated [10]. These countermeasures included masking (Mask-AES), noise generation (HW-NG and SW-NG), secret sharing (CoCo-Ibex), and dummy instructions (Secure-Ibex). Evaluating their effectiveness using CPA required acquiring extensive power traces, a process that highlighted the need for more streamlined acquisition methods and directly motivated the development of the toolkit presented in this thesis.

While these core-level countermeasures focus on direct hardware and software defenses for specific implementations, securing RISC-V architectures requires a broader approach. As discussed in [2], evaluating the security, testability and safety of RISC-V cores over its entire lifecycle is crucial for building safe open-source systems. This includes not only implementing countermeasures but also ensuring that the design and testing processes are robust and comprehensive. The research to the RISC-V cores on the PROACT SoC contributes to this broader understanding of RISC-V security, particularly in the context of side-channel vulnerabilities.

The necessity for streamlined power trace acquisition already became apparent during the research of Kocher et al. [6], which introduced the concept of Differential Power Analysis (DPA), demonstrating that statistical analysis of power consumption could extract secret keys without physical tampering. This work laid the foundation for subsequent advancements in side-channel analysis, including the development of CPA by Brier et al. [3]. CPA further refined the concept by employing Pearson's correlation coefficient. This enhancement allowed for more efficient key extraction, even in the presence of noise.

In depth explanation for both methods and corresponding countermeasures, can be found in the book by Mangard et al. [7].

To operationalize these side-channel analysis techniques, various tools have been developed, such as ChipWhisperer [9], an open source platform that provides both hardware and software for side-channel power analysis. ChipWhisperer has been instrumental in facilitating the research in

this thesis, providing a Python framework for implementing the CPA and a hardware platform for collecting power traces. The toolkit developed in this thesis builds upon the capabilities of ChipWhisperer, enhancing the efficiency of trace collection and experiment setup.

3 Background

This section explains the relevant background information needed to understand the rest of this thesis.

3.1 FPGA vs. ASIC

In this thesis, the PROACT SoC is implemented on a FPGA board. A FPGA is a reprogrammable chip, which can be used to implement different hardware designs. It consists of a large amount of logic blocks, like Look-Up Tables (LUTs) and flip-flops, which can be connected via the onboard routing the block RAM and to each other. In this thesis, the FPGA is described by the hardware description language Verilog, which is used to describe the hardware design of the PROACT SoC to the FPGA. This allows for a quick and easy way to implement and test new hardware designs, without needing to manufacture a new chip for every design.

To allow fast and easy testing of new designs, the PROACT SoC is prototyped on a FPGA board. This quick and simple prototyping is what allows the reduced time to market for the PROACT project, which is one of the main goals of the project. Crucially, a FPGA does not behave the same as an Application-Specific Integrated Circuit (ASIC). The PROACT SoC is nowadays also manufactured as an ASIC. An ASIC is a non-reprogrammable chip designed for a specific purpose (such as the PROACT project). Because each logic block in an ASIC chip is designed for a specific purpose, the area of the chip is used much more efficiently than in a FPGA, which allows for a higher clock frequency and lower power consumption. The downside of an ASIC chip is that is much more expensive to manufacture and time consuming to test, since a new chip needs to be manufactured for every design change.

The difference in onboard logic results in a different leakage magnitude and Signal-to-Noise Ratio (SNR) between the FPGA and ASIC. The power traces collected from the FPGA are thus not directly comparable to the power traces collected from an ASIC. Although these results are not directly comparable, the relative ordering of the different AES implementations is expected to be the same, since the base logic of the cores is the same. It is important to note that the used FPGA board is a board specifically designed for side-channel analysis. The board therefore has clean power rails, a low noise supply and an on-board power shunt, which allows for very clean power traces.

During the research period of this thesis, the ASIC of the PROACT SoC was not yet delivered, which is why only the FPGA implementation is analyzed in this thesis.

3.2 Power side-channel analysis and CPA

Every computer system consumes power when it is executing a program. This power consumption does not only come from logic switching, but also from all the data that needs to be written and stored. A computer stores data in memory components, like CMOS. When a CMOS cell is set from 0 to 1, the cell needs to be charged, which consumes power. When a CMOS cell is set from 1 to 0, the cell needs to be discharged, which also consumes power. This means that every transition of a bit directly correlates with the power consumption of the system.

3.2.1 Leakage

This power consumption can be measured, which can be used to analyze the data that is being processed by the system. This security evaluation method is called Side-Channel Analysis (SCA). In order to compare this measured power consumption with the data being processed, a leakage model is used. This leakage model is a mathematical model that describes the relationship between the data being processed and the power consumption of the system. The most common leakage models, which form the base of more complex models, are Hamming Weight (HW) and Hamming Distance (HD).

3.2.2 Hamming Weight & Hamming Distance

The HW leakage model is based on the number of bits that are set to 1 in a binary string. The more bits are set to 1, the more power is needed to drive a value on a precharged bus or to set a value in a register. This means that the power consumption of a system is directly correlated with the HW of the data being processed.

The HD leakage model is based on the number of bits that differ between a previous value and a current value. The more bits differ, the greater the expected change in power consumption. The HD is calculated by taking the XOR of the two values and then computing its Hamming weight.

$$\mathrm{HD}(v_{old}, v_{new}) = \mathrm{HW}(v_{old} \oplus v_{new})$$

3.2.3 Divide and conquer

In this thesis, 128-bit AES keys are used, which can be one of the $2^{128} = 3.4 \times 10^{38}$ possible keys. This number is far too large to attack, which is why a divide and conquer approach is used. These 128-bit keys consist of 16 bytes, which can be attacked independently. This reduces the search space to $16 \times 2^8 = 16 \times 256 = 4096$ possible keys, which is a feasible number to attack. The exact method of splitting these 16 bytes is explained in Section 4.6.

3.2.4 Pearson's correlation coefficient

These expected power consumptions are then compared to the measured power consumption by using Pearson's correlation coefficient. When a correct key is used, the expected power consumption should correlate with the measured power consumption, which results in a high correlation coefficient. This coefficient is calculated by the following formula, where $\rho_{k,t}$ is the correlation coefficient for key hypothesis k at sample point t over N traces.

$$\rho_{k,t} = \frac{\sum_{i=1}^{N}(h_{i,k} - \bar{h_k})(l_{i,t} - \bar{l_t})}{\sqrt{\sum_{i=1}^{N}(h_{i,k} - \bar{h_k})^2}\sqrt{\sum_{i=1}^{N}(l_{i,t} - \bar{l_t})^2}}$$

$h_{i,k}$ is the hypothetical power consumption for trace i and key hypothesis k, which is calculated by the leakage model. $\bar{h_k}$ is the mean of the hypothetical power consumption for key hypothesis k over all traces. $l_{i,t}$ is the measured power consumption for trace i at sample point t. $\bar{l_t}$ is the mean of the measured power consumption at sample point t over all traces.

The correct subkey can be recovered by taking $\hat{k} = \arg\max_k \max_t |\rho_{k,t}|$.

3.2.5 CPA

The combination of the leakage model and the correlation coefficient is called CPA [3]. CPA is a successor of DPA [6], which is a simpler method of analyzing the power consumption of a system. DPA splits the traces into two groups, based on the value of a specific bit in the hypothetical power consumption. CPA improves this method by using Pearson's correlation coefficient, which allows extracting the correct key with fewer traces while allowing more noise. This makes CPA a more effective method of analyzing the power consumption of a system.

3.2.6 Evaluation metrics

In order to evaluate the effectiveness of the CPA, the following metrics are used:

- *Key Rank*: The position of the correct subkey, when all 256 possibilities are sorted by $\max_t |\rho_{k,t}|$
- *Measurements to disclosure (MTD)*: The smallest number of measurements needed, such that each of the 16 subkeys stays at a key rank of 1.

3.3 AES Algorithm Structure and Attack Surfaces

The Advanced Encryption Standard (AES)[5, 8] is a symmetric-key algorithm, which can be implemented efficiently on both hardware as well as on software. These symmetric keys come in different sizes although only the 128-bit version is analyzed in this thesis.

An AES transformation from plaintext to ciphertext consists of an initial round key addition, followed by 9 full rounds, followed by a final round, which excludes the MixColumns step. These rounds consist of four steps: SubBytes, ShiftRows, MixColumns and AddRoundKey, which are explained in the following subsections. The order of these steps is shown in Figure 1.

3.3.1 SubBytes

In the byte substitution step, each byte is mapped to another byte using a substitution table (S-box). Since this mapping is a bijection, the operation can be reversed using the inverse mapping. To provide resistance against linear and differential cryptanalysis, the Substitution Box (S-Box) is designed to achieve a low maximum linear bias and a low maximum differential probability. The low linear bias hinders the discovery of linear approximations, while the low differential probability prevents specific input-to-output difference propagation. Consequently, a small change in the input produces a significant change in the output—a property known as diffusion—which minimizes correlations between input and output.

3.3.2 ShiftRows

In the row shift step, the 16-byte state array is arranged into a 4×4 grid. Each row is then cyclically shifted to the left by an offset equal to its row index: Row 0 remains unshifted, Row 1 is

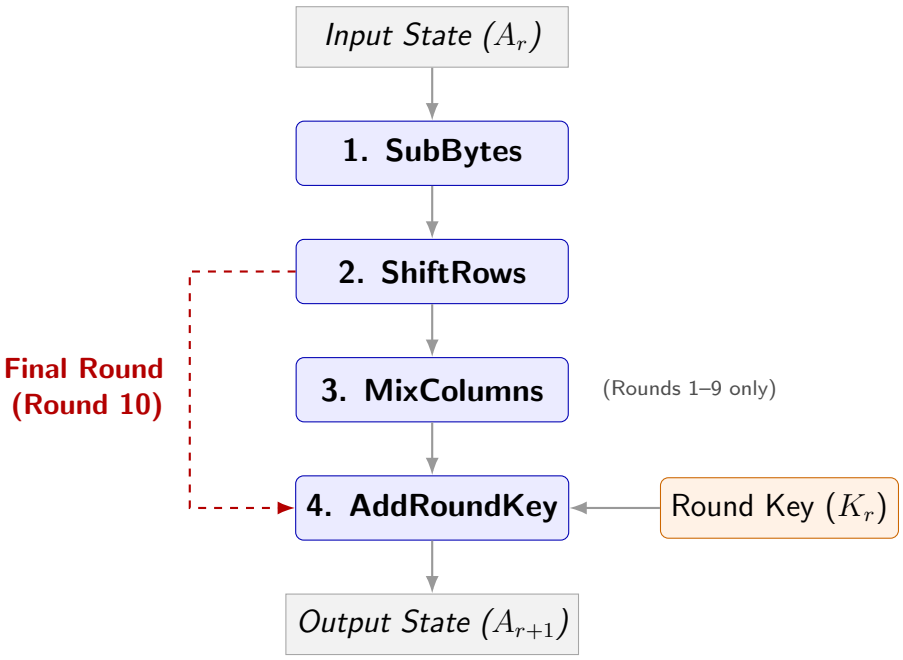

Figure 1: AES Round Structure

shifted left by 1 byte, Row 2 by 2 bytes, and Row 3 by 3 bytes.

$$\begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} \mapsto \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \end{bmatrix}$$

3.3.3 MixColumns

In the column mix step each column of the previous matrix is multiplied by the following matrix:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Because this matrix has a non-zero determinant in the Galois Field $GF(2^8)$, the matrix is invertible, which in turn makes this Mix Columns step reversible. The transformation is optimized for Maximum Distance Separable (MDS) to provide optimal diffusion. This results in that each output byte is dependent on all input bytes, while still keeping the number of operations low. To maintain this performance, the matrix only uses the field elements 1, 2 and 3. Because these operations take place in $GF(2^8)$ modulo the reduction polynomial $x^8 + x^4 + x^3 + x + 1$, these operations can be implemented efficiently. These operations can be implemented as follows:

- $1 \cdot s = s$
- $2 \cdot s = (s << 1) \oplus (0x1B \text{ if } s[7] = 1 \text{ else } 0x00)$

- $3 \cdot s = 2 \cdot s \oplus s$

The optional XOR with $0x1B$ is needed to reduce the result modulo the reduction polynomial, which is only needed when the most significant bit of the input is set and would result in a 9-bit number after the left shift.

3.3.4 AddRoundKey

In the Add Round Key step, the current 128-bit state is combined with a 128-bit round key using a bitwise XOR operation. Considering the XOR operation is its own inverse, this step is reversible by applying the same round key again. Each of these round keys is derived from the original 128-bit key using key expansion.

Key expansion is a process that takes the original key and generates the round keys without needing any additional input. In AES-128, 11 round keys are used: the initial key and one key for each of the 10 rounds. The key expansion starts by splitting the 128-bit key into 4 words (w_0, w_1, w_2, w_3) of 32 bits each. Then, for each round, a new word is generated by taking the previous word and XORing it with the word 4 positions earlier.

For every fourth word, the previous word is first rotated one byte to the left, then each byte is substituted using the S-box and finally the first byte is XORed with a round constant. This round constant is different for each round and is derived from the Galois field $GF(2^8)$.

$$\text{RotWord}(b_0, b_1, b_2, b_3) = (b_1, b_2, b_3, b_0)$$

$$\text{SubWord}(b_0, b_1, b_2, b_3) = (S(b_0), S(b_1), S(b_2), S(b_3))$$

$$\text{Rcon}_i = \begin{cases} \text{0x01} & \text{if } i = 1 \\ \text{0x02} \cdot \text{Rcon}_{i-1} & \text{if } i > 1 \text{ and } \text{Rcon}_{i-1} < \text{0x80} \\ (\text{0x02} \cdot \text{Rcon}_{i-1}) \oplus \text{0x1B} & \text{if } i > 1 \text{ and } \text{Rcon}_{i-1} \geq \text{0x80} \end{cases}$$

$$w_i = \begin{cases} w_{i-4} \oplus \text{SubWord}(\text{RotWord}(w_{i-1})) \oplus \text{Rcon}_{i/4} & \text{if } i \mod 4 = 0 \\ w_{i-4} \oplus w_{i-1} & \text{otherwise} \end{cases}$$

Because every operation in the key expansion is reversible, the original key can be derived from the round keys. This means that if an attacker is able to extract one of the round keys, the original key can be derived from this round key.

3.3.5 Key Attack Surfaces and Intermediate States

Two intermediate states are of primary interest to an attacker. In round 1, the output of the SubBytes step only depends on the input plaintext and the master key and do not have any bytes mixed. This renders the output of the SubBytes step in round 1 an instinctive target for a known plaintext attack. Conversely, the absence of the MixColumns step in the last round, restricts each ciphertext byte's dependency to exactly one byte of K_{10}, the round key for round 10. This makes this the ideal target for a known ciphertext attack. The vulnerability of these two intermediate states is further discussed in Section 4.6.

3.4 ASCON and Xoodyak

Two other algorithms present on the PROACT SoC are ASCON and Xoodyak. These algorithms are both lightweight Authenticated Encryption with Associated Data (AEAD) schemes.

ASCON [14] is selected by National Institute of Standards and Technology (NIST) as the official standard for lightweight authenticated encryption [14], providing both confidentiality and data integrity using a 320-bit internal state.

Xoodyak [4] is another NIST finalist for AEAD schemes, alongside hashing, pseudorandom number generation and MAC computation. It uses the Xoodoo permutation, which is a 384-bit state permutation specifically designed for lightweight cryptography on microcontrollers.

While traditional algorithms like AES are designed solely on confidentiality, AEAD schemes also focus on data integrity and authenticity. This means that these algorithms not only encrypt the data, but also provide a way to verify that the data has not been tampered with. This is done by generating a tag, which is sent alongside the ciphertext. In AEAD schemes, the encryption process takes four distinct inputs:

- *Key*: A secret symmetric key used for both encryption and decryption. This key is shared between the sender and the receiver.
- *Nonce*: A unique value that is used only once for encryption. This ensures that a same plaintext encrypted multiple times with the same key will produce different ciphertexts.
- *Plaintext*: The original data that needs to be encrypted.
- *Associated Data*: Additional data that is not encrypted but is authenticated. This data could contain information like headers or metadata, which needs to be verified for integrity but does not need to be kept secret.

During decryption, the receiver validates the authentication tag before revealing the plaintext. If the tag does not match, the receiver knows that the message has been tampered with and will reject the message.

While these algorithms are present on the PROACT SoC and are supported by the developed toolkit, they are not analyzed in this thesis; only the different AES implementations are attacked.

3.5 The PROACT SoC

The PROACT SoC consists of a main Ibex RISC-V CPU acting as a controller for the co-processors: Another Ibex RISC-V CPU, two AES cores, one ASCON core and one Xoodyak core. The main core controls the other cores over a shared system bus. The main core is connected to a controller consisting of a Serial Peripheral Interface (SPI) and a Universal Asynchronous Receiver-Transmitter (UART). The main core controls the other cores by setting the control register, which is read by the other cores. These cores communicate back by setting the status register, which is read by the main core.

The FPGA used for the implementation of the PROACT SoC is a CW305 Artix-7 board. This target runs at a clock frequency of 50 MHz, which is the maximum frequency that this FPGA can run at. The PROACT SoC is implemented on the FPGA using the Verilog hardware description language.

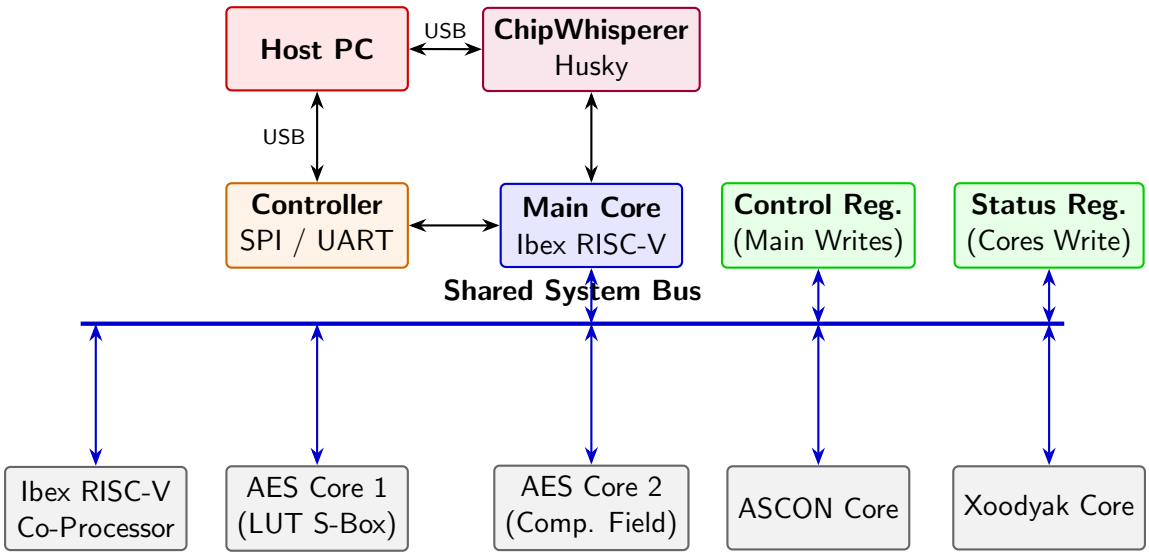

Figure 2: PROACT SoC Architecture.

3.5.1 Co-Processors: AES1 & AES2

On the PROACT SoC are two different AES-128 cores: The first AES core is a lookup table substitution box (LUT S-Box) [5] implementation. The other AES core is a composite field [12] implementation. These cores support both encryption and decryption at a latency of 11 clock cycles per 128-bit block.

Both cores have 3 bits on the control register: Enable, Start and Decrypt, controlling the state of the core. The core communicates back to the main core with 3 bits in the status register: Done, Trigger and Active. A loop consists of the following states: clear, reset, load, start and execute. These states with corresponding register values are shown in Table 1

State	Control register (E, S, D)	Status register (D, T, A)
Clear	1-0-x	x-x-1
Reset	0-0-x	x-0-0
Load	1-0-x	0-0-1
Start	1-1-x	0-1-1
Execute	1-1-x	1-0-1

Table 1: AES state register values [11].
Control bits: **E**nable, **S**tart, **D**ecrypt. Status bits: **D**one, **T**rigger, **A**ctive. (x = don't care)

In Table 2 the memory offsets are shown, which are needed by the main core to communicate with the AES cores. As seen in the table, the AES cores have room for 16 bytes of key, 16 bytes of data and 16 bytes of result. This means the AES cores can only encrypt one block of data at a time, which is exactly 16 bytes in AES-128.

Offset	Register
0x00	START
0x04	DONE
0x08–0x14	KEY
0x18–0x24	DATA
0x28–0x34	RESULT

Table 2: AES memory offsets [11]

Offset	Register	Notes
0x00	LEN	
0x04	KEY	4×4 bytes $\rightarrow$ 16 bytes FIFO
0x08	NPUB	4×4 bytes $\rightarrow$ 16 bytes FIFO
0x0C	AD	write FIFO
0x10	PT	write FIFO
0x14	CT	read FIFO
0x18	TAG	read FIFO

Table 3: ASCON and Xoodyak memory offsets [11]

3.5.2 ASCON and Xoodyak

Two other cores present on the PROACT SoC are an ASCON [14] core and a Xoodyak [4] core. These cores only support encryption; decryption needs to be done using software.

In Table 3 the memory offsets are specified, used for communication between the main core and the ASCON and Xoodyak cores.

3.5.3 Sw-RV

The last core present on the PROACT SoC is another Ibex RISC-V core. This core is used for software emulations, like AES. This core can not be accessed directly by the host and thus needs to be programmed via the main core. In this thesis, this core runs a portable byte-oriented tiny-AES-c implementation.

3.5.4 Controller

The PROACT SoC is controllable by two connections: by SPI for loading the code and resetting the chip and by Universal Asynchronous Receiver-Transmitter (UART), for communicating with the running program on the main core.

While the main core is active, data can be sent to the UART, which will set status register bit 0. The controller can check this bit and then read the UART. This way, the main controller can be programmed to execute commands without needing to change the firmware between every experiment.

4 Methodology

The aim of this thesis is to improve the efficiency of the PROACT testing setup, which is done by creating a new GUI to control the PROACT SoC. Then the new GUI is used to collect power traces from the different AES implementations. On these traces, CPA is performed to find the used key. This section explains the methodology used to achieve this.

4.1 The Requirements and the Baseline Workflow

In order to successfully perform the experiments, the following things are needed:

4.1.1 Requirements

- A ChipWhisperer Husky, which is used to collect the power traces.
- A CW305 Artix-7 FPGA board, which is used to implement the PROACT SoC.
- A host computer, which is used to control the ChipWhisperer and the PROACT SoC.
- The developed PROACT toolkit
- Cables to connect the ChipWhisperer, the FPGA and the host computer.

After the requirements are met, the following workflow is used to perform the experiments:

4.1.2 Baseline Workflow

1. Set up the ChipWhisperer Husky and FPGA, and connect the probe to the ChipWhisperer and to the shunt resistor on the Husky.
2. Connect the UART and SPI to the host by USB.
3. Connect the ChipWhisperer to the host by USB.
4. Connect the power supply to the FPGA.
5. Flash the FPGA with the PROACT SoC by using the GUI or the CLI.
6. Program the main core with the code to control the co-processors by using the GUI or the CLI.
7. Program the Sw-RV co-processor to allow software emulation of AES by using the GUI or the CLI.
8. Run one or multiple experiments by using the GUI or the CLI.
9. Analyze the collected traces

4.2 Mapping the PROACT Toolkit to the SoC Architecture

The PROACT toolkit is designed to control the PROACT SoC and acts like an Application Programming Interface (API); it provides a set of tools to setup different experiments without needing to reflash the FPGA or reprogram the cores. An experiment that is requested from the host is translated to a set of commands that are sent to the main core, which then controls the co-processors. This results in the following steps on the PROACT SoC:

4.2.1 Reset

The reset step is used to reset the co-processor to a known state. This is done by first clearing the co-processor by setting the enable bit to 1 and the start bit to 0. After the co-processor is cleared, it is reset by setting the enable bit to 0. This combination will reset the internal state of the co-processor and set the trigger and active register to 0.

4.2.2 Load

After the co-processor is reset, it can be set to loading mode. This is done by setting the enable bit to 1 and the start bit to 0. When applicable, the decryption bit is set to 1 to indicate that the co-processor should decrypt the data instead of encrypting it. After this, the active bit in the status register is set to 1, indicating that the co-processor is ready to be loaded with the key and data. In the case of the AEAD cores, the nonce and associated data are also loaded. After all the data is written, the co-processor is ready to be started.

4.2.3 Start and Execution

In order to start the co-processor, the start bit is set to 1, which will start the execution of the experiment and raise the trigger bit to 1. On this rise, the ChipWhisperer starts the set amount of samples. After the co-processor is done, the done bit is set to 1 and the trigger bit is dropped to 0.

On completion of a round, the steps are repeated until the set amount of traces and thus rounds are completed. Then all the collected traces are saved to a file, which can be used for further analysis.

4.3 The C Library

The API on the host is implemented in C and provides a set of functions to control the PROACT SoC. Counted with `cloc`, this library consists of 972 lines of code and compiles to a ≈ 29.86 kB library. The host-side C library API is listed in Appendix A.

An example of how to use the C library to perform a single AES encryption experiment is shown in Listing 1.

Listing 1: Example of a single AES encryption using the C library

```
#include "proact_aes.h"

void demo_aes1_encrypt_once(void) {
  static const uint32_t key[4] = {
```

```
    0xabcdef01, 0x12345678, 0xdeadbeef, 0x87654321
  };
  static const uint32_t pt[4] = {
    0x12345678, 0xabcdef01, 0x87654321, 0xdeadbeef
  };
  uint32_t ct[4];

  aes_run(PROACT_AES1, key, pt, ct, /*decrypt=*/0, /*verbose=*/0);

  /* ct now contains the 128-bit ciphertext */
  /* e.g., send it over UART or compare against a KAT vector */
}
```

This example shows that every experiment can be performed with a single function call, which takes care of all the steps needed to perform the experiment. This allows for easy integration of the PROACT toolkit into other software without needing to set every register described in Tables 1, 2 & 3.

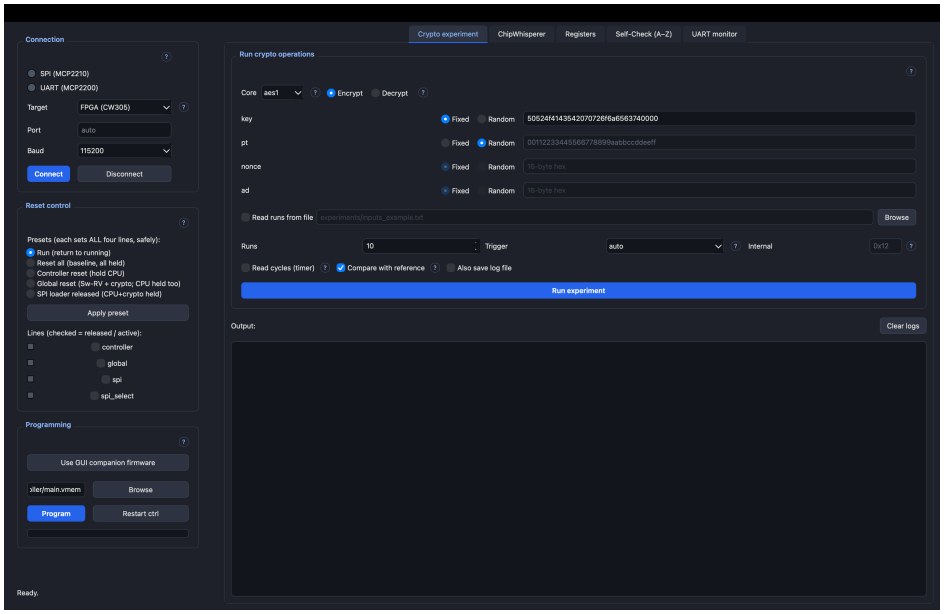

Figure 3: New GUI for the PROACT toolkit, which allows for easy control of the PROACT SoC.

4.4 The GUI

The GUI and CLI developed for the PROACT toolkit are implemented in Python and interface with the C library described in Section 4.3. The graphical interface is built using `PyQt6` (≥6.5), while core host communications rely on `pyserial` (≥3.5) for UART transport, `hidapi` (≥0.14) for USB-HID device detection, and `mcp2210-python` (≥0.1.4) as an SPI code loader. Data processing and storage are handled via `numpy` (≥1.23) and `h5py` (≥3.7). This toolkit is an evolved version of the Proof of Concept (PoC) toolkit, whose GUI was initially developed in this thesis to test the C library and accelerate the main core programming time. The new toolkit also provides a CLI and allows for easy experiment setup and execution and is shown in Figure 3. Before the PoC toolkit was created, an earlier legacy toolkit was used, but it lacked the modern interface and integration features developed in this work. These three generations of the toolkit are compared in Table 4.

Feature / Capability	**Legacy toolkit**	**PoC toolkit**	**PROACT toolkit**
Supported Interfaces	GUI only	GUI only	GUI & CLI
SPI Connection	Hardcoded location	Automated detection	Automated detection
UART Connection	Manual selection	Automated detection	Automated detection
FPGA configuration method	External Jupyter notebook	In GUI	In GUI & CLI
C-library version	Legacy	Restructured	Restructured & optimized for size
Author	Mr. A. Sajadi	Mr. N.L.M. Stijger	Mr. A. Sajadi & Mr. N.L.M. Stijger

Table 4: Feature comparison across the three generations of the toolkit.

In PROACT toolkit, different experiments can be set up and executed with a few clicks, without needing to reflash the FPGA or reprogram the cores. The new toolkit also allows for easy integration of the PROACT toolkit into other software, by providing a CLI and a C library. This allows for easy automation of the experiments and makes it possible to run multiple experiments sequentially without needing to manually set up each experiment.

4.5 Measurement Setup

To be able to do analysis on the different cores, power traces are needed. In this thesis, power traces are collected using a ChipWhisperer Husky, a device built for performing side-channel power analysis (as illustrated in Figure 4). It has a max sample rate of 200MHz, which is 4 times the clock rate set on the FPGA (50MHz). This clock rate is chosen such that the traces are aligned with the clock cycles of the FPGA and is synced over a 20-pin header. Every sample is collected by measuring the voltage drop over a shunt resistor, which is connected with a coaxial cable to the ChipWhisperer. For the experiments, the input gain is set to 10 dB, except the Sw-RV core, which is set to 20 dB. The higher gain is needed for the Sw-RV core, since this core has a lower power consumption and thus a lower voltage drop over the shunt resistor. It starts collecting on the rise of the co-processors trigger bit until the set amount of samples is collected. It also counts

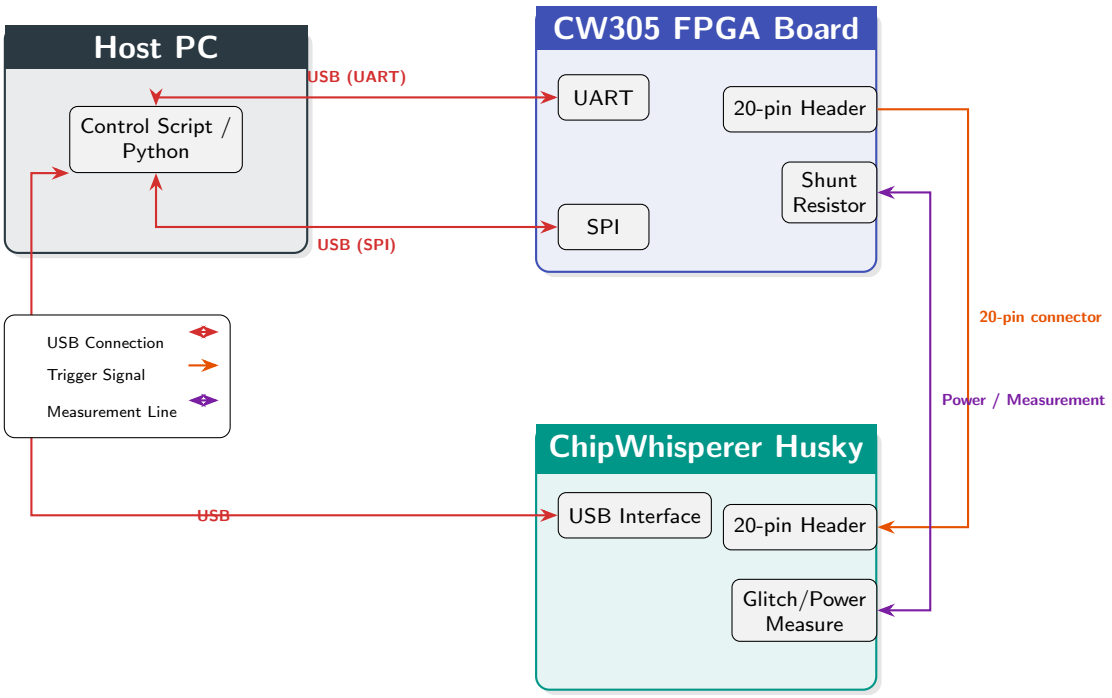

Figure 4: ChipWhisperer Husky Setup

the number of cycles between the trigger rise and trigger drop (adc.trig_count). This trigger signal is generated by the co-processor and is also communicated over the 20-pin header. Since the Husky knows both the clock and trigger signals, it can align the collected samples with the clock cycles of the co-processor and can count the number of clock cycles between the trigger rise and trigger drop (adc.trig_count). During recording, the decimation (adc.decimation) rate is set to 1, which means that every sample is saved. Per analyzed core, 20,000 power traces with each 5000 samples are collected, together with a random plaintext and trigger count. This count is used to trim the samples to exclude the idling of the core after the encryption is completed. These parameters are set to make sure that more than enough samples are collected to cover the entire encryption process, while also keeping the amount of data manageable. For each of the experiments on the different cores, the same key is used: `0x50524f4143542070726f6a6563740000`, which is "PROACT project" in ASCII, padded with zeros. The plaintexts are randomly generated for each round by using the `SystemRandom` library in Python and is seeded with random system bits. The random plaintexts are not unique, but the probability of a duplicate is very low, since the plaintexts are 128 bits long and the number of rounds is only 20,000 and should result in a uniform distribution of the plaintexts.

This configuration is chosen to ensure that there are more than enough power traces to perform a successful CPA attack. The sample size is the maximum amount of samples that can be collected by the ChipWhisperer Husky, while still being able to store all the traces in memory and does not affect the performance of the experiment. The decimation rate is set to 1 to make sure that no samples are skipped, which could result in missing important information. The trigger count of the last power trace is used to trim all the power traces to the same length and to exclude the idling of the core after the encryption is completed. This is done after the traces are collected, to make sure that the entire encryption process is captured in the power traces.

4.6 CPA Implementation and Leakage Models

In this thesis, CPA is implemented in Python using the `ChipWhisperer` library, which provides a set of tools to perform side-channel analysis. From this library, the `chipwhisperer.analyzer.cpa` module is used to perform the CPA attack. This module provides a set of functions to perform the CPA attack on the collected traces. The leakage models used in this thesis are the S-Box Output and Last Round State Difference models, which are explained in the following sections:

4.6.1 S-Box Output

The S-Box Output leakage model, is a model that exploits the non-linear spread of the S-Box (Section 3.3.1). Due to this non-linear spread, an incorrect guess generates a completely different output, which separates the correct key from the incorrect ones. The attack works by predicting the power usage, used to set the bits after the S-Box lookup in round 1 and can be calculated by using the Hamming Weight of the S-Box output.

$$h_{i,k_j} = \text{HW}(\text{S-Box}(p_i[j] \oplus k_j))$$

In which $h_{i,k}$ is the predicted power usage for the i-th plaintext and the k-th key byte guess, $p_i[j]$ is the j-th byte of the i-th plaintext and k is the key byte guess. The S-Box output is then used to calculate the correlation between the predicted power usage and the measured power usage.

This directly targets the master key, with no need to reverse the key expansion after the key has been found. The S-Box Output leakage modes is used for the SW-RV AES emulation, because the calculation is done in software and is computed and written byte-by-byte. This results in a clear separation between the different bytes, which can be used to find the correct key.

4.6.2 Last round state difference

In AES, the final round (Round 10 in AES-128) omits the MixColumns step. This creates a direct linear relationship between each byte of the final round state and the resulting ciphertext. This results in the ability to determine the last round key K_{10} byte-by-byte using known ciphertexts.

The power consumption can be predicted using the HW leakage model applied to the output of the inverse S-Box during the intermediate execution of the final round:

$$h_{i,k_{10,j}} = \text{HW}(\text{S-Box}^{-1}(c_i[j] \oplus k_{10,j}) \oplus c_i[\text{SR(j)}])$$

Where $h_{i,k_{10,j}}$ is the predicted power consumption for trace i given key byte guess $k_{10,j}$, c_i the i-th ciphertext block, $SR(j)$ the index mapping corresponding to the ShiftRows operation at byte position j, $k_{10,j}$ the candidate guess for the j-th byte of the 10th round key and S-Box^{-1} the inverse SubBytes transformation.

This leakage model is used for the AES 1 Look-Up Table (LUT) S-Box and AES 2 composite field cores, because these cores are implemented in hardware and the S-Box output is not directly accessible. But the entire 128-bit state is updated after every round and leaks the HW between the previous and current state. This allows for a byte-by-byte attack on the last round key, which can then be used to reverse the key expansion and find the master key.

The S-Box Output leakage model was also tried for the hardware cores, but this did not result in a successful attack. For the attack on AES 1, the attack gave the best results with a Sum of

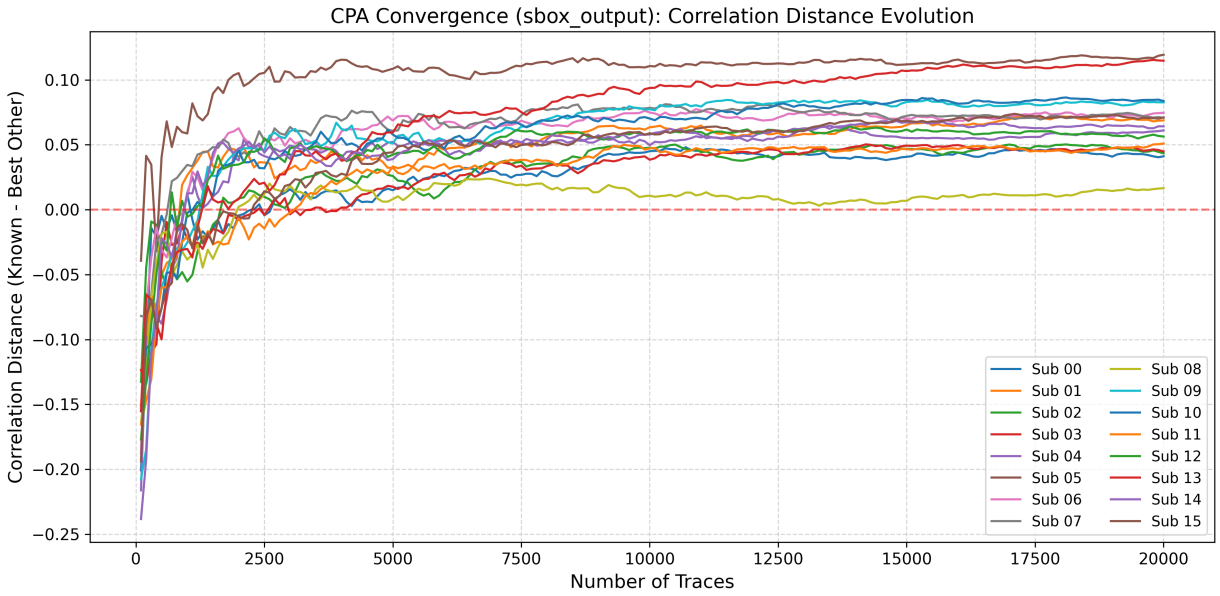

Figure 5: SW-RV AES emulation: S-Box output CPA, byte-by-byte key recovery compared to the best incorrect key

Absolute Differences (SAD) correction. For AES 2 and the SW-RV AES emulation, this correction did not improve the results. The SAD correction is a method to correct for misaligned traces, which can occur when the traces are desynced during the collection. Although the samples are synchronized with the FPGA clock, trigger jitter or variable latency between the trigger and the first round can shift the operation within the captured window. This correction is done by calculating the SAD between the traces and shifting them to align them.

5 Results

The results of the CPA on the different cores are shown in figures 5, 6 and 7, in which the difference in correlation between the correct key and the best incorrect key is shown against the number of traces used. For the SW-RV AES emulation the complete correct key is found after 4000 traces with the sbox_output leakage model 4.6.1. The complete correct key for the AES 1 LUT S-Box core is found after 5800 traces with the last_round_state_difference leakage model 4.6.2 and with the same method the key is found for the AES 2 composite field core after 7100 traces. This is the moment all the lines stay above the zero line. The key found in all experiments is the same as the key used in the experiments, which is `0x50524f4143542070726f6a6563740000`.

5.1 Acquisition results

To measure the improvement of the new GUIs, the time to program the controlling software to the main Ctrl-RV core is measured, which is shown in Table 5. This shows that the PoC GUI improves the programming time with respect to the legacy GUI by a factor of 16.1. The new GUI improves the programming time with respect to the legacy GUI by a factor of 18.1 and with respect to the

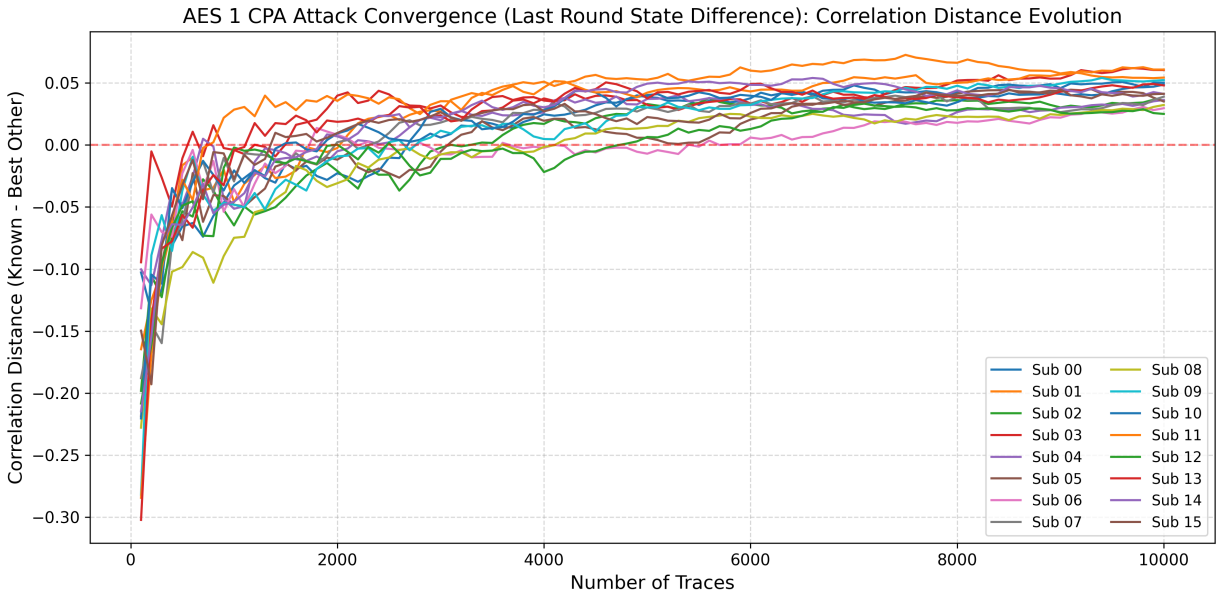

Figure 6: AES 1 LUT S-Box CPA: last round state difference with SAD correction, byte-by-byte key recovery compared to the best incorrect key

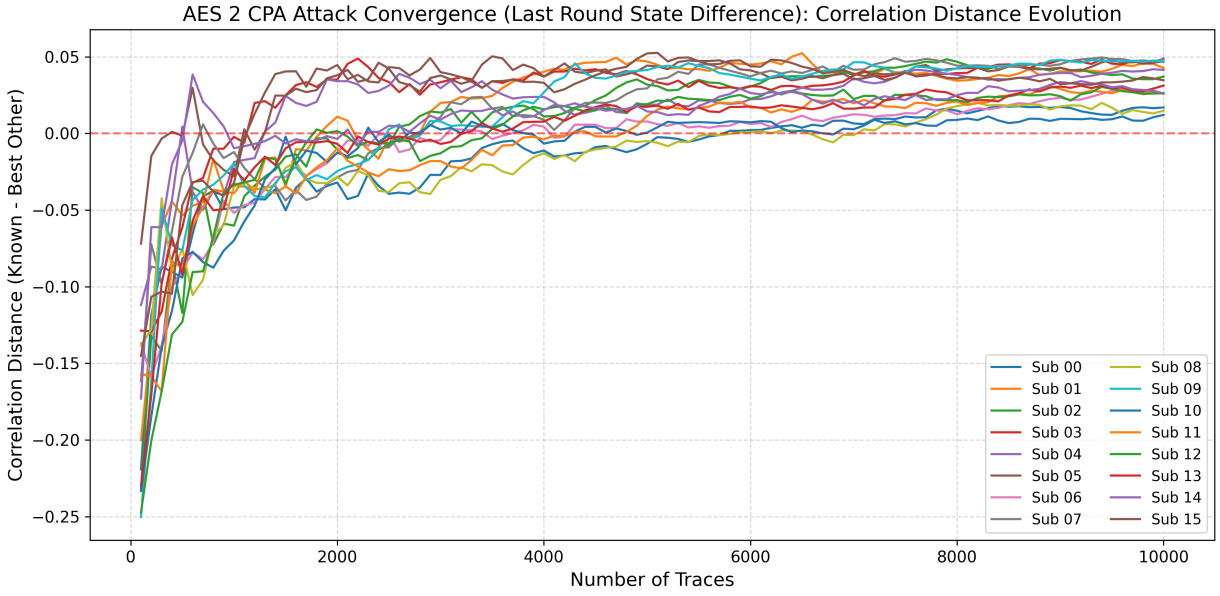

Figure 7: AES 2 composite field CPA: last round state difference without SAD correction, byte-by-byte key recovery compared to the best incorrect key

GUI	time (mm:ss)	byte count
Old	6:18.66	87485830
POC	0:23.45	87485830
New	0:20.92	30580

Table 5: Programming time for Ctrl-RV over a single acquisition

Core	byte count
AES1	223451526
AES2	191771010
Sw-RV	219281043

Table 6: Trace data size

PoC GUI by a factor of 1.1. The byte count is also shown, which is the amount of bytes sent over UART to program the main core. The new GUI uses a different format, which reduces the amount of bytes sent over SPI by a factor of 2860 with respect to the legacy and PoC GUIs. This new format only contains command-interpreter firmware instead of the full image. These timings are single measurements rather than an average of multiple measurements. Given the magnitude of difference, the variation between measurements should not affect the results.

This timing does not include the time needed to connect the GUI to the correct USB mapping and changed from a manual hardcoded location to an automatic detection of the correct USB mapping. This is done by using the `hidapi` library, which can detect the correct USB mapping by using the vendor and product ID of the device. This allows for a more user-friendly experience, since the user does not need to know the correct USB mapping and can just connect the device to any USB port.

As seen as in Table 6, the on disk size of 20,000 traces with 5000 samples, a plaintext, ciphertext and a key is around 200 MB. For each CPA attack, a python script is used to perform the attack on the collected traces. The time needed to perform the CPA attack is around 5 minutes on a modern laptop (tested on a MacBook Pro with a M1 chip), but this is can be improved by using a manual `NumPy` attack instead of the `ChipWhisperer` library, which is used in this thesis to only show that the correct key can be found.

5.2 Power Trace Analysis

In Figure 8, a single power trace of the AES 1 LUT S-Box core is shown, which shows the power consumption of the core during the encryption process. In Figures 9, 10 and 11, the mean power traces of the AES 1 LUT S-Box core, AES 2 composite field core and SW-RV AES emulation are shown. These mean traces are calculated by averaging the power consumption of all the collected traces. In Figures 9 and 10, the idling of the core can be seen after 1000 samples. This shows that a trim window of [0, 1000] samples is enough to capture the entire encryption process.

In Figure 11, a repetition of encryption rounds can be seen on the Sw-RV core, which confirms that the emulation takes multiple clock cycles to perform the encryption. This is due to the fact that the AES emulation is done in software and not in hardware, which results in a lower performance.

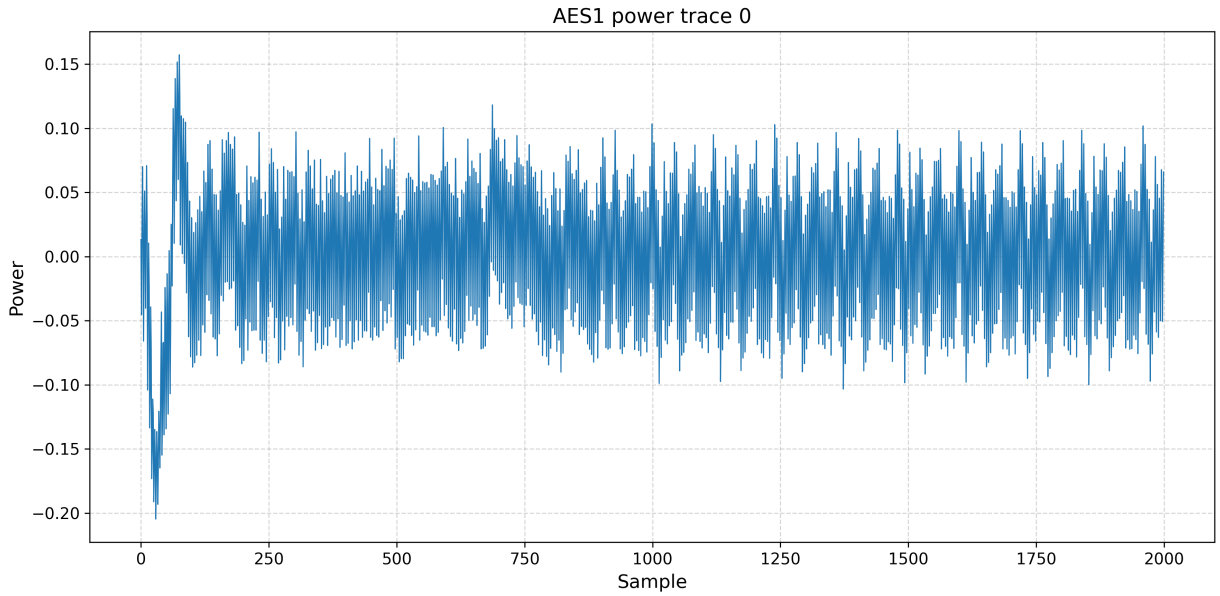

Figure 8: AES 1, Power Trace 0

Since the S-Box Output attack is an attack on the first round, a trim window of [15, 1800] is chosen.

5.3 AES 1 LUT S-Box

In Figure 12, the correlation of the first byte of the AES 1 LUT S-Box core is shown, which shows the correlation for byte 00 of the key for all 256 possible key guesses. The correct key guess is shown in black, which is the key used in the experiments. As seen as in Table 7, the maximum correlation of all bytes is around sample 57. Furthermore, the MTD of all bytes is shown, which is the number of traces needed to find the correct key byte. In Figure 13, the key ranks of all bytes are shown, which shows the rank of the correct key guess for all bytes over the number of traces used. This confirms that the correct key stays at the top rank after 5800 traces.

5.4 AES 2 Composite Field

In Figure 14, the correlation of the first byte of the AES 2 Composite Field core is shown, which shows the correlation for byte 00 of the key for all 256 possible key guesses. The correct key guess is shown in black, which is the key used in the experiments. As seen as in Table 8, the maximum correlation of all bytes is around sample 64. In Table **??**, the MTD of all bytes is shown, which is the number of traces needed to find the correct key byte. In Figure 15, the key ranks of all bytes are shown, which shows the rank of the correct key guess for all bytes over the number of traces used. This shows that the correct key stays at the top rank after 7100 traces.

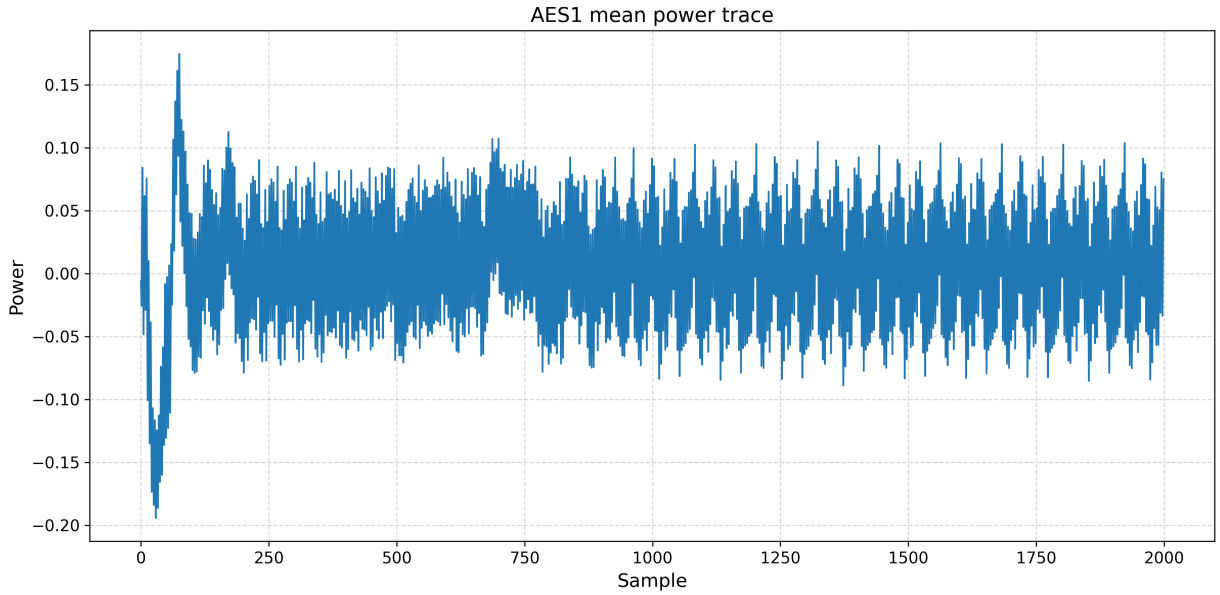

Figure 9: AES 1, Mean Power Trace

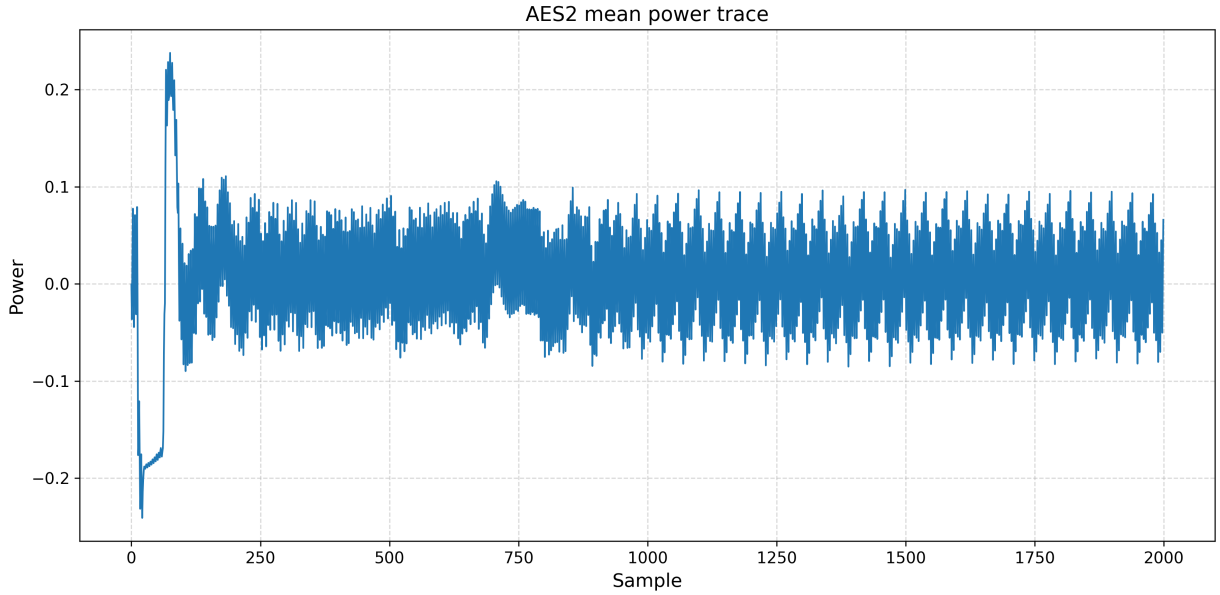

Figure 10: AES 2, Mean Power Trace

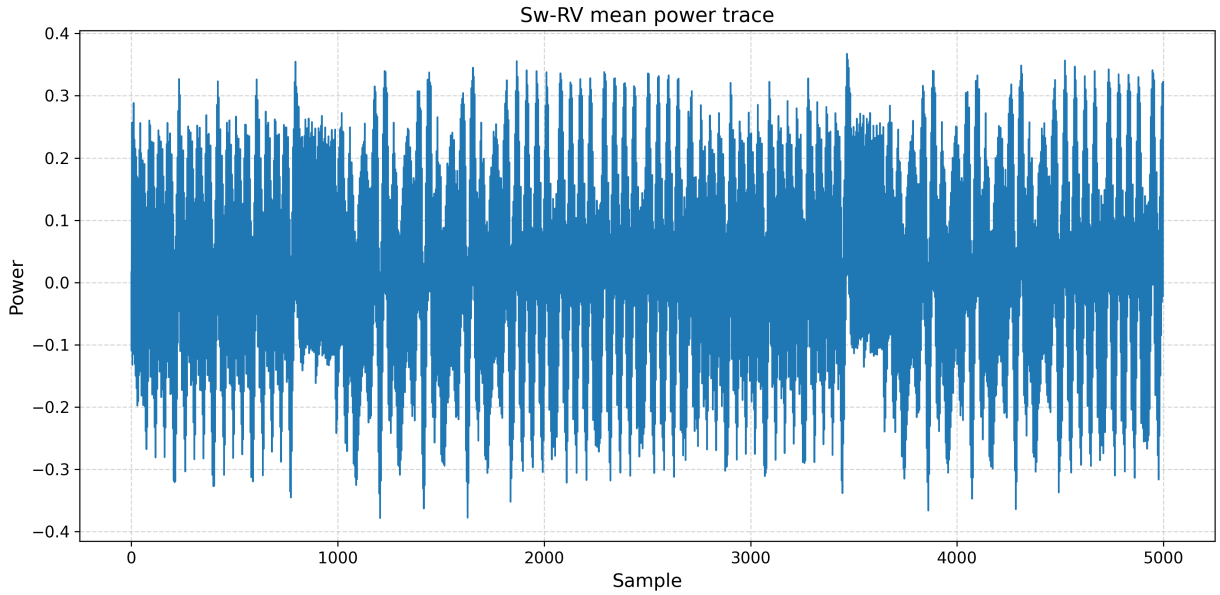

Figure 11: AES Sw-RV, Mean Power Trace

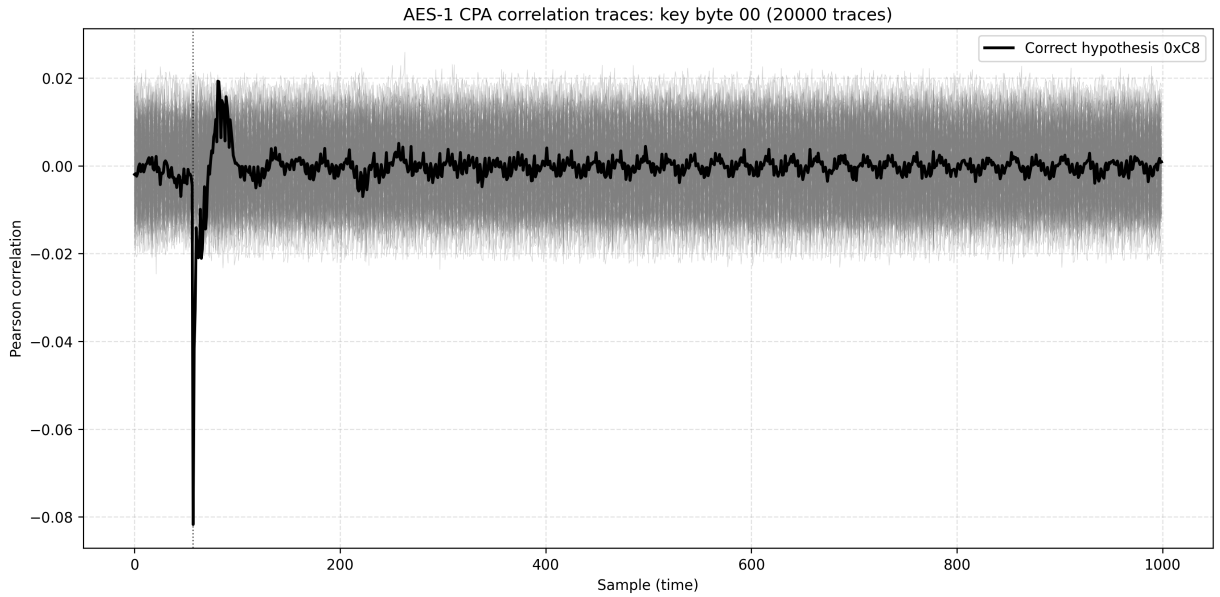

Figure 12: AES 1, LUT S-Box Correlations

Byte	Maximum Correlation	Sample Number	MTD (traces)
00	-0.0817	57	2700
01	-0.0848	57	2100
02	-0.0702	57	3300
03	-0.0937	57	1500
04	-0.0702	57	2200
05	-0.0690	57	3100
06	-0.0747	57	5800
07	-0.0761	57	2400
08	-0.0798	57	4200
09	-0.0777	57	2800
10	-0.0813	57	1900
11	-0.0857	57	800
12	-0.0682	57	4800
13	-0.0761	57	1200
14	-0.0781	57	1900
15	-0.0885	57	1400

Table 7: Maximum correlation (min=-0.0682, median=-0.0779, max=-0.0937) and MTD (min=800, median=2300, max=5800) of correct hypothesis per byte for AES 1 LUT S-Box

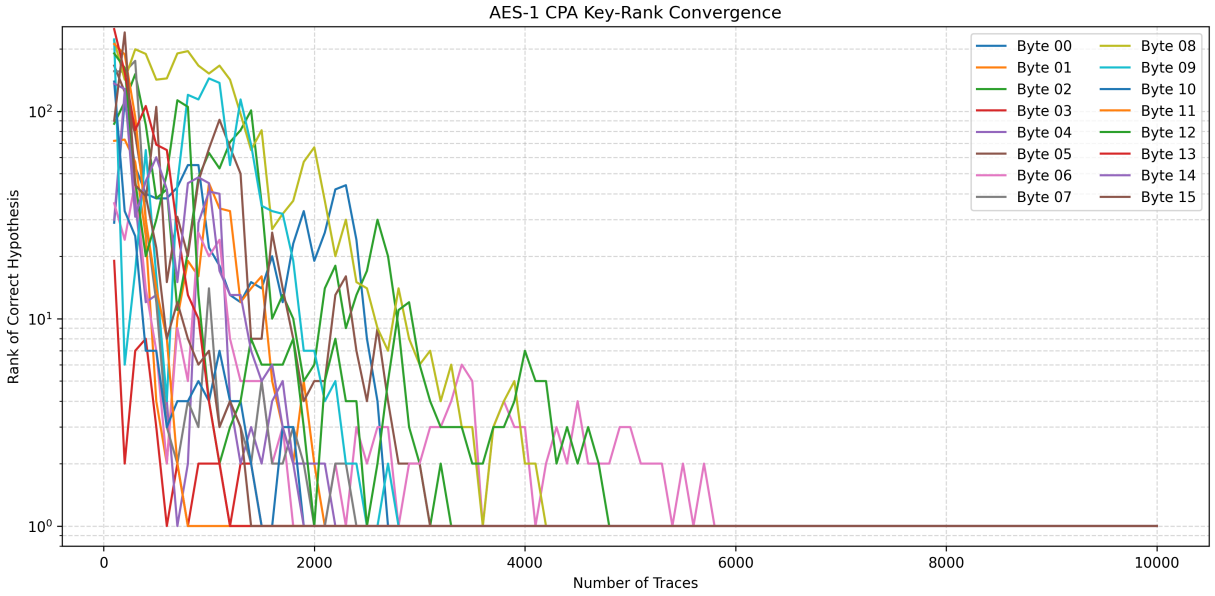

Figure 13: AES 1, LUT S-Box Key Ranks

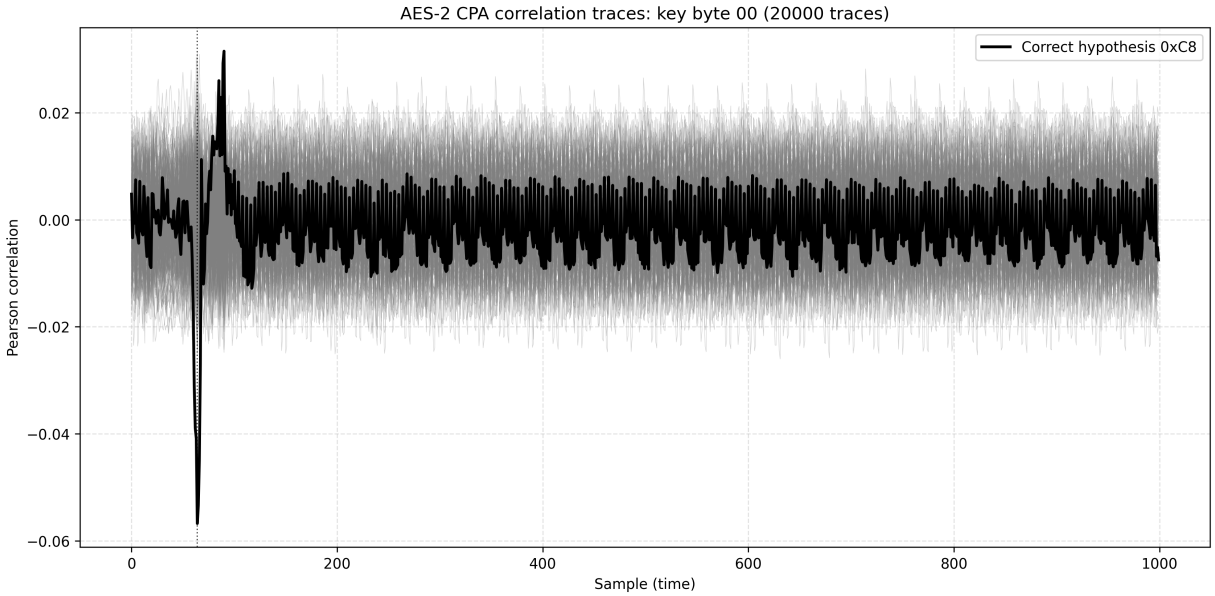

Figure 14: AES 2, Composite Field Correlations

Byte	Maximum Correlation	Sample Number	MTD (traces)
00	-0.0594	64	5800
01	-0.0795	64	2800
02	-0.0694	64	2600
03	-0.0867	64	1200
04	-0.0873	64	1300
05	-0.0820	64	1300
06	-0.0636	64	4300
07	-0.0862	64	2800
08	-0.0503	64	7100
09	-0.0884	64	2900
10	-0.0574	64	6900
11	-0.0663	64	4800
12	-0.0900	64	3500
13	-0.0724	64	3300
14	-0.0699	64	2300
15	-0.0844	64	1200

Table 8: Maximum correlation (min=-0.0503, median=-0.07595, max=-0.0900) and MTD (min=1200, median=2850, max=7100) of correct hypothesis per byte for AES 2 Composite Field

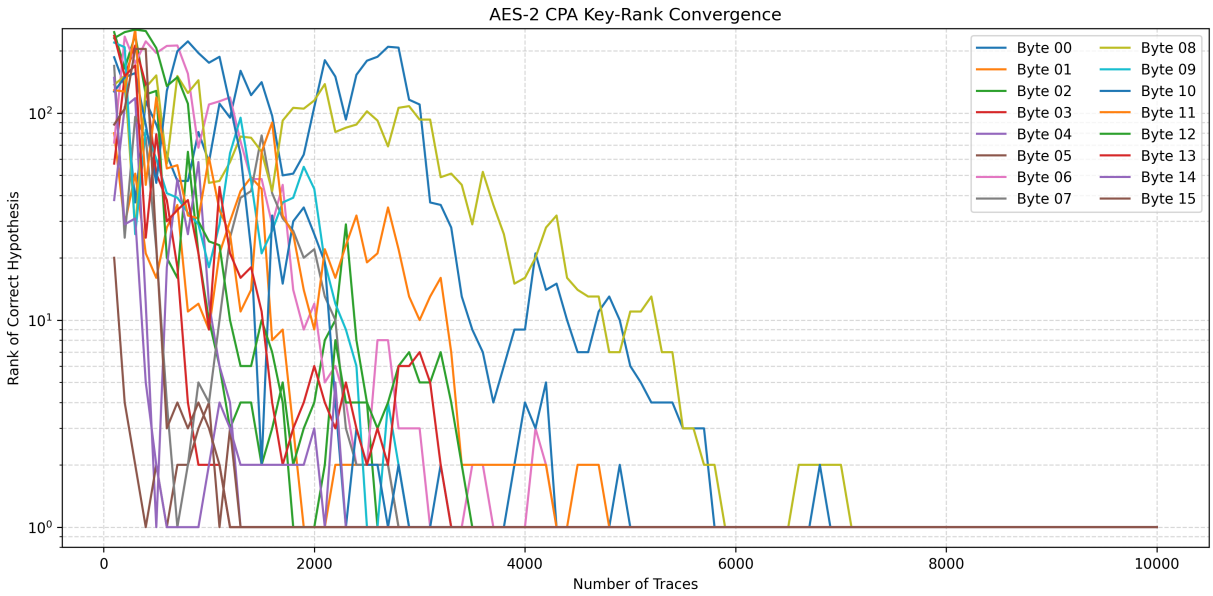

Figure 15: AES 2, Composite Field Key Ranks

5.5 AES Sw-RV Emulation

In Figure 16, the correlation of the first byte of the AES Sw-RV core is shown, which shows the correlation for byte 00 of the key for all 256 possible key guesses. The correct key guess is shown in black, which is the key used in the experiments. As seen as in Table 9, every byte has its own maximum correlation peak. Furthermore, the MTD of all bytes is shown, which is the number of traces needed to find the correct key byte. In Figure 17, the key ranks of all bytes are shown, which shows the rank of the correct key guess for all bytes over the number of traces used. This shows that the correct key stays at the top rank after 4000 traces.

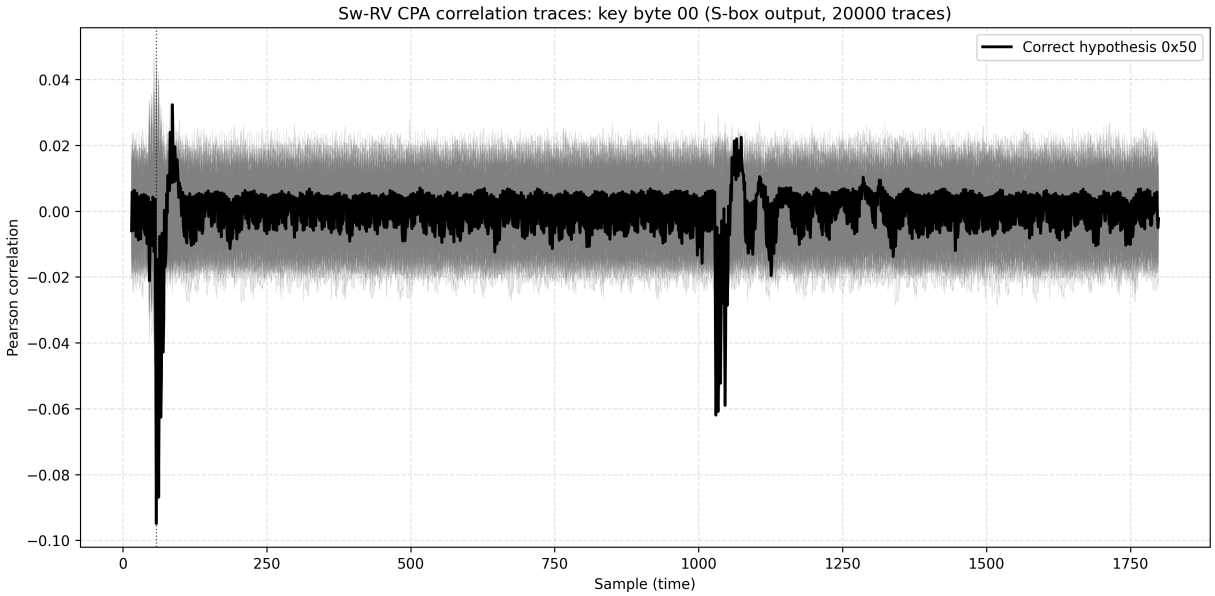

Figure 16: AES Sw-RV Emulation

Byte	Maximum Correlation	Sample Number	
00	-0.0949	58	2700
01	-0.1145	1666	900
02	-0.0911	1478	1700
03	-0.0816	1254	4000
04	-0.1117	110	1000
05	-0.1604	1030	500
06	-0.1223	486	1000
07	-0.1136	1454	700
08	-0.0768	150	2000
09	-0.1270	1242	1300
10	-0.1285	1046	1200
11	-0.0892	1678	3100
12	-0.1072	190	1200
13	-0.1542	1454	1300
14	-0.1161	566	1000
15	-0.1112	1038	2300

Table 9: Maximum correlation (min=-0.0768, median=-0.11265, max=-0.1604) and MTD (min=500, median=1250, max=4000) of correct hypothesis per byte for Sw-RV AES Emulation

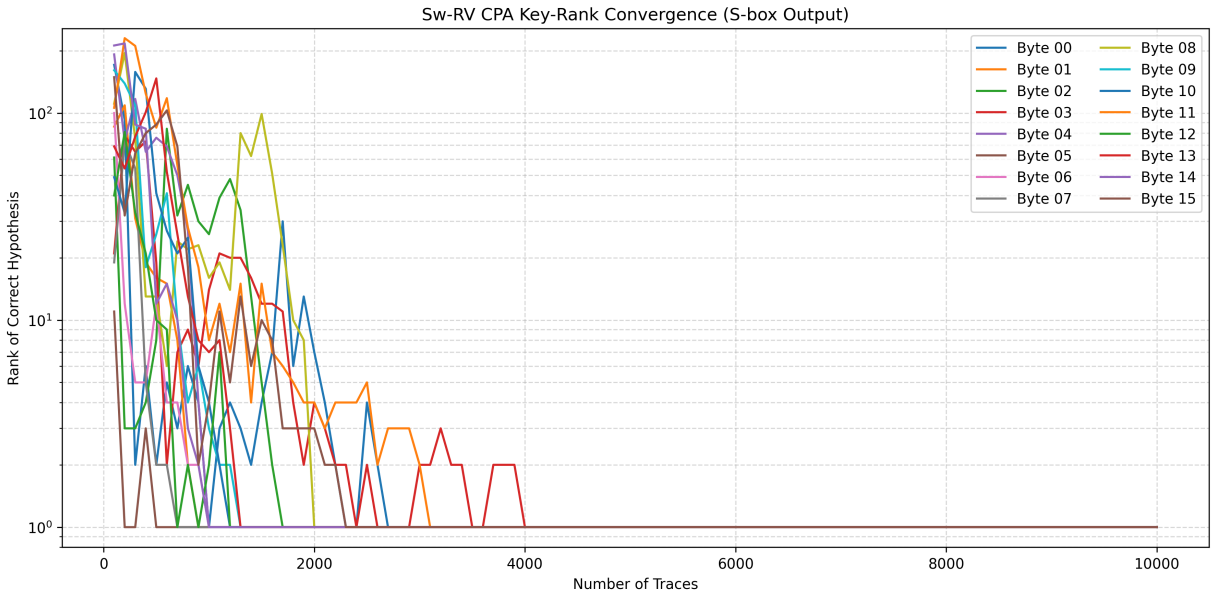

Figure 17: AES Sw-RV Key Ranks

6 Conclusion, Discussion and Further Research

In this thesis, a new toolkit was developed to perform side-channel analysis on the PROACT SoC. This toolkit includes a new firmware for the PROACT SoC, which allows for multiple experiments to be performed on the SoC without the need to reprogram the device. As shown in Section 5.1, a total speedup of 18.1x and a size reduction of 2860x was achieved with respect to the legacy toolkit. This makes it safe to say that the new toolkit is a significant improvement in efficiency and usability, which answers RQ 1.

Not only the speedup and size reduction is an improvement, but also the usability of the toolkit is improved. The new GUI allows automatic detection of the correct USB mapping and the ability to configure experiments without the need to reprogram the device. The CLI even allows a full setup with a single bash script, which can be used to perform a full experiment without the need for any manual steps. This makes the toolkit more fault-tolerant and reduces the chance of human error, which answers RQ 2.

After the toolkit was completed, a full side-channel analysis was performed on the AES implementations on the PROACT SoC. Traces were collected with a single bash script and the correct key was found for all three AES implementations. This shows that the new toolkit is fully functional and allows efficient collection of traces, which answers RQ 3.

6.1 Discussion

In this thesis, the CPA attack was performed on the AES implementations running on the PROACT SoC without any security countermeasures. However, these results are limited to experiments conducted on a single physical board using a single cryptographic key; testing on an ASIC or across multiple hardware samples and keys could yield different trace characteristics and attack performance. To mitigate these single-sample constraints and average out experimental anomalies,

collecting data across multiple measurement campaigns would provide a more generalized noise profile. Across the different cores, every byte exhibits a different MTD, which could be caused by this underlying measurement noise. A summary of the key comparison metrics across the three implementations is shown in Table 10.

Core	Model	MTD (min / med / max)	Corr. (min / med / max)	Peak sample
SW-RV AES emulation	S-Box output	500 / 1250 / 4000	-0.0768 / -0.11265 / -0.1604	58–1678
AES 1 LUT S-Box	Last round state difference	800 / 2300 / 5800	-0.0682 / -0.0779 / -0.0937	57
AES 2 Composite Field	Last round state difference	1200 / 2850 / 7100	-0.0503 / -0.07595 / -0.0900	64

Table 10: Summary of the CPA results for the three AES cores: leakage model, MTD, maximum correlation, and dominant peak sample.

The MTD for the SW-RV AES emulation is lower than the MTD for the hardware cores, which is caused by the fact that the AES emulation is done in software and not in hardware. This results in a lower performance, which allows for a more accurate measurement of the power consumption. Furthermore, every byte has a different peak sample (see Table 9), which contributes even more to the lower MTD.

The AES 1 LUT S-Box core has a lower MTD than the AES 2 composite field core, which is caused by the extra noise generated by the composite field implementation and is supported by the wider spread in Table 8 compared to Table 7.

6.2 Further Research

Future extensions of this project could be:

- Repeating the same experiments on an ASIC implementation of the PROACT SoC, which can be used to compare the results with the FPGA implementation.
- Run experiments for ASCON and Xoodyak, which are the other two cryptographic algorithms implemented on the PROACT SoC.
- Extending the GUI with an attack module, which can automatically perform CPA attacks on the collected traces.
- Allowing different AES implementations to be selected in the GUI for the SW-RV AES emulation, which can be used to compare the different implementations.
- Using other attack techniques such as deep learning SCA or fault injection attacks.
- Implementing security countermeasures on the AES implementations and testing the effectiveness of these countermeasures against SCA.

References

[1] Asmita Adhikary, Abraham Basurto, Lejla Batina, Ileana Buhan, Joan Daemen, Silvia Mella, Nele Mentens, Stjepan Picek, Durga Lakshmi Ramachandran, Abolfazl Sajadi, Todor Stefanov, Dennis Vermoen, and Nusa Zidaric. PROACT – physical attack resistance of cryptographic algorithms and circuits with reduced time to market. In *Applied Reconfigurable Computing. Architectures, Tools, and Applications (ARC 2024)*, Lecture Notes in Computer Science, pages 255–266. Springer Nature Switzerland, 2024.

[2] Jens Anders, Pablo Andreu, Bernd Becker, Steffen Becker, Riccardo Cantoro, Nikolaos I. Deligiannis, Nourhan Elhamawy, Tobias Faller, Carles Hernandez, Nele Mentens, Mahnaz Namazi Rizi, Ilia Polian, Abolfazl Sajadi, Mathias Sauer, Denis Schwachhofer, Matteo Sonza Reorda, Todor Stefanov, Ilya Tuzov, Stefan Wagner, and Nusa Zidaric. A survey of recent developments in testability, safety and security of RISC-V processors. In *2023 IEEE European Test Symposium (ETS)*, pages 1–10, 2023.

[3] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation Power Analysis with a Leakage Model. In *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer Berlin Heidelberg, 2004.

[4] Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Xoodyak, a lightweight cryptographic scheme. *IACR Transactions on Symmetric Cryptology*, 2020(S1):60–87, 2020.

[5] Joan Daemen and Vincent Rijmen. The Rijndael Block Cipher. *AES Proposal: Rijndael*, 1999.

[6] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power analysis. In *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer Berlin Heidelberg, 1999.

[7] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 2007.

[8] National Institute of Standards and Technology. Advanced Encryption Standard (AES). Technical Report FIPS 197, NIST, 2001.

[9] Colin O'Flynn and Zhizhang David Chen. ChipWhisperer: An open-source platform for hardware embedded security research. In *Constructive Side-Channel Analysis and Secure Design (COSADE)*, pages 243–260, 2014.

[10] Abolfazl Sajadi, Nusa Zidaric, Todor Stefanov, and Nele Mentens. A systematic comparison of side-channel countermeasures for RISC-V-based SoCs. In *2024 IEEE Nordic Circuits and Systems Conference (NorCAS)*, pages 1–7, 2024.

[11] Mr. A. Sajadi. Proact, the complete manual. Manual Version 1.0, Leiden University, LIACS, 2026.

[12] Akashi Satoh, Sumio Morioka, Kohji Takano, and Seiji Munetoh. A Compact Rijndael Hardware Architecture with S-Box Optimization. *Advances in Cryptology — ASIACRYPT 2001*, pages 239–254, 2001.

[13] Transforma Insights. Number of Internet of Things (IoT) connections worldwide from 2022 to 2023, with forecasts from 2024 to 2034 (in billions). Statista, 2025.

[14] Meltem Sönmez Turan, Kerry McKay, Jinkeon Kang, and John Kelsey. Ascon-Based Lightweight Cryptography Standards for Constrained Devices: Authenticated Encryption, Hash, and Extendable Output Functions. NIST Special Publication 800-232, National Institute of Standards and Technology, August 2025.

A C Library API

- `void ctrl_reset(void)`
  Clears the local shadow copy of the control register.
- `void ctrl_flush(void)`
  Writes the shadow control value to hardware.
- `void ctrl_flush_debug(int debug)`
  Flushes control register and optionally prints its value for debugging.
- `void ctrl_set(uint32_t bits)`
  Sets selected bits in the control shadow register.
- `void ctrl_clr(uint32_t bits)`
  Clears selected bits in the control shadow register.
- `uint32_t status_read(void)`
  Reads the shared status/control register value.
- `int status_test(uint32_t mask)`
  Checks whether all bits in a mask are set in the status register.
- `void status_wait(uint32_t mask)`
  Blocks until all requested status bits are set.
- `int status_wait_timeout(uint32_t mask, uint32_t spins)`
  Polls status bits with a bounded timeout; returns success/failure.
- `void aes_reset(proact_aes_core_t core)`
  Performs safe reset sequencing for an AES core.
- `void aes_run(proact_aes_core_t core, const uint32_t key[4], const uint32_t data[4], uint32_t out[4], uint32_t decrypt, uint32_t verbose)`
  Runs one AES operation end-to-end (configure, start, wait, read, reset).
- `void aead_reset(proact_aead_core_t core)`
  Performs safe reset sequencing for an AEAD core.
- `void aead_run(proact_aead_core_t core, const uint32_t key[4], const uint32_t nonce[4], const uint32_t *ad, const uint32_t *pt, uint32_t *ct, uint32_t *tag, uint32_t ad_len, uint32_t pt_len, uint32_t decrypt, uint32_t triggercfg, uint32_t verbose)`
  Runs one AEAD transaction (load inputs, pulse start, poll done, read CT/TAG).
- `int aead_kat_run(void)`
  Runs ASCON and Xoodyak known-answer tests and returns a bitmask result.
- `int aes_experiments(uint32_t debug)`
  Runs AES1/AES2 encrypt/decrypt self-tests against known vectors.

- `int ascon_experiments(uint32_t debug)`
  Executes the AEAD round-trip test for ASCON.

- `int xoodyak_experiments(uint32_t debug)`
  Executes the AEAD round-trip test for Xoodyak.

- `int run_all_experiments(uint32_t debug)`
  Runs all crypto self-tests and reports aggregate failures.

- `void swrv_enable(void)`
  Releases the software RISC-V target from reset.

- `void swrv_disable(void)`
  Holds the software RISC-V target in reset.

- `void swrv_select_trigger(void)`
  Routes trigger output to the software target trigger source.

- `void swrv_load_imem_word(uint32_t off, uint32_t word)`
  Writes one word to target instruction memory window.

- `void swrv_load_dmem_word(uint32_t addr, uint32_t word)`
  Writes one word to target data memory window.

- `int swrv_aes_block(const uint32_t key[4], const uint32_t in[4], uint32_t out[4], int decrypt, uint32_t timeout)`
  Commands target software AES, waits for completion, and reads result block.

- `void dev_write(const uintptr_t addr, const uint32_t val)`
  Performs a memory-mapped 32-bit device write.

- `uint32_t dev_read(const uintptr_t addr)`
  Performs a memory-mapped 32-bit device read.

- `int putchar(int c)`
  Writes one character to UART with pacing.

- `int puts(const char *str)`
  Writes a null-terminated string to UART.

- `void put_hex(uint32_t h)`
  Prints a 32-bit value as 8 hexadecimal characters.

- `int uart_rx_ready(void)`
  Checks if a UART byte is available via status register polling.

- `int uart_getchar(void)`
  Blocking UART receive of one byte.

- `int uart_getchar_timeout(uint32_t spins)`
  UART receive with bounded poll timeout.
- `void uart_set_baud_divisor(uint32_t divisor)`
  Programs UART baud-rate divisor register.