RAESCY

A power side channel assessment framework for pre **and** post-silicon evaluation

Msc Thesis

Laura Muntenaar

Delft University of Technology

TUDelft

RAESCY

A power side channel assessment framework for pre **and** post-silicon evaluation

by

Laura Muntenaar

Student Name	Student Number
Laura Muntenaar	4554213

Thesis Comittee:	prof. dr. ir. G.N. (Georgi) Gaydadjiev
	Dr.ir. M. (Motta) Taouil
	ir.Dr. C.J.M. (Chris) Verhoeven
	A. Aljuffri
Project Duration:	01, 2023 - 10, 2023
Faculty:	Faculty of Electrical Engineering, Delft

Cover image created with the assistance of DALL-E 2 (Modified)

Preface

That's it, I'm done. After spending many years at the TU Delft my time has finally come. This thesis was the crown jewel of my Tu Delft experience filled with very deep lows and occasional highs. But all's well that ends well. [34]

First off, to my supervisors Dr Ir M. Taouil and Prof Dr Ir G.Gaydadjiev, I had a lot of fun during our meetings. The genuine excitement and ideas brought up boosted my motivation and made me push as hard as I could. Then to Abdullah Aljuffri, thank you so much for your continuous support and motivation. When deadlines came really close or I got really stressed you never wavered and helped guide the project back on track.

I would personally like to thank my parents and siblings for tolerating and supporting me for all these years. Next to that I want to thank my incredible boyfriend Nicolaas du Plessis for supporting and comforting me through the last 10 months. Lastly my amazing friends, Thijs Nulle, David Vos, Sem Duveen, Natalia Struharova, Dan Andreescu, Ioana Savu, Stephan Olde, Julian Sanders, Ion Babalau, Konrad Ponichtera, Luke de Waal, Bob Morssink and many more. Thank you for supporting me and sticking with me until the end. To all you guys, Im sorry I was basically unbearable and unavailable during this thesis, I love you guys.

Laura Muntenaar
Delft, November 2023

Abstract

Demand for smart, Internet-connected devices and other electronics has increased dramatically in recent years. This increase in demand for technological devices, driven by advancements in Artificial Intelligence (AI), the Internet of Things (IoT), and autonomous systems, has exposed the digital system to potential security threats. As more devices access personal data, data protection and security have become increasingly challenging. The rise in security incidents, such as the MOVEit attack and the NeBu data breach, highlights the urgency for better security systems. Malicious attacks can manifest at different levels, ranging from the network and system to the circuit level. Power side-channel attacks, which exploit variations in power consumption to obtain sensitive information, have proven effective against modern cryptographic implementations. In response to the threats posed by such attacks, researchers have diligently focused on devising countermeasures. Current research efforts focus on developing these countermeasures, yet evaluation only happens in pre-silicon scenarios in simulation or on an FPGA. Evaluting the effectiveness of those countermeasures on real Systems on Chips (SoCs) is challenging.

The thesis introduces a novel framework to bridge the gap between pre and post-silicon power side channel assessment, allowing accurate characterization and comparison of cryptographic designs before and after manufacturing. This approach aims to provide comprehensive profiling and ensure the security of future cryptographic designs. The suggested platform was verified by employing a series of different versions of the Advanced Encryption Standard (AES) algorithm that were integrated into a full System on Chip (SoC). As a metric for determining whether or not the proposed platform is suitable, attacability is being measured via correlation power analysis (CPA). Pre-silicon enables the evaluation of each AES core in isolation from the other components of the system, hence having noise-free power traces. However, in post-silicon, in order to provide a correct analysis, the platform incorporates a wide variety of features, such as automation execution, trace alignment, and noise reduction of power traces. In order to improve the automated execution process, a Left Feedback Shift Register (LFSR) has been devised to introduce a level of randomness in the encrypted messages and keys used by the cryptographic engine. In the context of trace alignment, the platform has been specifically built to include a trigger signal that serves the purpose of identifying the start/stop of the execution. Clock-gating is used as a means of mitigating noise by freezing the operation of additional cores, hence preventing the generation of extraneous auditory disturbances. In addition, the suggested approach is specifically tailored for the manufacturing process using the 40nm TSMC technology, including a simulation evaluation. Finally, to facilitate power measurements on the platform, custom printed circuit boards(PCB's) and a software backend were designed to support real life measurements and the final correlation of the pre and post-silicon power side channel behaviour. The results of our study suggest that both Pre-silicon (i.e., standalone evaluation) and Post-silicon (i.e., system evaluation) provide similar levels of accuracy in assessing attackability.

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

Abbreviation	Definition
A	Accelerators
AES	Advanced Encryption Scheme
AHB	Advanced High-performance Bus
AHB3-lite	Advanced High-performance Bus 3-lite
AI	Artificial Intelligence
AMBA	Advanced Micro-controller Bus Architecture
ASIC	Application-Specific Integrated Circuit
BL	Bitline
CIA	Confidentiality, Integrity, and Availability
CG	Clock gating
CPU	Central Processing Unit
CISC	Complex Instruction Set Computer
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DPA	Differential Power Analysis
DRAM	Dynamic RAM
DUT	Device Under Test
ESP-01	Espressif System - 01
FPGA	Field Programmable Gate Array
FTDI	Future Technology Devices International
GHz	Gigahertz (a unit of frequency equal to one billion hertz)
GPIO	General-Purpose Input/Output
IDE	Integrated Development Environment
IO	Input/Output
JTAG	Joint Test Action Group
LFSR	Linear Feedback Shift Register
MHz	Megahertz (a unit of frequency equal to one million hertz)
MOSFETs	Metal-Oxide-Semiconductor Field Effect Transistors
MU	Memory Unit
NI	Network Interface
NIST	National Institute of Standards and Technology
PCB	Printed Circuit Board
PoR	Power on Reset
PU	Peripheral Units
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
RISC-V	Reduced Instruction Set Computer - Five
RI5CY	A RISC-V core developed by the open-hardware group
ROM	Read-Only Memory
RX	Receiver
SCCA	Side Channel Channel Analysis
SNR	Signal-to-Noise Ratio
SoC	System on Chip
SPI	Serial Peripheral Interface
SVM	Support Vector Machines
TX	Transmission

Abbreviation	Definition
UART	Universal asynchronous receiver / transmitter
USB	Universal Serial Bus
USB-C	Universal Serial Bus - Type C
WL	Wordline

1 Introduction

This chapter briefly covers the topic of this thesis, why it is important, what work has been done on it and the contributions of this thesis. First, the motivation for this thesis is explained, and then the current state-of-the-art research is discussed. Then the contributions of this thesis are explained and lastly, an overview of the thesis organization is given.

1.1. Motivation

The world is experiencing a growing trend of interconnectivity, with an increasing demand for ”smart” devices, which are internet-connected and capable of accessing specific data from every corner of the world. This relentless pursuit of technical innovation is driven by explorations into technologies such as Artificial Intelligence (AI), the Internet of Things (IoT), and autonomous systems. However, the exponential growth of connected devices also exposes this digital infrastructure to potential security threats. With more and more devices accessing personal data, data protection and security have ballooned over the years.

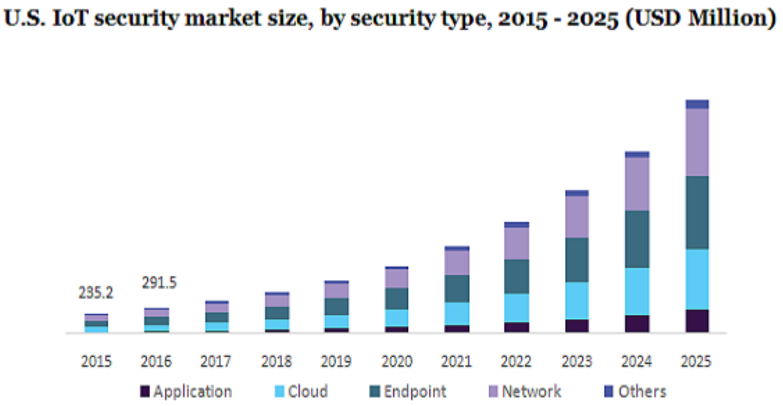


Figure 1.1: Rise in IoT security market [5]

Securing and protecting private information in an ever-growing digital space becomes more difficult not only for individuals but also for organizations. A good example of failing security is the recent MOVEit attack [33] where over four hundred organization’s security systems got breached leading to company, personnel and customer data getting stolen with simple third-party file transfer software. Another example closer to home is the recent data breach NeBu [46] which led to the release of personal information of thousands of Dutch citizens. Security breaches and data leaks get more frequent and are almost routine.

With the rise of data leaks and security breaches, there is an urgent need for better security systems and better security infrastructures. This becomes evident by the exponential growth in the IoT security market (see Figure 1.1). Companies are desperate for better and innovative ways of securing their and their customer's information.

When it comes to malicious security attacks, they are not solely isolated to software or cloud-based attacks. Generally, security attacks happen on one of three levels, network level, device level and chip level. Network-level attacks refer to attacks targeting inter-device communication or cloud-based applications. Device-level attacks refer to attacks targeting devices such as cars, phones or computers. Lastly, chip level or side-band attacks refer to the physical attacks of individual chips.

Focusing on the device-level attacks, side channel attacks have emerged as a significant concern. Side channels refer to unintended information leakage channels that provide valuable insights into the internal operations of a system, enabling attackers to extract sensitive information without directly accessing the protected data. Side channel attacks, particularly those exploiting power consumption patterns, have shown to be rather effective against modern cryptographic implementations. An example of how side channels can be used maliciously is shown in figure 1.2.

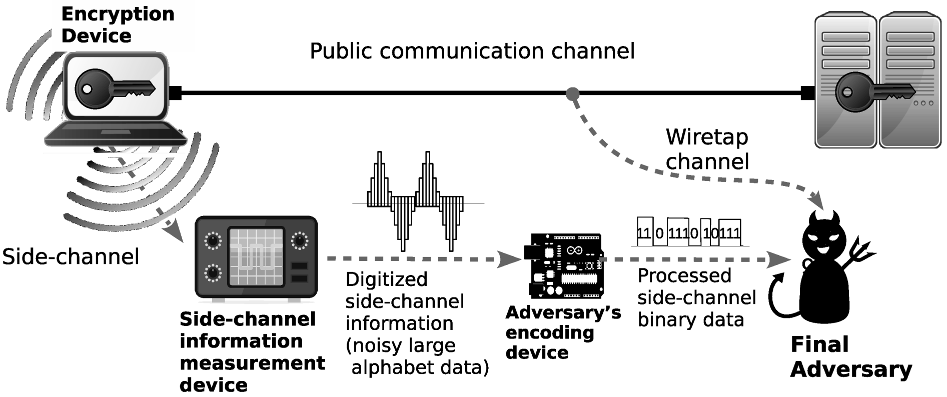

Encryption Device
Public communication channel
Wiretap channel
Side-channel
11 0 111 0 10 111
Digitized side-channel information (noisy large alphabet data)
Side-channel information measurement device
Adversary's encoding device
Processed side-channel binary data
Final Adversary

Figure 1.2: Example side channel attack [72]

Different types of side channels can be exploited. Figure 1.3 shows the different side channels of a device that can be exploited. Different side channels can be characterized by their success rate in retrieving sensitive information.

The most powerful type of side-channel attack is the power side-channel attack. Power side-channel attacks exploit the power consumption and power profile of a device to retrieve encryption/decryption keys. By observing fluctuations in the power lines during encryption or decryption the attacker can gain knowledge of what encryption is running and by knowing what data is being encrypted, the attacker can try to re-create the key.

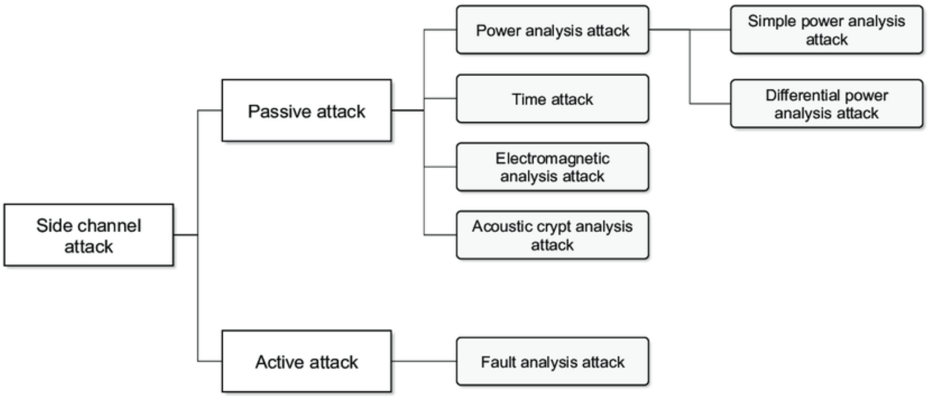

Side channel attack
Passive attack
Active attack
Power analysis attack
Time attack
Electromagnetic analysis attack
Acoustic crypt analysis attack
Fault analysis attack
Simple power analysis attack
Differential power analysis attack

Figure 1.3: Classification of side-channel attacks [63]

To combat power side-channel attacks, substantial research efforts have been devoted to developing countermeasures. Various techniques have been proposed, aiming to mitigate vulnerabilities in the design and hardware implementation of these cryptographic algorithms. This research, unfortunately, has one big flaw, namely, it is difficult to understand and quantify how secure the designed countermeasures are when implemented on real systems on chips(SoCs).

Currently either designs are purely characterized before they are manufactured or companies invest in post-silicon characterizations and certifications to verify if the designs or countermeasures are as secure as proposed and designed. This, however, also implies that designs need to be manufactured to verify designs. If the resulting chip is inadequate, the company has already invested a lot of money into research, design and manufacturing of a non-functional chip.

This thesis introduces a novel framework to bridge the gap between pre and post-silicon power side channel assessment to provide accurate pre **and** post-silicon characterization and comparison. Ensuring that future cryptographic designs can be fully characterized and profiled before they need to be manufactured by characterizing the differences in pre and post silicon side channel security. The novel platform includes a specific set of different cryptographic designs as case studies and includes a full architecture from SoC to hardware design to give a complete platform for the pre-and post-silicon side channel assessment.

1.2. State of the art

Current research into power side channel analysis is primarily focused on the mitigation of side channel leakage in the pre-silicon phase. The assessment of the power side channel on either field programmable gate arrays(FPGA) is preferred over the assessment at a post-silicon phase like a System on Chip(SoC) design as they are more cost-efficient and flexible. Certain post-silicon side channel assessment frameworks do exist, however, they are limited in their capabilities and flexibility.

Platforms for pre-silicon comparison do exist. The oldest and most known implementation is the SASEBO board [42]. The SASEBO board was the first effort to design a standardized platform for the evalution of side-channel attacks. The SASEBO series supports various AES implementations with a dual FPGA setup where one FPGA facilitates the loading and setting up of the experiment and the other FPGA for implementing the AES under test. The limitations of SASEBO are as follows. Firstly it is outdated, in time and technology. Given the rapid advancements in the field of cryptography, the solutions and techniques presented are not the most current or effective ones. Lastly, the SASEBO design only facilitates pre-silicon side channel assessment. Even the SAKURA series, which is the next generation of SASEBO also facilitates only pre-silicon side channel assessment with the help of a Kintex-7 FPGA. [32]

In 2014, the ChipWhisperer platform [57] was introduced as the new standard in power-side channel assessment platforms. ChipWhisperer includes an FPGA-based target device which can be reconfigured to fit any other piece of hardware. ChipWhisperer also includes its own Python software package. There are two issues with ChipWhisperer, firstly ChipWhisperer hardware is costly. The ChipWhisperer software is free to use but to use the software properly, customers have to buy their hardware. Although it claims to be open source, getting the platform is difficult. Next to that, the ChipWhisperer architecture is fixed and not easily extended, meaning that customization is difficult without having to re-design the entire platform.

Next to Chipwhisperer and Sasebo/sakura, the last board in the realm of power side channel evaluation is the HaHa board [86]is a microcontroller/FPGA hybrid board with the intent to be used for education. In this platform, a microcontroller is used to run a custom design on an FPGA. The limitation is again that the post-silicon phase is not facilitated.

Currently, only one platform can facilitate post-silicon side channel assessment. In 2021 Platform Saidoyoki [44] was proposed as a new dual comparison between pre and post-silicon side channel assessment. Saidoyoki is a dual platform that includes two custom 180nm SoCs with each 2 different implementations of AES on a custom printed circuit board (PCB). With the results of the post-silicon power side channel assessment, they can revert and compare with the pre-silicon simulation results. The limitations of this platform are that is not flexible or provides comparison between AES algorithms.

Next to that, they don't facilitate the power side channel assessment in a pre-silicon stage on a reconfigurable device but only in simulation. But simulation is never real life, especially when assuming ideal noise characteristics.

Other platforms that do power side channel assessment mostly have the aim to facilitate and maximize the measurement part rather than doing power side channel measurement on specific algorithms. SCLAF [40] is a good example of this. Where the platform is primarily focused on obtaining the highest signal-to-noise ratio for any algorithm. The limitation of these platforms is that they are not made for comparison between different engines as the software-to-hardware pipeline is often custom and runs one engine at a time.

Looking at the above existing platforms, there is a gap in the research that is yet to be explored. Designing a platform which can evaluate and analyse a complete set of cryptographic engines both in pre and post-silicon settings. With the aim of providing a correlation between the pre and post-silicon security.

1.3. Contributions

1. **Proposal of a novel framework that can facilitate pre- and post-silicon leakage assessment of cryptographic algorithms against power attacks:** In addition to the pre-silicon leakage assessment methods, we devised a new framework that provides an accurate post-silicon evaluation by having an authentication executed, a power traces alignment-based trigger, and noise reduction using clock-gating of the system during the measurement of the traces.
2. **Design and implementation of a SoC Based on the Proposed Framework:** As a case study to validate the proposed framework, a SoC system was designed and implemented, which consists of a RISCV microprocessor, six AES implementations for power side channel analysis, and perphirals IPs such as UART and SPI for control and monitoring.
3. **Design and implemenation of printed circuit board hardware for performing power side channel measurements with the SoC platform** Printed circuits boards (PCB's) were designed to provide realistic and reliable testing scenarios for the custom SoC in a pre and post silicon setting. The PCB's consists of a PCB for the Post-silicon setting, a PCB for the pre-silicon setting an a general board to interface with the aforementioned and facilitate a good comparitive study.
4. **Chip manufacturing of the designed SoC using 40nm TSMC technology:** As the framework intends to validate the system post-manufacturing, the implemented design was built for fabrication using 40nm TSMC technology and Cadence EDA tools.
5. **Evaluation of the designed SoC using CAD tools:** To evaluate the proposed framework features (i.e., execution automation, trace alignment, and noise reduction), the implemented design was simulated by enabling one or more of these features to study their effects on the side channel attackability.
6. **Validation of the security of cryptography implementations using the proposed framework:** Validation of the implemented cryptographic's power side channel security and classification of countermeasures.

1.4. Thesis organization

This thesis has been organized into seven distinct sections, each crafted to provide a comprehensive and in-depth exploration of the subject matter. The arrangement of these sections serves as a strategic roadmap and is laid out as follows.

Chapter 1: Chapter one outlines the motivation, state-of-the-art analysis, methodology and contributions made.

Chapter 2: Chapter two outlines the background information needed. This consists of a background in cryptography, power side channel analysis and power side channel measurement techniques.

Chapter 3: Chapter three outlines the background information concerning SoC design. Including processor design, interconnect busses, memory architectures and software design.

Chapter 4: Chapter three outlines the design and requirements of the full system architecture. The requirements include requirements for every stage of the system implementation.

Chapter 5: Chapter four outlines and the implementation of the full SoC system, software architecture and hardware architecture.

Chapter 6: Chapter five outlines the functional results of the SoC, the validation of the power side channel behaviour of the SoC in simulation and a security assessment of the implemented cryptographic engines.

Chapter 7: Chapter six consists of the conclusion. Which includes a summary and future work.

2 Hardware Security

This chapter provides the background concerning power side channel assessment and security. First, an introduction to Security, cryptography and the Advanced Encryption Scheme (AES) is given. Then an introduction to Power side channel analysis, leakage models, different ways of doing power side channel assessment and what quantifies as a good measurement setup is given.

2.1. Secure Systems and Security

The security of a system can be defined as the ability of the system to protect its components from unauthorized access, attacks, damage or disruption. To define a system as secure in the cybersecurity space, there are three criteria which are employed; Confidentiality, Integrity and Availability. They compose the so-called CIA-Triad which can be seen in figure 2.1. Confidentiality refers to limited access to sensitive information, integrity refers to a system being trustworthy and availability refers to the ability of a system to access data. [56]

INTEGRITY
AVAILABILITY
CONFIDENTIALITY

Figure 2.1: The CIA triad pyramid [56]

To adhere to the above three criteria, encryption or cryptography is employed. To fulfil confidentiality, encryption can be employed to obscure sensitive information, to fulfil integrity good encryption facilitates the system being trusted and to fulfil availability data can be accessed safely.

2.2. Cryptography

Cryptography refers to any method of protecting information with the usage of codes so that only those for whom the information is intended can read and process it[79]. The first example of cryptography can be traced back to the Egyptians using codes in the form of non-standard hieroglyphs, which were carved on monuments around 1900 BC[62]. Another example of ancient cryptography is the usage of a scytale by the Greeks and Romans to communicate secret war strategies. By using a stick with a specific diameter, only a person who knows the same diameter stick can read to message.

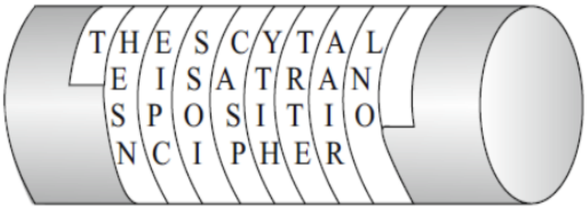


Figure 2.2: The Scytale transposition cipher [60]

A cypher refers to the process of abstracting data through the application of a particular algorithm that alters the data. In the field of cryptography, the original, unencrypted data that is intended for comprehension by another person is referred to as the Plaintext. Conversely, the data that is produced after encryption is implemented is known as the Ciphertext.

2.2.1. Cryptography classification

Over time, encryption and cryptography have evolved from simple algorithms and codes to complex mathematical operations which need to be performed in a specific order. Present day there a numerous encryption algorithms available, which can generally be categorized into three categories; **symmetric, asymmetric and hash**.

Figure 2.3 shows the classification of cryptographic algorithms with examples per category. The distinction between the three categories is their usage of a key. Modern cryptographic algorithms often use a key to perform encryption and decryption. The key can be seen exactly like a door key, only one successfully performs encryption. The most important part of encryption is thus to keep this key hidden.

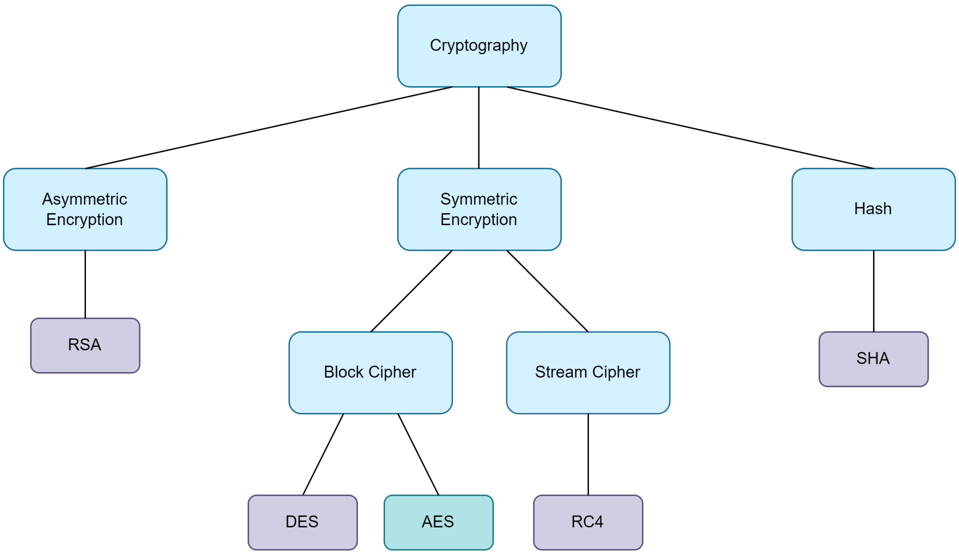


Figure 2.3: Classification of Cryptographic algorithms with examples [6]

Between the three categories, the functionality and usage of the key is what differentiates them.

- **Symmetric** In symmetric encryption, the key for encryption and decryption is the same. This property means that the key must be shared between the encryption and decryption urging the need for a secure key. Withing symmetric encryption the distinction can be made between block and stream cipher. These are differentiated by how the data is processed. **Block Cipher** processes data per block, meaning the input text is split up into blocks and then encrypted/decrypted per block. **Stream Chiper** perform encryption and decryption on a bit-by-bit bases.
- **Asymmetric** In asymmetric encryption the key for encryption and decryption is different meaning that keys don't have to be shared between encryption and decryption.

- **Hash** Hash algorithms don't use keys, they use mathematical operations to do a one-way encryption of the data. The one-way indicates that the operation is irreversible. Use cases of hash functions include the following; Generating checksum and generating of pseudo-random numbers.

Security of Cryptographic Systems
For modern-day encryption systems, two parameters are important to determine the security of cryptographic algorithms; Confusion and Diffusion [16]

Confusion: Confusion is the concealment of the relationship between the secret key and the ciphertext. [16]

Diffusion: Diffusion is the complexity of the relationship between the plaintext and the cipher text.

2.2.2. Advanced Encryption Standard (AES)

In response to increasing security demands, extensive evaluations led to the selection of Advanced encryption standard(AES)[17] by the Nation Institute of Standards and Technology (NIST) as the standard for encryption. Before being adopted as the official advanced encryption standard, what is known as AES was introduced as the Rijndael algorithm. Rijndael was developed by Vicent Rijmen and Joan Daemen in 1998 as the follow-up to the then data encryption standard (DES). As computers evolved, the 56-bit key length of DES became not secure enough anymore [19]. Unlike DES, AES support three key lengths: 128,196, and 256 bits. Making it harder to crack. The purpose of the Rijndael algorithm was to design an algorithm that was secure to all known attacks, efficiently implemented across all platforms and simple by design[17].

AES is a symmetric block cipher. As mentioned above, the word symmetric here indicates that the key for the encryption is the same as is used for the decryption. The word block cipher here means that the input data is encrypted per block (in AES this can be 128 or 256 bits) rather than one bit at a time. Figure 2.4 shows the encryption steps on the left and the decryption steps on the right. The AES algorithm consists of four main operations: SubBytes, ShiftRows, MixColumns and the addition of a Roundkey.

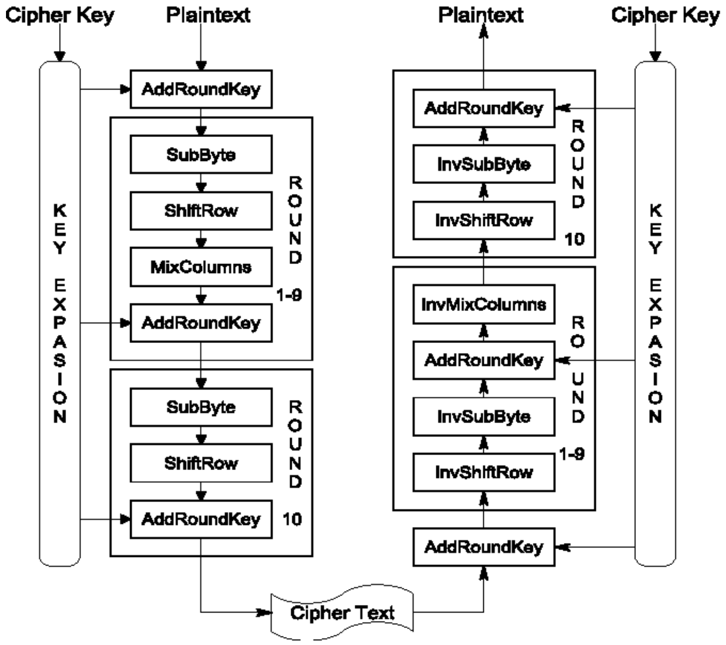


Figure 2.4: Advanced Encryption Standard Algorithm [43]

SubBytes
The subBytes also known as S-box substitution is the first step in the AES algorithm. The 128-bit input block comes in as a four-by-four-byte matrix (4x4 matrix where each value is 8 bits, reaching 128 bits).

It consists of a simple look-up table (also known as the S-box or lut) and a substitution. The function of subBytes is the remapping of the input matrix to another value. This is done using a look-up table and then using the input byte values to index it to a new value.

Looking at figure 2.5, imagine the value of a2,2 is *00101010*, then the first four bits will indicate the x location in the look-up table and the last four bits indicate the y location. Looking at the imaginary lut at position (0010, 1010) gives us the value b2,2. This value is substituted in the output block.

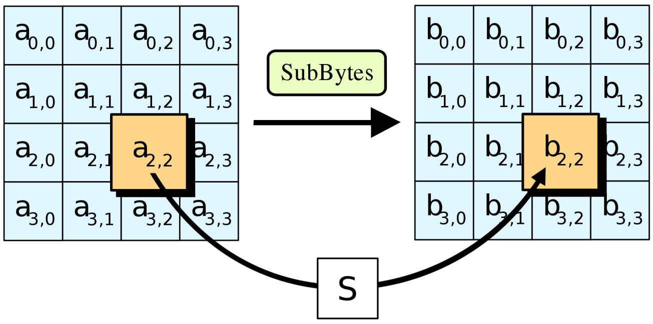

$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
SubBytes
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$
S

Figure 2.5: Substitution of bytes with S-Box look up [83]

Security of SubBytes

The SubBytes operation works on both confusion and diffusion. The splitting of the input bytes and remapping based on bit values indicates that even a single bitflip will result in a random different output.

Next to that SubBytes the splitting of the input blocks into bytes ensures that the correlation between input and output data is more complex. Output data now correlates on a byte-to-byte level rather than all 128-bit at the same time.

Shift Rows

After the subBytes operation, the next operation is the shift rows. The 128-bit input block comes in as a four-by-four-byte matrix. Every row in the input matrix is shifted left based on the row number. So row 0 is shifted left by 0, row 1 is shifted left by 1 and so on. Figure 2.6 shows the shifting of each row by its respective row number.

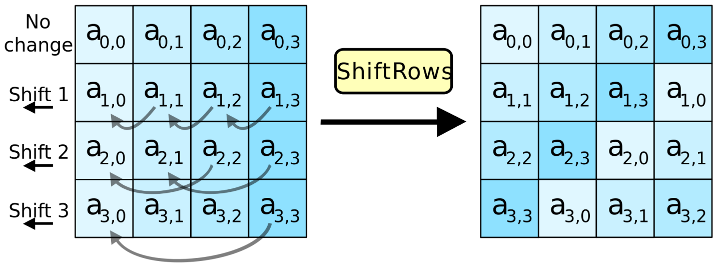

No change
Shift 1
Shift 2
Shift 3
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
ShiftRows
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,0}$
$a_{2,2}$ $a_{2,3}$ $a_{2,0}$ $a_{2,1}$
$a_{3,3}$ $a_{3,0}$ $a_{3,1}$ $a_{3,2}$

Figure 2.6: Shift rows operation [83]

Security of Shift Rows

The Shift Rows operation works mainly on diffusion. By re-arranging the rows per byte, the relation between the plaintext and the cipher becomes more complex. Changing the input plain text data has a rippling effect on multiple output bytes on the output.

Although Shift Rows mainly work on diffusion, it also adds to confusion. By re-arranging the byte position in each row an extra layer of complexity is added to the encryption algorithm.

MixColumns

After the shift rows, the next operation is the mix columns. The mix columns are a complex mathematical transformation in which each column of the state matrix is independently operated. For each column, a mathematical transformation is applied using a fixed set of coefficients. The transformation involves treating the column as a vector and multiplying it with a fixed vector. The byte results then are mixed and combined (XOR). A new byte is then calculated for each output column byte. Figure 2.7 shows the abstract operation between an input column, a fixed vector C(x) and the output vector.

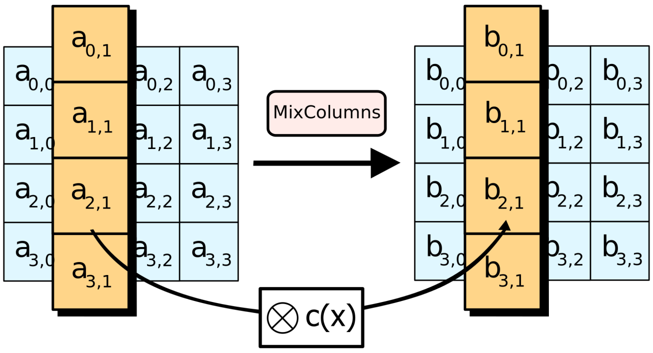

$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
MixColumns
$\otimes$ c(x)
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

Figure 2.7: Mix Columns operation [83]

Security of Mix Columns

The mix columns operation mainly contributes to confusion by the introduction of complex mathematical operations. The fixed vector coefficient and multiplication on a row base greatly disconnect the input bytes from the output bytes. Changing the input by one bit results in a non-linear change in multiple output bytes.

Roundkey addition

Looking at figure 2.4 shows that before the first round and at the end of every round there is the roundkey addition. The round key operation introduces the round key, which is a derivative from the original key and adds this. The AddRoundKey is an XOR operation between the roundkey matrix and the input data byte matrix. The bitwise XOR ensures that if the input data or the key is changed, the output data will be different. Figure 2.8 shows the XOR'ing of the round key matrix with the corresponding location in the input data matrix.

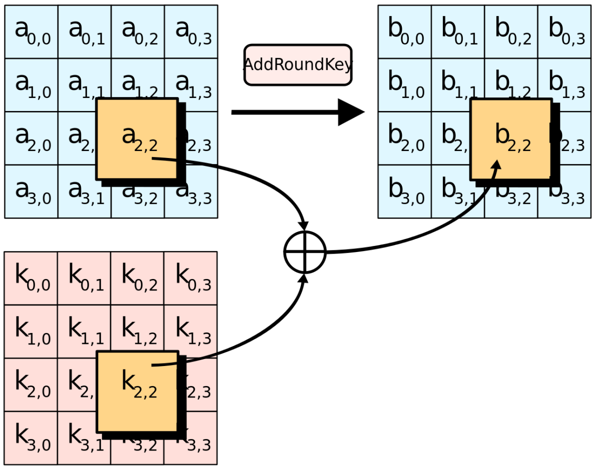

$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
AddRoundKey
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$
$\oplus$
$k_{0,0}$ $k_{0,1}$ $k_{0,2}$ $k_{0,3}$
$k_{1,0}$ $k_{1,1}$ $k_{1,2}$ $k_{1,3}$
$k_{2,0}$ $k_{2,2}$ $k_{2,3}$
$k_{3,0}$ $k_{3,1}$ $k_{3,2}$ $k_{3,3}$

Figure 2.8: Mix Columns operation [83]

Security of Round-key addition

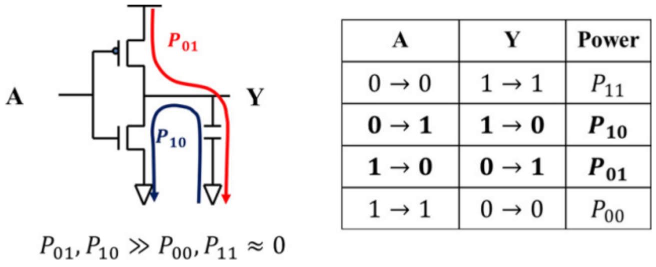
The round key addition ensures that the input data is mixed with the key data thus ensuring that the result is unique and dependent on a unique key. This contributes to the confusion as the complexity and even with the cipher text and knowing the algorithm, deriving the key is still difficult.

2.3. Power side channel analysis

Power side-channel analysis (PSCA) consists of exploiting and characterizing power side-channels to recover encryption keys. Power side-channel analysis consists of observing the power consumption of a chip, during encryption and decryption. Research has shown that power consumption can be directly correlated to the data being processed which, when analyzed, can lead to the stealing of the encryption key.[69]

For performing power side-channel analysis, power measurements need to be taken. Each power signal that is measured is called a power trace. Usually, large amounts of power traces are needed to fully analyse the design. The power analysis and power measurements are quite powerful due to the nature of the chip design. In hardware, millions of metal-oxide-semiconductor field effect transistors (MOSFETs) are packed per sq-mm. Each of them carries a charge (Coulombs) and consumes power. There are typically two factors that affect the power consumption of a device, static and dynamic power. Static power can also be observed as leakage power which is power that the device or transistor consumes when it is in its off state. Dynamic power is the power that a transistor consumes during switching activities (charged ('1') to discharged ('0')).

During power side channel analysis this dynamic power is of the greatest interest. It tells us the switching activity of a device during encryption, this can translated back into what operation is performed.[11]. An example of this is the Sbox substitution of AES. If the input and output are known, it is possible to reconstruct the operation by looking at how many transistors switched during the operation and to what state. To explain this further, figure 2.9 shows a simple inverter, if a '1' is put on the input, a '0' will come out and vice versa. The power consumption of that inverter can be modelled in each of the 4 states; The '1' off state, the '0' off state, the $0 \rightarrow 1$ transition and the $1 \rightarrow 0$ transition.



A	Y	Power
$0 \rightarrow 0$	$1 \rightarrow 1$	P_{11}
$0 \rightarrow 1$	**$1 \rightarrow 0$**	**P_{10}**
$1 \rightarrow 0$	**$0 \rightarrow 1$**	**P_{01}**
$1 \rightarrow 1$	$0 \rightarrow 0$	P_{00}

Figure 2.9: Power consumption of an inverter between states [11]

2.3.1. Hardware leakage models

Theoretical security parameters like the previously mentioned confusion and diffusion, when translated to the hardware domain, don't necessarily ensure a secure design. To do a proper analysis of a hardware implementation, a leakage model needs to be created. These leakage models indicate what the analysis will show. To profile the hardware security of a device, different places of leakage can be observed.

- Power profile of each performed operation: More complex operations that involve a lot of variables to change have a higher power consumption than simpler operations
- Timing profile of each performed operation: In general, the more complex the operation, the longer it will take.
- Hamming distance: Hamming distance refers to the amount of different bits between two binary sequences. As the amount of '1's in an operation means the operation consumes more power, knowing the hamming distance between the guessed value and the actual value, secret data can be extracted.
- Hamming Weight and intermediate values: Hamming weight counts the number of '1' bits in a binary sequence and intermediate values refer to the data values that are processed within a

cryptographic algorithm during its execution. These intermediate values can include key expansions, intermediate results of computations, or any other data, decryption, or other cryptographic operations. When the secret data contains more ’1’s (high Hamming Weight), certain operations consume more power than when the data contains more ’0’s (low Hamming Weight). With the extraction of the intermediate values and the hamming weight of an operation, keys can be extracted.

2.3.2. Power side channel Classification

Power side-channel analysis is widely regarded as the most powerful side-channel attack. Within power side-channel attacks, multiple variations exist. Figure 2.10 shows the classification of all power side-channel analysis methods. A distinction can be made between profiled and un-profiled analysis:

- ***Profiled analysis*** Profiled analysis is when the attacker has access to the device for an extended period and can run different key-data inputs and gather traces for each case.
- ***Un-profiled analysis*** Un-profiled analysis is when the attacker does not have access to the device but instead relies on generic models of power consumption.

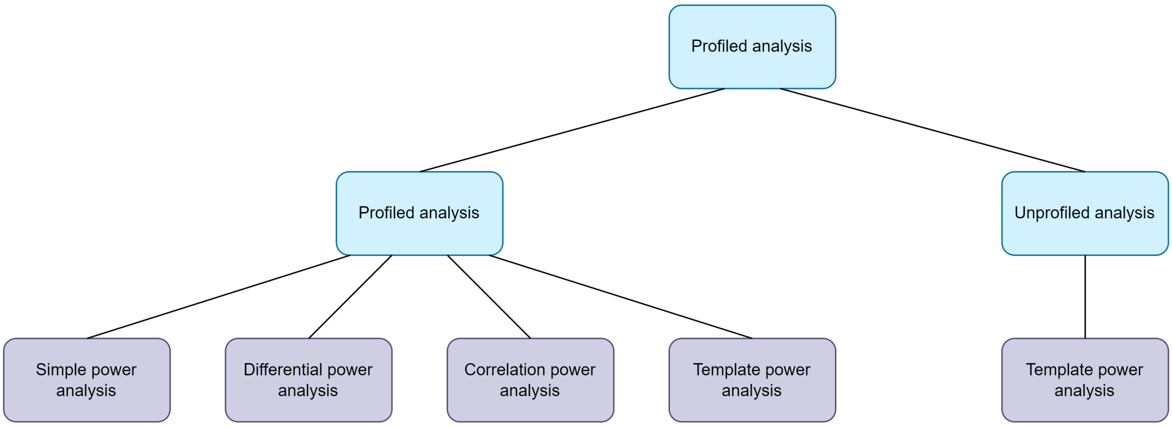


Figure 2.10: Classification of Power side channel analysis

2.3.3. Simple power analysis (SPA)

The simple power analysis is, as the name suggests, simple. By directly observing the power consumption of a chip over time, the observer can see fluctuations in the power line. By understanding what instructions are being performed on the chip, the correlation between power fluctuation per instruction/operation can be made [69]. Figure 2.11 shows an example of a fluctuating power line over a full 10 rounds inside a complete AES operation. The power fluctuates in one round depending on what operation is run.

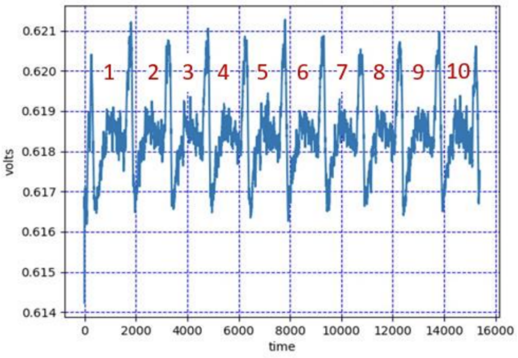


Figure 2.11: Example of SPA on an entire AES operation [69]

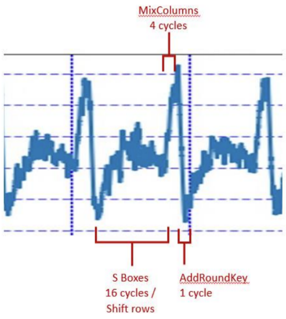


Figure 2.12: Breakdown of voltage difference per operation inside AES [69]

Every collection of power consumption measurements over time is referred to as a trace. Looking at figure 2.11 again, it becomes visible that the variations in voltage are millivolts. Next to that, the current difference average is in the microvolt range(e.g., µA). In practice, much noise is induced by the measuring device itself. Luckily, by collecting a lot of traces, the noise can be filtered out.

2.3.4. Differential power analysis (DPA)

Instead of observing large-scale power variations and correlating that to operations being executed, it is also possible to correlate the power variation over different traces to different key inputs. With differential power analysis (DPA) the observer observes different traces corresponding to a different inputted key values (guess). The power variation per key guess then correlates to how wrong the guessed key is. In other words, DPA uses a univariate statistical power model to observe the likelihood of the key based on power measurements [47], a guessed key value and a known output cipher text. Figure 2.13 shows an example of three different DPA traces and one reference at the top.

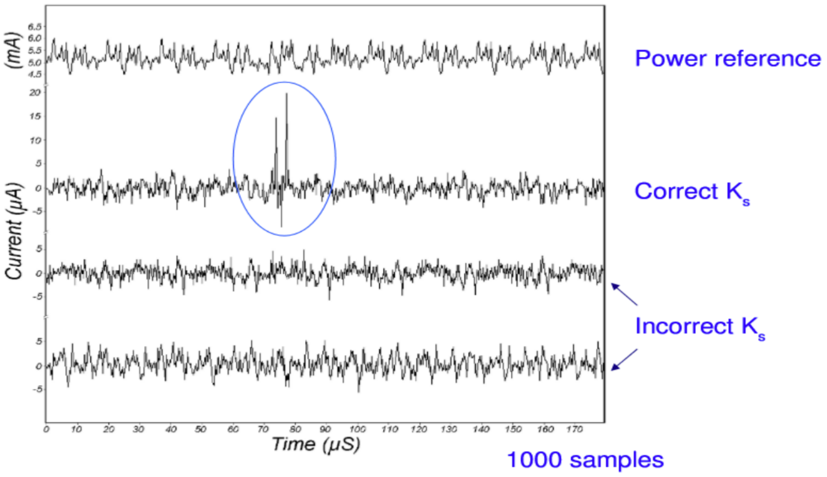


Figure 2.13: Differential power analysis with one correct key guess and two wrong guesses [47]

DPA procedure

Differential power analysis is powerful in its complexity and resilience against previous countermeasures. DPA start with the collection of m number of traces. Every trace is an array of power measurements per time stamp $T_i[j]$ where the i indicates the trace number and j indicates the time sample in that trace. m is suggested to be around 1000 but increasing sample size will lead to better results. Every $T_i[j]$ also gets its corresponding output ciphertext recorded and categorized as C_i.

It is now possible to design a function which can model the success of a key guess or how many bits a trace has guessed correctly. Imagine a function $D(C_i, K_n)$ which correlates a key guess to a correct output. In simpler terms, function D will return a "1" if the key guess leads to the correct cipher text output and a "0" if it doesn't. This probability of the outcome can be described for the collection of traces as follows.

$$P(\text{"}1\text{"}) = \sum_{i=1}^{m} D(C_i, K_n) \tag{2.1}$$

$$P(\text{"}0\text{"}) = \sum_{i=1}^{m} (1 - D(C_i, K_n)) \tag{2.2}$$

Then from that, the average power of a trace can be determined from the total power of the trace times the total time ($D * T_i[j]$) over the power at the measured point. This will give the power of a successful full bit guess and the power of an unsuccessfully bit guess.

$$Pwr(''1'') = \frac{\sum_{i=1}^{m} D(C_i, K_n) * T_i[j]}{\sum_{i=1}^{m} D(C_i, K_n)} \tag{2.3}$$

$$Pwr(''0'') = \frac{\sum_{i=1}^{m} (1 - D(C_i, K_n)) * T_i[j]}{\sum_{i=1}^{m} (1 - D(C_i, K_n))} \tag{2.4}$$

These equations give the success or failure at each point in time per trace. To get a full prediction at each point in time, the above can be subtracted.

$$T[j] = \frac{\sum_{i=1}^{m} D(C_i, K_n) * T_i[j]]}{\sum_{i=1}^{m} D(C_i, K_n)} - \frac{\sum_{i=1}^{m} (1 - D(C_i, K_n)) * T_i[j]}{\sum_{i=1}^{m} (1 - D(C_i, K_n))} \tag{2.5}$$

The correct key in equation 2.5 will then show up visually as a peak. It indicates the point where the success is maximum and thus the failure is minimum, generating the highest possible power difference.

2.3.5. Correlation power analysis (CPA)

Furthering the work of differential power analysis and template analysis the next step is the correlation power analysis. Rather than assessing traces concerning each other or modelling noise in-depth, correlation power analysis works based on correlation (surprise).

Correlation can be described as a statistical measure of the size and direction between data. Figure 2.14 shows different types of correlation, for example, if variable A increases then variable B increases as well indicating a positive correlation between them. The same can be done for the power traces that are collected. When guessing a key, power is only consumed when a bit is a logical ”1”, not the amount of bits that are present. This can be visualized like in figure 2.15. Five different numbers in an 8-bit system generate the same amount of ”on” LEDs. Meaning their power consumption is the same. This means that the model can’t be based on just the amounts of ’1’s in the trace. The model described before is called the hamming weight model which is grouping the amount of non-zero entries in a set.

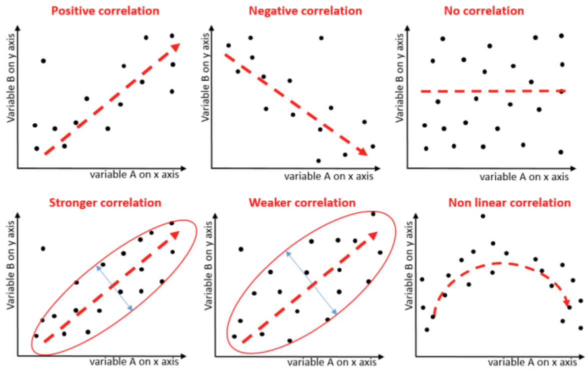


Figure 2.14: Correlation patterns [50]

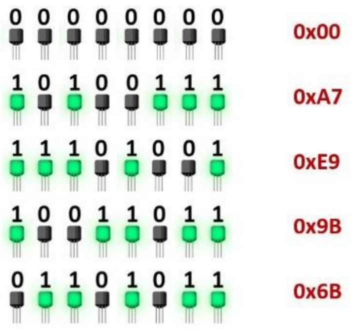


Figure 2.15: 8 bit numbers with same power density [69]

CPA procedure

The CPA procedure is simple but complex in mathematical terms. As the scope of this thesis is on the assessment of power side channel behaviour rather than the complexity of the analysis algorithm, the mathematical approach is replaced by a visual simple approach.

Figure 2.16 shows the analysis setup. Known input data is sent to the device that is being attacked. Then power traces are recorded and that will give us output data with corresponding power measurements (trace). For each input data, a key value is then guessed and the key value and input data are XOR’ed to create a hypothetical output value.

The hypothetical output values are then binary wised compared and their hamming weight is recorded (amount of ’1’s). Meaning that for every data value per hypothetical key, the hamming weight is recorded. This can be seen in figure 2.17.

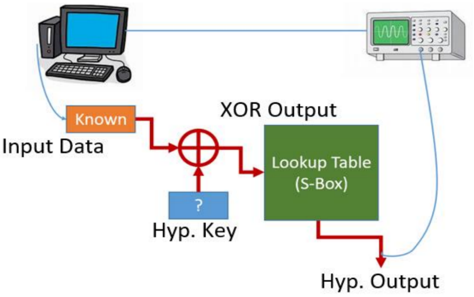
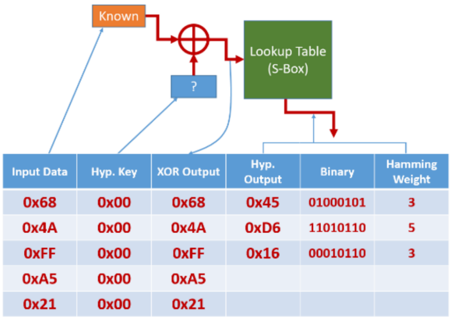


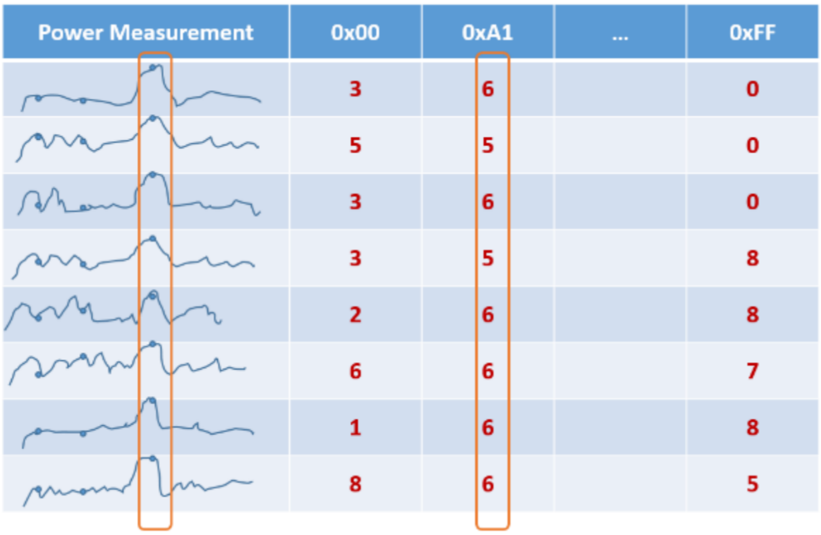
Figure 2.16: CPA setup [69]



Input Data	Hyp. Key	XOR Output	Hyp. Output	Binary	Hamming Weight
0x68	0x00	0x68	0x45	01000101	3
0x4A	0x00	0x4A	0xD6	11010110	5
0xFF	0x00	0xFF	0x16	00010110	3
0xA5	0x00	0xA5			
0x21	0x00	0x21			

Figure 2.17: Visualization of CPA where each column is a different key guess[69]

Then the final correlation comparison can be made. Every key value and their corresponding hamming weights are compared to the original recorded power trace. Then the one that is the closest to the power trace is predicted to be the result. Figure 2.18 shows this final comparison and the power traces on the left.



Power Measurement	0x00	0xA1	...	0xFF
	3	6		0
	5	5		0
	3	6		0
	3	5		8
	2	6		8
	6	6		7
	1	6		8
	8	6		5

Figure 2.18: Final comparison between different key possibilities and power traces [69]

2.3.6. Machine learning based power analysis

Most recently, the newest approach to power analysis is with the use of machine learning. Machine learning is the art of pattern recognition based on a large dataset. In power-side channel analysis, machine learning can be used in two stages.

Pattern Recognition and Leakage Models

- **Feature Extraction** Machine learning algorithms can help in extracting relevant features from the power consumption data. These features might include mean, standard deviation, peak values, or frequency components.
- **Leakage Models** Machine learning models can be trained to understand the relationship between power consumption patterns and cryptographic operations. These models can capture leakage behaviour, allowing the identification of sensitive operations based on power consumption patterns.

Side-Channel Attack

- **Classification** Machine learning algorithms, especially classifiers like Support Vector Machines (SVM), Decision Trees, or Neural Networks, can be trained to classify power consumption traces into different classes representing different operations or keys.

- **Template Attacks** Machine learning techniques, such as clustering algorithms (k-means, DBSCAN), can be used to create templates that represent the power consumption profiles associated with specific cryptography keys or operations. During an attack, incoming traces can be compared with these templates to infer the key being used.

Figure 2.19 shows the position of machine learning in the context of power side-channel analysis.

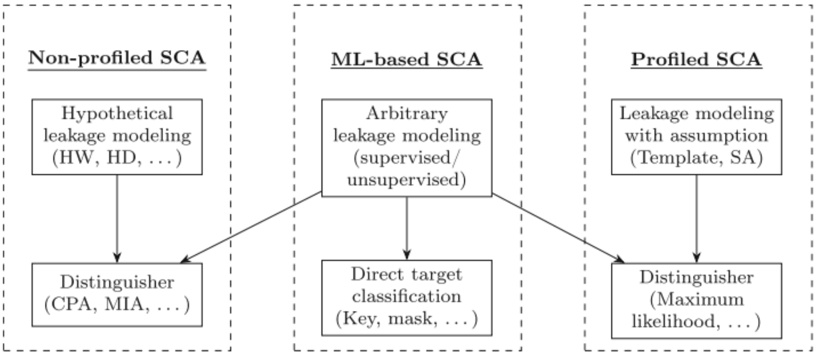

Non-profiled SCA
Hypothetical leakage modeling (HW, HD, ...)
Distinguisher (CPA, MIA, ...)
ML-based SCA
Arbitrary leakage modeling (supervised/ unsupervised)
Direct target classification (Key, mask, ...)
Profiled SCA
Leakage modeling with assumption (Template, SA)
Distinguisher (Maximum likelihood, ...)

Figure 2.19: Machine learning in the context of power side channel analysis[37]

2.4. Power side channel Countermeasures

After investigating different power side channel analysis techniques to break encryption the next question now is: *"How can a system be protected against these attacks?"*. Throughout the years, whenever an attack was designed, a countermeasure was developed. The countermeasure would then lead to a new attack etc. This means that various countermeasures exist for each analysis type. First, it is important to understand what defines a good countermeasure.

2.4.1. Quantification of Countermeasures

Before explaining different ways of protecting a system against power side channel analysis it is important to understand how to quantify and compare different countermeasures. All power-side channel analysis techniques depend on retrieving information from power traces by filtering out/modelling the noise and unwanted data. This means that the more clear the system's data is, the better the analysis techniques will work. There are generally three techniques that can be employed to quantify the security of a countermeasure (SNR, Timing disarrangement and Signal Masking and scrambling). For this thesis, the Signal-to-noise ratio and timing disarrangement is the main parameter, but all measures are briefly mentioned.

Signal to Noise Ratio (SNR)

This ratio is called the signal-noise ratio (SNR). Figure 2.20 shows an example of four different signals with different SNR ratios. It can be observed that the higher the SNR, the more signal over noise the result is. Mathematically the SNR can be expressed as follows.

$$SNR = \frac{P_{signal}}{P_{noise}} \tag{2.6}$$

Generally in electronics, the SNR would be as high as possible, resulting in a clean signal with the least amount of noise. For power-side channel security, however, and decrease in SNR means a less clear signal[31]. A less clear signal leads to an increase in the difficulty of retreivin the key with power side-channel analysis. With this the following statement can be made: ***The lower the SNR, the more secure a system is to power side channel assessment***.

Timing Disarrangement

Another parameter to quantify countermeasures is timing disarrangement[31]. Most side channel analysis techniques are based on the analysis of different data sets of points which are assumed to be

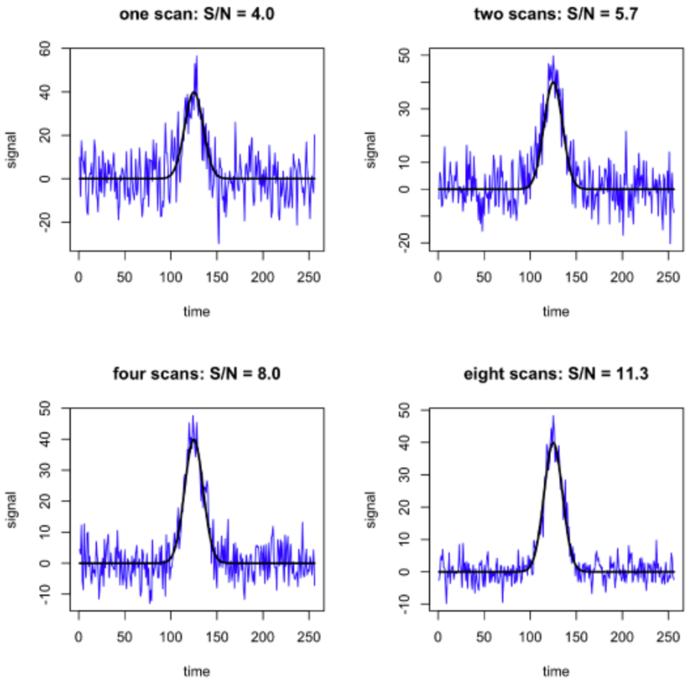


Figure 2.20: Four different signals with four different SNR ratios [36]

sampled at the same time. By scrambling the data points in time and misaligning them, it becomes more complicated to process the data and retrieve useful information.

Signal Masking and Scrambling

One of the last parameters that can be considered is the hiding of entire parts of the cryptographic operation in the power trace. Unfortunately, this strategy is usually very specific, requires expert knowledge and involves significant changes to the cryptographic algorithm at the additional cost of reduced performance and increased resource consumption.[31]

2.4.2. Classification of Countermeasures

Power side-channel countermeasures all operate based on working on one of the afore mentioned parameters to increase security. Generally, they can be grouped into two groups; Injection of noise (NI) or reduction/suppression of correlation signature (randomization of data). [18]. That being said, a lot of different countermeasures exist, a few are outlined below.

Algorithmic Countermeasures

Masking Masking refers to the insertion of random values (masks) in the cryptographic operation, meaning it makes it more complex to retrieve the original operation. This decreases the signal-to-noise ratio and works on the signal masking and scrambling as well. Examples of masking are [38] where a fixed-set masking is applied on top of the original algorithm to increase security. Another example of masking is the introduction of random code injection [7].

Dithering Dithering adds controlled noise to the power consumption, obscuring the correlation between power traces and data. Examples of dithering are using random fast voltage dithering (RFVD) voltage regulators to add random variations in the power lines. [75].

Blinding Similar to masking, blinding involves introducing constant power leakage to cryptographic operations to hinder power-based attacks. This can result in different keys having the same leakage pattern thus making it harder to find the right key.

Architectural Countermeasures

Dual-Rail Logic Dual-rail logic encoding ensures that each bit is represented as two complementary signals, reducing information leakage.

Shuffling Shuffling involves changing the order of operations or data to obscure the power consumption patterns.

Power Gating Power gating involves selectively turning off parts of the device when they are not in use to reduce power leakage.

Randomization Techniques

Random Delay Introducing random delays in cryptography operations disrupts the correlation between power consumption and data.

Random Function Execution Randomly selecting functions or instructions from a set of alternatives adds uncertainty to power traces.

2.5. Power side channel security measurement

After discussing power-side channel leakage, power-side channel analysis and countermeasures it is also important to understand how these power-side channel measurements are taken in a practical sense. Figure 2.21 shows a block diagram of a generalized power side channel measurement setup.

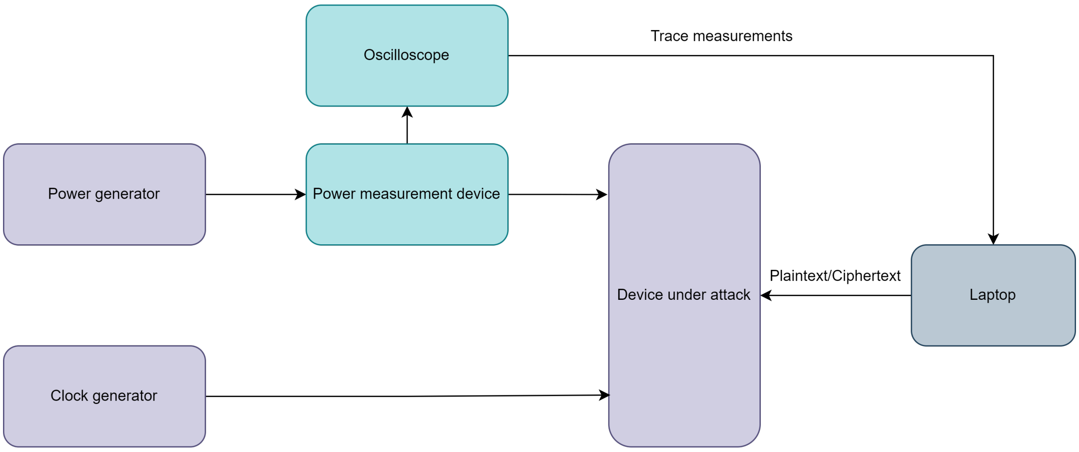


Figure 2.21: Generalized measurement setup of power side channel [51]

The key components of the above setup are described as follows

Power generator A power generator generates the correct power for the device under attack. Usually, the power generator is in line with the power measurement to measure the power of the device under attack's power consumption.

Clock generator A clock generator creates the clock/time base for the device under attack. It can either be internal or external depending on the device under attack.

Power measurement device The power measurement device is the bridge between the device under attack and the power generator. It is directly connected to the device's main power line and thus can collect the power traces directly at the source. Usually, an **Oscilloscope** is attached to observe the power lines and send them to a laptop/personal computer (PC).

Device under attack The device under attack is the cryptographic engine that is encrypting and decrypts values received from an external laptop/PC.

2.5.1. Quantification of the quality of a measurement setup

The main purpose of a measurement setup is to facilitate the gathering of valuable data. For power side channel assessment this means achieving the highest Signal to noise ratio possible. This leads to the following conclusion: **”A measurement setup should not interfere with the measurement and aim to add as little noise to the measurement as possible.”** The addition of noise can be further quantified down to the following parameters. [51]

Power generator noise Power generators usually include buck/boost converters who, by nature, generate a lot of switching noise (Higher frequency noise). This power generator noise should not interfere with the measurement.

Clock generator noise Clock generator noise can lead to clock instability and timing issues. Meaning that an unstable clock generator leads to misaligned measurements.

Introduction of timing dissarangement The main setup should not add any randomization or timing dissarangement to the measurement.

Power measurement device quantization noise Quantization noise in a measurement setup leads to small rounding errors in the measurements meaning that readings are not always as accurate.

Overall noise performance inside measurement frequency The main purpose of the measurement setup is to not interfere with the measurement, this means that all the system noise needs to be filtered out at the frequency that the measurements are taken at.

3

System on Chip Overview

The design of a custom system on chip (SoC) or ASIC requires in-depth knowledge of computer and system architecture. This chapter aims to provide an overview and introduction to all components that make up a complete chip architecture. First and brief overview of a general chip architecture is given, and then the different sub-components are explained. Lastly, the software design flow of an SoC is explained.

3.1. SoC design

Roughly 65 years ago, the first general-purpose computer SoC was designed. That moment marked a turning point in modern-day electrical and computer engineering. Ever since SoC design has transformed and evolved from designing simple personal computers to increasingly smaller and more complex chips. Almost every modern-day device is equipped with an SoC. These SoCs are designed by companies who have been at the frontier of the computer evolution, companies as Intel, Apple, and advanced micro Devices (AMD). Companies who have monetized their technology and advancements to ensure their livelihood. Making a custom SoC design is a complex multi-level puzzle which includes semi-conductor fundamentals, chip architectural components and software design.

Figure 3.1 shows an overview of a simple modern-day SoC. In the image, all the different blocks define different functions that the system can do. All these blocks can be generally subdivided into five groups. Figure 3.2 shows a simplified breakdown of the modern-day chip into a generic SoC. Some key architectural components can be briefly mentioned in this figure. At the heart of a chip lies the Central Processing Unit (CPU). The CPU is responsible for the execution of a specified set of operations. When writing a piece of software, this code is assembled into different operations which are sequentially executed by the CPU.

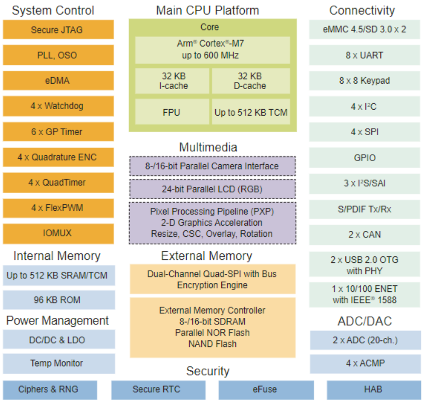


Figure 3.1: Block diagram of RT1050

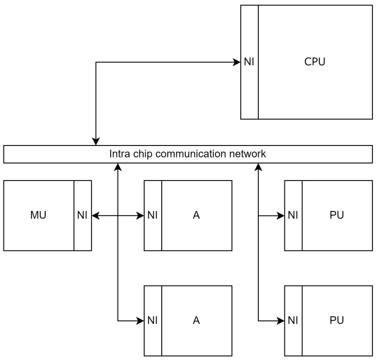


Figure 3.2: Simplified block diagram

Next to the CPU, four different components make up a chip. First a Memory Unit (MU), the memory unit is responsible for storing and loading data from and to other parts of the chip. In line with the memory unit, there is the intra-chip communication network. This network is responsible for facilitating all communication between different parts of the chip. This is facilitated by having Network Interfaces(NI) at every part.

Lastly, designated parts of a chip that are not the CPU can be divided into two parts. Accelerators (A) and Peripheral Units (PU). Accelerators are specific pieces of hardware that are designed to do one task very fast. This can include cryptography or data processing. Peripheral communication units provide a connection between the chip and the outside world. This can include USB ports or IO pins.

3.2. Processing Unit

With the ever-growing need for faster and more complex applications, CPU design has evolved from simple single-core processing units that can execute one instruction per clock cycle, to complex (multi-)core processing units with the possibility to schedule and parallelize instructions. All to facilitate the need for faster and more efficient applications. Figure 3.3 shows the evolution of CPU design, the rise of multi-core CPUs and the evolution of changing SoC's frequency range and power consumption. Here it becomes visible that designers are consistently searching out new techniques to facilitate faster processing.

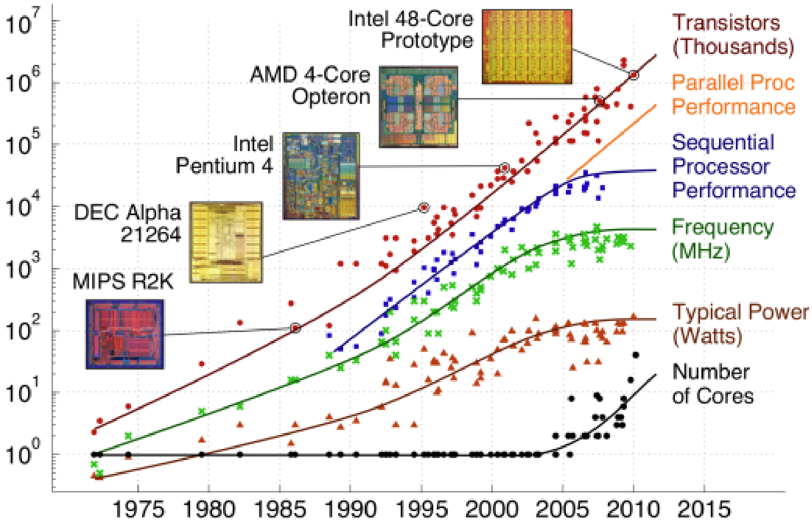


Figure 3.3: Evolution of CPU design [80]

The performance of a CPU can be quantified by the following parameters:

Throughput the speed at which a CPU can execute a program. This includes the frequency at which the CPU can run and how many instructions can be executed per clock cycle. This is modelled by the following equation:

$$\frac{seconds}{program} = \frac{instructions}{program} * \frac{cycles}{instruction} * \frac{seconds}{cycle} \tag{3.1}$$

Power dissipation The power consumed by a CPU during executing a program indicates the energy efficiency of the CPU. The higher the power consumption, the shorter the battery life of a phone and the shorter the life cycle of the CPU. The dynamic power dissipated by a CPU can be modelled by the following equation [53]:

$$P_{dyn} = \frac{1}{2}CV^2 f \tag{3.2}$$

The above parameters hide a subtle contradiction. If the end goal is high speed, the time per cycle (the clock frequency) can be increased. But that also increases the power dissipation. This indicates the biggest problem in modern CPU design. **”In CPU design there is no one size fits all solution, CPU design is often a compromise between speed, power, available area and many other parameters”**.

3.2.1. Instruction set architecture (ISA)

The key component of any computer is the central processing unit (CPU). The processing unit executes instructions. Instructions can be seen as the lowest level language a CPU can interpret. Examples of instructions are: ADD, SUBtract, MULtiply and DIVide. A piece of high-level code (for example in C), when run on a CPU, is assembled into an ordered list of instructions that that specific CPU can execute. An example of this can be seen below, listing 3.2.1 shows a simple C function that adds two numbers and returns the result. Listing 3.2.1 shows the exact equivalent of that addition into ARM core assembly (One flavour of assembly). In general, all high-level code can be assembled into a finite set of instructions.

```c
int add_numbers(int num1, int num2) {
    int result = num1 + num2;
    return result;
}
```

Listing 3.1: C example

```
.data
result: .word 0      @ Initialize a variable to store the result
.text
.global main

main:
    @ Input parameters:
    @ num1 in r0
    @ num2 in r1

    @ Add num1 and num2
    ldr r2, [r0]      @ Load num1 from memory
    ldr r3, [r1]      @ Load num2 from memory
    add r4, r2, r3    @ Add num1 and num2, store in r4

    @ Store the result in memory (result variable)
    str r4, result    @ Store the result in the "result" variable in memory

    @ Load the result from memory into r0 (return value)
    ldr r0, result

    @ Exit the program
    mov r7, #1         @ Exit syscall number
    swi 0              @ Invoke the syscall
```

Listing 3.2: Assembly example

The question now arises, what instructions can a CPU execute? CPU design usually starts with the Instruction Set Architecture (ISA), or what set of instructions is a CPU going to execute. The ISA can be seen as the computer architecture outline where the actual CPU design is an implementation of the ISA. The ISA plays a crucial role in determining the capabilities and strengths of computers. ISAs serve as the bridge between software and hardware components within a computer system, dictating the available instructions for a processor, the accessible registers, and the organization of memory. Figure 3.4 shows what layer the ISA is located at between the high-level code applications and the hardware transistor level.

ISAs can be practically divided into complex instruction sets (CISC) and reduced instruction set (RISC) based ISAs[12]. Complex instruction sets focus mainly on the reduction of the number of instructions a CPU needs to execute per program. This comes at the cost of an increase in the number of clock cycles per instruction and a more complex hardware design. Reduced instruction set-based ISAs aim to reduce the number of cycles per instruction at the cost of an increased number of instructions and

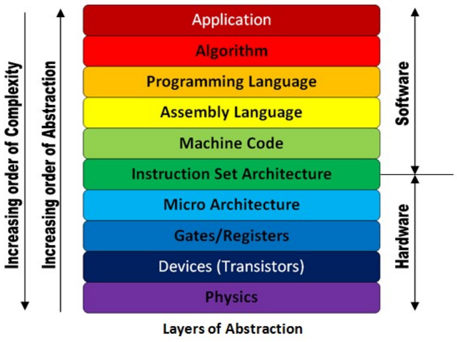


Figure 3.4: The ISA concerning software and hardware

simpler hardware per instruction. Examples of a RISC architecture are the ARM[73] architectures which can be seen in Apple™or low-level embedded cores. An example of a CISC architecture is the Intel x86 architecture which can be found in most laptops and PCs. For SoC design, RISC designs are often simpler and thus will be the focus of this thesis.

3.2.2. RISC-V architecture

With an ever-growing need for custom systems on chips, microprocessors can be found everywhere. From smartwatches to keychains to phones, custom SoCs are ubiquitous. Yet, trying to implement an existing ISA is physically almost impossible. Most ISA like ARM and x86 are proprietary, meaning licensing is needed. Licensing can cost millions of euros, months of negotiation and limits to customization and innovation. Therefore an urgent need for an open source ISA was growing. RISC-V (Reduced instruction set - Five) was introduced by the University of Berkeley as the new standard RISC open-source ISA [82].

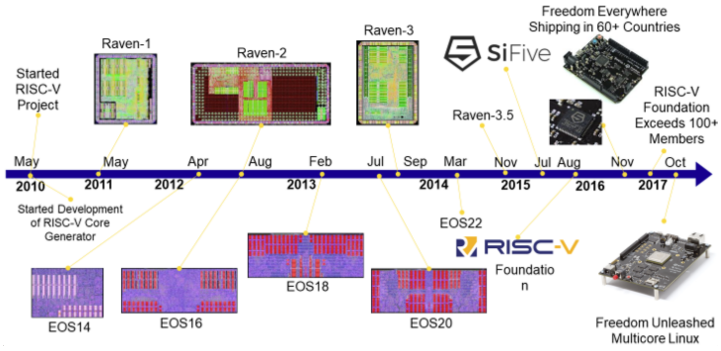


Figure 3.5: Evolution of RISC-V project [1]

(RISC-V) CPU architecture

In contrast to other ISAs, RISC-V is freely accessible, utilizable, and customizable by anyone. With this, there is a community-driven innovation rather than all the power being in the hands of a few companies. Figure 3.6 shows an implementation of the RISC-V core which is freely available for implementation. This means anybody can take the design and create their own SoC.

The architecture in figure 3.6 is quite common to most modern-day CPU architectures. Rather than being simple and executing one instruction after the other, they process instructions in stages to facilitate the execution of more instructions per time unit. The combination of these stages is known as a pipeline. Using the above architecture as an example the following stages can be identified.

Instruction Fetch In this stage, the CPU fetches the next instruction from memory. In the CPU, a special register the program counter (PC) holds the address of the next instruction in memory. When an instruction is fetched, the PC will increment itself to the next instruction that needs to be fetched.

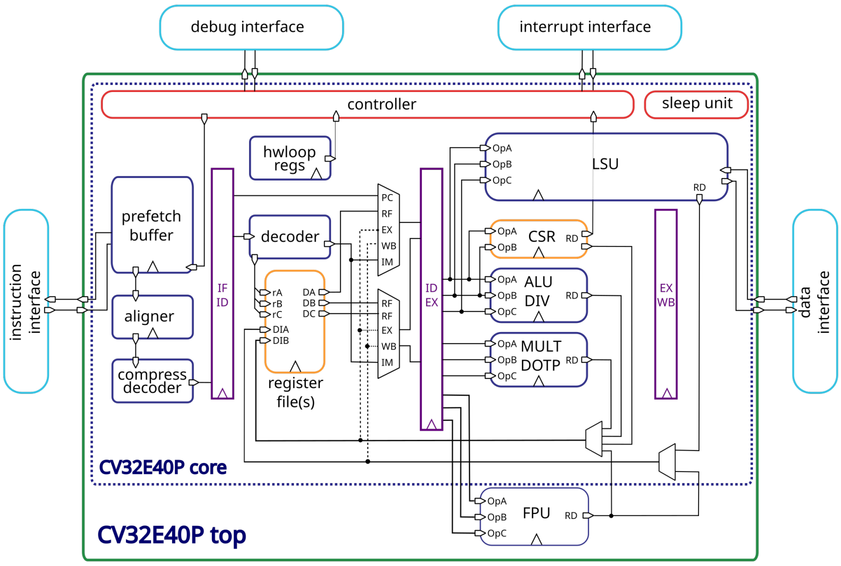


Figure 3.6: Example of (RISC-V) core [2]

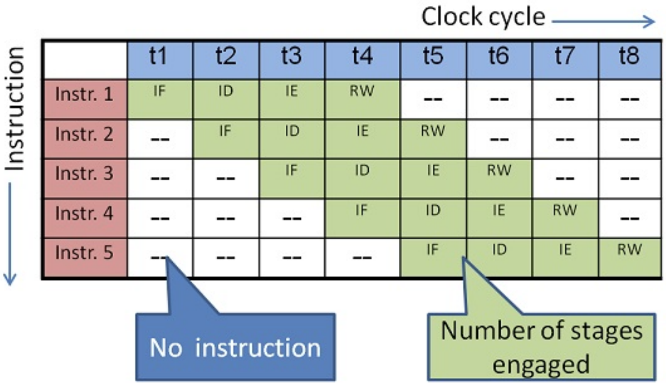
Instruction Decode In this stage, the CPU decodes the instruction into the operation that needs to be performed.

Execute In this stage the CPU executes the operation that was fetched. The result of the operation is computed and sent to the next stage.

Memory fetch In this stage, the CPU fetches or writes data to the data memory if the instruction that was executed involved memory. Any memory-related executions are performed in this stage.

Write back The last stage in the pipeline is the write back, in this stage the CPU writes back any result of the executed instruction to its register file or back to memory.

In figure 3.6 between each set of blocks, a purple block is shown. This block indicates a register meaning that when one stage is done with one instruction, another instruction can already be started. Figure 3.7 shows this concept in clock cycles. At t2 when the data from Instruction 1 has been fetched and pushed into the register between the IF-ID, instruction 2 can already fetched. Pipelining speeds up the processing of a set of instructions significantly.


Clock cycle
Instruction

	t1	t2	t3	t4	t5	t6	t7	t8
Instr. 1	IF	ID	IE	RW	--	--	--	--
Instr. 2	--	IF	ID	IE	RW	--	--	--
Instr. 3	--	--	IF	ID	IE	RW	--	--
Instr. 4	--	--	--	IF	ID	IE	RW	--
Instr. 5	--	--	--	--	IF	ID	IE	RW

No instruction

Number of stages engaged

Figure 3.7: Pipelining in CPU architecture [8]

3.3. Memory architectures

For computers, memory is a key component of a functioning system. With the rise of artificial intelligence and machine learning comes a need for faster and larger data sets and thus larger storage as close to the processing unit as possible. Current innovation techniques in computer architecture almost solely focus on more efficient and faster memory architectures as fetching and moving large amounts of data is the main bottleneck in fast applications. Various different memory types can be exploited for different applications. Figure 3.8 represents a basic classification of computer memory.

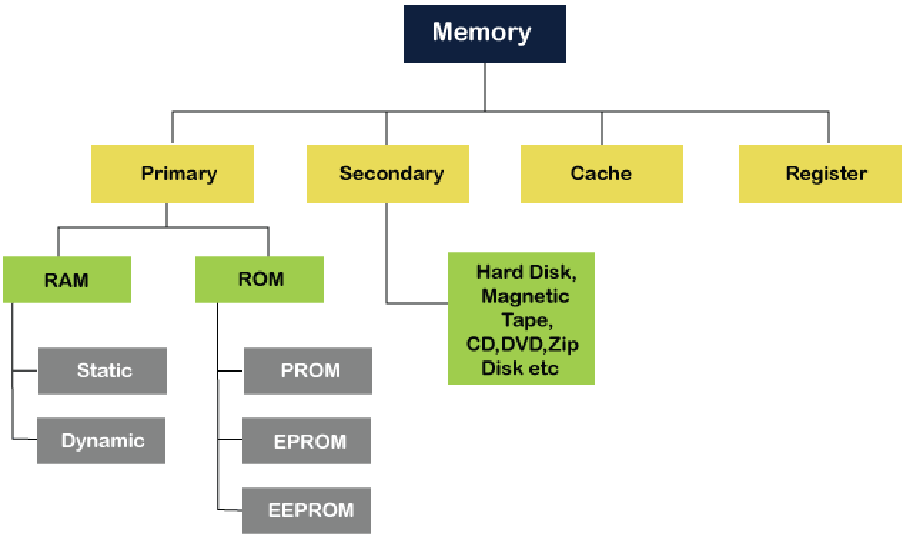


Figure 3.8: Classification of computer memory [3]

Differentiation between the different memory architectures can be made on the following key parameters:

- **Performance** the number of clock cycles it takes to load/store data and how fast the data can be moved inside the memory.
- **Size** How much data can a memory hold and the resulting area of that memory, the more data a memory can hold, the larger in physical size the memory is.
- **Location** Spatial location with regards to the CPU, faster memory is situated closer to a CPU

These parameters influence a chip designer's choice of what memory to select for their application. To understand more about the different types, each type will be discussed after which a comparison based on the above parameters will be made.

3.3.1. Primary Memory

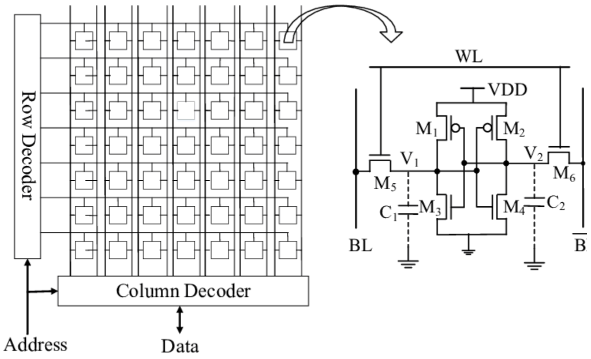
Primary memory can be defined as a chip's main memory. It usually has a direct connection to the CPU and is situated as close as possible to it. Primary memory contains all the instructions and data needed for the CPU to run its execution programs. Within primary memory, two types can be distinguished, random access memory (RAM) and read-only memory (ROM).

3.3.2. Random access memory

Random access memory acts as the CPU's internal memory. It stores data, programs, and program results from the moment the chip is turned on. RAM is volatile storage meaning that when the power is cut, the data in the memory will be lost. RAM can be viewed as the CPU's short-term memory where it can store data short-term for near-future use. There are two types of RAM widely used, SRAM and DRAM.

Static RAM

Static RAM also known as SRAM is used for the static storage of data. Each cell in a SRAM memory array can hold one bit of information. An SRAM cell is generally made up of 6 or 11 transistors. During the read of this bit, the bitlines get driven high (BL in figure 3.9) then the wordline is driven high (WL in figure 3.9). Depending on if the bit is a 0 or 1, either the BL or the B is pulled lower than the other, giving a slight voltage offset. If the cell is programmed to hold a ”1” then V2 is high, meaning that when WL is driving high, BL is bumped a bit higher than B. And the reverse for when a ”0” is programmed.



(a) A typical SRAM array. (b) A six-transistor SRAM cel

Figure 3.9: Physical layout of SRAM cell[30]

Looking at the size and complexity of the SRAM cell, it becomes clear that six or eleven transistors are needed per bit stored. Meaning it is not an efficient size-to-memory capability ratio. When it comes to speed, the SRAM cell is relatively fast as it is only determined by how fast the transistors can switch. When it comes to cost, having six transistors per bit means it is quite expensive to make.

Dynamic RAM

Dynamic RAM also known as DRAM is used for the dynamic storage of data. Each cell in a DRAM memory can hold one bit of information. One DRAM cell is made up of one capacitor and one transistor. The access transistor is connected to the word line and acts as a switch. The capacitor stores each bit of data as a negative or positive electrical charge. The memory state is read by sensing the stored charge on the capacitor via the bit line, which is set to the operating voltage/2 with the transistor closed. When the access transistor is on, the stored charge carriers flow into the bit line, which changes its potential. This voltage change is detected and amplified by the sense amplifier connected to the bit line [45]. Figure 3.10 shows the layout of a single DRAM cell and how its size has changed over the years.

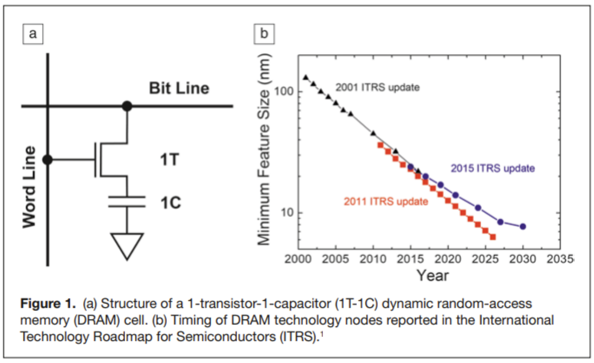


Figure 1. (a) Structure of a 1-transistor-1-capacitor (1T-1C) dynamic random-access memory (DRAM) cell. (b) Timing of DRAM technology nodes reported in the International Technology Roadmap for Semiconductors (ITRS).[1]

Figure 3.10: Physical layout and size of DRAM cell[45]

Looking at the difference in layout and number of components between SRAM and DRAM, it becomes clear that a DRAM cell is a lot smaller than the SRAM cell. Meaning that DRAM cells can be packed more efficiently leading to larger memory capabilities per area. The cost of this is that the DRAM access speed is determined by the speed with which the capacitor can charge and discharge as well as the speed with which the transistor can switch. Meaning that DRAM is significantly slower than SRAM.

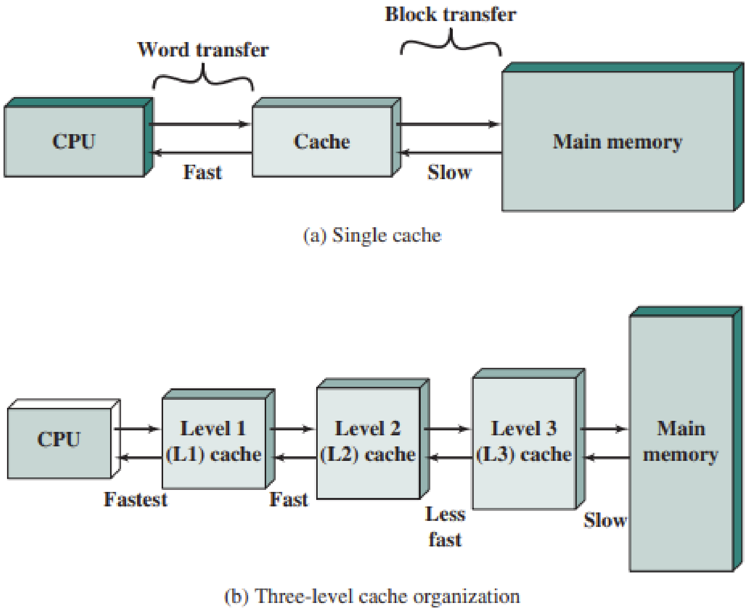
3.3.3. Read only memory

Read-only memory (ROM) is a specialized type of non-volatile memory. The name read-only memory refers to the fact that ROM is permanent memory that can not be erased even when power to the memory is cut off. The information stored in ROM memory is programmed once and can't be overwritten by the CPU. ROM is widely used for important device information and start-up sequence data, for example storing firmware or holding boot sequencing. The moment a CPU is powered on, the CPU will start reading the ROM boot data on how/where to find the first instruction to execute.

There are four main types of ROM available, programmable ROM, Erasable Programmable ROM, Electrically Erasable Programmable ROM and Flash ROM. All four are based on the same architecture but employ different strategies to facilitate some type of re-programming.

3.3.4. Cache

Cache memory is designed to reduce the latency between a CPU and the main memory by inserting itself between the two and acting as a closer buffer. Figure 3.11 shows where the cache is located. The theory of cache is simple; cache contains copies of parts of the main memory that are likely to be used by the CPU. When the CPU requests data, it is first checked if the data is in the cache. If it is in the cache then the data is outputted to the CPU, if it is not then the data is fetched from the main memory.[76] The benefit of this is the fact that, if the data is in the cache, the fetching of data is a lot faster.



(a) Single cache

(b) Three-level cache organization

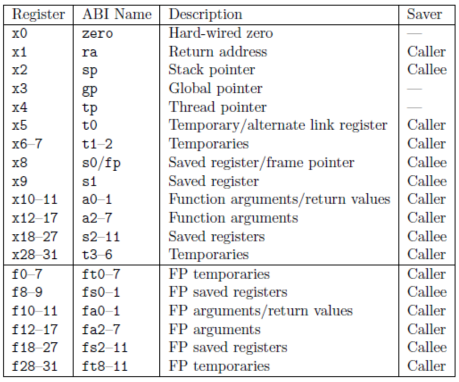
Figure 3.11: Cache with respect to the CPU and main memory [76]

Cache cells are often implemented as SRAM cells within a smart access protocol.

3.3.5. Registers

CPU registers are registers which are usually inside the CPU itself. They are used to store intermediate results or final results of operations before they are pushed to memory. Conventional CPUs have a finite set of registers which have a fixed purpose. Figure 3.12 shows an example of the registers inside a RISC-V CPU.

Each register is often directly connected to the CPU itself, meaning that having a lot of registers makes the CPU design more complex.



Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5	t0	Temporary/alternate link register	Caller
x6–7	t1–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller
f0–7	ft0–7	FP temporaries	Caller
f8–9	fs0–1	FP saved registers	Callee
f10–11	fa0–1	FP arguments/return values	Caller
f12–17	fa2–7	FP arguments	Caller
f18–27	fs2–11	FP saved registers	Callee
f28–31	ft8–11	FP temporaries	Caller

Figure 3.12: Registers inside a RISCV-CPU

3.3.6. Memory comparison

Based on the aforementioned parameters, the different memory types can be compared on their size, location and performance. Figure 3.13 shows the memory pyramid and the parameters. On the left, the vicinity to the CPU decreases going from registers to secondary memory. On the right, the performance increases from secondary memory to registers. At the bottom, the width of the pyramid slice indicates the maximum size of the memory.

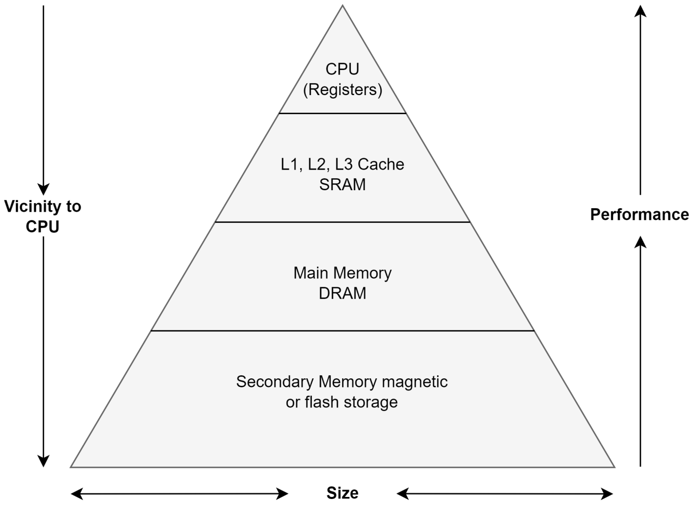


Figure 3.13: Classification of computer memory with regards to parameters

From this pyramid, some important conclusions can be established. Registers have the lowest latency and highest performance **but** they are also the smallest memory. Secondary memory is the slowest memory **but** it can hold the most data. These insights become of great importance when deciding what type of memory to put into a chip. But it should not be forgotten that there are more parameters affecting the decision, parameters like ”what memory is available” and ”what memory can fit inside the chip”.

3.4. Intra-chip Communication

The key to a fast and well-functioning chip is rooted in the intra-chip communication. Intra-chip communication is the way data is transferred between different parts of the chip. The speed at which data can be transferred from any peripheral part of the chip to the CPU dictates the overall speed at which the CPU can execute commands. When it comes to intra-chip communication there have been significant improvements, from simply putting custom routing architectures to designing complete networks and hierarchical systems. Figure 3.14 shows the evolution of different intra-chip architectures over the years. Where in the 1990's a chip only consisted of a few components, modern chips contain hundreds of components that need to be available on the chip. Therefore more complex and structured interconnect architectures were developed.

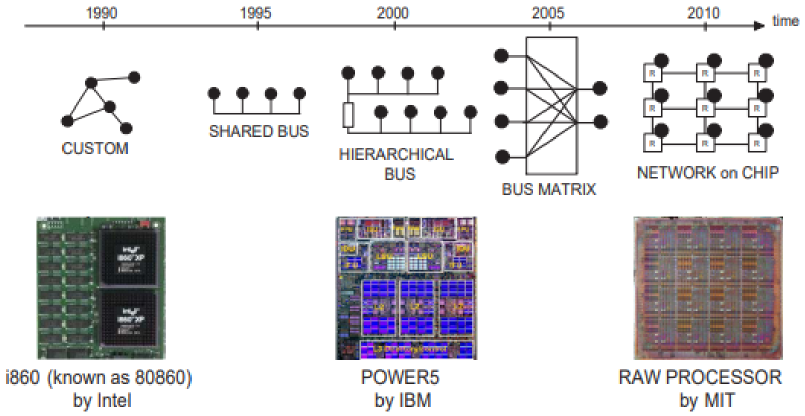


Figure 3.14: Evolution of intra-chip communication infrastructures [61]

Three main approaches can be taken when designing an intra-chip communication infrastructure. The bus, crossbar and recent innovation, the Network on Chip. The following sections describe the three approaches. Before diving into the main specifics, it is important to understand the parameters that determine a good bus interface. The main aspects of a good infrastructure are considered to be the following:

- **Throughput** How much data can be transferred over the bus versus the amount of cycles it takes (latency).
- **Complexity** The design complexity of the bus and thus the resulting difficulty in creating the infrastructure.
- **Area overhead** The amount of space (area) that is needed to facilitate the infrastructure.

3.4.1. Bus

The bus infrastructure is a high-speed data highway, allowing data and control signals to flow between different components such as processors, memory units, input-output modules, and specialized accelerators. This communication is essential for the execution of tasks and the delivery of performance in modern electronic systems. The design and optimization of bus architectures have a profound impact on overall system performance, power efficiency, and scalability. Figure 3.15 shows the structure of a bus inside an SoC.

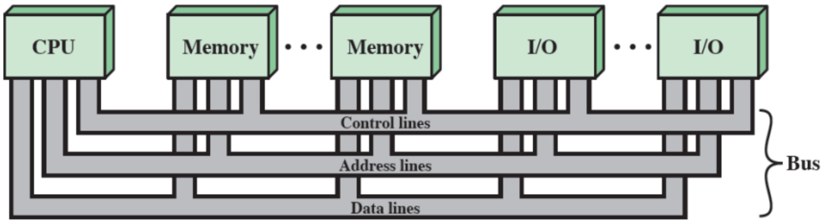
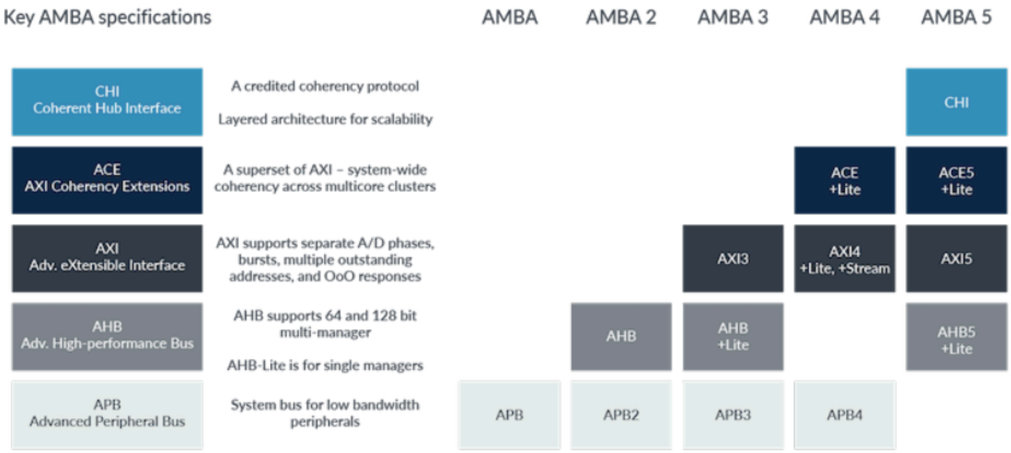


Figure 3.15: Bus infrastructure [61]

The evolution of bus infrastructures has been driven by an increasing demand for higher performance and enhanced functionality. Early chip designs featured simple bus architectures with limited bandwidth and basic protocols. As transistor density increased according to Moore's law, over time designers were able to integrate more complex systems onto a single chip, which resulted in the need for more sophisticated interconnection schemes.

Historically, buses were characterized by a shared bus architecture, where multiple components accessed a common communication channel. However, this approach presented limitations in terms of scalability and bandwidth due to multiple devices wanting a shared resource. To address these challenges, various advanced bus architectures have emerged. The most notable and most used is the AMBA (Advanced Micro-controller Bus Architecture) bus.



Key AMBA specifications		AMBA	AMBA 2	AMBA 3	AMBA 4	AMBA 5
CHI Coherent Hub Interface	A credited coherency protocol Layered architecture for scalability					CHI
ACE AXI Coherency Extensions	A superset of AXI – system-wide coherency across multicore clusters				ACE +Lite	ACE5 +Lite
AXI Adv. eXtensible Interface	AXI supports separate A/D phases, bursts, multiple outstanding addresses, and OoO responses			AXI3	AXI4 +Lite, +Stream	AXI5
AHB Adv. High-performance Bus	AHB supports 64 and 128 bit multi-manager AHB-Lite is for single managers		AHB	AHB +Lite		AHB5 +Lite
APB Advanced Peripheral Bus	System bus for low bandwidth peripherals	APB	APB2	APB3	APB4	

Figure 3.16: Evolution of AMBA busses [10]

The AMBA bus protocol, designed and standardized by ARM (Advanced RISC Machines) Holdings, has emerged as a dominant and versatile interconnect framework. The AMBA bus standard includes five main bus interfaces. The older AMBA 1 version consists of the advanced system Bus (ASB) and the advanced peripheral Bus(APB). The AMBA 2.0 version, the AMBA High-performance Bus(AHB), the latest AMBA 3.0 and 4.0, the Advanced eXtensible Interface (AXI) and the AXI4 bus. Of these, the most commonly used are the AHB and the AXI bus. Figure 3.16 shows the evolution of the amba busses and the key characteristics of the interfaces.

AHB bus

The AMBA advance high-performance bus (AHB) [4] protocol is a protocol designed by ARM to facilitate high-performance chip designs. The AHB bus is a complex bus architecture which connects different components like processors, memory and peripherals within an SoC. It offers a high bandwidth, low latency, and can support multiple masters and slaves. The AHB bus implements the features required for high-performance, high-clock frequency systems including:

- **Burst transfers** Sequential data accesses are facilitated without the need for a separate address and data cycle per packet.
- **Bus Pipe-lining** New bus transfers can already start during the sending of the previous data.
- **Configurable address and data bus widths** AHB facilitates address and data bus widths from 32 to 64 bits.

Figure 3.17 shows a block diagram of the AHB bus structure. The manager or bus master is the interface that dictates the transfer of data. In the case of a chip, this is the CPU or main processor. The subordinates are the peripheral units on the bus that the CPU needs to interface with.

The AHB bus functions on the principle of addressing. Every subordinate has its own 16-bit base address space and 16 lower bits for memory within that peripheral. For example, subordinate 1 can have address 0x001A_0000 (hex format) and subordinate 2 can then have 0x001B_0000. If the CPU wants to write to address 0x001A_0004, it is clear that that is in subordinate 1's address space. The decoder decodes the requested address based on the 16 higher bits and pulls high the HSEL of subordinate 1. Subordinate 1 has its HSEL high and now is ready to receive data. When decoding the address, the decoder also gives the multiplexer select which data line must be selected. If it is a read, subordinate 1 will put the requested data onto its HRDATA line.

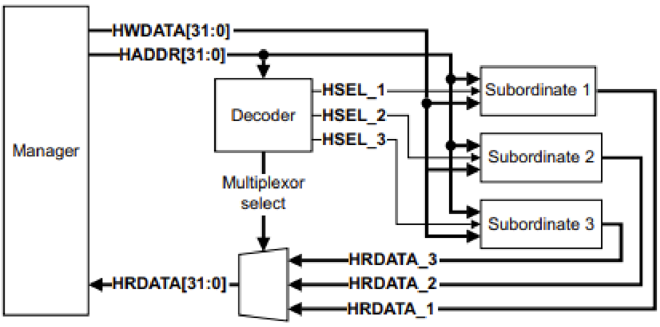


Figure 3.17: AHB bus architecture [4]

When it comes to performance, the AHB bus takes 3 cycles per data transfer, one to send the address, one for the decoding and acknowledgement and one for the actual data transfer. Usually, the AHB bus can facilitate multiple masters, to facilitate a lighter and simpler high-speed design, the AHB-Lite protocol can be used.

AXI bus

The advanced extensible interface (AXI) [10] is one of the newer additions to the AMBA protocol. The sole purpose of the AXI is to keep interconnect speed to the growing trend in embedded processor speed. The AXI protocol is quite a bit more complex and involved than the AHB bus.

The AXI bus implements the features required for high-performance, high-clock frequency systems including:

- **Burst transfers** Sequential data accesses are facilitated without the need for a separate address and data cycle per packet.
- **Multiple Channels** AXI protocol involves separate read and write channels, which allows for concurrent read and write operations. Improving overall system performance.
- **Out-of-Order Data Completion** AXI protocol allows for out-of-order data completion, meaning data packets can be sent in any order. This enables peak efficiency in the handling of data transfers.
- **Flexible address width** AXI protocol supports various address widths, meaning the interface can address a lot of different memory spaces and devices.
- **Flexible data width** AXI protocol supports large data widths (32-bit, 64-bit, 128-bit, etc.).

Figure 3.18 shows an example implementation of the axi interconnect. It becomes clear that the AXI bus is a lot more involved than the AHB bus. Every subordinate has a subordinate and a manager interface connecting it to the bus.

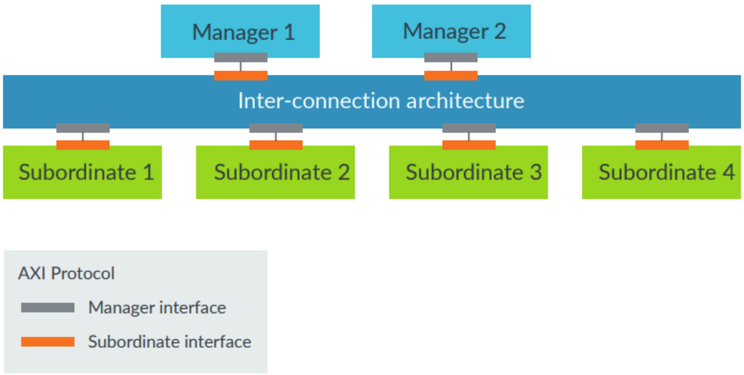


Figure 3.18: AXI Overview in example SoC [10]

The AXI protocol is a multi-channel interface consisting of a write address, write data, read address and read data channel. The read-and-write address channels initiate the read-and-write operation. It includes things like target address, data width and control signals. The read-and-write data channels are the transfer channels for the actual transfer of data. Figure 3.19 shows the channels between the manager and the subordinate.

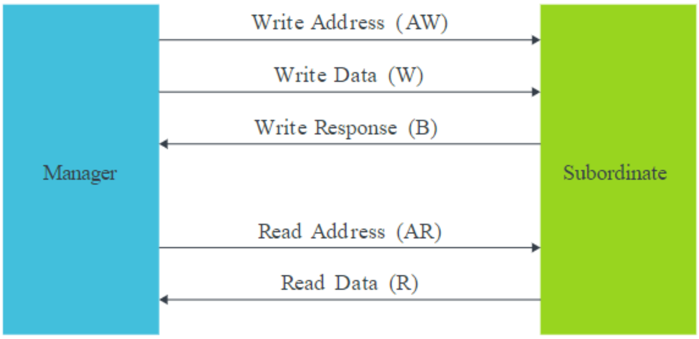


Figure 3.19: AXI bus channels

3.4.2. Switch based interconnects

Rather than having a bus between all the components on an SoC, a switch-based interconnect consists of a complex network of switches to facilitate easy routing between inputs(CPUs) and outputs.

Crossbar interconnect

Figure 3.20 shows an example crossbar interconnect. Crossbar interconnects can accommodate multiple sources (say CPUs) and multiple destinations. Therefore crossbars are usually implemented in multi-core applications and complex memory architecture. The advantage of a crossbar interconnect is that operations are non-blocking. Connections can be made between any input and output without contention, ensuring efficient communication even under heavy loads.

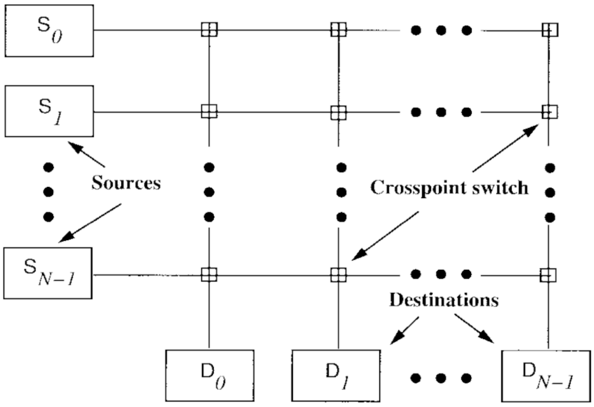


Figure 3.20: Crossbar interconnect example [23]

The advantage of crossbar interconnects is the low latency. As operations are non-blocking, a connection can always be established, meaning data can always be transferred. The disadvantage of crossbar interconnects is the area and control complexity. To have a fully functioning crossbar network, a lot of control lines need to be available. Next to that, if only a limited amount of components need to interface, the area overhead to implement the switch network is big.

Multistage Network on Chip (NoC)

To mitigate the availability of a single path in the single bus architecture and limit the implementation overhead of crossbar busses, the multistage switch interconnect was created[39].

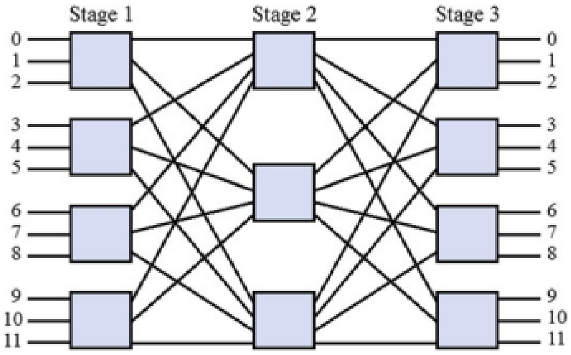


Figure 3.21: Multistage switch interconnect example [39]

The advantages of multistage interconnects are the low latency and scalability. As connections are still mostly non-blocking, a connection can always be established, meaning data can always be transferred. Next to that, adding a node to the network is easily done by extending the switch interface without having to implement a full new row of switches. The disadvantages of multistage interconnects are the control complexity and the blocking operations. To have a fully functioning crossbar network, a lot of control lines need to be available. Next to that, as the multistage network is not a full crossbar interconnect, connections can be blocked, and latency is higher than that of a crossbar network.

3.4.3. Static interconnects

Static interconnection networks have fixed unidirectional or bidirectional paths between components. Static networks can be divided into two groups: completely connected networks and limited connection networks[39]. Figure 3.22 shows a static completely connected interconnect. The benefit of having a static interconnect is that the system behaviour is predictable, either there is a direct path between the processor and the peripheral or the path is defined in a routing table. The downside of having a static interconnect is that it is not easily scalable. Adding a node to the architecture means re-doing the routing or adding a lot more connections.

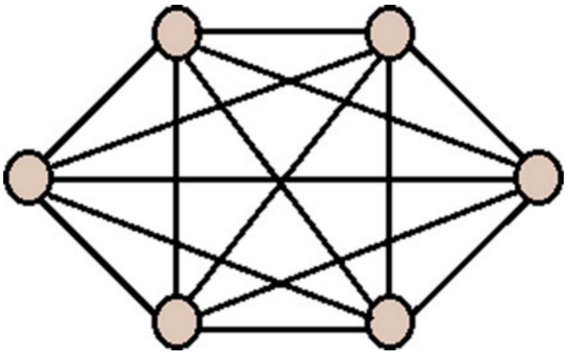


Figure 3.22: Static interconnect example [39]

3.5. Peripheral Interfaces

Next to the CPU, the memory unit and the interconnect, a chip generally consists of multiple peripheral units. Peripheral units aid the CPU by either aiding in the execution of specific tasks or interfacing with the outside world. Peripheral units generally can be divided into either hardware accelerators or peripheral communication interfaces.

3.5.1. Hardware Accelerators

With the rise in technological advances, computer chips become more and more advanced. A modern-day CPU executes millions of tasks in seconds. But a CPU still can only do one operation per clock cycle and increasing the frequency has negative impacts on the function of the entire computer, creative innovations had to be made. Two innovative ideas have become at the core of chip innovation. The addition of multiple CPUs in one chip (heterogeneous computing) and the offloading of tasks to specific parts of a chip designed for the execution of said task. Hardware accelerators are chip parts specifically designed for the execution of one simple task or one simple set of tasks. They are hardware implementations designed to do a specific job faster, a very crude example of a hardware accelerator is a Graphical processing unit (GPU) which is a piece of hardware built for the compute-intensive operations which come with graphics.

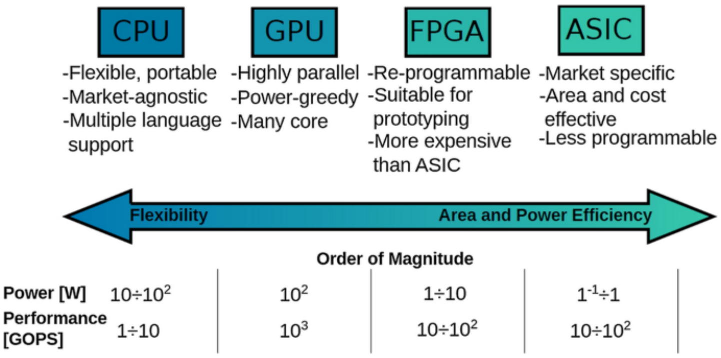


Figure 3.23: Differences between CPU's, GPU's, FPGAs and ASIC's [13]

Cryptography engines

Cryptographic engines are specialized hardware or software components designed to perform cryptographic operations that ensure data confidentiality, integrity, and authentication in digital communication and storage. There are generally two types of cryptographic engines, software and hardware engines.

Hardware engines are dedicated pieces of hardware designed to accelerate cryptographic operations like AES encryption and decryption. They are generally used inside network chips, security appliances and storage systems which need data encryption. Software-based engines are often cryptographic libraries that provide cryptographic functions more efficiently, not requiring dedicated hardware.

Cryptography engines can be applied in many fields. Examples are secure communication where the engines enable secure communication when browsing, sending emails and using virtual private networks(VPNs). Another example is healthcare where patient information needs to be secure.

3.5.2. Communication interfaces

Rather than letting the SoC execute tasks in silence, it is often required and useful that the SoC user can interact with the chip. Meaning, uploading code, setting pins to either high or low or asking for updates from the SoC. This is where communication interfaces come in. Communication interfaces open the CPU up to the outside world. There are numerous communication interfaces available, each with its purpose and use case. The key to reliable communication is understanding and selecting the right interface per use case. Below two of the most common inter-chip communication interfaces are explained.

Serial peripheral interface (SPI)

The serial peripheral interface is a synchronous communication interface which can facilitate communication between various devices and sensors. It is widely known and used by many SoCs and is based on the controller-peripheral protocol. This means one device on the bus controls all the interactions between itself and its peripheral devices.

SPI is a serial protocol that in its simplest form works on four wires; The SCK or the clock, the PICO for the data transfer from the bus controller to the peripheral devices, the POCI for the data transfer from the peripherals to the bus controller and the CS which is used by the controller to select the peripheral it wants to talk to. Next to four-wire SPI, there exists also a variation that is called Quad-SPI where, instead of 1 data line for a peripheral device to send data on, four wires are available for communication. This allows higher data rates.

Figure 3.24 shows an interaction between a peripheral device and the bus controller in a simple four-wire SPI. The controller pulls the CS of the peripheral low and starts sending the clock over the SCK line. Then the controller starts sending data to the peripheral, after which the peripheral starts sending data back.

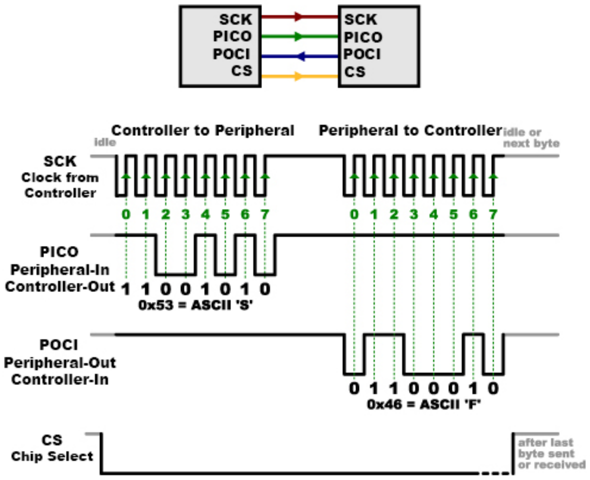


Figure 3.24: SPI communication overview [29]

SPI is often used for its flexibility, simplicity and high speed. SPI is re-configurable in its data sampling, meaning data can be sent at the positive clock edge and negative clock edge.

The Universal Asynchronous Receiver-Transmitter (UART)

The UART interface is a fundamental serial communication interface used in embedded systems, connecting devices and facilitating data exchange over short distances. Unlike synchronous interfaces like SPI or I2C, UART operates asynchronously, allowing devices with different clock frequencies and timings to communicate effectively.

UART is a two-wire protocol with a receiver(RX) and a transmission(TX) wire. Figure 3.25 shows the receive of one device coming from the transmission wire of another device. UART works based on a start bit, indicating that a data packet is coming and a stop bit, indicating the end of a data packet. The rate at which the data is sent is handled by a re-configurable baud rate, both devices need to facilitate the same baud rate. This baud rate is independent of the clock frequency of the device.

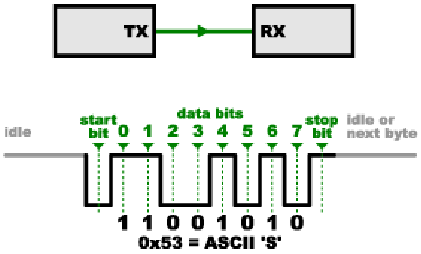


Figure 3.25: UART communication overview [29]

The downside of UART is that the speed is rather slow but the fact that the clock frequencies of both devices can be different and that only 2 wires are needed, UART is commonly used between devices where speed is not necessary but robustness and simplicity are needed.

3.6. SoC software design

In line with the SoC design comes software design. Running code on conventional SoCs like ARM cores [73] in base integrated developer environments(IDE's) like the Arduino IDE [9] is relatively straightforward. The entire hardware layer is abstracted and the user only has to write code and upload it to the device. This is not the case when designing a custom SoC with a custom core. For a custom implementation, the hardware abstraction also needs to be done in a custom way. Figure 3.26 shows how the abstraction works in a pipeline from .c code to memory files for the memory. In general, the pipeline consists of 3 main steps; Compiling, Assembling and Linking.

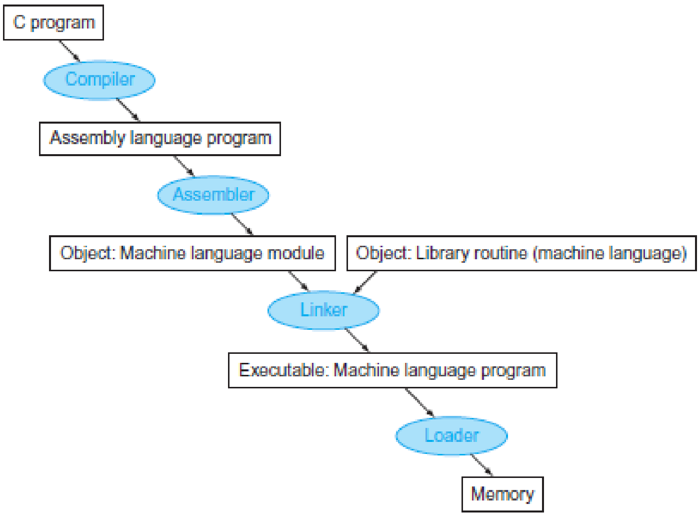

C program
Compiler
Assembly language program
Assembler
Object: Machine language module
Object: Library routine (machine language)
Linker
Executable: Machine language program
Loader
Memory

Figure 3.26: Complete software from code to memory pipeline [64]

3.6.1. Compiler

A compiler compiles code into assembly code which consist of the base instructions a CPU can understand[64]. In the past, code was written directly into the assembly language as code needed to be small and efficient. As technology got better, compilers were able to compile more user-friendly code into assembly code whilst optimizing the code. Compilers only compile the code for a specific computer architecture. Taking an x86 compiler and attempting to run it on an ARM architecture is therefore not possible.

3.6.2. Assembler

After compiling the generated assembly code is passed to the assembler. The assembler translates the assembly instructions into machine code which is a binary representation of the instructions. Below is an example of assembly code in block 3.6.2 and its equivalent machine code in 3.6.2. This binary representation is called an object file.

```
MOV A, 5    ; Move the value 5 into register A
ADD B, A    ; Add the value in register A to register B
```

Listing 3.3: Assembly example

```
A205    ; Move 5 into register A
1A0B    ; Add contents of register A to register B
```

Listing 3.4: Machine code example

Next to generating the object files, the assembler also gives the memory location of each variable and instruction in offsets and a list of unsolved references to functions which are defined in other libraries (eg printf function). A typical object file contains the program text (instructions) and data (constants and strings), information about instructions and data that depend on absolute addresses, a symbol table of unresolved references, and possibly some debugging information[81].

3.6.3. Linker

The last step in the pipeline is the linker stage. In the linker stage, the memory locations of each variable and instruction are translated from offsets to absolute locations. This is done with the help of a linker script. A linker script defines the entire memory architecture of your hardware into regions. These regions are defined by the user and usually consist of the base address of the instruction and data memory, the size of the memory, what data goes into the memory and in what order it is arranged in the memory. Figure 3.6.3 shows an example of a linker script for a RISC-V engine. In the beginning, the memory is defined after which the order or what goes where is detailed.

```

OUTPUT_FORMAT("elf32-littleriscv", "elf32-littleriscv", "elf32-littleriscv")
OUTPUT_ARCH(rv32im)  /* Specify the RISC-V architecture and ABI */

ENTRY(_start)  /* Entry point of the program */

MEMORY
{
    ram (rwx) : ORIGIN = 0x00000000, LENGTH = 8K  /* Define the memory region (8K) for RAM */
}

SECTIONS
{
    .text :
    {
        _text_start = .;  /* Symbol pointing to the start of the text section */
        *(.text)  /* .text section, where the program code resides */
        _text_end = .;  /* Symbol pointing to the end of the text section */
    } > ram

    .data :
    {
        _data_start = .;  /* Symbol pointing to the start of the data section */
        *(.data)  /* .data section, where initialized data resides */
        _data_end = .;  /* Symbol pointing to the end of the data section */
    } > ram

    .bss :
    {
        _bss_start = .;  /* Symbol pointing to the start of the bss section */
        *(.bss)  /* .bss section, where uninitialized (zero-initialized) data resides */
        _bss_end = .;  /* Symbol pointing to the end of the bss section */
    } > ram
}
```

Listing 3.5: Linker script example

After the linker, the code is finally ready to be put into the memory of the SoC. The compiler, assembling and linker is something which need to be made custom for each core that the SoC supports.

4 System Architecture

This chapter outlines the full proposed system architecture. The chapter starts with explaining the motivation for needing the system. Then the final system objectives are elaborated on after which the practical test cases are explained. After the system objectives and the test cases, the design and implementation requirements are quantified. After this, the proposed system architecture is explained and shown.

4.1. Motivation

There are several reasons to motivate the design of a novel system to perform power side-channel analysis. The most important one is the absence of a good and versatile platform for our specific use cases. First, it is important to understand that a lot of platforms are available which can perform some form of power side-channel analysis. However, these are focused on their specific application and research area and are often centred around pre-silicon validation.

Post-silicon leakage assessment, crucial for securing semiconductor devices against power-based attacks, has received limited attention in research. Although numerous studies have been carried out on the assessment of pre-silicon leakage, there remains a lack of emphasis and focus on the development of post-silicon techniques. The lack of research into developing a more reliable and efficient mechanism of quantifying and correlating post-silicon security, makes existing methods seem more expensive and time-consuming. As a result, some chip manufacturers are skipping post-silicon leakage assessment altogether.

This is a dangerous trend that may result in unidentified vulnerabilities that can jeopardize the privacy of individuals using the end products. Therefore, it is important to invest more in research and development on post-silicon leakage assessment. Next to that, it is also necessary to make this technology more accessible and affordable for chip manufacturers. By doing so, future SoCs will be better protected against power attacks and the data privacy of users is safeguarded.

Besides the absence of a fitting platform for post-silicon side channel assessment, a secondary reason can be extracted. Practically, existing platforms are proprietary, expensive and tool-specific. Examples of this are ChipWhisperer [58], of which the software is easy to use yet the hardware is proprietary. Meaning that not everybody has access to perform side-channel analysis. Designing a new platform which is versatile, tool agnostic, and open-source allows for future side-channel assessment innovations to grow at the TU Delft.

4.1.1. System Architecture Objectives

The motivation of a novel platform shows the final goals of the new system. Qualitative wants can be translated into certain system objectives. System objectives spell out what the system should do, how well it should do it, and what people should expect from it. The system objects facilitate setting up concrete test scenarios and defining quantifiable system requirements later on.

For the system architecture, the system objectives can be listed as follows:

Comparing and correlation of pre and post The proposed architecture needs to close the gap between pre-silicon side channel assessment and post-silicon side channel assessment by providing a custom SoC platform as well as an equivalent FPGA comparison platform.

Validation in pre and post The proposed architecture aims to provide validation of a select number of AES implementations in pre and post-silicon settings which will work as a reference comparison for later AES implementations.

Validation of countermeasures implementation The proposed architecture facilitates the analysis of different countermeasures in both the pre and post-silicon setting by introducing AES engines with countermeasures in the custom SoC platform for comparison.

Extensible to various future AES implementations The proposed architecture is extensible to various AES implementations by employing a common interface for all implementations. Meaning the platform can facilitate the testing of future implementations.

Faster assessment and analysis of power measurement The proposed architecture aims to reduce simulation and measurement time by ensuring that only relevant measurement portions will need to be measured.

Better quality measurement traces The proposed architecture aims to provide realistic high-resolution power measurements that are timing aligned for easy analysis.

4.2. Test Cases

The system objectives dictate the expectations and aims of the novel system. To navigate the end goals of the system, a baseline set of test cases can be devised to ensure that the system can facilitate these. These test cases serve as the main framework capturing the system's functionality, performance and ultimately the system requirements.

1. **Power side-channel evaluation in simulation with custom SoC** The simplest test that the platform needs to facilitate is doing a pre-silicon evaluation on the custom SoC with the use of a technology library and simulation power data. The platform needs to be able to show the differences between AES implementations based on simulation power analysis.
2. **Power side channel assessment with custom SoC** The most important test is the post-silicon side channel assessment. Figure 4.1 shows the post-silicon side channel assessment test setup. Power is externally provided by a standard power supply. A laptop uploads a program which is then programmed on to the SoC. The SoC then runs the AES algorithm and measurements are taken with a probe and oscilloscope.

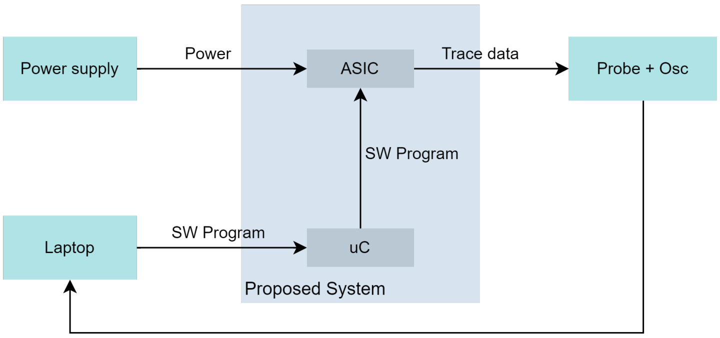


Figure 4.1: Pre-silicon side channel assessment

3. **Power side channel assessment with FPGA** To facilitate the pre-silicon side channel assessment, measurements with the FPGA also need to be facilitated with the system. Figure 4.2 shows the setup for this test case. For this also an external power supply is needed. Then the FPGA needs to be reconfigured when needed by the laptop. Next to that, to provide an accurate comparison between FPGA and SoC, the FPGA should be able to run the same program and the same capture of measurements.

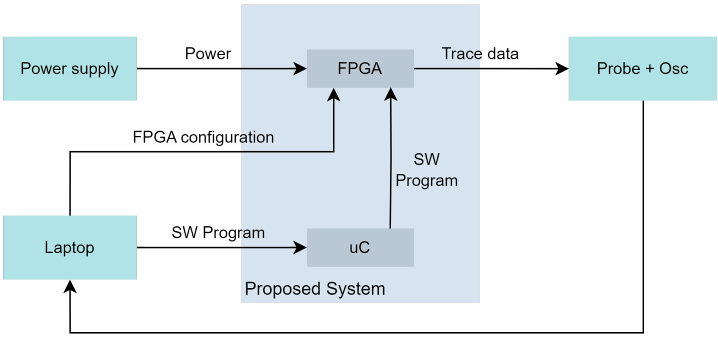


Figure 4.2: Post-silicon side channel assessment

4. **Power side channel comparison and evaluation** The last test case that needs to be facilitated is the possibility of the evaluation of the platform itself. Figure 4.3 shows the setup for this test case. This is done by introducing debug capabilities into the platform which can show the current program, and send updates on encryption and decryption outputs to verify correct encryption.

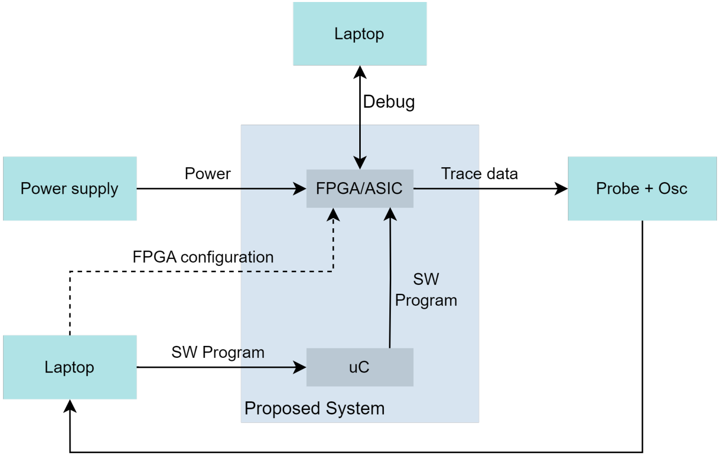


Figure 4.3: Testing of the test platform itself

4.3. System requirements

After discussing the motivation for a novel platform design and discussing the system's main objectives and expected test cases, strict requirements can extracted. These requirements detail the system objectives and are split up into two parts: Design requirements and implementation requirements.

4.3.1. Design requirements

The design requirements detail the firm requirements which concern the entire system design. These outline what a system should achieve and how it should function.

1. **The system design must facilitate pre and post-silicon side channel analysis** The system must be able to facilitate the comparison and correlation between pre and post-silicon side channel behaviour.

2. **The system design facilitates different hardware implementations of AES** To provide realistic comparisons between different countermeasures, the design needs to facilitate different hardware implementations.
3. **The system design must provide realistic power side measurements for analysis** Following the main objective, the output measurements need to be realistic. Realistic measurements can then be used to quantify CPU and peripheral interface noise and understand better the impact of the rest of the chip on the security of the AES engine.
4. **The system design does not interfere with the power side channel measurements** Apart from the functional parts of the SoC, the rest of the SoC and hardware should not interfere with the power side channel measurements by introducing noise or time skew.
5. **The system design be easily understood by an MSc student** The system is to be designed for future research and future data analysis. Therefore it will also be operated by other MSc students or Ph.D. students. To facilitate this, it is assumed that the student has some knowledge of computer engineering but does not understand every aspect fully.
6. **The platform must be able to be operated by an MSc student** As previously mentioned, a fellow student with some knowledge of computer engineering should be able to operate the system with little help from outside.
7. **The system design must be reliable/repeatable in its operation and behaviour** The system must behave in a predictable manner which can be repeated run after run to provide good comparisons.
8. **The system design should be redundant to external influence like power fluctuations** Small input power fluctuations should not affect the operation of the system as these are normal occurrences in everyday life.
9. **The system design should be redundant enough for single points of failure** A single point of failure should not destroy the system as in, it might change the way it operates but it still needs to be operable and provide reliable measurements.

4.3.2. Implementation requirements

For the implementation requirements a breakdown of implementations can be made. Looking at an existing platform like Chip wisperer in figure 4.4, the following components can be seen. A target device running an SoC design, a hardware PCB facilitating all the interfaces bewteen the laptop/oscilloscope and the target device and a connection to a laptop on the right.

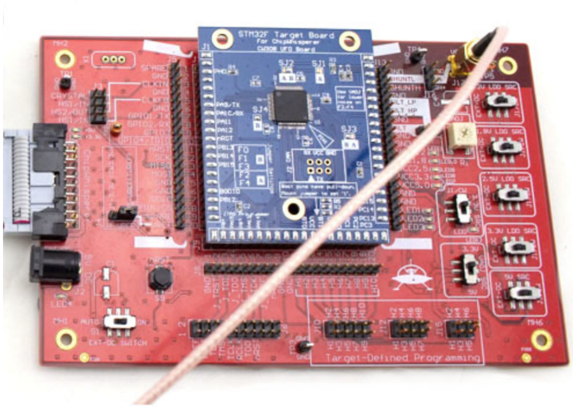


Figure 4.4: Chip Wisperer Husky [57]

Based on these existing platforms, the implementation can be split into three parts with seperate requirements. Firstly the SoC/FPGA target design implementation requirements, the software implementation requirements and the PCB implementation requirements. Figure 4.5 shows the breakdown of the entire system into these three parts and the responsibilities of each part in the system.

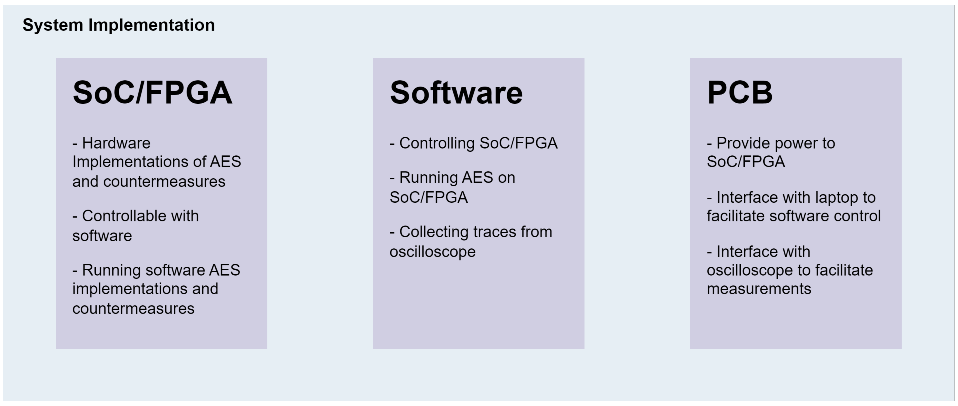

System Implementation
SoC/FPGA
- Hardware Implementations of AES and countermeasures
- Controllable with software
- Running software AES implementations and countermeasures
Software
- Controlling SoC/FPGA
- Running AES on SoC/FPGA
- Collecting traces from oscilloscope
PCB
- Provide power to SoC/FPGA
- Interface with laptop to facilitate software control
- Interface with oscilloscope to facilitate measurements

Figure 4.5: Breakdown of the system implementation

SoC/FPGA target implementation requirements

The SoC/FPGA target implementation outlines the requirements for the target chip that will be the comparison between the pre and post-silicon side channel assessment. This means the SoC target implementation includes the chip design that will be in the pre-silicon side channel assessment (on an FPGA-based assessment) and the post-silicon side channel assessment (SoC-based assessment).

1. **The SoC/FPGA implementation must include a minimum of three distinct AES implementations** To provide realistic comparisons and quantifiable measurements of security between pre and post and between different AES architectures, a variety of protected and unprotected AES engines are needed.
2. **The SoC/FPGA implementation must include a core with a minimum processing speed of 100 MHz and capable of running small software programs** The AES engines need to be configurable and adaptable from the outside world. The 100MHz is set a guideline extracted from embedded core designs.
3. **The SoC/FPGA implementation must provide reliable and repetitive measurements of the AES implementation** To facilitate proper analysis and characterization the system should be able to repeat different test scenarios.
4. **The SoC/FPGA implementation must be able to successfully store the 128-bit AES output in its internal data memory** To validate the function of an AES engine and do analysis on the correctness, the system should be able to store the output.
5. **The SoC/FPGA implementation must include basic SoC components to visualize the system noise** One goal of post-silicon side channel assessment is to see the effects other SoC components have on the security of encryption. Following chapter 3's components of an SoC, at least one of each component should be available.
6. **The SoC/FPGA implementation should include a debug interface that is capable of running the AES from its port** As the system must be reliable, redundant and provide full core transparency, a back up core is needed. This consists of having a debug interface to backup the core in case of malfunctioning.
7. **The SoC/FPGA implementation must be easily operated and understood by any peer student** As this platform's goal is to be used in future experiments, the SoC/FPA implementation should be kept simple and standardised to make operating easy. To quantify this, a usability survey can be held at which the platform should get at least an 8/10.

Software implementation requirements

The software implementation requirements of the platform are the requirements for the software that is to interface with the SoC/FPGA target and the operator. This includes the starting up, the interfacing with peripherals on the target and the running of different tests. It does not refer to the measurement pipeline.

1. **The Software implementation must be able to boot the SoC/FPGA under 1 minute for at least 1000 repetitions** The software must be able to boot up the SoC/FPGA design meaning it must be able to safely power on and start running a program repetitively and reliably.
2. **The Software implementation must be able to run on the implemented core** As the SoC/FPGA design needs to facilitate a RISC-V core, the software must be able to compile and assemble for the implemented core.
3. **The Software implementation must be able to interface with the hardware AES implementations for at least 1000 repetitions** The software implementation needs to be able to start and stop the AES implementation. Next to that it needs to feed the AES with the encryption and Decryption data as well as read out the final encrypted or decrypted result.
4. **The Software implementation must be able to interface with all other peripherals on the SoC/FPGA design** The software should be able to control all parts of the SoC/FPGA design reliably and repetitively.
5. **The Software implementation should be less small enough to fit inside the SoC internal instruction memory** As the final SoC memory will be limited, the software implementation should fit in the memory space of the SoC.
6. **The Software implementation should be flexible and adaptable between tests** Different tests have a need for different software, the software should be able to change between tests.

PCB implementation requirements

The PCB implementation requirements of the platform are the requirements for the PCB to facilitate the power side channel assessment on a target chip on the PCB. Next to that, the PCB also facilitates the interaction between the outside world and the SoC/FPGA design.

(a) **The PCB implementation must provide the correct power and be able to power up the SoC/FPGA** The target device needs to receive clean and correct power to safely start-up repetitively.

(b) **The PCB implementation must isolate the noise the SoC design for the power side channel measurement** The noise of the peripheral devices on the PCB should not interfere with the power side channel measurements of the SoC.

(c) **The PCB implementation must include visual debugging indicators to facilitate debugging like test points and LEDS** As this is a test platform, visual indicators that indicate any issue need to be on the PCB.

(d) **The PCB implementation must include connectors for the power side channel assessment and trigger pins** To facilitate interfacing with either an oscilloscope or any other assessment device, connectors must be available.

(e) **The PCB implementation must safeguard the FPGA and SoC at all stages of operation** As the FPGA/SoC are quite expensive, they must be protected from any type of system failure. This includes, power dips and short circuits.

(f) **The PCB implementation must be easy to operate by an MSc Student** The operation of the PCB should be clear and easily understood by a peer student.

4.4. Proposed System Overview

The above design and implementation requirements can be combined into a full power side channel test framework and measurement setup architecture. Figure 4.6 shows the complete overview of the entire proposed system architecture. On the left, the setup with the custom ASIC is displayed. The SoC includes a fixed set of AES implementations and a RISC-V core. On the right, the same setup is displayed for the FPGA. The FPGA will be configured with the exact same SoC design in the same PCB layout. This is to ensure the testing on the FPGA is as comparable as possible to the SoC setup. The red dotted line indicates the two separate SoC platforms the system needs to facilitate.

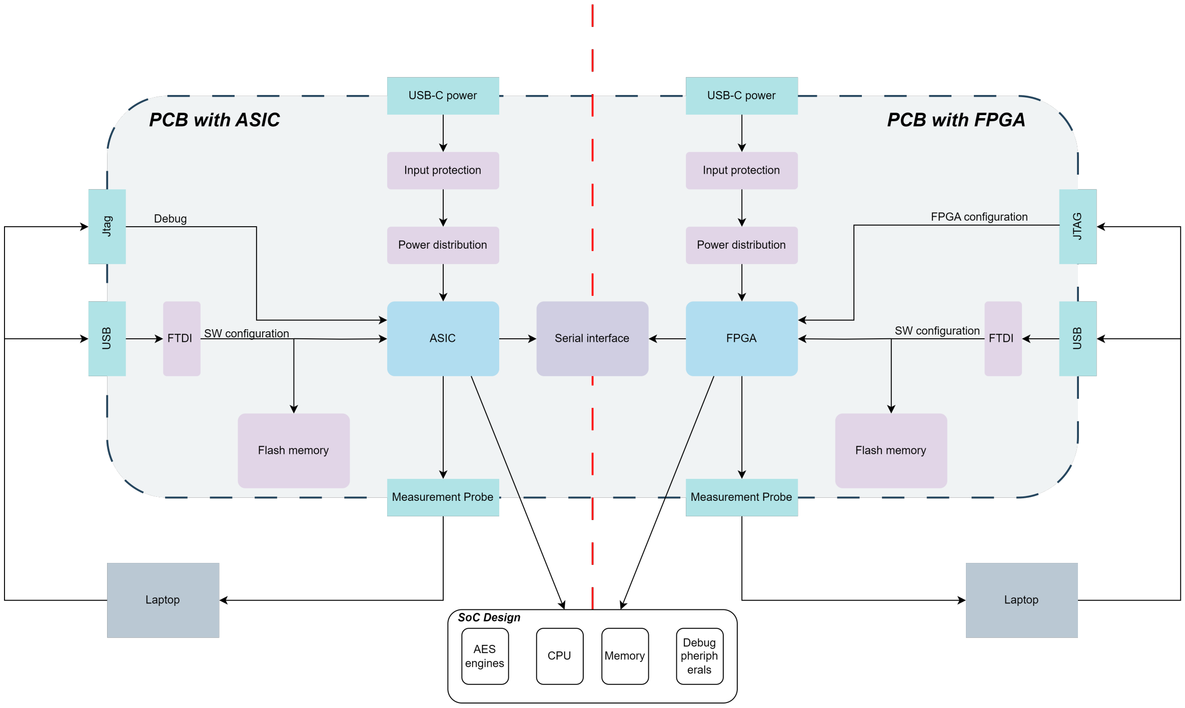


Figure 4.6: Complete proposed system overview

4.4.1. System breakdown

Observing the full proposed system in figure 4.6, the system can be broken down into different parts.

Hardware Components The input power should be applied with a standard cable like USB-C. This facilitates the ease of use as well as ensuring enough power (5V, 3A) can be supplied. The input power needs to go through input protection such as to not break anything as well as keeping the FPGA and ASIC safe. The power distributions provide clean and isolated power to the SoC target and the rest of the system. Next to that, the power distribution enables isolated power measurements for the SoC implementation.

The Measurement probe will be put as close to the SoC design as possible to ensure a maximum signal-to-noise ratio of the encryption/decryption power. An FTDI [28] chip is used to load programs over USB into external flash memory which can be accessed by the SoC design. The datalines of the FTDI and the SoC design are muxed to the flash memory to facilitate reconfiguring and reflashing of programs.

Lastly a standard debug port like JTAG [41] from the Joint Test Action Group (JTAG) is used for standardized debugging and a serial interface like an ESP-01 is used for sending serial updates from the core to the laptop.

Software Components The software components include the software for the entire SoC design. This software then needs to interface over USB with the FTDI chip so a program can be uploaded into the memory. The software also includes the boot code that will, at start-up, fetch the program from external flash memory on the PCB.

SoC Components The SoC design consists of a CPU that can run software next to a set of different AES engines for running and characterizing different architectures and countermeasures. Also some memory for storing variables and some debug peripherals like a communication peripheral that can fetch the program from external memory and a communication peripheral that can update over a serial bus.

5 Design and Implementation

This chapter outlines the full proposed system implementation. The implementation is done in 3 phases, the SoC design, the Software design and the PCB design. Section 5.2 explains the full implementation of the SoC design by going over each part. Then the complete software backend is explained. Lastly, the PCB design is explained.

5.1. Implementation Overview

Following the proposed system architecture and system requirements, the design and implementation of the complete system architecture can be divided into three sub-parts. The design of the custom SoC design which can be implemented on an SoC as well as on an FPGA, the design of the software architecture to interface with the custom SoC and finally the design of a printed circuit board (PCB) to facilitate the measuring of power side channel measurements on the SoC and FPGA.

5.2. SoC design Overview

The SoC design is a RISC-V-based AES test SoC design. In short, the SoC design is implemented using an R32I RISC-V core built on the RI5CY core from the open-hardware group [66]. The implemented design is an extrapolation/adaptation of the Pulpino SoC made by the open hardware group. Next to the RISC-V, the design consists of an AHB3-lite interconnect bus with a combined 32kB data and instruction SRAM. Lastly, a small piece of ROM memory is implemented where a bootloader is stored which copies external programs over Quad-SPI.

Figure 5.1 shows the full overview of the SoC design. In short, the SoC design's AHB3 lite bus features a 32-bit wide data and address bus which are facilitated with a bus decoder and mux. Connected to the bus are a select number of peripherals. JTAG is featured for bus debugging and can be used as a backup bus master, SPI is used for external flash loading and features Quad and Serial mode loading. Six different AES engines are used for power comparison, they can be used individually. Two hardware timers are implemented for future more complex software AES implementations. Lastly, UART is used as the serial in/out, allowing for dynamic data and key configuration or receiving status updates and sending updates like final AES cipher text over a serial bus to a laptop.

The following sections provide detailed exploration and elaboration of each peripheral, outlining their custom implementations in this specific platform.

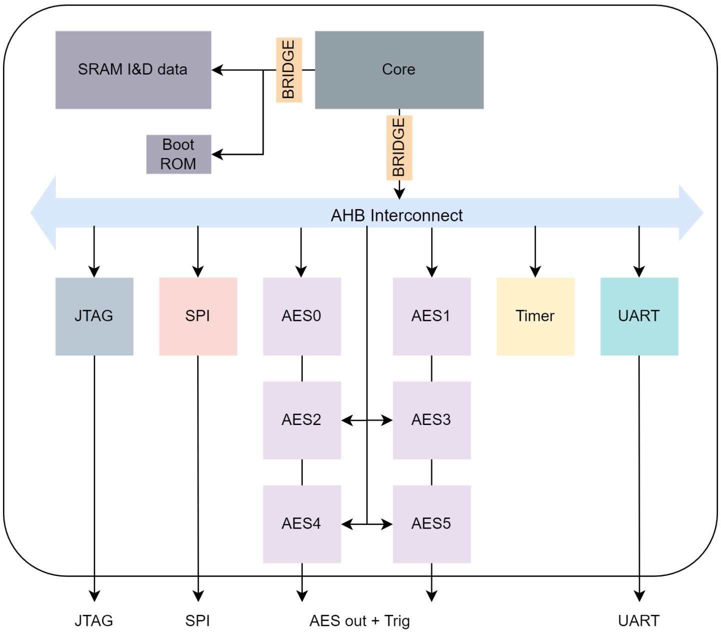


Figure 5.1: SoC design simplified block diagram

5.2.1. Baseline AES engine

In this thesis, the focus is on a variety set of AES engines. Table 5.1 shows an overview of all the AES engines and their respective design. There are three AES implementations with various sbox sizes. Then there are some protected AES implementations which include three with the domain-oriented masking (DOM) countermeasure and one with a power balancing countermeasure. The Domain-oriented masking engines also vary in sbox size. The design of these AES DOM engines are done by a fellow master student Mr Hang [35]. His thesis showed a area efficient secure DOM-based AES implementation. AES engine 2 is the baseline AES engine with no protection and no countermeasures.

Table 5.1: Set of implemented AES engines

AES number	Design	Full design description
AES0	AES	Balanced SBox AES
AES1	***DOM***	DOM 4s design
AES2	AES	AES baseline design
AES3	AES	AES 16s design
AES4	***DOM***	DOM 8s design
AES5	***DOM***	DOM 16s full pipe design

An AES engine can generally be divided into two parts, encryption and decryption. On the encryption side, a key and plain text enter and a ciphertext comes out. On the decryption side, ciphertext and the key enter and the plaintext comes out. Both decryption and encryption together form a full AES engine as shown in figure 5.2.

Conventional assessment of power side security focuses on running the AES engine numerous amounts of time and then assessing the power line fluctuations over all the iterations (Eg [48]). These traces need to be aligned for analysis as the beginning and ending of each iteration needs to be visible. This is an intensive process as perceived random fluctuations in time need to be mapped to a set of encryption iterations[52]. Meaning it is difficult to find and do analysis on only relevant information in a power trace.

Secondly, to facilitate analyses like correlation power side or template-based, every iteration of encryption/decryption consists of a key, plaintext, ciphertext and the power data. This means

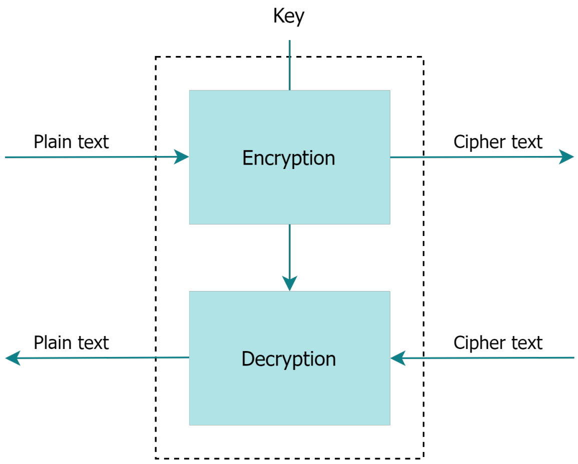


Figure 5.2: AES simplified block diagram

every iteration new key and plaintext needs to be inputted into the AES engine. Resulting that for analysis which need large trace sets, large input data sets are needed and thus large on chip memory is needed. This is not recommended as memory is area and power extensive.

5.2.2. Improved AES engine

To enhance the AES engines for various power side-channel analysis methods, the following objectives must be achieved:

- Traces need to be aligned for easy analysis and mapping
- Only relevant information should be available in the analysis to make analysis faster and easier
- The amount of memory needs to be limited to minimize the area and power of the SoC

Linear feedback shift register (LFSR)

Looking at the baseline AES engine block diagram, every iteration a plaintext and a key is needed. Meaning that for 1000 traces, 2000 128-bit input data points are needed. Meaning that when the trace requirement of an experiment increases, the necessary data and thus the necessary memory buffer increases. As memory consumes a large amount of power and area, having large memories on power side channel platforms does not work [27].

For power side channel analyses, only the power lines are observed. What is being decrypted or encrypted is of little importance. As long as the inputs can differ each cycle (meaning different power line variations can be observed). To facilitate this, cryptographic engines often employ linear shift feedback registers (LFSR's). LFSRs are used as building blocks for many stream ciphers and other cryptographic primitives as they can generate pseudo-random numbers, in fast digital counters, whitening sequences etc[15].

LFSR's work on a seed. A seed is an initial input value. Based on this initial input value, a new value can be created from a polynomial operation on the initial value. Figure 5.3 shows such a polynomial operation. In this figure the next input value is the previous input shifted right by one position plus the XOR of bit 0, 1 and 21 and 31. Generally LFSR's come in two types, Fibonacci and Galois, both operate on the same principle but differ in their polynomial function. In this design Fibonacci was used as it proved to be less prone to against power side channel analysis. [15]

Trigger signal

When taking a power side channel measurements in conventional platforms, the measurement starts from when the device is turned on. This to not miss the encryption or decryption. That means a lot of unwanted data is recorded and analysed making measurements long. To align the

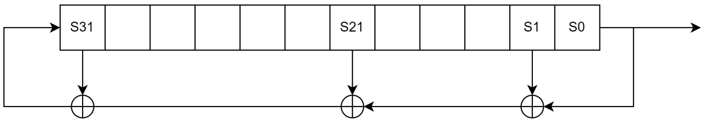


Figure 5.3: Lineal feedback shift register operation

trace data as well as ensuring only relevant information is in the analysis, a trigger signal can be implemented that will indicate when measurements need to be taken.

To facilitate this trigger, and make quicker and aligned measurements, a special register is created. This register can be configured using software and is directly connected to an an output pin. This pin can be configured to go high when the encryption starts and go low when encryption is done. This trigger signal can then interface with the oscilloscope to trigger the oscilloscope to start recording measurements based on this trigger.

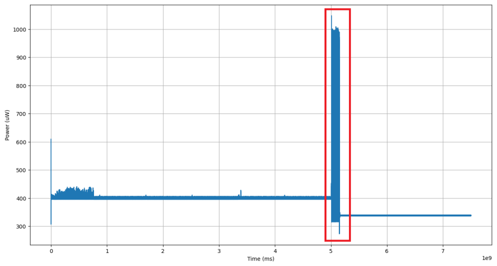


Figure 5.4: Conventional power measurement

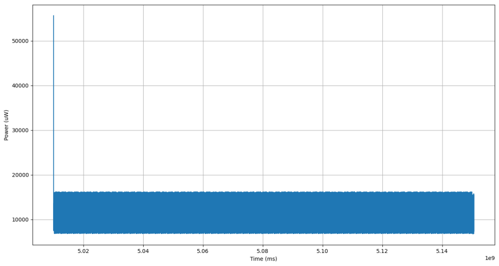


Figure 5.5: Power measurement with Trigger

Figure 5.4 shows a conventional power measurement where the red box indicates the AES running. Here it becomes visible that of the entire recorded trace only a fraction is valuable data. Figure 5.5 shows a trigger based power measurement which is only the AES running.

Controlling the AES

The last modification done to the AES engine is making it easily controllable and configurable in software. To facilitate this, an set of control variables can be implemented. These can then be put into a register which the core writes to. Table 5.2 shows the configuration register with the start signal which is used to start the encryption, the ENC/DEC to set the mode, the LFSR_RST which initializes the seeds of the LFSR's, the Restart which will restart the test if needed and the TRIG which will pull the trigger high.

Registername	Address	Reset	Bit definitions				
CONFIG_REG	0x1D10 0004	0	START[2]	ENC/DEC[3]	LFSR_RST[4]	RESTART[5]	TRIG[6]

Table 5.2: Config register of AES engine to control LFSR and AES

These variables go in line with a custom FSM which used the above signals to start the LFSR, pass the value to the AES and do this for the amount of iterations set by the user. Figure 5.6 shows the implemented FSM.

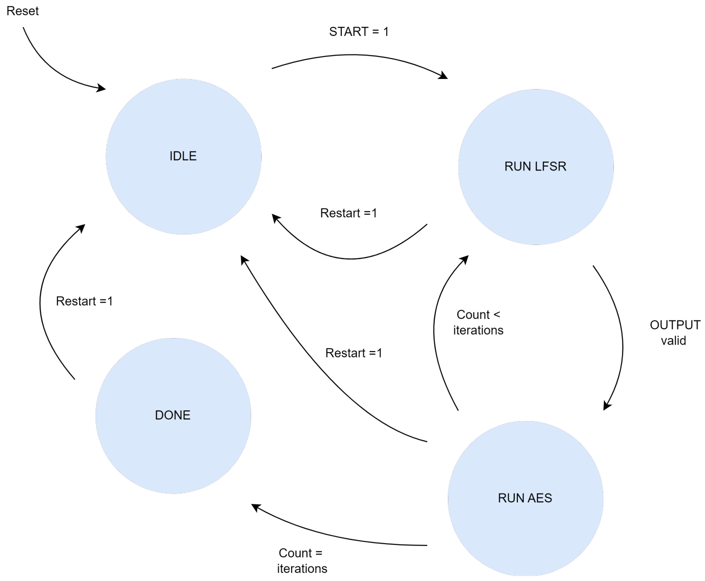


Figure 5.6: Finite state machine for LFSR+AES interaction

Final AES design

Figure 5.7 shows the complete improved AES implementation including all the above modifications. These modification are non-invasive and AES implementation agnostic. Meaning that the previously mentioned set of AES engines can be swapped by newly designed engines and be implemented on the platform.

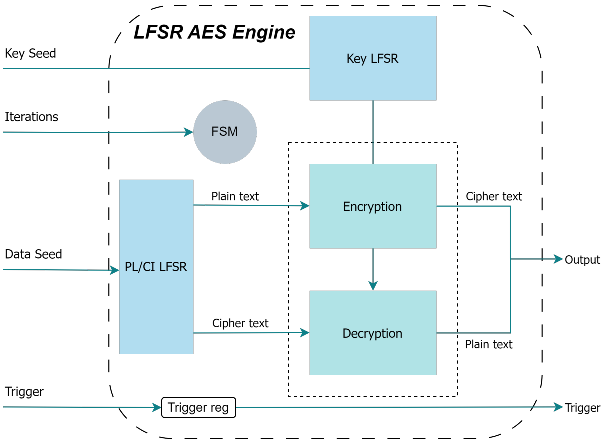


Figure 5.7: Final AES implementation

5.2.3. RISC-V processor

Referring back to the SoC requirements, to facilitate running of software algorithms and to interface with the AES implementations, a RISC-V core was implemented. The base RISCY core from the PULP-Platform [66] was used for this platform it provides the good balance between size and complexity. Figure 5.8 shows the architecture of the CPU. RISCY is a 4 stage pipelined core with a 32-bit data and instruction interface.

To enable interfacing with peripherals of the core, an instruction and data wrapper are created to fetch the instructions from memory at the instruction interface and the data from the interconnect bus and the peripherals on the data interface. The instruction wrapper is located at the I$ blue interface in figure 5.8. The data wrapper is located at the right interconnect blue box. The debug interface on the left was left not-connected as implementing this is quite complex and prone to

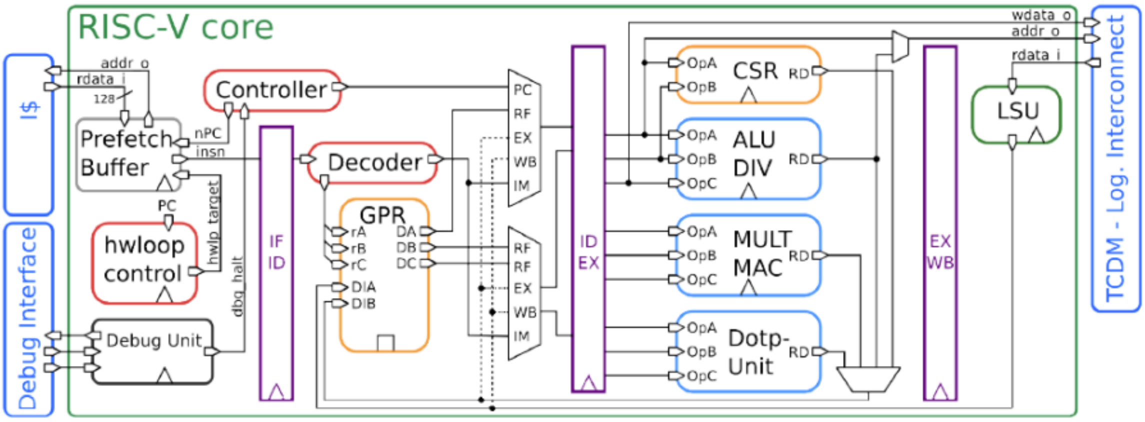


Figure 5.8: RISCY Core

failure.

Lastly the RISC-V has a vectored interrupt controller which can, based on a 32 bit vector, interrupt the RISC-V core to focus on a specific peripheral. The interrupt vector implemented is as follows. Priority is given to the peripheral with the highest bit in the vector. So in this case the priority is the AES peripherals, the timer, the uart and then the SPI (as this is only used for boot).

Interrupt vector = [22'd0, AES0_IRQ, AES1_IRQ, AES2_IRQ, AES3_IRQ, AES4_IRQ, AES5_IRQ, TIMER_IRQ, UART_IRQ, SPI_IRQ]

5.2.4. Interconnect Bus

To facilitate the data transfer between the RISCY core and the data memory, and the RISCY core and the SoC design peripherals, an interconnect bus is needed. This interconnect bus is crucial to the design of a reliable SoC. As every interaction between pieces on the SoC happen via the interconnect bus, it needs to be very reliable.

In order to make the interconnect bus as reliable as possible, a standard bus interface was chosen. As the interconnect bus on the SoC needs to facilitate under 16 peripherals, 1 bus master at a 100MHz clock, the AHB-lite interconnect was chosen as the to be implemented interconnect. The AHB-lite interface is a pipe-lined 32-bit wide interconnect bus with separate read and write lines but only one address line. The function of the AHB bus is described in chapter 3 in depth.

Normally, slower bus peripherals are facilitated by making a bridge and a slower bus. Although some slower peripherals are available on the bus, the overhead of also implementing an slower APB bridge and APB bridge outweigh the advantage of having one simpler bus and adding some extra memory registers at the slower peripherals to facilitate this.

5.2.5. Memory architecture

Memory architecture determines how well your core runs and the scope of programs you can run. For the RISC-V core there are two distinct memories that need to be taken into account. Instruction memory and data memory. To facilitate small programs on the core but also keep the area as small as possible, it was determined that 16kB of instruction memory and 16kB of data memory was enough to facilitate all resulting test-cases. This is implemented as SRAM block on the chip.

Next to core memory, all peripherals (IO) in the SoC design also need some registers to store variables. An example of this can be, if you want an AES engine to start running you need to be able to send a start signal to it. In Raescy this is facilitated by a concept knows as memory mapped IO. Meaning all IO devices share the same memory space but different memory locations. Figure 5.9 shows the complete memory map of Raescy. Every peripheral located on the interconnectbus of the RISCY core needs to have an address in the higher 16 bits to facilitate the AHB decoder. Everything that does not have a 16 bit addressing is directly interfaced with the RISCY core.

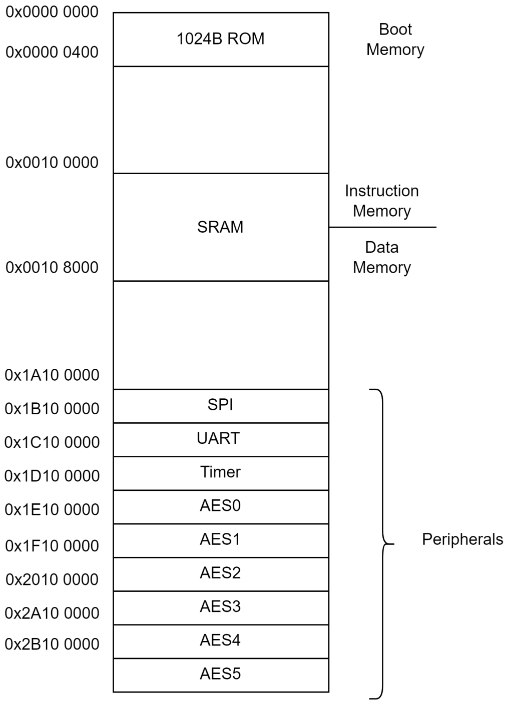


Figure 5.9: REASCY Memory map

Next to the data and instruction memory there is one other type of memory that needs to be taken into account. In order to make the RISC-V core flexible in its programs it can run, the program needs to be able to change and be loaded into the main memory every run. To facilitate this, a boot program, which is a program which initializes the core can be implemented. The main purpose of this boot program is that, upon startup, the RISC-V core fetches the new program externally and copies it to the internal instruction and data memory. After the copying the boot program points the core where to find the first instruction and ensures the core jumps to that address.

Boot loader code is usually hardcoded in Read only Memory (ROM) on the SoC. Meaning its hard coded directly in the design. In the platform it was determined that about 1kB was needed to facilitate the full boot program. Figure 5.9 shows the boot ROM at the location 00 which is the reset address for the RISC-V meaning that, upon start up, the bootloader will be run.

5.2.6. SPI

To facilitate the loading of instructions and copying them to internal memory, Quad SPI can be used. This is due to the nature of external memory. Most external memories implement a Quad SPI standard [26]. This design implements an adaptation of the Pulp-Platform QSPI [66] implementation for this. To fully understand SPI, chapter 3 can be used.

5.2.7. UART

To make the design not only functional but also flexible and transparent, a UART peripheral is added. Over the UART bus, updates and the final AES outputs can be communicated to the outside world over serial bus. UART was used at it made the resulting external interface flexible as it is supported by a lot of serial devices. Meaning that either a microcontroller or any other controller can communicate with the SoC design later. To fully understand UART, chapter 3 can be used.

5.2.8. Timer

To further future proof the design, two hardware timers where implemented. These hardware timers consist of a simple 1x clock timer and an upscaled x16 timer. These timers faciliate future software designs by enable programmers to set delays and create system ticks.

5.2.9. JTAG

To make the design more redundant and reliable, a debug interface is implemented in the form a JTAG debug port. The JTAG module can take control of the design in case the RISC-V does not function properly. Figure 5.10 shows the function of the JTAG interface. JTAG can be used to fully control the entire AHB bus, meaning it can put instructions into memory in case the SPI doesnt work, it can send updates and readout the AES output if needed and run the AES if needed.

In this SoC, the CPU sub-Module is not connected for safety purposes. Next to that, the AXI submodule is changed to a AHB-Lite submodule. This module can be used with OpenOCD software [59].

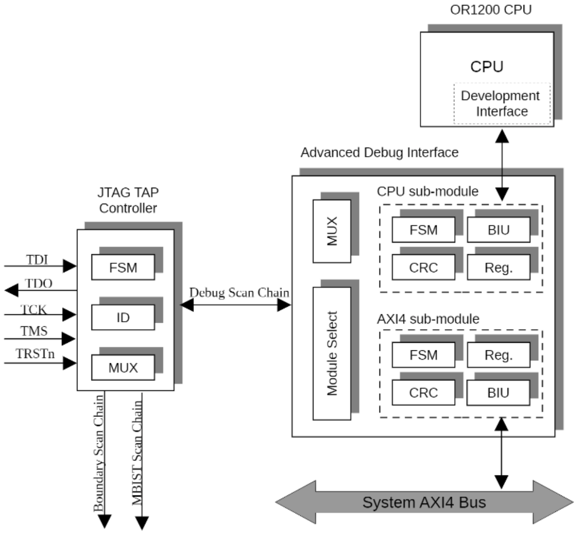


Figure 5.10: Jtag debug interface

5.3. Software Design

In parallel with the SoC design, it is critical to ensure that the software for the platform can in fact be run on the platform. To ensure that software can be run on the platform, all the platform peripherals needs to be translated into code.

5.3.1. Software backend Pipeline

As the core, interconnect and memory implementation are custom, the compiling, assembling and linking also needed to be custom. Figure 5.11 shows the pipeline from c code on the left to memory .dat files on the right.

As the design of a software pipeline is extremely complicated, for this implementation pulp platform's [66] pulpino board implementation was used as an example template. Although they implement it according to their architecture, having the same core it gave a good starting point.

Compiling

For the compilation, the GNU RISC-none-elf compiler was used from the GNU RISC-V Embedded GCC platform [67]. To facilitate the RISC-V architecture in the design, the instruction set was chosen to bethe RV32I with the multiply and the Zicrs extension according the RISC-V ISA manual [82].

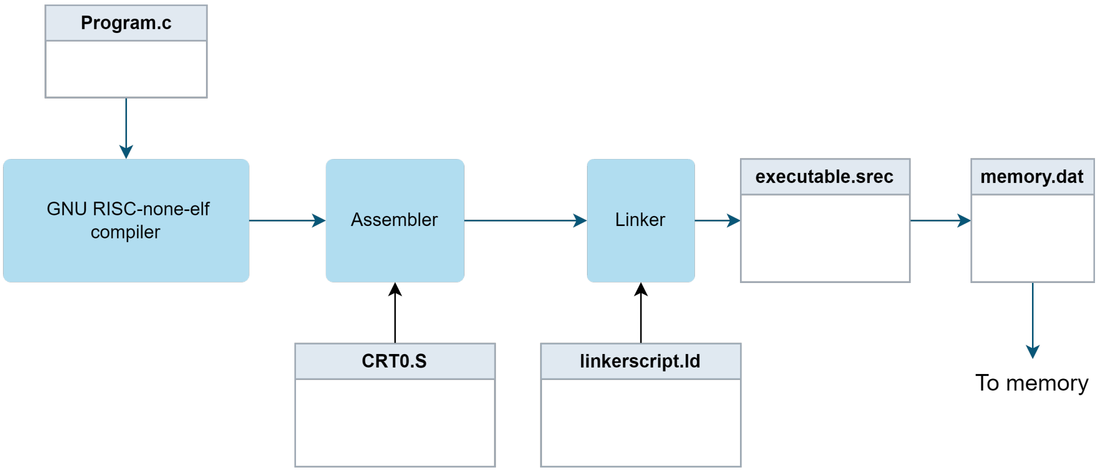


Figure 5.11: Software pipeline

Assembling

After compiling, the next stage is assembling. For the assembly, a start up assembly script needs to be made. This is generally known as the crt0.S script. The crt0 signifies the initial start up file. It contains the symbol _start that is both the default base address for the application and the first symbol in the executable binary image. Next to that, it clears the stack and defines the reset and illegal instruction handling which is important in case of reset or illegal instructions.

Figure 5.3.1 shows a tiny portion of the start up script where the _start symbol is defined and, in the main entry, the jump to the main code of the program is made.

```
_start:
  .global _start

  /* clear BSS */
  la x26, _bss_start
  la x27, _bss_end
  addi x10, x0, 0
  addi x11, x0, 0
  bge x26, x27, zero_loop_end

main_entry:
  addi   x10, x0, 0
  addi   x11, x0, 0x1
  //jal  uart_set_cfg;

  /* jump to main program entry point (argc = argv = 0) */
  addi x10, x0, 0
  addi x11, x0, 0

  jal x1, main

  mv s0, a0
  mv a0, s0
  /* if program exits call exit routine from library */
  jal  x1, exit
```

Listing 5.1: Assembly start up script

Linking

The linking phases patches all assembly code into the different memory regions of the platform. As discussed in the section 3.6.3. For the linker of the platform, the memory architecture in section 5.2.5 needs to be incorporated. This is done my augmenting the pulpino linker script [65] , which uses the same core, with the platform's custom memory architecture.

```
MEMORY
{
    instrram     : ORIGIN = 0x00000000, LENGTH = 0x4000
    dataram      : ORIGIN = 0x00104000, LENGTH = 0x2000
    stack        : ORIGIN = 0x00106000, LENGTH = 0x2000
}
```

Listing 5.2: Memory linking script

5.3.2. Pheripheral code

In order to abstract each peripheral into code which can be used to control it a few things needs to be implemented. Firstly a complete register map needs to be made of each peripheral. These registers control the complete execution of the peripheral. Then with the registers, functions need to be created which set or readout these registers. In the following paragraphs the focus will be on the AES engine implementation for simplicity. The full register-maps and coding can be found in the appendix.

Registermap

Each peripheral interface on the interconnect bus has a certain set of 32 bit registers in its design, these registers are used to control the entire peripheral, store input/output values in and send the current status of the that peripheral. An example of this is if the AES needs to be started, a seed value needs to be send to the AES engine. In this engine, a register is assigned with the sole purpose of storing this seed.

Table 5.3 shows the register map of an AES engine. First the base address is shows as $0x1D10\_0000$. The first register is then the status register. Currently the status register is not used. Then the config register which determines the entire state of the engine. Bit 2 initializes the engines, bit 3 set the mode to encryption or decryption, bit 4 resets just the LFRS's, bit 5 restarts the engine if the keys need to be updated midway through and bit 6 set the output trigger to the Oscilloscope. The rest of the register are to load a 128 bit key and data seed or if an extended 256 bit seed is needed, extra registers are added. Lastly the Count register stores the amount of iterations that need to be performed.

Table 5.3: Register map of AES engine 0

Pheripheral	Registername	Description	Address	Reset	Bit definitions				
LFSR+AES0			0x1D10						
	STATUS_REG		0x1D10 0000	0	DONE				
	CONFIG_REG		0x1D10 0004	0	START[2]	ENC/DEC[3]	LFSR_RST[4]	RESTART[5]	TRIG[6]
	KEY_REG[31:0]		0x1D10 0008	0	KEY_REG[31:0]				
	KEY_REG[63:32]		0x1D10 000C	0	KEY_REG[63:32]				
	KEY_REG[95:64]		0x1D10 0010	0	KEY_REG[95:64]				
	KEY_REG[127:96]		0x1D10 0014	0	KEY_REG[127:96]				
	PL_REGISTER[31:0]		0x1D10 0018	0	PL_REGISTER[31:0]				
	PL_REGISTER[63:32]		0x1D10 001C	0	PL_REGISTER[63:32]				
	PL_REGISTER[95:64]		0x1D10 0020	0	PL_REGISTER[95:64]				
	PL_REGISTER[127:96]		0x1D10 0024	0	PL_REGISTER[127:96]				
	KEY_REG2[31:0]		0x1D10 0028	0	KEY_REG2[31:0]				
	KEY_REG2[63:32]		0x1D10 002C	0	KEY_REG2[63:32]				
	KEY_REG2[95:64]		0x1D10 0030	0	KEY_REG2[95:64]				
	KEY_REG2[127:96]		0x1D10 0034	0	KEY_REG2[127:96]				
	PL2_REGISTER[31:0]		0x1D10 0038	0	PL2_REGISTER[31:0]				
	PL2_REGISTER[63:32]		0x1D10 003C	0	PL2_REGISTER[63:32]				
	PL2_REGISTER[95:64]		0x1D10 0040	0	PL2_REGISTER[95:64]				
	PL2_REGISTER[127:96]		0x1D10 0044	0	PL2_REGISTER[127:96]				
	COUNT		0x1D10 0048	0	COUNT[31:0]				
	OUT_REGISTER[31:0]		0x1D10 004C	0	OUT_REGISTER[31:0]				
	OUT_REGISTER[63:32]		0x1D10 0050	0	OUT_REGISTER[63:32]				
	OUT_REGISTER[95:64]		0x1D10 0054	0	OUT_REGISTER[95:64]				
	OUT_REGISTER[127:96]		0x1D10 0058	0	OUT_REGISTER[127:96]				

Function code

Having the register map of the AES engine, and understanding what bit does what means it is now possible to start parameterize this into functional code. Say the AES needs to do 1000 cycles of encryption with a seed of key_seed and data_seed this can now be parameterized the following way.

```

    //Set mode to encryption
  *(volatile int*) (AES_REG_CONFIG) =  ((1 << 3) & 0x08);

  // Setup count
    *(volatile int*) (AES_REG_COUNT) = 1000;

  // Initialize seeds
    *(volatile int*) (AES_REG_KEY0) = key_seed[3];
    *(volatile int*) (AES_REG_KEY1) = key_seed[2];
    *(volatile int*) (AES_REG_KEY2) = key_seed[1];
    *(volatile int*) (AES_REG_KEY3) = key_seed[0];

    *(volatile int*) (AES_REG_PL0) = data_seed[3];
    *(volatile int*) (AES_REG_PL1) = data_seed[2];
    *(volatile int*) (AES_REG_PL2) = data_seed[1];
    *(volatile int*) (AES_REG_PL3) = data_seed[0];

  // Start the engine
  *(volatile int*) (AES_REG_CONFIG) =  ((1 << 2) & 0x04)| ((1 << 4) & 0x10)| ((1 << 6) & 0x40)

  // Wait for interrupt to tell us that the AES is done
  while(!AES_get_status()){
    asm volatile ("nop");
  }
    // Get the results back from the engine
  out[0] = *(volatile int*) (AES_REG_OUT0);
  out[1] = *(volatile int*) (AES_REG_OUT1);
  out[2] = *(volatile int*) (AES_REG_OUT2);
  out[3] = *(volatile int*) (AES_REG_OUT3);
```

Listing 5.3: AES running C code

5.3.3. Bootloader architecture

After writing all the peripheral code and setting up the pipeline there is only one part left, the bootloader code. For the bootloader code, as it works with a different memory than the conventional code, the linker script need to be redesigned to fit the memory architecture to only the ROM memory. The linker script can adapted as in the following code block.

```
    rom         : ORIGIN = 0x00008000, LENGTH = 0x400
```

Listing 5.4: Linker for bootloader

Then the bootloader code itself goes through the following steps. First the SPI communication is checked, to verify that a connection can be established with the external memory. The functional part can boiled down to two for loops and a jump, the for loops include the copying the instruction ram first and then the data ram. Codeblock 5.3.3 shows the copying of the instruction memory over QSPI and the final jump function to the instruction start address which is a basic assembly jump to a pointer.

```
  addr = instr_start;
  spi_setup_dummy(8, 0);
  for (int i = 0; i < instr_blocks; i++) {

    // cmd 0xEB fast read in Quad Mode, needs 8 dummy cycles
    spi_setup_cmd_addr(0xEB, 8, ((addr << 8) & 0xFFFFFF00), 32);
    spi_set_datalen(32768);
    spi_start_transaction(SPI_CMD_QRD, SPI_CSN0);
    spi_read_fifo(instr, 32768);

    instr += 0x400;  // new address = old address + 1024 words
    addr  += 0x1000; // new address = old address + 4KB
  }
 jump_and_start((volatile int *)(INSTR_RAM_START_ADDR));

-------
void jump_and_start(volatile int *ptr)
{

  asm("jalr x0, %0\n"
      "nop\n"
```

```
        "nop\n"
        "nop\n"
        : : "r" (ptr) );
}
```

Listing 5.5: Copying of data ram over SPI, C code

5.4. Final ASIC Design

After designing the SoC design, the next step was the translation into an ASIC architecture. As the final ASIC will be printed at TSMC on 40nm technology the logic gates in the RTL design now need to be translated onto the technology. This can be done using the TSMC 40 nm library and the Cadence design suite [8]. Next to technology mapping, to make ASIC synthesis easier, a few adaptations need to be made to the design.

Looking at the memory architecture in section 5.2.5, note that the Instruction and Data memory are combined in one SRAM but in different regions. This was implemented for the practical reason of being easier to place in the ASIC place and route as memory blocks are instantiated as black boxes. This is the same for ROM architecture.

Then the last tweaks that need to be done to the ASIC design is the implementation of a clock buffer and the level shifting of the digital pins.

5.4.1. Final ASIC layout and pinout

Figure 5.12 shows the final ASIC layout after the place and route. Next to that, the pin out is visible. To facilitate and isolate every peripheral, grounds and power lines are added on each corner of the design.

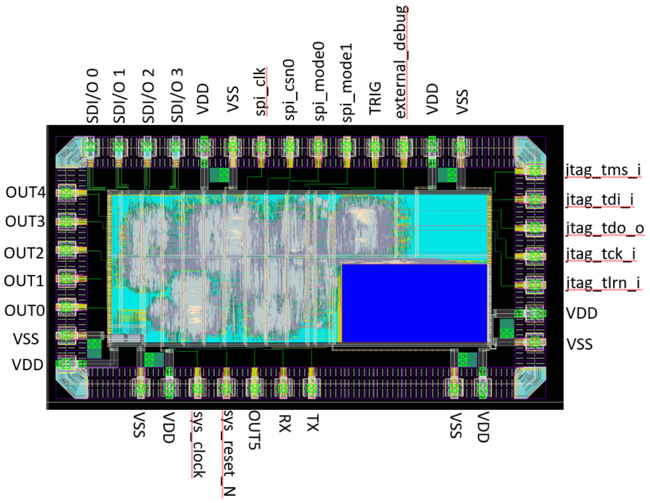


Figure 5.12: Final ASIC layout and pinout

The final design has the following specs:

IO voltage 1.0V supply voltage.

Pin out 38 pins, 14 digital pins capable up to 3.3V and 24 analog pins capable up to 1.0V.

Clock 100Mhz single clock interface with a clock tree.

5.5. PCB Design

After designing the SoC design and the software architecture, the last step in the platform design is the design of a Printed circuit board (PCB). To facilitate the power side channel measurements on an FPGA or SoC ASIC, the PCB design needs to be agnostic of the resulting DUT. In the following sections the main focus is on the pre-silicon PCB design as the ASIC silicon has not yet returned and thus the resulting design has not be finalized. It was decided to split the PCB design up into two separate PCB's, a Carrier board for all the peripheral interfaces and a Daughter target board with either an FPGA or an ASIC purely for measuring the power. There are several factors that led to this decision, the following being the most important:

- **Modularity and Scalability** The most important thing is that the platform is not a custom solution for a single problem, it is a solution on which the next solution can extend on. As the platform evolves, additional modules can be added or modified without affecting the entire system. A new chip can be easily facilitated then.
- **Ease of Debugging and Testing** By splitting the problem into different PCB's the platform will be easier to debug and test. When issues arise, it is easier to see where the problem is. Next to that, it isolated tested on each PCB helps finding issues earlier and without breaking expensive parts.
- **Protection and Reliability** Protection of expensive is easier to implement when separating a design into modules. If the carrier pcb breaks, the daughter board can be more easily shielded from that failure. Additionally, in case of a failure, it's simpler to replace or repair a single module than a complex integrated board.
- **Design Simplicity and Flexibility** By separating one complex design into 2 smaller simpler parts, each part is quite simple. Meaning it can more easily be understood. Next to that, with having a carrier board facilitating all common interfaces means that new daughter board designs are easily made on top of the common interfaces.
- **Improved power domains** The biggest issue with power side channel assessment is the influence of platform noise on the power side measurements. Referring back to section 2.5.1, a good measurement setup adds minimal noise to the power side measurement. By splitting the design into a carrier board with all common interfaces and a small daughter board with only the DUT or measurements it is easier to isolate the power lines and decouple system noise.

5.5.1. FPGA Target board Overview

The FPGA board is the daughter board for the pre-silicon power side channel analyses. The FPGA chosen is the Kintex K70T, this is the smallest kintex series FPGA which is caple of holding the complete SoC design. This FPGA is a Xilinx made FPGA meaning the integration between the rtl design and uploading designs to the FPGA is as seamless and straightforward as possible. Leading to future PHD and Msc students having an easy operation.

Figure 5.13 shows the complete block diagram of the FPGA pcb. On the left a stacking connector to the carrier board is depicted. On the right, the connectors to the specific interfaces on the FPGA board are depicted. The following subsections detail the specific blocks inside the design.

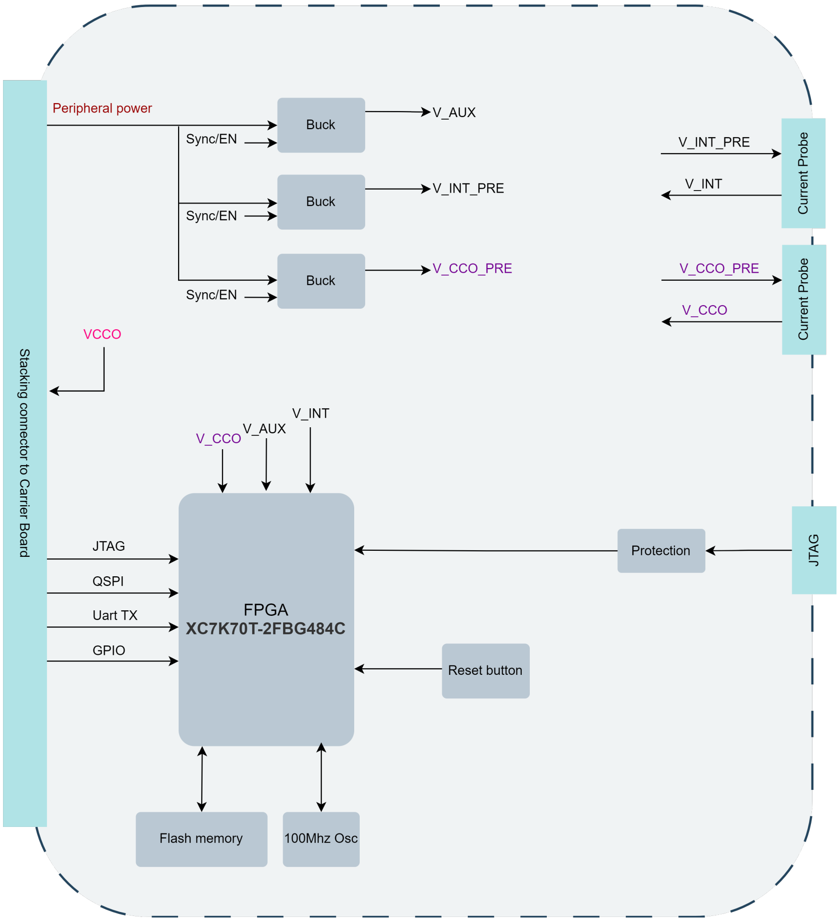


Figure 5.13: FPGA board block diagram

Power consumption

The most tricky part of a design with an FPGA is the power consumption and distribution. The kintex-7 series comes with three separate power domains; The VInt which is the internal supply voltage powering the internal core and interconnect, the VCCO which powers the different GPIO banks of the FPGA and the VAUX which is the auxiliary power powering transceivers, internal clocks and phase lock loops.

Each of these rails have different power consumption profiles and different operation voltage levels. Table 5.4 shows the different supply currents during static and operational mode for each rail.

Power Rail	Operating voltage (V)	Quiescent Supply Current (mA)	Power On Current (mA)	Min current in Design (mA)
VInt	1.0	241	450	691
VCCO	2.5	1	40 per Bank	161
VAUX	1.8	2	40	42

Table 5.4: Power consumption of FPGA power rails

To facilitate this and ensure the power lines have enough margin, a 3A max current consumption was chosen for the internal line and a 2A max current consumption for the VCCO and the VAUX. This results in a power budget of 12W Peak power.

Power line stability

To facilitate clean power line measurements and ensure reliable operation of the FPGA, the different buck converters powering each line need to be phase synced and sequenced. Following the data sheet of the FPGA the recommended power-on sequence is VCCINT, VCCAUX, and

VCCO with 5ms delays to achieve minimum current draw and ensure that the I/Os are 3-stated at power-on. The recommended power-off sequence is the reverse of the power-on sequence [84]. This is faciliiltated using a power sequencer chip with adjustable delay; the LM3881. The LM3881 can be set to the exact 5ms delay but to ensure enough margin, 50ms was chosen instead. Next to sequencing, to minimize the effect of each power line's influence on the others, the phases of the buck converters can be controlled by a 3-phase oscillator, spreading the spectrum over a larger bandwith.

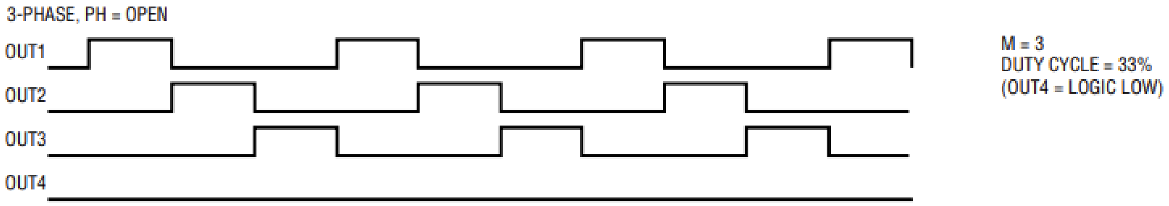


Figure 5.14: Multi-Phase output [55]

Power Measurements

To facilitate the power measurements on the FPGA it is important to know the interface for the measurements, in this case the power side channel measurements will be done with the Riscure current probe [70]. The big question is thus, which power rails of the FPGA needs to be measured in the power side channel measurements. Looking at the FPGA datasheet, the VCCO powers the banks which is where the custom SoC design will be programmed on. This means that the encryption and dycryption happens here. Meaning this is the line that needs to be observed. Following the riscure probe datasheet, the probe needs to be placed in line with the power of the to be measured power supply. Figure 5.15 shows this.

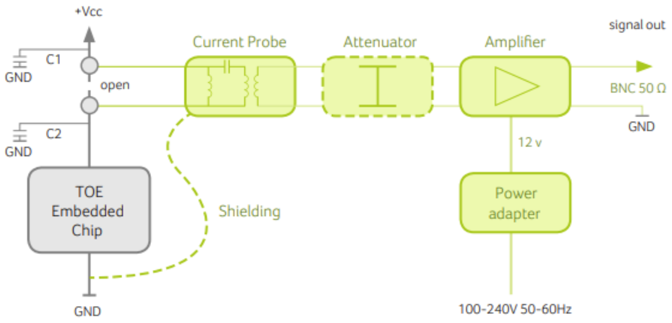


Figure 5.15: Riscure probe implementation

But to also enable characterization of the FPGA's internal noise, an extra probe interface was put on the VCCINT line, both can be easily bypassed by inserting a short.

Programming

The programming of the FPGA can be done in different modes called Boot modes, meaning where the RTL design is loaded from can be configured. To facilitate this, the FPGA has three mode pins which can be set to 0 or 1.

Configuration Mode	M[2]	M[1]	M[0]
JTAG	1	0	1
QSPI	0	0	1

Table 5.5: Configuration bit modes on the Kintex-7

To reconfigure the FPGA, the JTAG mode is needed. With JTAG it is possible to configure the FPGA using a diligent JTAG programmer [22] right from the Vivado window.

Then for repeated testing, to facilitate not having to re-upload a new configuration upon each power cycle, a piece of QSPI flash memory can be used in the QSPI programming mode. The flash memory can store the previously loaded configuration. The PCB has a switch that can be set to which boot mode is necessary.

Final FPGA PCB layout

Figure 5.16 shows the final PCB design. On the right, the green connectors are the interfaces with the Riscure probe. On the bottom, the QSPI and some test points are located. On the top left, the JTAG interface is located with the bootmode switch underneath. On the bottom of the PCB, the interface with the carrier board is located.

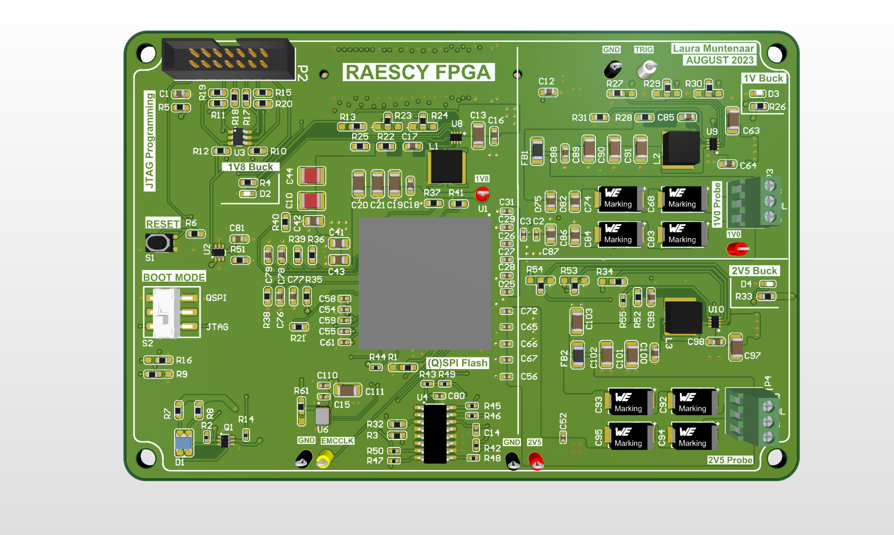
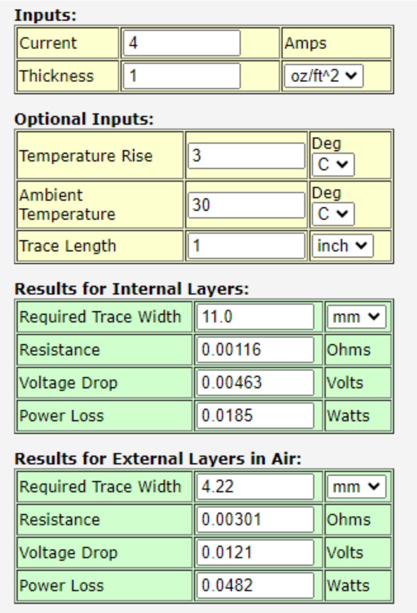


Figure 5.16: Final FPGA PCB

The final PCB is a six layer pcb to facilitate all the FPGA pins. To maintain good signal quality, two of them are implemented as ground. And the other four are signal layers. To facilitate the best possible signal integrity the grounds are placed surrounding the signal layers of the FPGA. This protects them from the power layers at the top and bottom of the PCB.

When laying out a PCB like this one, the most important part is the layout of the power rails. When laying out a PCB, one vital aspect that needs to be taken into account is the current running trough a wire (called a trace) The current running through a trace has effects on the temperature of that trace, if the trace is too small, it might burn off because it gets too hot. As this PCB facilitates power lines of 3A, the width of the trace connecting the input and the FPGA need to be around 4mm in our current configuration to get a 3 degree temperature rise in 30deg ambient. Figure 5.17 shows these calculations using the tracewidth calculator [77].



Inputs:

Current	4	Amps
Thickness	1	oz/ft^2

Optional Inputs:

Temperature Rise	3	Deg C
Ambient Temperature	30	Deg C
Trace Length	1	inch

Results for Internal Layers:

Required Trace Width	11.0	mm
Resistance	0.00116	Ohms
Voltage Drop	0.00463	Volts
Power Loss	0.0185	Watts

Results for External Layers in Air:

Required Trace Width	4.22	mm
Resistance	0.00301	Ohms
Voltage Drop	0.0121	Volts
Power Loss	0.0482	Watts

Figure 5.17: Trace width calculations

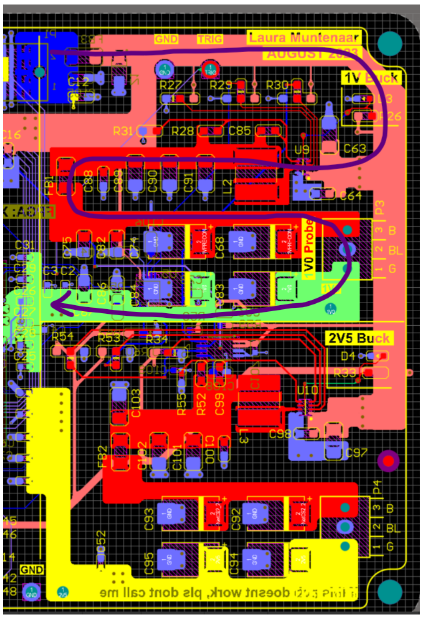


Figure 5.18: Polygon based Power traces

To facilitate even better power transfer, polygon regions can be used. These polygons are defined copper regions in stead of small traces. Figure 5.18 shows the implementation of these polygons on the 1V line and the power transfer from the input to the FPGA on the left. Here also the layout of the capacitors can be seen, which are directly laid out on the trace and directly connected to ground.

The rest of the layout can be summarized by following the layout example of every chip and facilitate the shortest datapath as possible between devices.

5.5.2. Carrier board Overview

The carrier board is the main interface between the outside world and the measurement setup, it receives the input power, houses the flash memory for uploading software and provides enough IO interfaces for custom configurations or tests.

Figure 5.19 shows the full block diagram for the carrier PCB. The design mainly consists of five parts which are the power, communication, software programming, GPIO's and Level shifting.

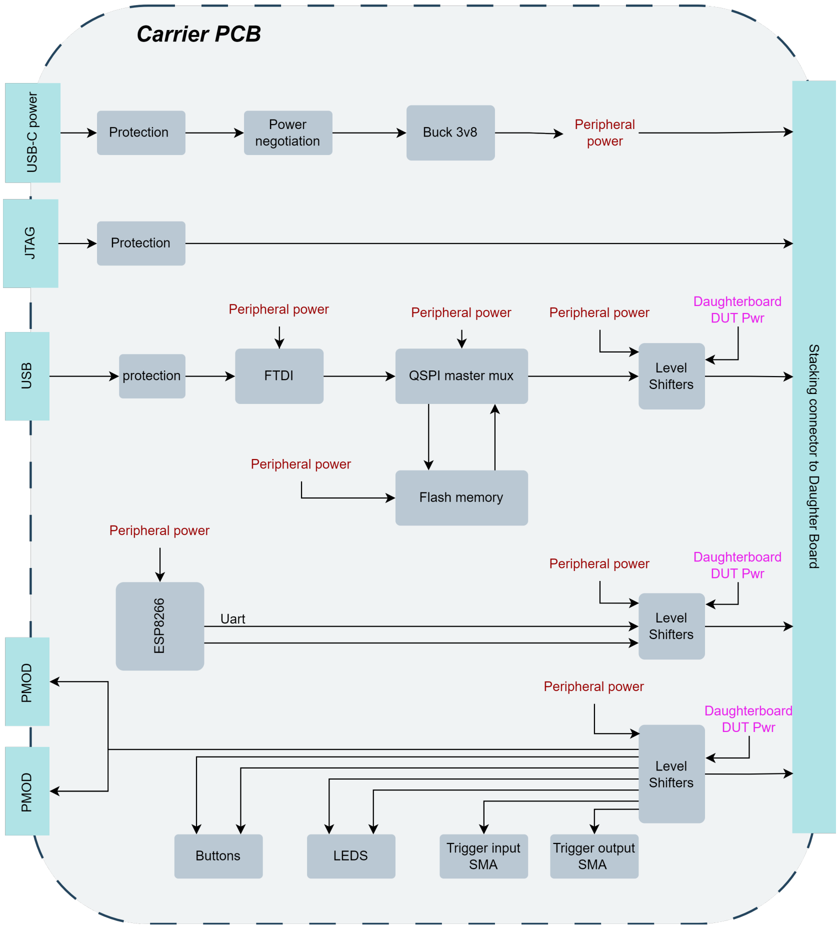


Figure 5.19: Carrier board block diagram

Power input and rail power

For the power input, a standard USB type C connector is used as this is a standard port and ensures ease of operation. Next to that, the USB type C protocol [71] [25] support the delivery of high power at relatively low voltage.

Referring back to the power consumption of the FPGA and assuming the power of the SoC will be lower, it becomes visible that the peak supply power needs to be around 15W to be safe. Looking at the USBC protocols, the USB Type-C 1.2 protocol seems to be perfect to facilitate this as it asks for 5V, 3A.

As the platform is going to be operated by other Msc or PHD students, the design needs to be resistant to receiving wrong USB-C protocol power input and wrong cable orientation. To prevent the board from starting up with wrong power, the USB-C Data and CC pins can be used. The Data pins can negotiate with the input power supply to give the maximum current where the CC pins can detect correct cable orientation and port control. From this, a complete visual characterization of what power is supply can be given to the user.

The last thing that the power needs to protect for is the case of under voltage. Under voltage occurs when the load on the power rails is too high. Devices on the rail will then operate on lower

Status	USB connection Invalid	USB-C connection Invalid	USB 1.5A connection	USB-C 1.5A connection	USB-C 3A connection
Related led	Error (Red)	Error (Red)	Valid (Green)	Valid (Green)	Correct (Green)

Table 5.6: Power status Leds

voltage but as their power consumption remains the same, they will start drawing higher current. This can lead to fatigue and device failure. To detect this, the input and output voltage rails need to be monitored closely. When the voltage drops below the preset under voltage threshold, the power rails should be disconnected until the system is power-cycled. In this way, the issue can be observed, investigated and then safely re-engaged. To facilitate this, a power on reset (PoR) is implemented which, in case under voltage happens, the system is automatically safely powered down. Once started up, the PoR button can be pressed, this will reset the under voltage condition and the power can safely be applied.

All the above can be combined into a complex power on logic which is visualized with basic logic gates in figure 5.20.

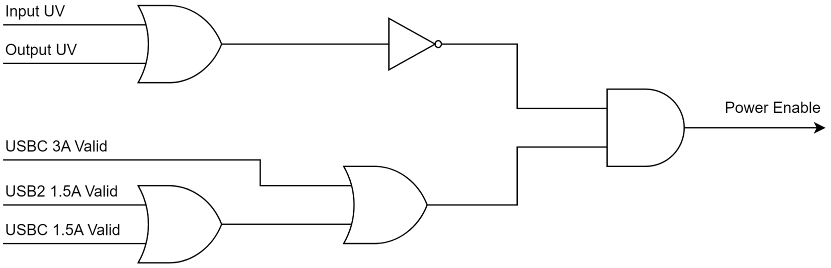


Figure 5.20: Power on Logic

Programming of SoC software

As previously mentioned the bootcode of the SoC includes the copying of external flash memory to the internal memory over spi. For this, external SPI flash memory is needed. But how is the software uploaded onto this SPI flash memory? For this, an FTDI(Future Technology Devices International) chip [28] with USB port and an SPI mux can be used. An FTDI chip can interface with USB over a standard USB port. Meaning a direct connection between a laptop and the FTDI chip can be made.

The SPI of the FTDI and the SPI bus of the SoC can then be muxed into one bus which interfaces with the Flash memory. This muxing is done based on if the USB port is connected or not. If the USB is present, the mux will select the FTDI chip as main SPI bus to upload data to the memory. If the USB is not present the mux will default to select the SoC design to extract the data from the memory.

UART serial interface

To facilitate the SoC serial interface for updates a UART interface needs to be interfaced with. This is done with the use of an ESP-01 which can interface with UART over wifi. Meaning the operator can connect with the SoC over the network by simply opening a terminal port. The communication is established by sending simple commands over uart from the SoC to set the IP address and the port number.

PMOD connectors and GPIO's

Lastly to increase the versatility and allow for future projects or different FPGA usages, two diligent pmod interfaces[21] like the VGA module in figure 5.22 are facilitated by implementing the correct connectors. These PMOD interfaces are designed by diligent and have versatile applications. If, in the future, a extra SD card is needed for example, a PMOD connector for this can be directly connected to the connectors.

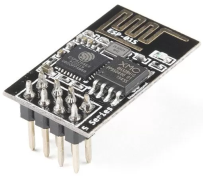


Figure 5.21: ESP 01 Wifi UART module

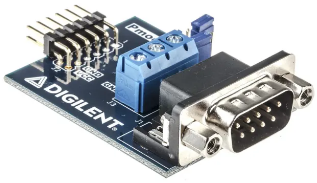


Figure 5.22: Example PMOD interface [20]

Next to PMOD connectors, 6 GPIO's are implemented. Figure 5.23 shows the gpio implementation. Out of the six, four GPIO can be implemented as a testpoint, led or button dependent on what the user wants. These are implemented to facilitate any type of extra interface that is needed on the target board. (eg. the out pins of the AES on the SoC design).

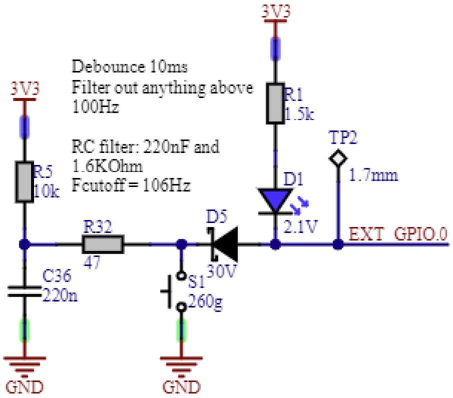


Figure 5.23: GPIO architecture

Lastly, the two extra GPIO's are connected with an SMA connector if, in the future, extra measurement points are needed on the target boards.

Level Shifters

To facilitate both the voltage level of the FPGA as well as the voltage level of the SoC or any other target board, level shifter or level translators can be used. These level shifters are used on any signal between the target board and the carrier board. This includes the JTAG, UART, QSPI and GPIO's. This makes the design completely independent on the target device's voltage. For the level shifters, the LSF0108 from texas instruments is used. This is an auto-bidrectional level shifter able to shift voltage in between the range of 0.9V-5V. Figure 5.24 shows the internal architecture of the levelshifter shifting from 1.8 to 3.3V.

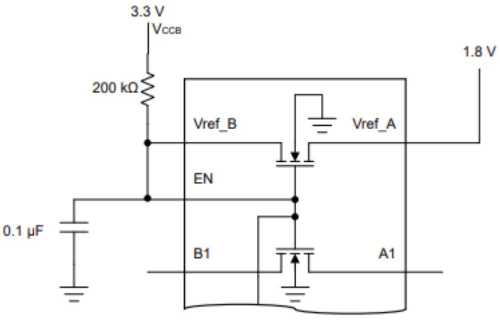


Figure 5.24: Level-shifter internal architecture

The levelshifting detects a high voltage on one side and will pull that line up to the required voltage level on the other side.

Final Carrier PCB layout

Figure 5.25 shows the final PCB design. On the left, the USB-C power, leds and its testpoints are placed. On the bottom and right side, all the interface peripherals with the target board are placed, the JTAG, QSPI-flash, the PMODS, the SMA GPIO's and the normal GPIO's. In the middle area all the levelshifters ar located.

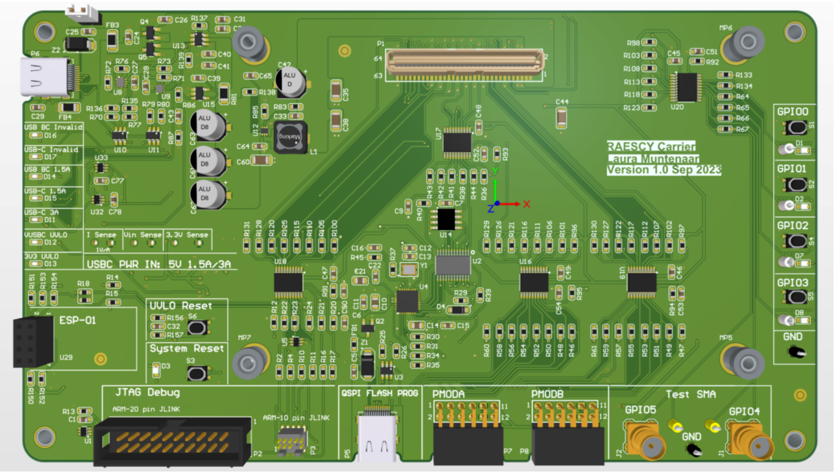


Figure 5.25: Final Carrier PCB

The final PCB is a four layer pcb with two trace layers and two GND layers. This to facilitate good isolation between both signal layers on the top and bottom. To facilitate minimal interference with the target PCB, most "noise" generating components are places outside of where the target PCB will go. The layout of this pcb was only tricky in making all the interfaces usuable. So ensuring that the testhooks of the GPIO's can be used whilst a user presses a button etc.

5.5.3. PCB combination result

Figure 5.26 shows the complete platform stack up. On top, the carrier board with the FPGA and on the bottom the carrier board. To connect these two boards, mezzanine connectors are used. Mezzanine connectors facilitate high speed data tranfer and specifically made for board to board connections. To keep the target board on top, 4 standoffs are implemented which have their own mounting holes on the target board.

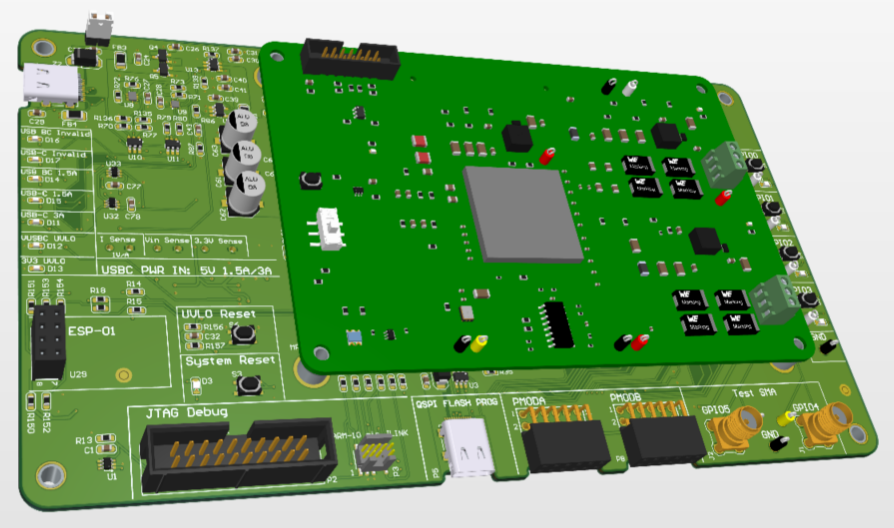


Figure 5.26: Stackup of carrier and target PCB

6

Validation and Results

This chapter outlines the test setups and validation of the platform. First, the functionality of the platform is verified. Then simulation power side channel measurements are taken and evaluated. ***The results presented in this chapter are bounded by just the SoC implementation design.*** *This is because the PCB and the SoC will not be back soon enough to commence full testing*

6.1. SoC Functional testing platform

Testing of the SoC/FPGA design happens in two phases. The first phase is testing the full functionality of the design. This includes testing all peripherals in combination with the core and the software. For this, the following test setup can be used.

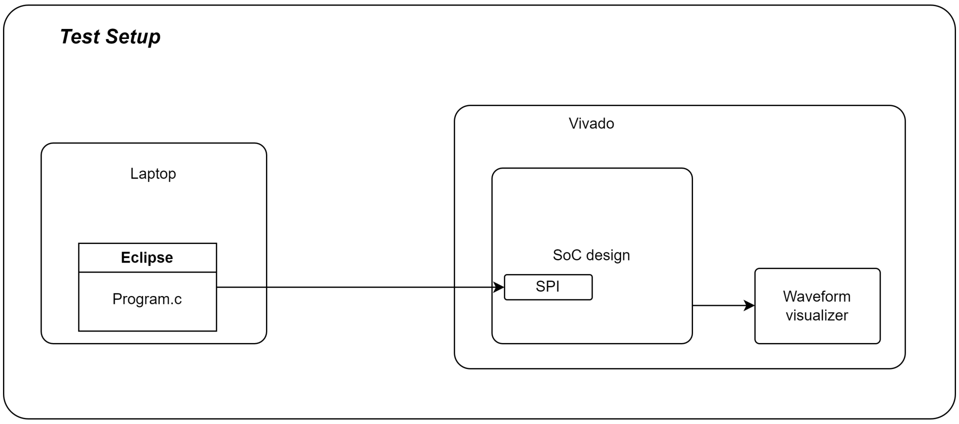


Figure 6.1: Experimental test setup for functionality testing

Figure 6.1 shows the experimental test setup used for the validation. In the image, certain pieces of software can be identified.

- **Eclipse** Eclipse C/C++ IDE [24] is an open-source developer environment that allows for maximum freedom for the user. This includes using custom compilers and generating linkers and assembly code.
- **Vivado** Vivado [85] is a software environment for the analysis and implementation of hardware description languages.

6.2. SoC functional results

Testing the design functionality requires the testing of each peripheral individually. The next sections describe each peripheral's functionality about the power side channel assessment. All peripherals are of course tested, verified and profiled more in-depth during the design and verification place but to avoid overloading information, displayed functional tests are limited to the end goal of the platform.

6.2.1. AES engines

To determine the functionality of the AES engine, the code in section 5.3.2 can be used. This encapsulates all the different parts of the improved AES engine. In this code, the AES engine is toggled to Encryption, and the number of cycles and the seeds are pushed into the engine. Then the engine is started and when it is done, the engine's interrupt will go high and the result will be passed back onto the bus. For this functionality example, AES engine 2 is used. All other engines work the same.

Looking at the code in section 5.3.2, the first step in the process is setting the mode to encryption and putting in the number of iterations that need to be run. In this case, it is set to 1000. Figure 6.2 shows the interconnect bus address on the top, the databus data underneath that and then the config and count register. It becomes clear that the RISC-V puts the correct address on the interconnect bus, the bus selects the correct peripheral and then the data is correctly written to the registers in that peripheral.

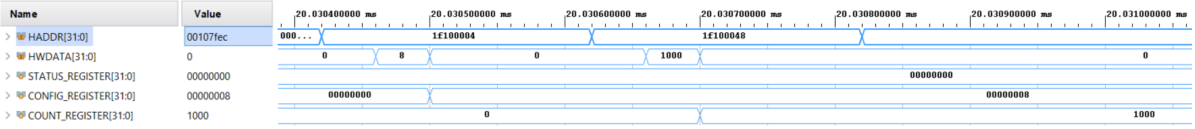


Figure 6.2: Setting mode and setting number of iterations

The next step is loading the key and plaintext seeds into the engine. As the bus is only 32-bit wide, the 128-bit keys and plaintext are written in four different bus cycles. Figure 6.3 shows the writing in of the key 32-bits at a time on the Hwrite bus.

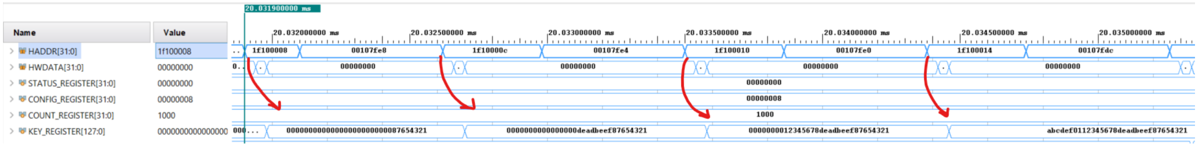


Figure 6.3: Writing in the key

After the key and plaintext seeds are loaded into the engine is it time to start the engine, this is done by resetting the LFSR with their new seeds and then starting the FSM and waiting for the 1000 cycles to be done. In the meantime, the core keeps checking if the status register is not yet set to 1 (indicating the engine is done).

If the 1000 cycles are done, the interrupt will go high and then the core will start requesting the result back from the engine. Figure 6.4 shows the write-back of the result to the core on the Hread bus.

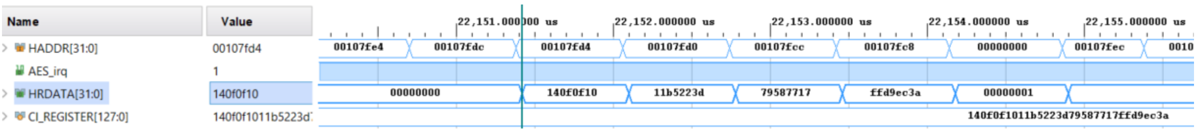


Figure 6.4: Reading back the result

Operation	Time (us)	Clock cycles	% of Time
Total execution	2.134,960	213469	
Encryption time	2.079,440	207944	97.4%
Overhead	55.520	5552	2.6%

Table 6.1: Execution time and overhead

Performance analysis

For the above engine, the speed of the encryption is limited by the overhead of parsing values in and out from the engine takes up clock cycles. Table 6.1 shows the total execution time and the amount that the bus transactions have with the system clock running at 100MHz.

As can be seen, the overhead of the bus interactions is minimal concerning the encryption time. The only issue is that this overhead is fixed. No matter the amount of iterations, these 55 microseconds will be there.

6.2.2. UART

The UART is only used for sending back updates or data to the outside world. To test this, a couple of bytes can be sent over the bus and then observed at the UART TX pin. The code described in 6.2.2 shows the setting of the baud rate to 115200 and then five chars or 5 bytes are sent after each other over the bus.

```
uart_set_brate(1);
uart_sendchar('D');
uart_sendchar('o');
uart_sendchar('n');
uart_sendchar('e');
uart_sendchar('1');
```

Listing 6.1: UART Code

Uploading this code on the platform and observing the UART TX line, the result is shown in figure 6.5. Here it becomes visible that the Brate is set to 1 and then the bytes are parsed bytes by byte and they are translated into binary on the RsTX line.

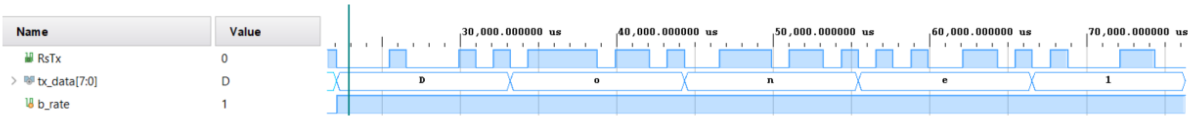


Figure 6.5: Writing bytes over UART

6.2.3. SPI Bootloader

The SPI's main purpose is the interaction with the external flash memory and the copying of the external memory into the main memory. To verify this, the bootloader can be run and then the memory can be checked to see if the bytes in memory are the same as the bytes of code uploaded.

Taking again the code described in 5.3.2, a couple of binaries of the code that come out of the assembler are displayed below. The binaries consist of the address_data where the address is the location in the memory space. As the SRAM memory implemented is 128bit registers memory, the expected output data will be as displayed in code on the right or 4 consecutive bytes in one memory register.

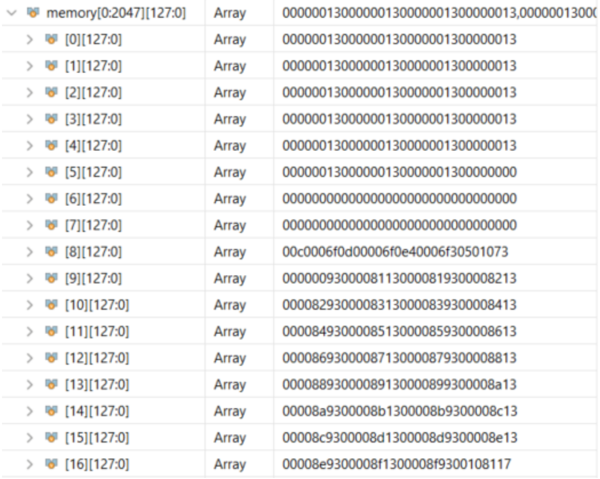
Now observe the memory layout in the sram and decode the addresses. At memory location 000000a0 or in memory terms 28, meaning register number 8. When looking at the memory after running the bootloader the following can be observed. Looking at the bootloading, it takes about 42ms to complete which is under the requirement of half a second we said.

```
000000a0_00C0006F
000000a4_0D00006F
000000a8_0E40006F
000000ac_30501073
000000b0_00000093
000000b4_00008113
000000b8_00008193
000000bc_00008213
000000c0_00008293
000000c4_00008313
000000c8_00008393
000000cc_00008413
000000d0_00008493
000000d4_00008513
000000d8_00008593
```

Listing 6.2: Binary code

```
00C0006F0D00006F0E40006F30501073
00000093000081130000819300008213
00008293000083130000839300008413
000084930000851300008593
```

Listing 6.3: Memory layout



memory[0:2047][127:0]	Array	00000013000000130000001300000013,0000001300(
[0][127:0]	Array	00000013000000130000001300000013
[1][127:0]	Array	00000013000000130000001300000013
[2][127:0]	Array	00000013000000130000001300000013
[3][127:0]	Array	00000013000000130000001300000013
[4][127:0]	Array	00000013000000130000001300000013
[5][127:0]	Array	00000013000000130000001300000000
[6][127:0]	Array	00000000000000000000000000000000
[7][127:0]	Array	00000000000000000000000000000000
[8][127:0]	Array	00c0006f0d00006f0e40006f30501073
[9][127:0]	Array	00000093000081130000819300008213
[10][127:0]	Array	00008293000083130000839300008413
[11][127:0]	Array	00008493000085130000859300008613
[12][127:0]	Array	00008693000087130000879300008813
[13][127:0]	Array	000088930000891300008993000008a13
[14][127:0]	Array	00008a9300008b1300008b9300008c13
[15][127:0]	Array	00008c9300008d1300008d9300008e13
[16][127:0]	Array	00008e9300008f1300008f9300108117

Figure 6.6: Memory layout after bootloader

6.2.4. JTAG

Next to loading a program into memory over spi, the JTAG debug peripheral can also be used to run the AES engine. Figure 6.7 shows the functionality of starting AES0 over JTAG.

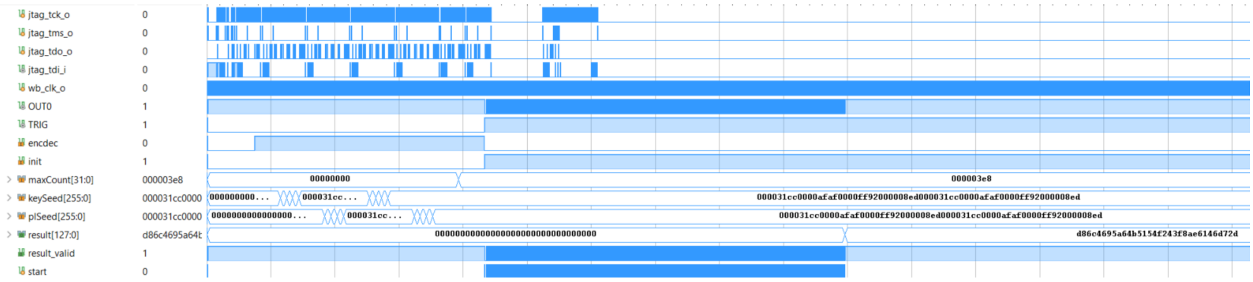

jtag_tck_o 0
jtag_tms_o 0
jtag_tdo_o 0
jtag_tdi_i 0
wb_clk_o 0
OUT0 1
TRIG 1
encdec 0
init 1
maxCount[31:0] 000003e8
keySeed[255:0] 000031cc0000
plSeed[255:0] 000031cc0000
result[127:0] d86c4695a64t
result_valid 1
start 0
00000000
000003e8
000000000...
000031cc...
000031cc0000afaf0000ff92000008ed000031cc0000afaf0000ff92000008ed
0000000000000000...
000031cc...
000031cc0000afaf0000ff92000008ed000031cc0000afaf0000ff92000008ed
00000000000000000000000000000000
d86c4695a64b5154f243f8ae6146d72d

Figure 6.7: Starting AES engine over JTAG

In the figure, the following can be seen. The data is parsed over the test data out(TDO) line, and the number of iterations as well as the key and plaintext seed are inputted. Then the engine is started and encryption runs until the done pin is pulled up. The JTAG works completely independent of the RISC-V core.

6.3. SoC Power side channel behaviour experimental platform

The goal of the platform is to evaluate the side channel performance of different AES engines. To deem the platform functional, the power side channel analysis must be reliably performed. To do this in simulation, the following setup can be used.

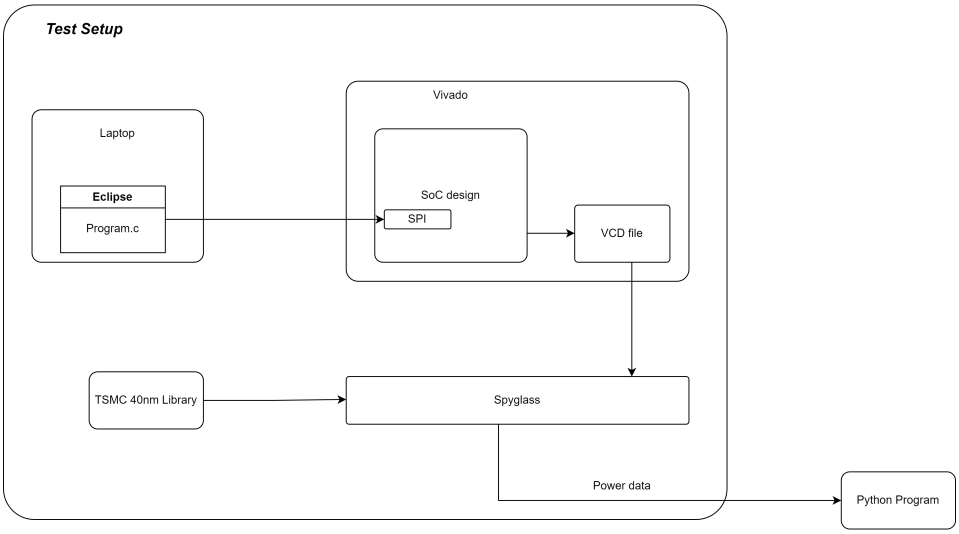


Figure 6.8: Test setup for power side channel analysis

Figure 6.8 shows the experimental test setup used for the validation. In the image, certain pieces of software can be identified next to the ones mentioned in the previous section.

- **QuestaSim** QuestaSim is a hardware description language simulator [68]. It can be used for the simulation of RTL designs
- **VCD file** A Value change dump(vcd) file is a file that consists of all the switching activity in a design, so if a register updates its value, the vcd file will show the time at which this happened and what happened.
- **Spyglass** Spyglass is software which allows designers with insights about their RTL design in simulation [78], in this case, with the technology library and the vcd file, the power behaviour of the RTL design can be observed.
- **Python** Python is a programming language which is used, in this case, for the visualization of the power reports from Spyglass.

6.4. SoC Power side channel behaviour

With the previously mentioned experimental platform, the power behaviour of the SoC platform can be observed. First, the power behaviour of the different AES implementations needs to be observed. From that, the entire platform's power behaviour can be observed during encryption. From this, the platform noise can be identified and a mitigation strategy can be implemented. Lastly to completely verify the SoC, a power side-channel attack can be performed. For all the experiments, the set of implemented AES engines on the SoC is used. In the below sections, AES2 and AES4 are used for comparison as these are the least and most secure implementations. All other engines's results are in the appendix.

Table 6.2: Set of implemented AES engines

AES number	Design	Full design description
AES0	AES	Balanced SBox AES
AES1	***DOM***	DOM 4s design
AES2	AES	AES baseline design
AES3	AES	AES 16s design
AES4	***DOM***	DOM 8s design
AES5	***DOM***	DOM 16s full pipe design

6.4.1. AES baseline power side channel behaviour

Figure 6.9 shows the power side channel behaviour of an unprotected AES engine 2 implementation on the left and the power side channel behaviour of a protected AES engine 4 implementation on the right.

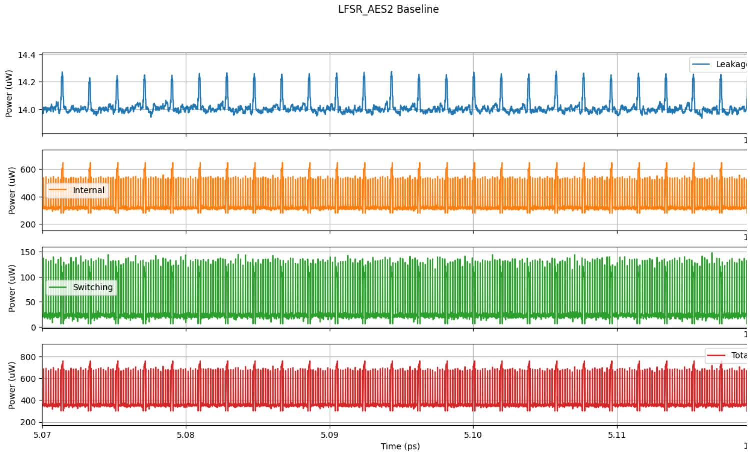


Figure 6.9: Unprotected AES implementation

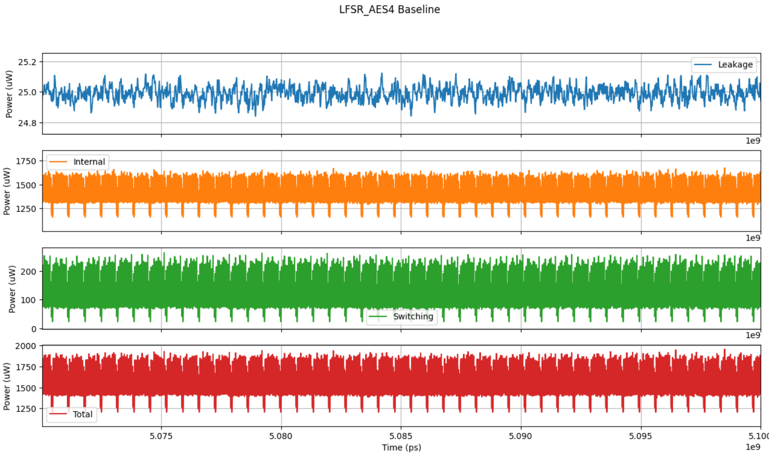


Figure 6.10: Protected AES implementation

Analysis

The figures display 4 different powers, leakage, internal, switching and total. The leakage power is the static power that is leaked when the device is not in operation. Meaning the static dissipation power. The internal power is the power dissipated by the internal charge and discharging of the transistors. The switching power is the power dissipated by the driving of output loads outside of the transistor cells.

From the above figures, the unprotected one shows a repetitive peak in the leakage and total power. The unprotected, however, doesn't show this repetitive behaviour. From this, the security of the AES engine can be correlated to the observation of the repetitiveness of the power behaviour. If the repetitiveness is observed, the SNR ratio is higher than if it cannot be observed.

To analyse this further the following technique was used. First, it is known that 1000 traces are used. The first and last peak in the signal can be denoted as P_0 and P_{1000} with time stamps T_0

and T_{1000}. The time between these peaks is the total time encryption time. Dividing that total time by 1000 gives the average time for each encryption. Then comparing the average time for each encryption with the average time between the peaks will give us how repetitive each encryption is.

Engine	Total Time/1000 (us)	Peak to peak time (us)
AES0	189.15	191.061
AES1	87.824	127.281
AES2	190.939	190.939
AES3	14.007	23.580
AES4	68.047	69.734
AES5	57.972	64.271

Table 6.3: Average calculated time per encryption and actual average time per encryption

Table 6.3 shows the repetitiveness in the signals. Before diving deeper into the results it is important to understand that the repetitiveness of the signal does not mean it is not secure, it only indicates that an attack could retrieve the key if enough traces are provided.

6.4.2. AES platform power side channel behaviour

After looking only at the power behaviour of the LFSR and AES, the next step is to observe the main power line of the entire SoC. This includes the improved LFSR_AES. To get the simulation, the power simulation will only be done on the triggered part, so only the part where encryption is ran. This also simultaniously verifies the platforms function.

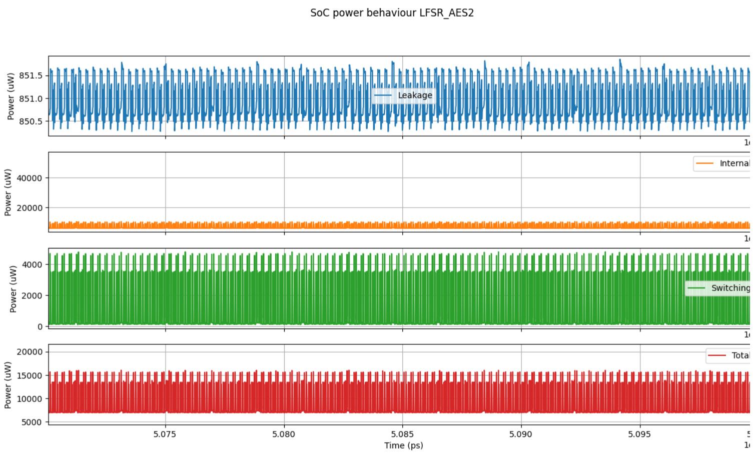


Figure 6.11: Unprotected AES implementation

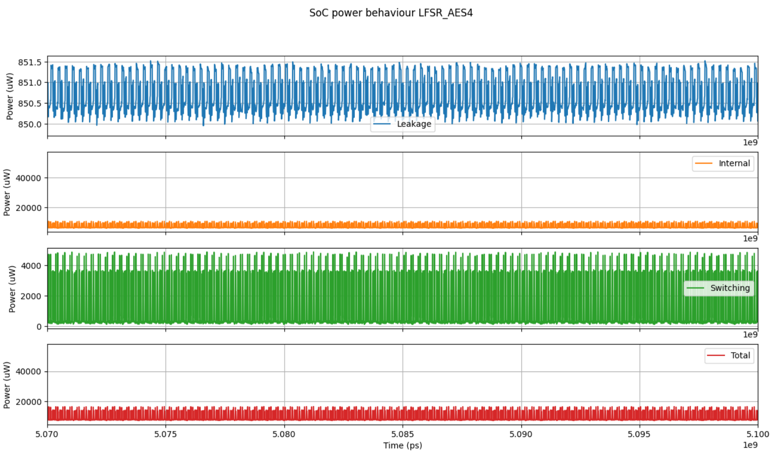


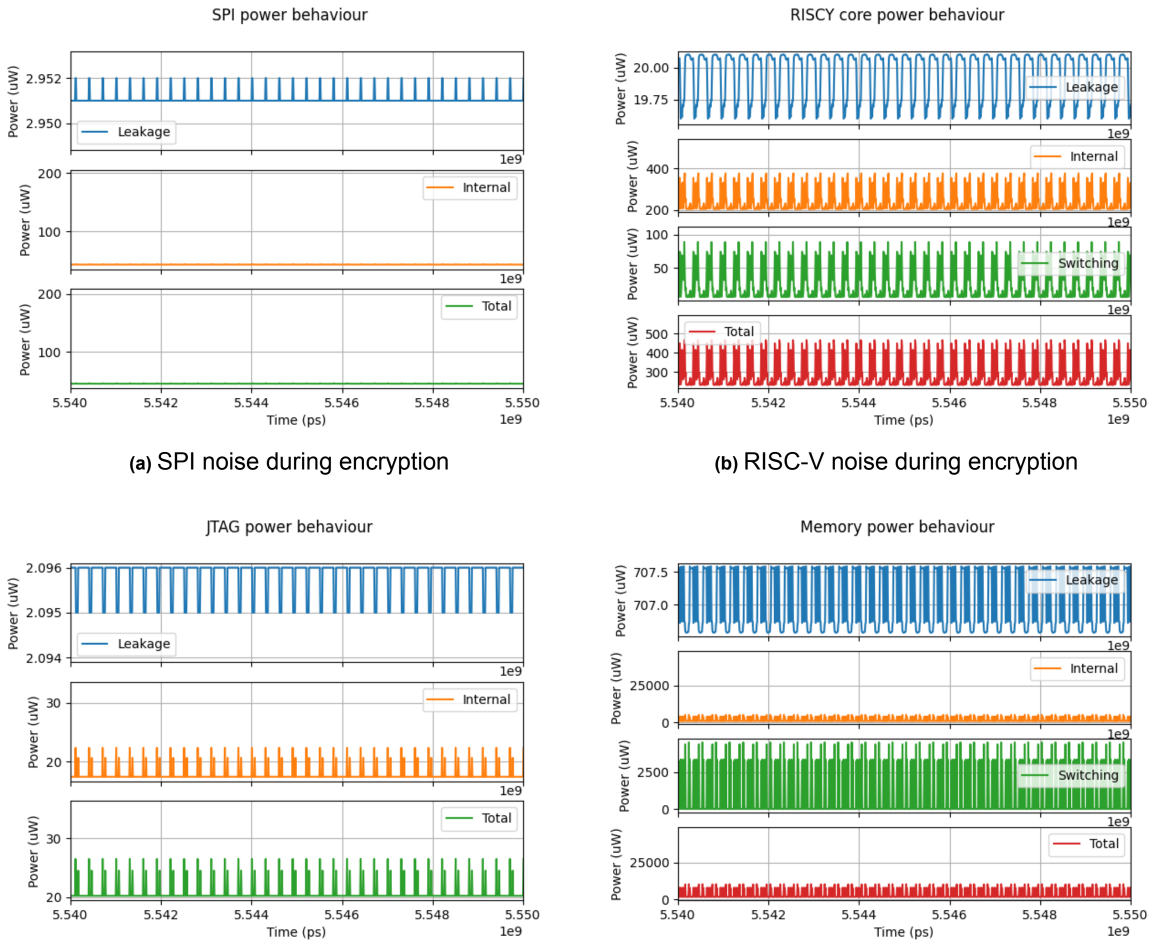
Figure 6.12: Protected AES implementation

Analysis

Observing the above images and comparing them with the images from the baseline, shows that the system is adding a lot of noise. From the baseline, it is visible that the baseline AES2 has peaks around 14uW and AES4 around 25uW. With all the system noise added there is now around 851uW of power. Meaning that it will be significantly harder to visualise the effect of encryption with all this added noise. The first step in mitigating this is to analyze the noise of all the extra parts of the system.

6.4.3. Noise analysis of platform components

As previously mentioned, the platform seems to generate quite some noise. Luckily with this platform, the noise can be identified, quantified and later profiled. When looking at the SoC architecture, next to the AES engines, there are a few peripherals that are still on during encryption. This includes the memory and interconnect bus, the RISC-V core and the SPI, JTAG and UART.



(a) SPI noise during encryption

(b) RISC-V noise during encryption

(c) JTAG noise during encryption

(d) Memory noise during encryption

Figure 6.13: The power behaviour of SoC components during encryption of AES 2

Analysis

Observing the power behaviour of other SoC components except for the AES engine it becomes visible what is causing all the noise. The internal memory of the SoC is generating about 710uW of leakage power. Which was to be expected. The design facilitates 32kB of 40nm SRAM next to a small 1kB of ROM. Observing the average power dissipation of 40nm 6T SRAM cells to be in the range 4pW-16nW per bit [54], the total power can be determined as below. Doing this calculation gives indeed a power within the specified range.

$$P_{Leakage} = 710uW = PSRAM + P_{RomLeak} = 33000 * 6 * 3.5nW$$

6.4.4. Clock-gating countermeasure implementation

As previously mentioned, the platform seems to generate a lot of noise and switching power, especially the memory. To mitigate this, a few techniques can be employed to lower the dynamic power consumption of the platform. One way of doing this is by using clock gating [74]. Power consumption can be determined by the following formula.

$$P_{dynamic} = C * V^2 * F_{clock}$$

Reducing the clock frequency is key to reducing dynamic power. Implementing clock gating is rather easy and straightforward. The clock of the Core, Memory SPI and UART can be operated with the Trigger signal. This means that when the trigger goes high, the peripherals will be clock-gated. Then when the interrupt of the AES engine goes high, the clock gating can be stopped. The core can check the status of the engine and the result can be brought back safely. In logic this will be:

$$Clock_{peri} = Clock_{system} \wedge (Trigger \vee AES_{irq})$$

Doing this on the AES engines gives the following result. Figure 6.15 shows the protected clock-gated implementation and figure 6.14 shows the unprotected clockgated implementation.

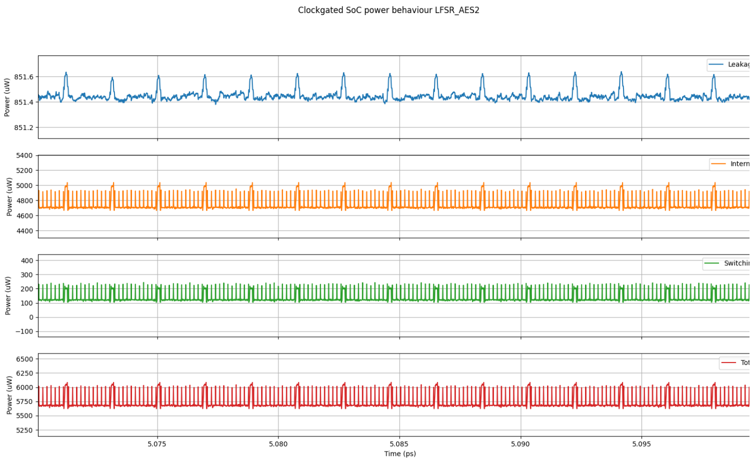


Figure 6.14: Unprotected clock gated AES implementation

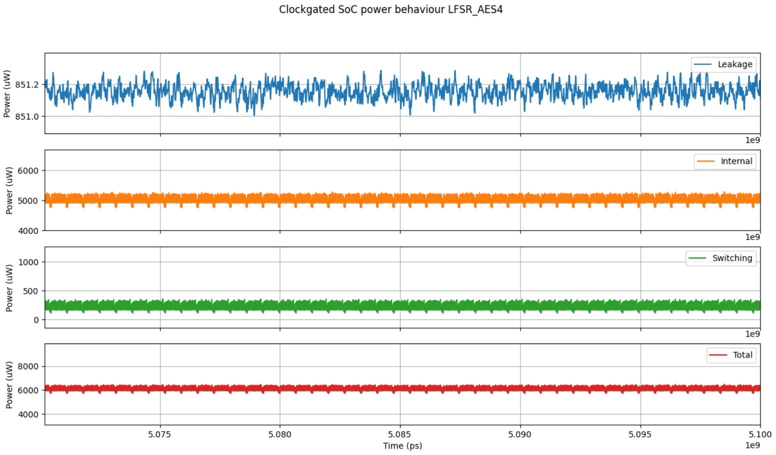


Figure 6.15: Protected clock gated AES implementation

Analysis

Comparing the initial result to the clock-gated result and subtracting the static power, it becomes visible that the original signal has been reconstructed but now with a higher static power. Which indicates that platform's static power. Figure 6.16 and 6.17 shows the comparison between the orginal signal and the clockgated version when subtracting the system's static power from the clockgated version. It is possible to reconstruct the original signal.

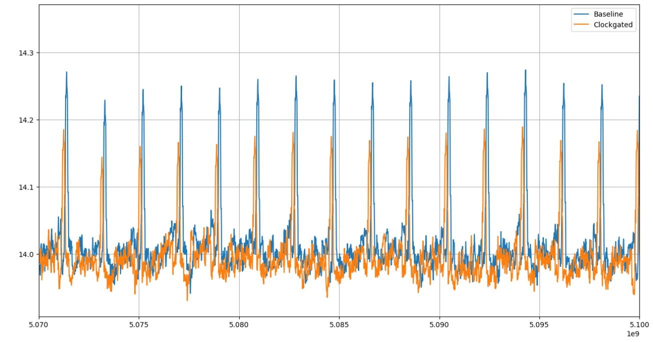


Figure 6.16: Comparison clock gated and baseline protected AES implementation

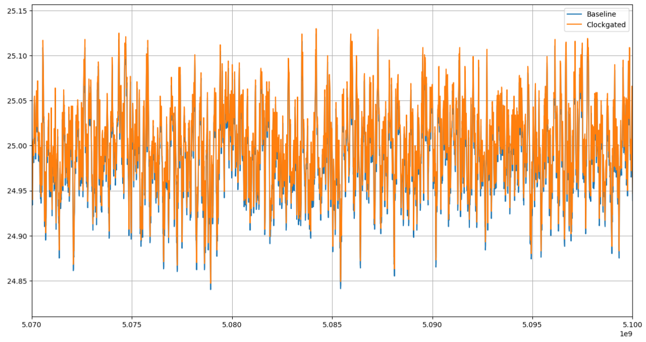


Figure 6.17: Comparison clock gated and baseline protected AES implementation

6.5. Attackability evaluation

After analysing power side channel behaviour and system noise, the last thing that can be done is observing the attackability. As the platform's ultimate goal is to evaluate power side channel behaviour and compare between different implementations, it is important that the power side channel behaviour can indeed be observed and evaluated. One way of doing this is by doing a power side-channel attack or assessment. In this section, the focus is on AES engine 2, which is the baseline unprotected engine, and AES engine 4 which is the most secure. For each of the previous scenarios, the baseline, the triggered version and the clock-gated version, an attack can be made. All implementations did undergo the attack, these results can be viewed in Appendix D.

6.5.1. Attack model

For this power side channel assessment, the attack chosen was Correlation Power Analysis(Section 2.3.5) with 1000 traces. CPA is chosen as it does not require as many traces to get a good analysis in contrary to DPA. This method is based on the correlation of predicted power and the actual obtained power trace. The attack consists of four main steps:

Gathering of power traces The power traces are collected in a set of 1000 and can be represented the following where d equals 1000 and N equals the amount of samples per trace. (for example AES 2 has 191 samples per trace and AES 4 has 68)

$$T = \begin{bmatrix} t_{00} & t_{01} & t_{02} & t_{0N} \\ t_{10} & t_{11} & t_{12} & t_{1N} \\ .. & .. & .. & .. \\ t_{d0} & t_{d1} & t_{d2} & t_{dN} \end{bmatrix}$$

Generation of potential keys For the Sbox operation of AES, keys are inputted per byte leading to the potential keys for each byte being in the range of 0-255 or 2^8. From this a potential key vector S can be made where m is 256 in this case.

$$S = \begin{bmatrix} k_0 & k_1 & k_2 & k_3 & .. & k_{m-1} \end{bmatrix}$$

Generation of leakage power model For the attack as the attack location is chosen as the Sbox values, the potential keys *S* can converted into a power representation of the sbox operation on the inputted plaintext byte *P[i]*.

$$H = sbox[P[i] \wedge S_i] \wedge P[i] \wedge S_i$$

This will give us the power profile of the sbox operation with every potential key candidate. Again, m is 256 in this case.

$$H = \begin{bmatrix} h_0 & h_1 & h_2 & h_3 & .. & h_{m-1} \end{bmatrix}$$

Correlation between leakage power and actual power With all the possible leakage powers they now can be compared and correlated to the actual power consumption using Pearson's correlation equations which practically looks like this [14]

$$G_{i,j} = \frac{(h_{d,i} - \overline{h_i})(t_{d,j} - \overline{t_j})}{\sqrt{(h_{d,i} - \overline{h_i})^2 (t_{d,j} - \overline{t_j})^2}}$$

Where t is our actual power traces and h is our calculated trace power.

6.5.2. Baseline version

The triggered version is the power consumption of the entire SoC platform with all system noise. The engines are run for 1000 iterations to gather all the traces. Then the CPA attack is done based on those 1000 traces. Figure 6.18 and 6.19 gives the following partial guessing entrophy per key byte. Partial guessing entropy(PGE) quantifies the unpredictability of the different bytes of the key. It is measured in bytes and represents the average number of guesses an attacker needs to make to correctly guess that part of the secret key. Higher partial guessing entropy indicates higher unpredictability and stronger security for that specific part of the system.

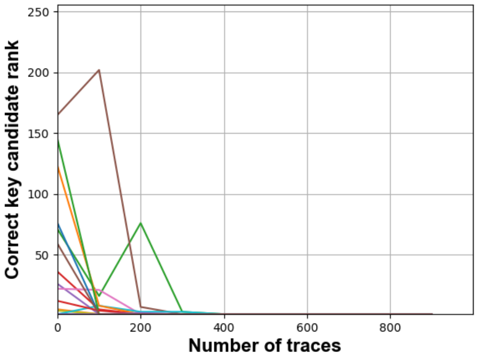


Figure 6.18: Correct key guessing on unprotected AES engine 2 baselines

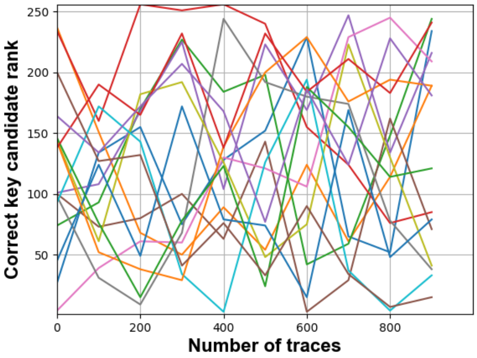


Figure 6.19: Correct key guessing on protected AES engine 4 baseline

In the above figure and appendix D it becomes visible that the PGE of the unprotected engines go down really fast. This indicates that the key can be retreived easily. The protected engines, the PGE does not go down but rather scrambles, indicating that the engine is protected and the key can not be retreived with just 1000 traces.

6.5.3. Triggered version

The baseline version is the power consumption of only the LFSR+AES engines. The engines are run for 4000 iterations to gather all the traces, this number is different due to all the noise from the various platform components. Then the CPA attack is done based on those 4000 traces.

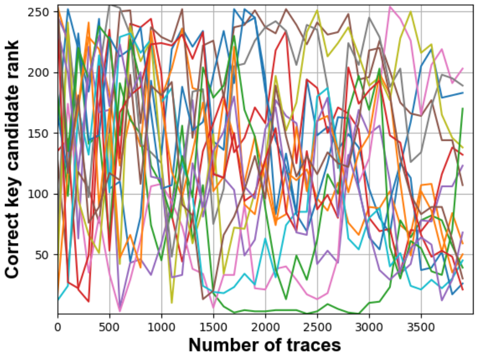


Figure 6.20: Correct key guessing on unprotected AES engine 2 with system noise

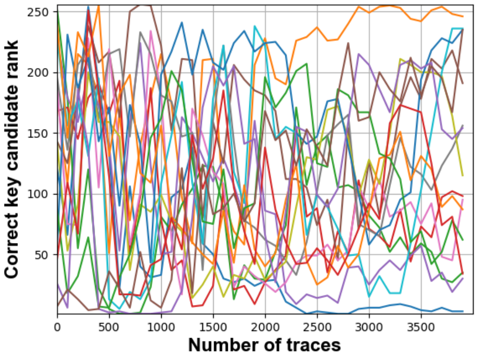


Figure 6.21: Correct key guessing on protected AES engine 4 with system noise

In the above image shows that the system noise hides the power fluctuations of the AES engines quite severely. With unprotected engine 2, the key is now not retrievable except for one byte. Also AES engine 4, with all the noise, has one byte for which the PGE goes to 0. Meaning that even with all the noise, the engine is still not completely secure. As 4000 traces was not enough to get the key in both engines, figure 6.22 and 6.23 gives the trend in the PGE as an average over all 16 key bytes.

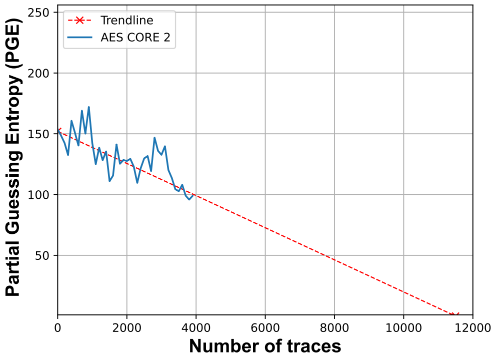


Figure 6.22: Correct key guessing on unprotected AES engine 2 with system noise

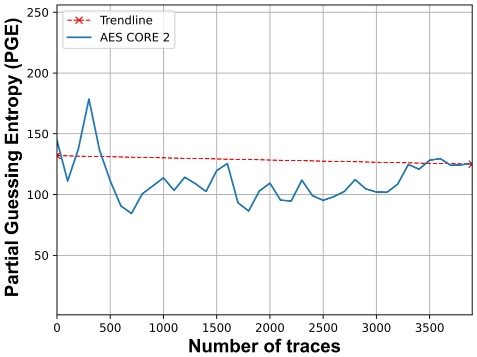


Figure 6.23: Correct key guessing on protected AES engine 4 with system noise

Based on the trend observed, it can be said that AES engine 2 has a downwards regression in its PGE, meaning that with enough traces it could be attackable. In AES engine 4, the trend is only marginally downwards meaning that with all the system noise it will be difficult to attack it.

6.5.4. Clockgated version

For the clock-gated version, some system noise is physically turned off. Figures 6.24 and 6.25 shows the effect on the partial guessing entropy of the engines when the dynamic system noise is turned off.

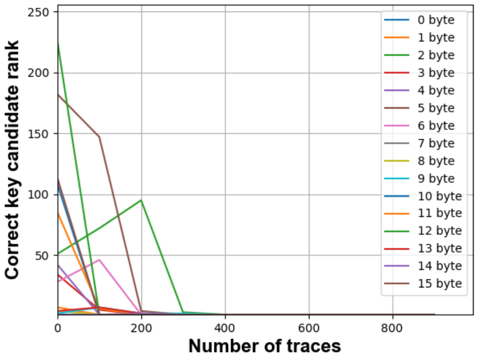


Figure 6.24: Correct key guessing on AES engine 2 with clock gated system

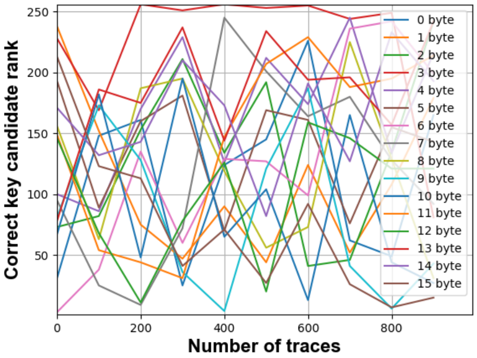


Figure 6.25: Correct key guessing on AES engine 4 with clock gated system

When clockgating the noisy platform components, the unprotected AES engine becomes easily attackable again. The protected AES engines still is not attackable.

6.5.5. Analysis

The previous sections provide 3 different scenarios. The first is the baseline which provides an unrealistic scenario but clear behaviour and good analysis. The second one is the triggered version with all the system noise which provides a realistic scenario but it is hard to do attackability analysis on this with the limited amount of traces. The last scenario is the clockgated scenario in which some system peripherals are clockgated, this provides a compromise between a realistic scenario but with good attackability analyis.

Based on the above results the following short conclusion can be made. The unprotected engines, AES 2 and AES 0 are fairly easily attackable with the platform. In the baseline and the clockgated version, the number of traces needed to crack the engine is really low meaning that it is indeed an unprotected AES implementation. With the protected engine, the number of input traces was not sufficient to crack the engine.

Lastly, looking at appendix D a last observation can be made. In the protected engines, some bytes seem to be attackable. This can be due to various reasons, an example of this is the key that is inputted. Some bytes of the key might be easy to retreive due to their specific power profile of the implementation. In this platform, we dont really care about the specific implementation of the AES, we just try to show the power security.

6.5.6. Final classification of AES engines

To compare the different AES engines, the traces can be averaged for each engine. Averaging refers to the process of taking all the PGE's of every key byte and averaging them to indicate the attackabilty of the key as a whole. Figure 6.26 shows the attack on all AES engines when averaging the traces, this is done in the clockgated scenario meaning results will be quite realistic and with sufficient analysis.

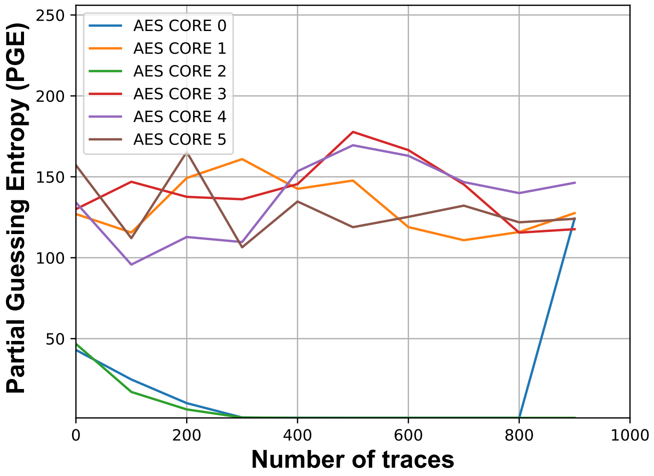


Figure 6.26: Averaging of traces

Based on the results from the attackability of all the AES engines in appendix D and the result shown in figure 6.26 the following classification can be made in table 6.4. The conclusion can then be as follows: the unprotected engines can be cracked with the platform, the protected engines can not be cracked in the current test format. More research is needed to fully characterize and check this.

Engine	Attackable with 1000 traces on the Platform
AES0	Yes
AES1	No
AES2	Yes
AES3	No
AES4	No
AES5	No

Table 6.4: Attackability of implemented AES engines

6.6. Discussion

Although preliminary results shows promise in characterizing and evaluating AES engines with the platform. With all the system noise on it was not possible to crack all the engines with just a 1000 traces or even 4000 traces. This means the platform has some limitations.

6.6.1. Limitations

The SoC platform has some limitation in what it can do, the following aspects might be cause influence the performance of the platform.

Larger data set needed A larger amount of power traces are needed to perform an attack on the AES engine. In simulation running a lot of iterations take weeks to complete. Therefore it was not possible to do it in this thesis. In real life, taking the measurements will be a lot faster and provide accurate results.

Wrong power profile of technology library Simulation is not real life and simulating memory noise is often not perse usefull. In this case the system noise perceived in simulation could be completely different than the system noise that will be seen in the actual platform. Only when the platform is tested concrete conclusions can be made about this.

Interfering software The software that is currenly run during encryption inserts nops every cycle that the engine is not done. This could lead to the RISC-V fetchin from memory polling the interconnect bus every cycle to check the peripheral register. A better way of doing this would be to only do something when the AES interrupt goes up, or only periodically check the status of the engine. There was, however, not enough time to implement and check this properly.

Test data set The test data set like the key and inputted plaintext seed might influence the performance. Due to the nature of the SBOX operation and the AES implementation, some key bytes might be easier to retreive than others.

7 Conclusion

This chapter provides a quick overview of the thesis and the achievements after which future work is discussed. Section 7.1 presents a summary of each chapter. Section 7.2 provides recommendations and topics which can be explored after this thesis.

7.1. Summary

The goal of this thesis was to design and create a platform that can accurately compare and correlate the power behaviour of AES engines in the pre and post-silicon stages. Although both sides had existing platforms there was a gap between the two stages. In this thesis, a proposal and design of a system architecture that bridges this gap is made. Although a lot more testing is necessary the first results showed that the platform is capable of running software, running AES engines and capturing power data from them. Next that, the platform shows evalution of power side channel security can be done.

Chapter one presents the motivation for this thesis, the need for accurate and realistic side channel assessment as well as the gap in current side channel assessment platforms. This chapter discusses the relevance of side channel assessment and explains the current state of the art.

Chapter two presents the necessary background information about security and power side channel assessment. It explains the beginnings of security, the classification of cryptography algorithms and AES. With regards to Power side channel assessment, it discusses the different ways of doing power side channel analysis like CPA and DPA, a brief overview of countermeasures and the quantification of a good measurement setup.

Chapter three presents the necessary background information concerning system-on-chip design. It explains everything about chip architecture. This includes the CPU, different memory architectures, intra-chip communication busses and peripheral interfaces like hardware accelerators and communication interfaces.

Chapter four sets the motivation and methodology for designing the proposed system. It explains the end test cases that need to be facilitated, and details all the requirements for the system. Lastly, the proposed system architecture is presented and elaborated on.

Chapter five provides a detailed methodology of the implementation of the proposed architecture. It elaborates all the implementation steps and decisions about the design of the SoC, the software and the PCBs.

Chapter Six elaborates the results of the SoC design, both on functionality and on a power side channel level. Only the SoC design is mentioned as the PCB and the SoC are not available for testing. On the functional level, all the different peripherals are shown to function with simple software and uploading that to the SoC. On the power side channel side, it is shown that the system influences the power side channel measurements but this is part of providing realistic measurements as well as techniques that can be established to isolate the AES engine. Lastly a correlation power attack was which showed that the unprotected engines the key could be retrieved easily and with the protected engines, this was harder. Even with all the system noise getting the key did show to be possible if a larger dataset was provided.

7.2. Future Work

This thesis is only the tip of the iceberg when it comes to designing and validating a realistic pre and post-silicon power side channel platform. There is still a lot that can be investigated.

SoC/FPGA Design

(a) **Full testing on the SoC/FPGA Design on the platform** Currently tests are only done in simulation, it is going to be interesting to see how the actual design performs and works in real life. With this, the correlations between simulation and real life can then also be made.

(b) **Different AES designs.** Currently, there is a finite set of AES designs implemented. But many more countermeasures and novel implementations are available.

(c) **Implementation of clock gating in new custom SoC** Previous sections have already shown that clock gating is effective in mitigating platform noise but to fully see the effect of that, a new SoC can be made that includes clock gating.

Software Design

(a) **Extensive code and software exploration** For the proposed architecture and functionality testing, only simple bare metal code was written to facilitate small experiments. As the system also consists of a timer, future software programs could include a tiny operating system which can turn the core to sleep and imitate the clock gating.

(b) **Better debug capabilities** Another thing which can be explored is the more elaborate debug options like building a solid jtag interface in C which can run the debug.

(c) **Extensive analysis of the platform** In this thesis only very limited actual analysis is done but it would be interesting to see how attackable the system is. Recent developments have shown that deep learning can be quite effective for power side channel analysis [49].

PCB Design

(a) **The first step in the PCB future work is the full testing of the PCB** Fully testing the power behaviour of just the PCB will give the characterization of the effect of the PCB on the power measurements. As the PCB includes a lot of measures that should remove unwanted noise it is interesting to see how they perform.

(b) **Doing the power side channel analysis on the ASIC and the FPGA with the PCBs** To finally see the correlation and end goal of the platform, analyzing both the ASIC and FPGA gives the full performance of the platform.

(c) **Quatification of PCB added noise** The last step that can be done is the quantification of system noise of the PCB to the power side channel measurements.

Bibliography

[1] Mar. 2019. URL: `https://www.sifive.com/blog/the-revolution-evolution-continues---sifive-risc-v`.

[2] URL: `https://docs.openhwgroup.org/projects/cv32e40p-user-manual/en/latest/intro.html`.

[3] URL: `https://www.javatpoint.com/classification-of-memory`.

[4] URL: `https://developer.arm.com/documentation/ihi0033/latest/`.

[5] Admin and Admin. *Internet of Things (IoT) Security Market Size to Surpass $9.88 Billion by 2025 – Component Segment Analysis, Vendor Landscape and Revenue Drivers Overview: Million Insights*. 2021. URL: `https://www.abnewswire.com/pressreleases/internet-of-things-iot-security-market-size-to-surpass-988-billion-by-2025-component-segment-analysis-vendor-landscape-and-revenue-drivers-overview-million-insights_541702.html`.

[6] Maghrib Alrammahi and Harleen Kaur. *Development of Advanced Encryption Standard (AES) Cryptography Algorithm for Wi-Fi Security Protocol*. Apr. 2014. DOI: `10.13140/RG.2.2.20993.97124`.

[7] Jude Angelo Ambrose, Roshan G. Ragel, and Sri Parameswaran. "RIJID: Random Code Injection to Mask Power Analysis Based Side Channel Attacks". In: *Proceedings of the 44th Annual Design Automation Conference*. DAC '07. San Diego, California: Association for Computing Machinery, 2007, pp. 489–492. ISBN: 9781595936271. DOI: `10.1145/1278480.1278606`. URL: `https://doi.org/10.1145/1278480.1278606`.

[8] Witscad Compter Architecture. *Concepts of Pipelining*. Accessed: 10 10, 2023. 2023. URL: `https://witscad.com/course/computer-architecture/chapter/concepts-of-pipelining`.

[9] Arduino. *Arduino*. Version 2022. Feb. 28, 2014. URL: `https://www.arduino.cc/en/software`.

[10] ARM. *AXI Protocol Overview*. Accessed: 12 10, 2023. 2018. URL: `https://developer.arm.com/documentation/102202/0300/AXI-protocol-overview`.

[11] Swarup Bhunia and Mark M Tehranipoor. *Hardware security: a hands-on learning approach*. Morgan Kaufmann, 2018.

[12] Emily Blem, Jaikrishnan Menon, and Karthikeyan Sankaralingam. "Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures". In: *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. 2013, pp. 1–12. DOI: `10.1109/HPCA.2013.6522302`.

[13] Hamidreza Bolhasani, Somayyeh Jafarali Jassbi, and Arash Sharifi. "DLA-E: a deep learning accelerator for endoscopic images classification". In: *Journal of Big Data* 10 (May 2023). DOI: `10.1186/s40537-023-00775-8`.

[14] Eric Brier, Christophe Clavier, and Francis Olivier. "Correlation power analysis with a leakage model". In: *Cryptographic Hardware and Embedded Systems-CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings 6*. Springer. 2004, pp. 16–29.

[15] Abhishek Chakraborty, Bodhisatwa Mazumdar, and Debdeep Mukhopadhyay. "Fibonacci lfsr vs. galois lfsr: Which is more vulnerable to power attacks?" In: *Security, Privacy, and Applied Cryptography Engineering: 4th International Conference, SPACE 2014, Pune, India, October 18-22, 2014. Proceedings 4*. Springer. 2014, pp. 14–27.

[16] Baris Coskun and Nasir Memon. “Confusion/diffusion capabilities of some robust hash functions”. In: *2006 40th Annual Conference on Information Sciences and Systems*. IEEE. 2006, pp. 1188–1193.

[17] Joan Daemen and Vincent Rijmen. “AES proposal: Rijndael”. In: (1999).

[18] Debayan Das et al. “EM/Power Side-Channel Attack: White-Box Modeling and Signature Attenuation Countermeasures”. In: *IEEE Design and Test* PP (Mar. 2021), pp. 1–1. DOI: 10.1109/MDAT.2021.3065189.

[19] Whitfield Diffie and Martin E Hellman. “Exhaustive cryptanalysis of the NBS data encryption standard”. In: *Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman*. 2022, pp. 391–414.

[20] *Diligent Pmode interface*. https://www.digikey.nl/nl/supplier-centers/digilent. Accessed: October 9, 2023.

[21] *Diligent Pmode interfaces*. https://digilent.com/reference/pmod/start. Accessed: October 9, 2023.

[22] *Diligent programmer*. https://digilent.com/shop/jtag-hs2-programming-cable/. Accessed: October 9, 2023.

[23] Santanu Dutta, Wayne Wolf, and Andrew Wolfe. “A methodology to evaluate memory architecture design tradeoffs for video signal processors”. In: *Circuits and Systems for Video Technology, IEEE Transactions on* 8 (Mar. 1998), pp. 36–53. DOI: 10.1109/76.660828.

[24] Eclipse Org. *Eclipse C/C++*. Version 2023-03. Feb. 28, 2014. URL: https://www.eclipse.org/downloads/packages/release/kepler/sr2/eclipse-ide-cc-developers.

[25] Nate Enos and Brian Gosselin. “A primer on USB Type-C and Power Delivery applications and requirements”. In: *Texas Instruments* (2016).

[26] F Foust. “Secure digital card interface for the MSP430”. In: *Michigan State University* (2004).

[27] Richard Fromm et al. “The Energy Efficiency of IRAM Architectures”. In: *ACM SIGARCH Computer Architecture News* 25 (Sept. 1997). DOI: 10.1109/ISCA.1997.604742.

[28] *FTDI Chip*. http://www.ftdichip.com. Accessed: October 9, 2023.

[29] MIKE GRUSIN. *Serial Peripheral Interface (SPI)*. Accessed: 10 10, 2023. 2020. URL: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all.

[30] Ujjwal Guin et al. “Detecting Recycled SoCs by Exploiting Aging Induced Biases in Memory Cells”. In: May 2019. DOI: 10.1109/HST.2019.8741032.

[31] Tim Güneysu and Amir Moradi. “Generic side-channel countermeasures for reconfigurable devices”. In: *International workshop on cryptographic hardware and embedded systems*. Springer. 2011, pp. 33–48.

[32] Hendra Guntur, Jun Ishii, and Akashi Satoh. “Side-channel AttacK User Reference Architecture board SAKURA-G”. In: *2014 IEEE 3rd Global Conference on Consumer Electronics (GCCE)*. 2014, pp. 271–274. DOI: 10.1109/GCCE.2014.7031104.

[33] Gurubaran Guru. *Moveit Hack: Over 400 organizations’ hacked by CL0P Ransomware Group*. 2023. URL: https://cybersecuritynews.com/moveit-hack-mass-hack/.

[34] Jay L Halio. “All’s Well That Ends Well”. In: *Shakespeare Quarterly* 15.1 (1964), pp. 33–43.

[35] Ruoyu Hang. “Securing an Efficient Lightweight AES Accelerator”. Accessed: October 9, 2023. MA thesis. Delft University of Technology, 2023. URL: https://repository.tudelft.nl/islandora/object/uuid:b009e5b3-03a8-4b2f-b89b-d8471fbdc30c?collection=education.

[36] David T Harvey. “Chemometrics Using R”. In: (2021).

[37] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. “Applications of machine learning techniques in side-channel attacks: a survey”. In: *Journal of Cryptographic Engineering* 10 (2020), pp. 135–162.

[38] Kouichi Itoh, Masahiko Takenaka, and Naoya Torii. “DPA countermeasure based on the “masking method””. In: *Information Security and Cryptology—ICISC 2001: 4th International Conference Seoul, Korea, December 6–7, 2001 Proceedings 4*. Springer. 2002, pp. 440–456.

[39] Mohsen Jahanshahi and Fathollah Bistouni. “Crossbar-based interconnection networks”. In: *Series: Computer Communications and Networks* 12 (2018), pp. 164–173.

[40] Brandon John. “Algorithm-Agnostic System for Measuring Susceptibility of Cryptographic Accelerators to Power Side Channel Attacks”. PhD thesis. Massachusetts Institute of Technology, 2022.

[41] JTAG Technologies. *JTAG Technologies - Official Website*. `https://www.jtag.com`. Accessed: October 9, 2023.

[42] Toshihiro Katashita et al. “Development of side-channel attack standard evaluation environment”. In: *2009 European Conference on Circuit Theory and Design*. IEEE. 2009, pp. 403–408.

[43] Pritamkumar N. Khose and Vrushali G. Raut. “HARDWARE IMPLEMENTATION OF AES ENCRYPTION AND DECRYPTION FOR LOW AREA AND POWER CONSUMPTION”. In: *International Journal of Research in Engineering and Technology* 03 (2014), pp. 480–484.

[44] Pantea Kiaei et al. “Saidoyoki: Evaluating side-channel leakage in pre-and post-silicon setting”. In: *Cryptology ePrint Archive* (2021).

[45] Seong Keun Kim and Mihaela Popovici. “Future of dynamic random-access memory as main memory”. In: *MRS Bulletin* 43.5 (2018), pp. 334–339.

[46] Erik van Klinken. *”Nebu-Datalek trof 16 overheidsorganisaties’*. May 2023. URL: `https://www.techzine.nl/nieuws/privacy-compliance/524046/nebu-datalek-trof-16-overheidsorganisaties/`.

[47] Paul Kocher, Joshua Jaffe, and Benjamin Jun. “Differential power analysis”. In: *Advances in Cryptology—CRYPTO’99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*. Springer. 1999, pp. 388–397.

[48] Takaya Kubota et al. “Deep Learning Side-Channel Attack Against Hardware Implementations of AES”. In: *2019 22nd Euromicro Conference on Digital System Design (DSD)*. 2019, pp. 261–268. DOI: `10.1109/DSD.2019.00046`.

[49] Takaya Kubota et al. “Deep learning side-channel attack against hardware implementations of AES”. In: *Microprocessors and Microsystems* 87 (2021), p. 103383.

[50] Sunil Kumar and Ilyoung Chong. “Correlation Analysis to Identify the Effective Data in Machine Learning: Prediction of Depressive Disorder and Emotion States”. In: *International Journal of Environmental Research and Public Health* 15.12 (2018). ISSN: 1660-4601. DOI: `10.3390/ijerph15122907`. URL: `https://www.mdpi.com/1660-4601/15/12/2907`.

[51] Alvin Cai Kunming et al. *Comparison of side channel analysis measurement setups*. Tech. rep. Technical report, Tech. Rep., 2015.[Online]. Available: http://alexandria …, 2015.

[52] Thanh Ha Le et al. “Efficient Solution for Misalignment of Signal in Side Channel Analysis”. In: vol. 2. May 2007, pp. II–257. ISBN: 1-4244-0728-1. DOI: `10.1109/ICASSP.2007.366221`.

[53] Margaret Martonosi, David Brooks, and Pradip Bose. “Modeling and analyzing CPU power and performance: Metrics, methods, and abstractions”. In: *SIGMETRICS 2001/Performance 2001-Tutorials* (2001).

[54] Jitendra Kumar Mishra et al. “A 40nm low power high stable SRAM cell using separate read port and sleep transistor methodology”. In: *2018 IEEE International Symposium on Smart Electronic Systems (iSES)(Formerly iNiS)*. IEEE. 2018, pp. 1–5.

[55] *Multiphase Oscillator*. https://www.analog.com/media/en/technical-documentation/data-sheets/6902f.pdf. Accessed: October 15, 2023.

[56] National Institute of Standards and Technology (NIST). *Cybersecurity: A Critical Component for Industry 4.0 Implementation*. Accessed on: October 10, 2023.

[57] Colin O'Flynn and Zhizhang Chen. "ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research". In: vol. 8622. Apr. 2014. ISBN: 978-3-319-10174-3. DOI: `10.1007/978-3-319-10175-0_17`.

[58] Colin O'flynn and Zhizhang David Chen. "Chipwhisperer: An open-source platform for hardware embedded security research". In: *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer. 2014, pp. 243–260.

[59] *OpenOCD*. https://openocd.org/pages/getting-openocd.html. Accessed: October 8, 2023. 2019.

[60] Christof Paar and Jan Pelzl. *Understanding cryptography: a textbook for students and practitioners*. Springer Science and Business Media, 2009.

[61] Francesca Palumbo, Danilo Pani, and Luigi Raffo. "Computer Science Research and the Internet". In: Jan. 2010, pp. 301–340. ISBN: 9781617610448.

[62] Dwiti Pandya et al. "Brief history of encryption". In: *International Journal of Computer Applications* 131.9 (2015), pp. 28–31.

[63] Jangyong Park et al. "A Survey on Air-Gap Attacks: Fundamentals, Transport Means, Attack Scenarios and Challenges". In: *Sensors* 23 (Mar. 2023), p. 3215. DOI: `10.3390/s23063215`.

[64] David A. Patterson and John L. Hennessy. *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*. 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2017. ISBN: 0128122757.

[65] Pulp Platform. *Pulpino linker script*. Accessed on: 06/10/23. 2021.

[66] PULP platform. *Pulp Platform*. URL: `https://pulp-platform.org/`.

[67] xPack project. *RISC-V None Embedding GCC Toolchain*. Accessed on: 06/10/23. 2021.

[68] *QuestaSim*. https://eda.sw.siemens.com/en-US/ic/questa/simulation/advanced-simulator/. Accessed: October 6, 2023. 2021.

[69] Mark Randolph and William Diehl. "Power side-channel attack analysis: A review of 20 years of study for the layman". In: *Cryptography* 4.2 (2020), p. 15.

[70] Riscure. *Riscure Current Probe*. Accessed: October 11, 2023. 2021. URL: `https://getquote.riscure.com/en/quote/2101059/current-probe.htm`.

[71] Andrew Rogers. "Introduction to USB Type-C™". In: *Microchip Inc., Chapters* 3.4 (2015).

[72] Bagus Santoso and Yasutada Oohama. "Information theoretic security for Shannon cipher system under side-channel attacks". In: *Entropy* 21.5 (2019), p. 469.

[73] David Seal. *ARM architecture reference manual*. Pearson Education, 2001.

[74] Jitesh Shinde and S. S. Salankar. "Clock gating — A power optimizing technique for VLSI circuits". In: *2011 Annual IEEE India Conference*. 2011, pp. 1–4. DOI: `10.1109/INDCON.2011.6139440`.

[75] Arvind Singh et al. "Improved Power/EM Side-Channel Attack Resistance of 128-Bit AES Engines With Random Fast Voltage Dithering". In: *IEEE Journal of Solid-State Circuits* PP (Nov. 2018), pp. 1–15. DOI: `10.1109/JSSC.2018.2875112`.

[76] William Stallings. *Computer Organization and Architecture*. 6th. Prentice Hall Professional Technical Reference, 2002. ISBN: 0130351199.

[77] Brad Suppanz. *PCB Trace Width Calculator*. Accessed: October 11, 2023. 2018. URL: `https://www.4pcb.com/trace-width-calculator.html`.

[78] *Synopsis Spyglas*. https://www.synopsys.com/verification/static-and-formal-verification/spyglass.html. Accessed: October 6, 2023. 2017.

[79] TechTarget. *Cryptography*. Accessed on: October 10, 2023.

[80] Pranav Tendulkar. "Mapping and Scheduling on Multi-core Processors using SMT Solvers". In: (Oct. 2014).

[81] University of Washington. *Compilers, Assemblers, Linkers, Loaders: A Short Course.* Accessed on: October 6, 2023. 1998. URL: https://courses.cs.washington.edu/courses/cse378/98wi/help/compilation.html.

[82] Andrew Waterman et al. "The RISC-V instruction set manual". In: *Volume I: User-Level ISA', version* 2 (2014).

[83] Wikipedia. *Advanced Encryption Standard — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Advanced%20Encryption%20Standard&oldid=1173129769. [Online; accessed 06-September-2023]. 2023.

[84] *Xilinx Kintex-7 FPGA Data Sheet*. https://docs.xilinx.com/v/u/en-US/ds182_Kintex_7_Data_Sheet. Accessed: October 9, 2023.

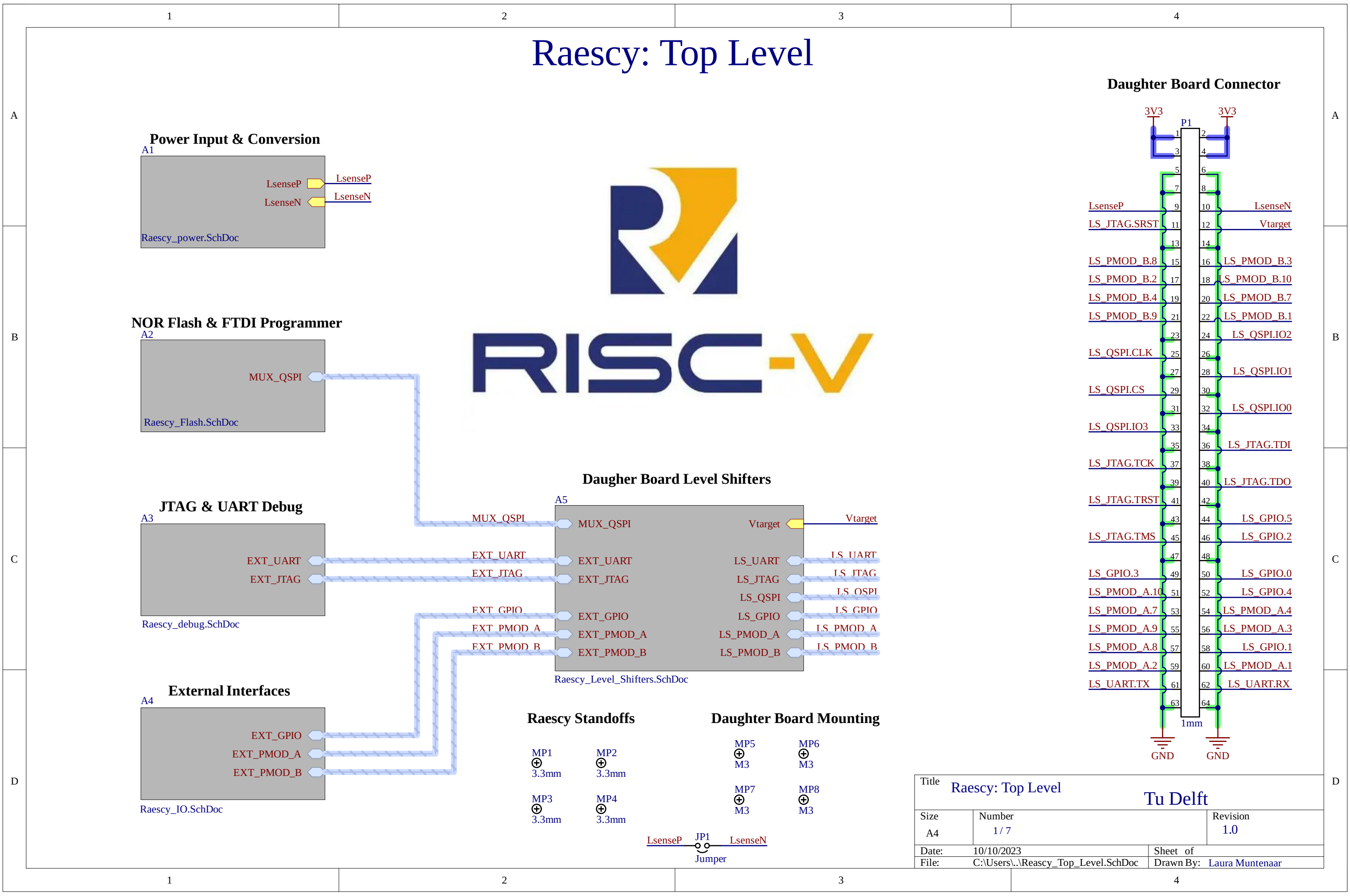
[85] Xilinx Vivado. *Vivado ML*. Version 2023. Feb. 28, 2014. URL: https://www.xilinx.com/support/download.html.

[86] Shuo Yang, Shubhra Deb Paul, and Swarup Bhunia. "Hands-On Learning of Hardware and Systems Security." In: *Advances in Engineering Education* 9.2 (2021), n2.

A

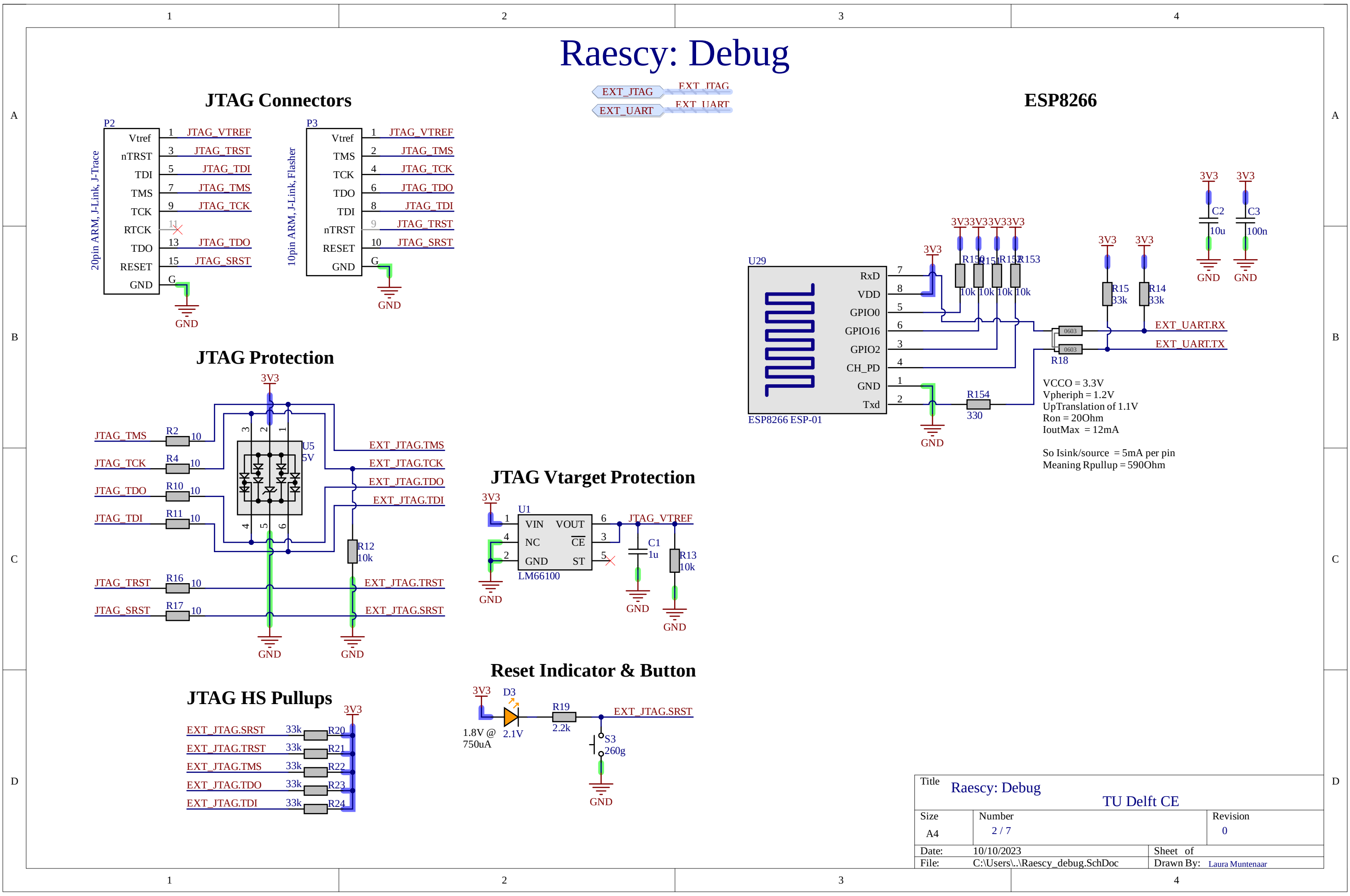
Raescy Carrier and FPGA pcb

Raescy: Top Level



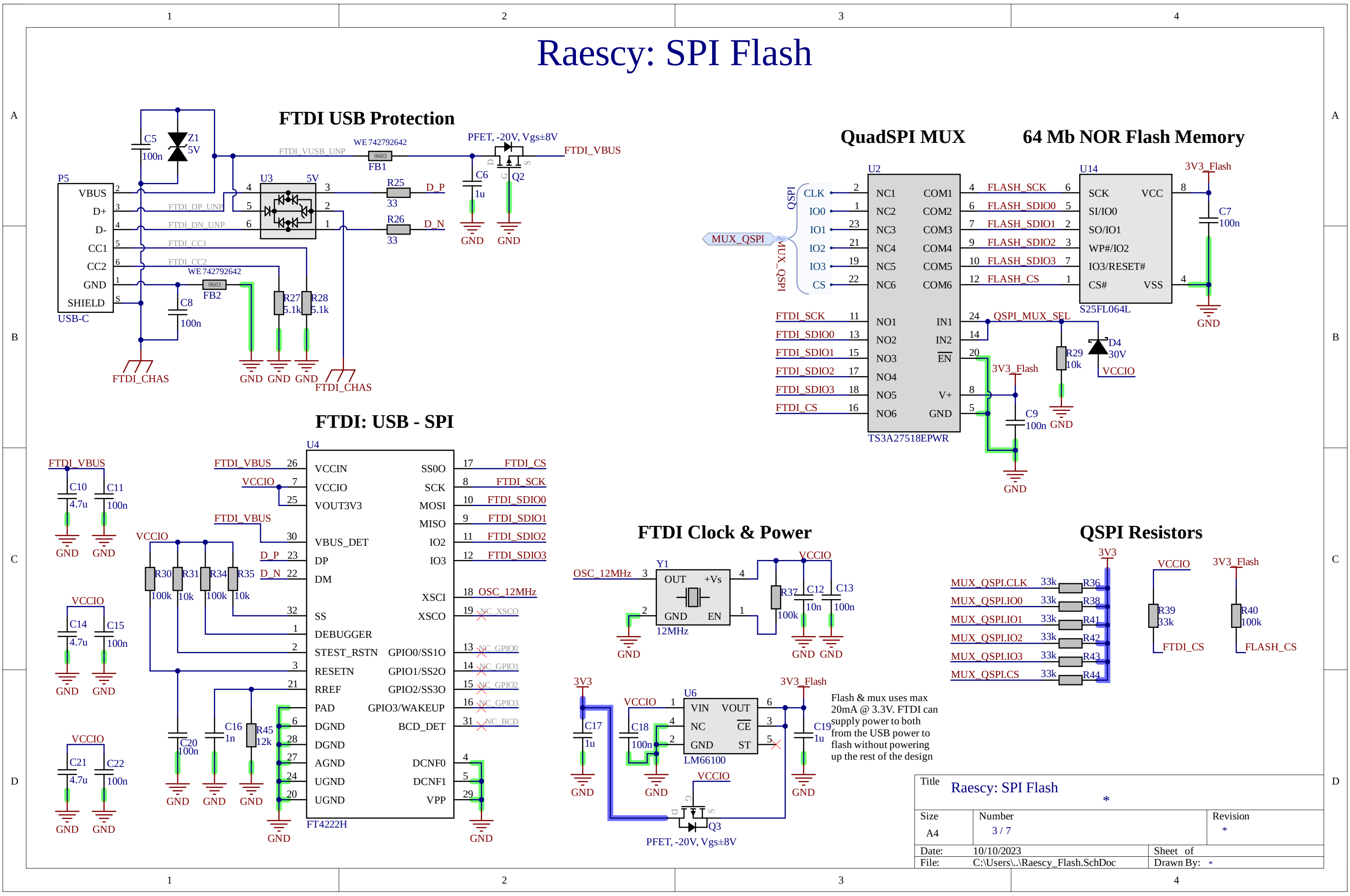
Title	Raescy: Top Level		Tu Delft
Size A4	Number 1 / 7		Revision 1.0
Date:	10/10/2023	Sheet of	
File:	C:\Users\..\Raescy_Top_Level.SchDoc	Drawn By:	Laura Muntenaar

Raescy: Debug



Title	Raescy: Debug		TU Delft CE
Size A4	Number 2 / 7		Revision 0
Date:	10/10/2023	Sheet of	
File:	C:\Users\..\Raescy_debug.SchDoc	Drawn By:	Laura Muntenaar

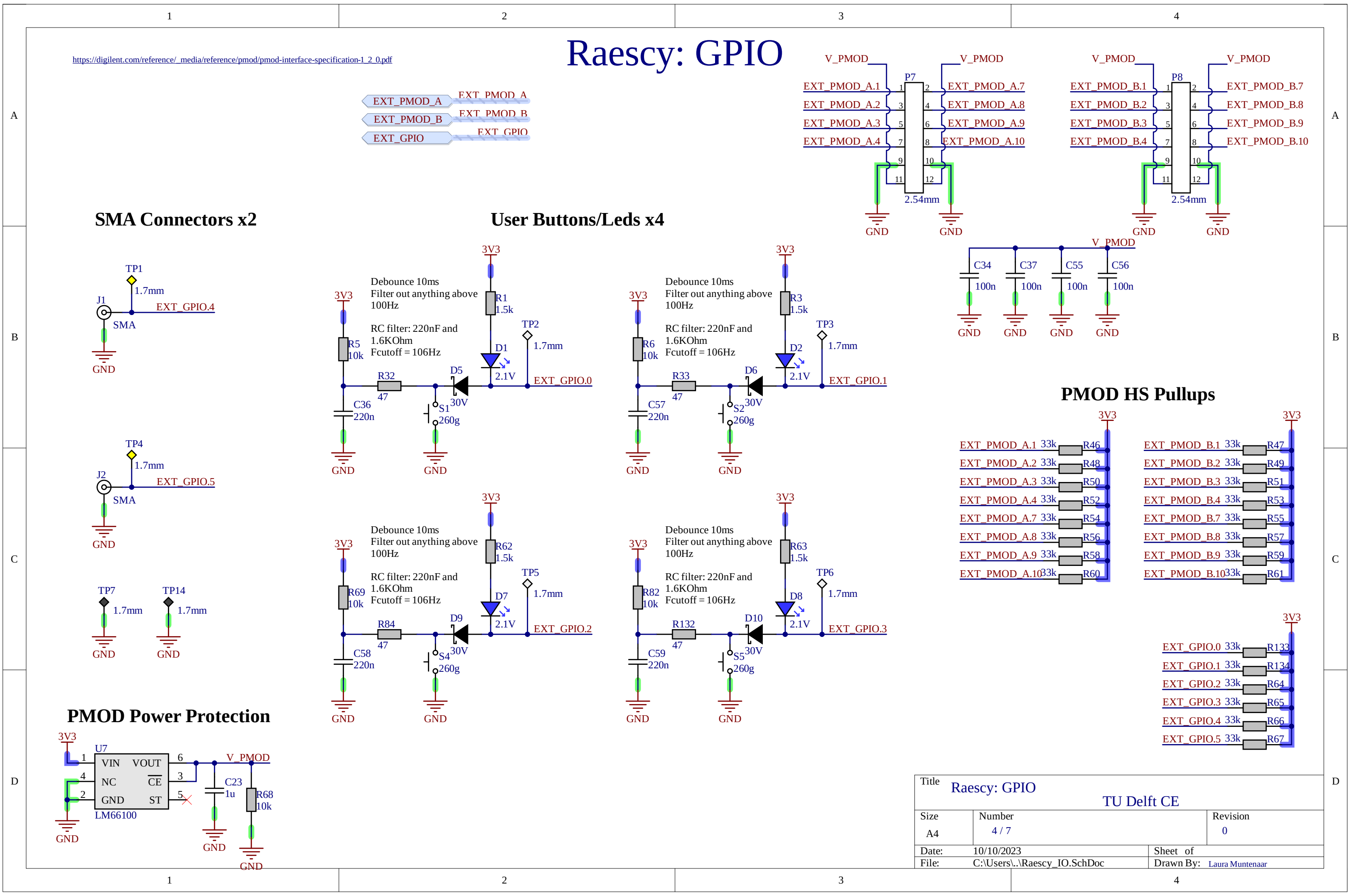
Raescy: SPI Flash



Flash & mux uses max 20mA @ 3.3V. FTDI can supply power to both from the USB power to flash without powering up the rest of the design

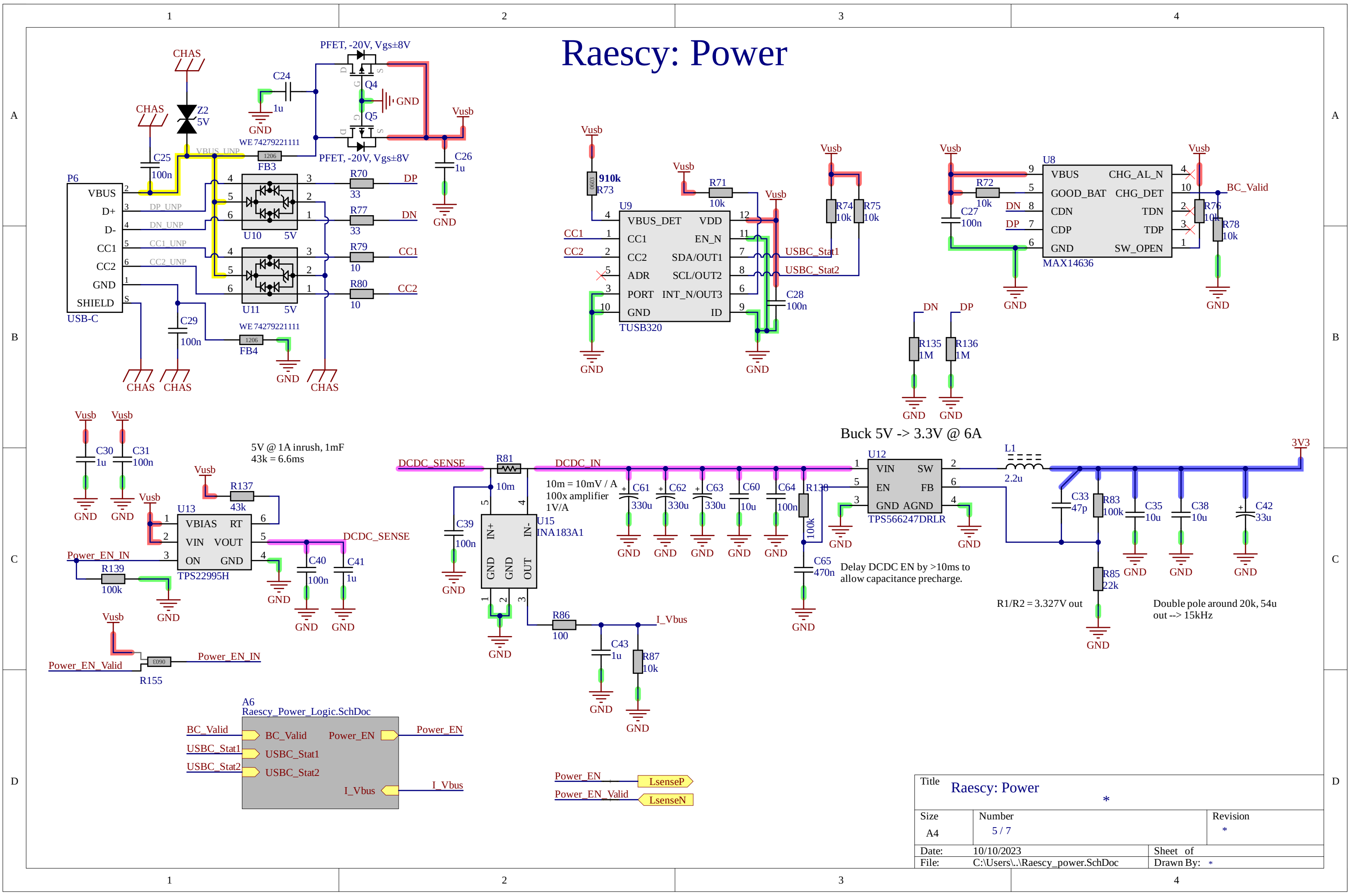
Title	Raescy: SPI Flash	*
Size A4	Number 3 / 7	Revision *
Date: 10/10/2023		Sheet of
File: C:\Users\..\Raescy_Flash.SchDoc		Drawn By: *

Raescy: GPIO



Title	Raescy: GPIO	TU Delft CE
Size A4	Number 4 / 7	Revision 0
Date: 10/10/2023		Sheet of
File: C:\Users\..\Raescy_IO.SchDoc		Drawn By: Laura Muntenaar

Raescy: Power



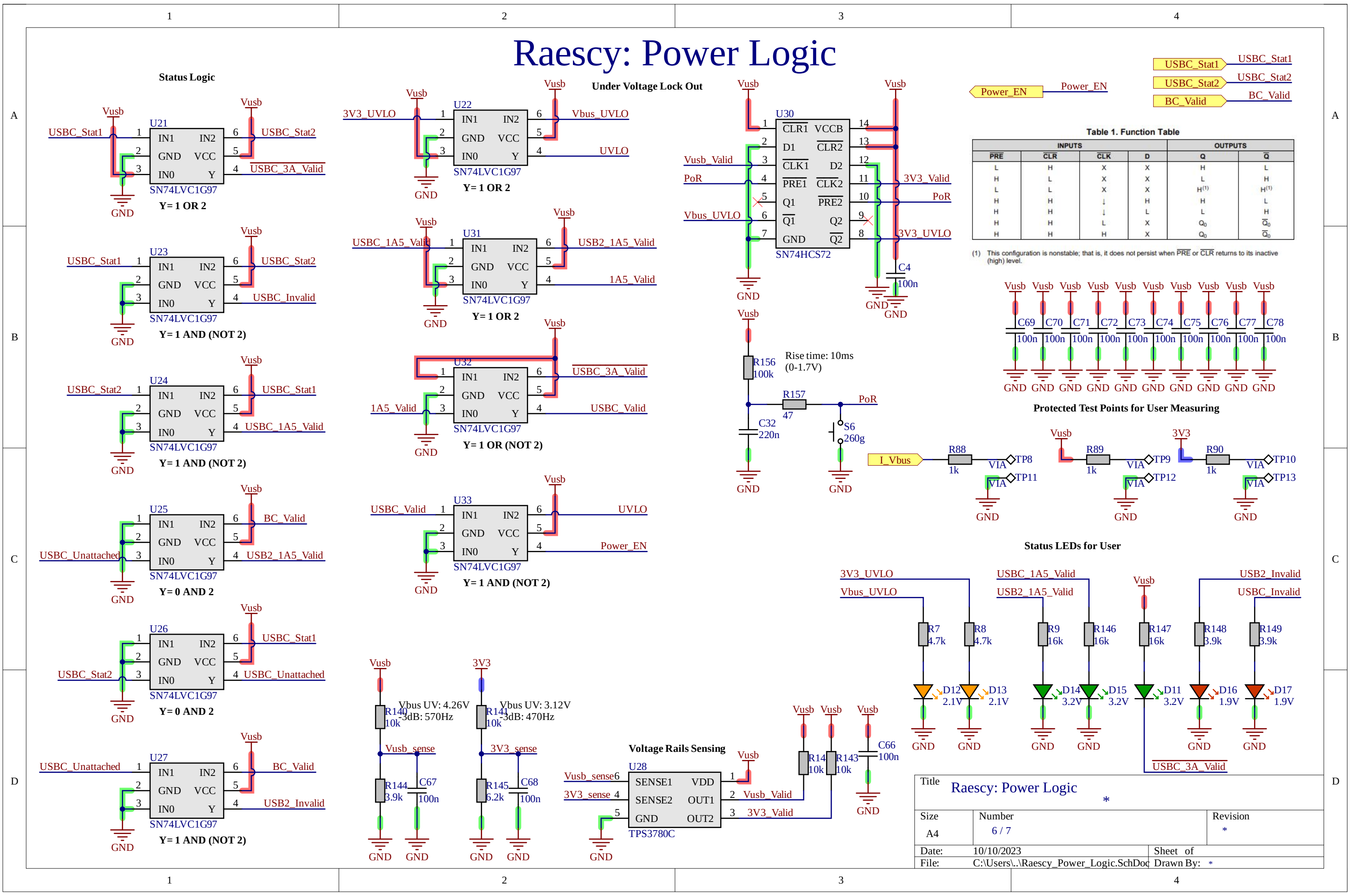
Title Raescy: Power *

Size	Number	Revision
A4	5 / 7	*

Date: 10/10/2023 Sheet of

File: C:\Users\..\Raescy_power.SchDoc Drawn By: *

Raescy: Power Logic

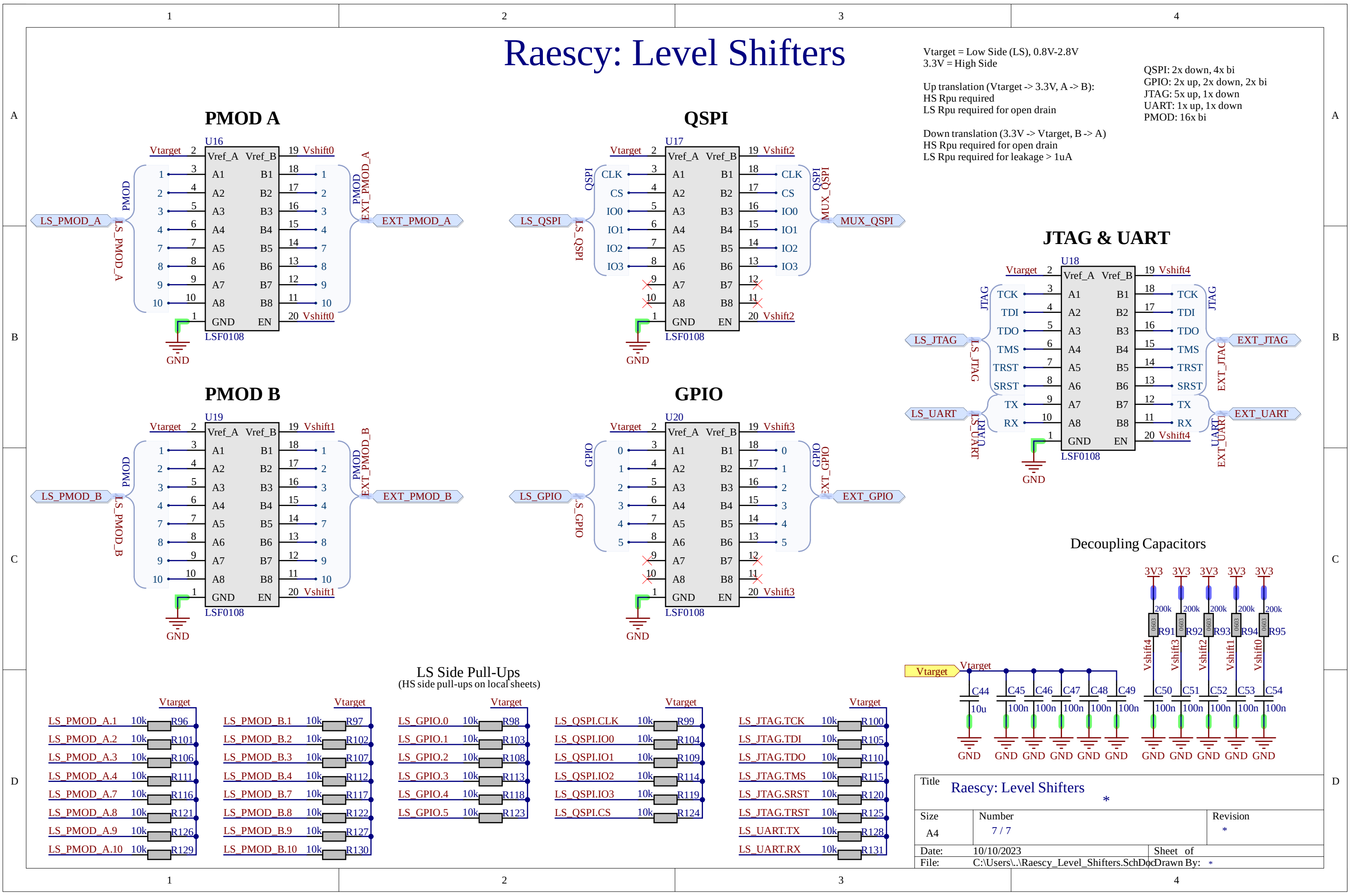


Title Raescy: Power Logic *

Size	Number	Revision
A4	6 / 7	*

Date: 10/10/2023 Sheet of

File: C:\Users\..\Raescy_Power_Logic.SchDoc Drawn By: *



Raescy: Level Shifters

Vtarget = Low Side (LS), 0.8V-2.8V
3.3V = High Side

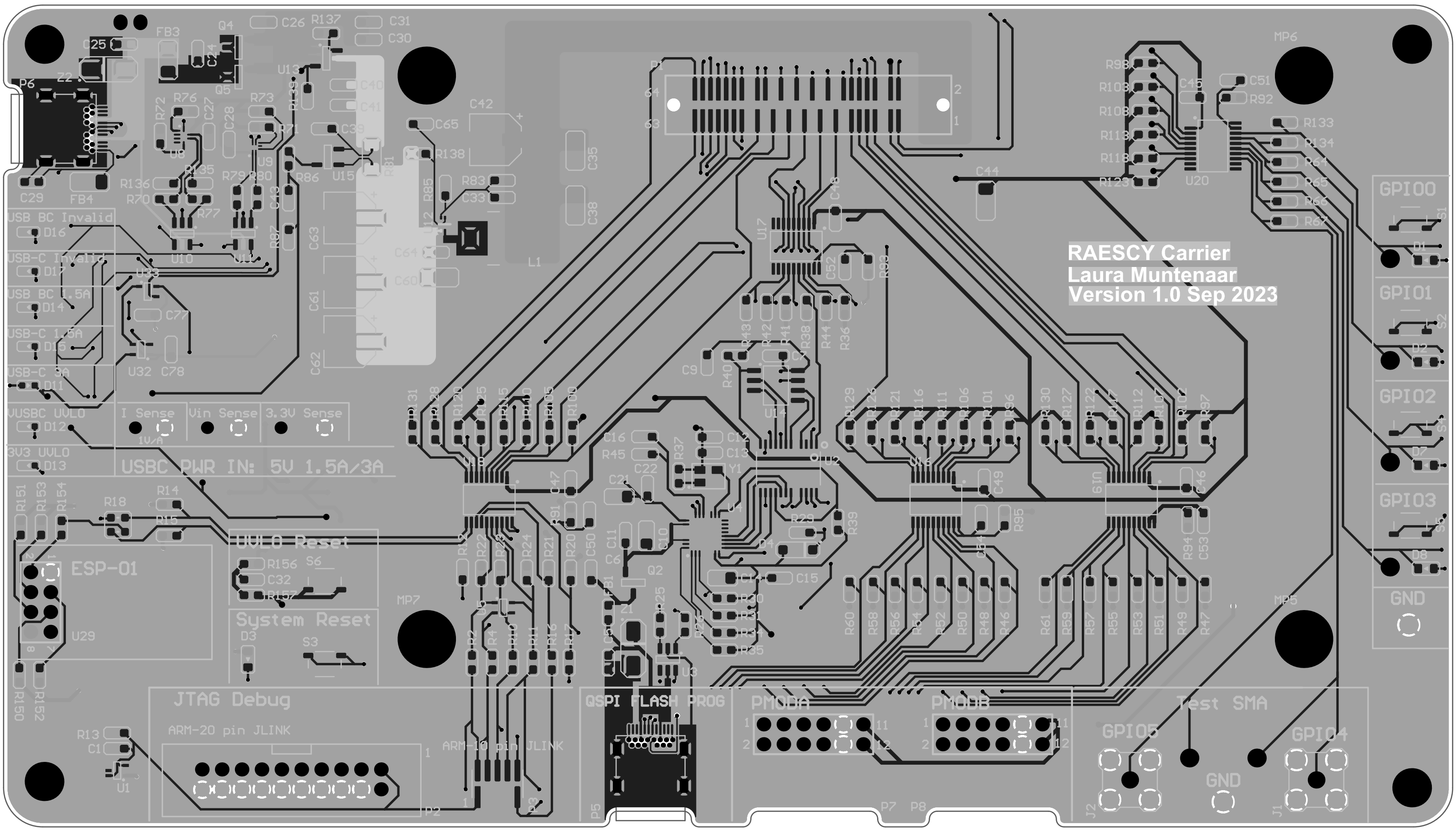
Up translation (Vtarget -> 3.3V, A -> B):
HS Rpu required
LS Rpu required for open drain

Down translation (3.3V -> Vtarget, B -> A)
HS Rpu required for open drain
LS Rpu required for leakage > 1uA

QSPI: 2x down, 4x bi
GPIO: 2x up, 2x down, 2x bi
JTAG: 5x up, 1x down
UART: 1x up, 1x down
PMOD: 16x bi

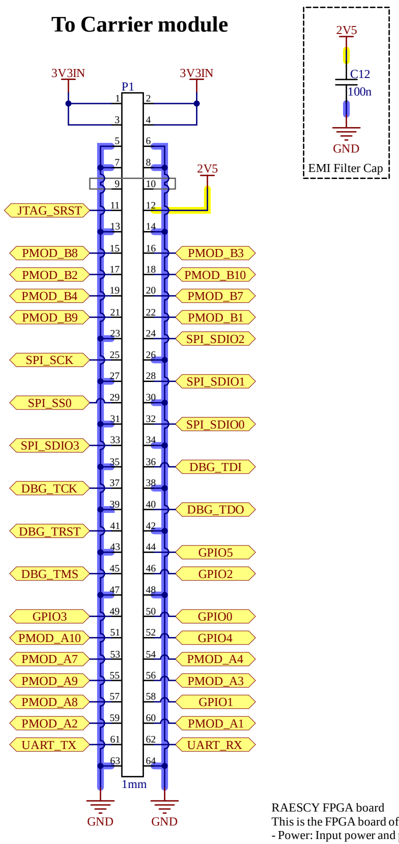
Title	Raescy: Level Shifters	
Size	Number	Revision
A4	7 / 7	*
Date:	10/10/2023	Sheet of
File:	C:\Users\..\Raescy_Level_Shifters.SchDoc	Drawn By: *

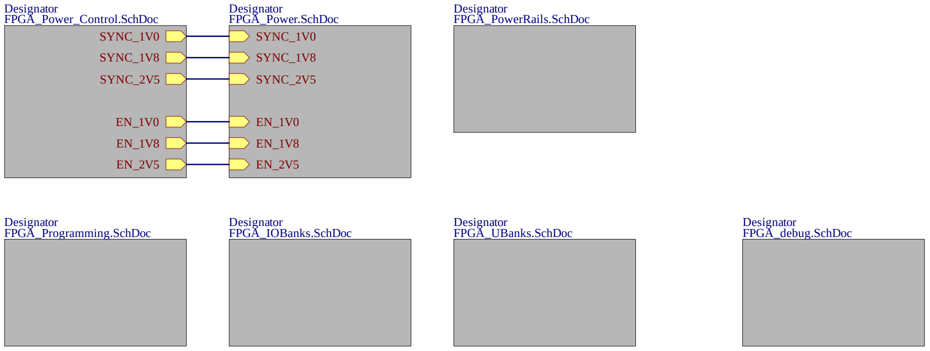
Board Outline



Raescy: FPGA Top

To Carrier module





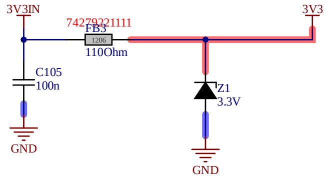
To Do
- Change all votlage rail names
- Add top level ports to Power contol & Power
- Add DNP caps
- Add 330u, 100u, and 4.7u caps
- Reannotated and validate all signals
- Add netlabels to single pin nets

RAESCY FPGA board
This is the FPGA board of the RAESCY framework. This pcb consists of the following sheets:
- Power: Input power and power distribution
- PowerRails: FPGA power rails and decoupling
- Programming: FPGA bank 0 for configuring the FPGA
- debug: Extra pheripherals for the FPGA
- Measurements: the Power side channel measurement connectors

Title	Raescy: FPGA Top TU Delft	
Size A4	Number 1 / 8	Revision *
Date: 10/10/2023		Sheet of
File: C:\Users\..\FPGA_Top_Level.SchDoc		Drawn By: Laura Muntenaar

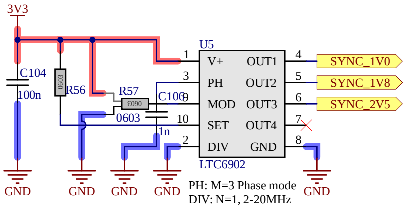
Raescy FPGA: Power Control

Input Protection



Refine comment:
Input UVLO required externally when exceeding current limit (4A) or when voltage drops below 2.9V.

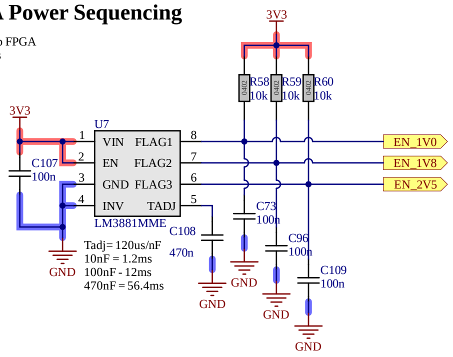
Power phase sync



PH: M=3 Phase mode
DIV: N=1, 2-20MHz
SET: 2.25MHz = 29.6k
MOD: GND=SSFM off

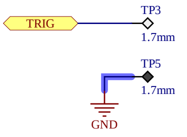
FPGA Power Sequencing

Max Tramp FPGA lines: 50ms

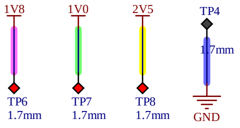


Tadj= 120us/nF
10nF = 1.2ms
100nF - 12ms
470nF = 56.4ms

External Trigger

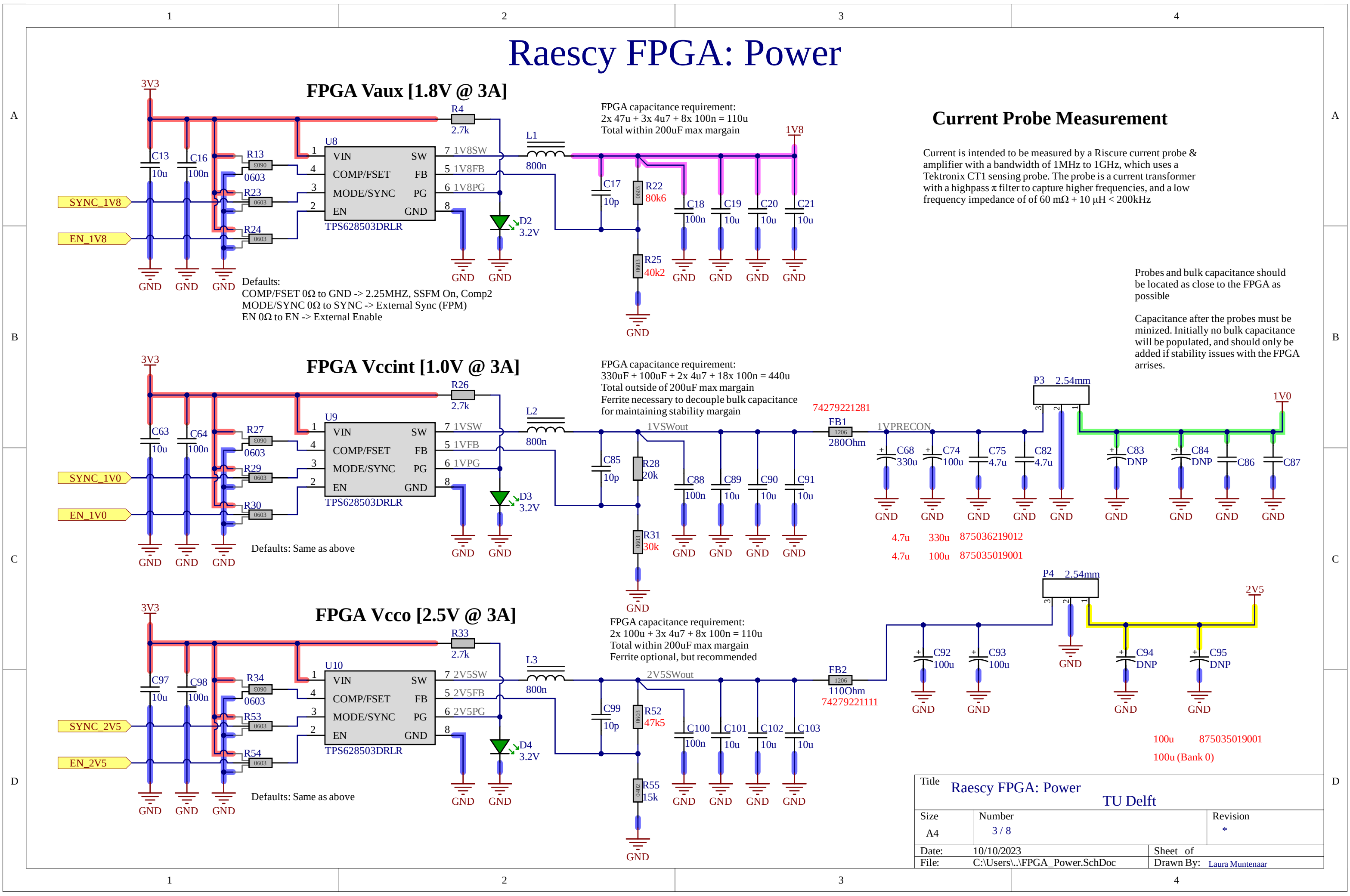


Power testpoints



Title	Raescy FPGA: Power Control *	
Size A4	Number 2 / 8	Revision *
Date: 10/10/2023		Sheet of
File: C:\Users\..\FPGA_Power_Control.SchDoc		Drawn By: *

Raescy FPGA: Power



Defaults:
COMP/FSET 0Ω to GND -> 2.25MHZ, SSFM On, Comp2
MODE/SYNC 0Ω to SYNC -> External Sync (FPM)
EN 0Ω to EN -> External Enable

Defaults: Same as above

Defaults: Same as above

FPGA capacitance requirement:
2x 47u + 3x 4u7 + 8x 100n = 110u
Total within 200uF max margain

FPGA capacitance requirement:
330uF + 100uF + 2x 4u7 + 18x 100n = 440u
Total outside of 200uF max margain
Ferrite necessary to decouple bulk capacitance
for maintaining stability margain

FPGA capacitance requirement:
2x 100u + 3x 4u7 + 8x 100n = 110u
Total within 200uF max margain
Ferrite optional, but recommended

Current Probe Measurement

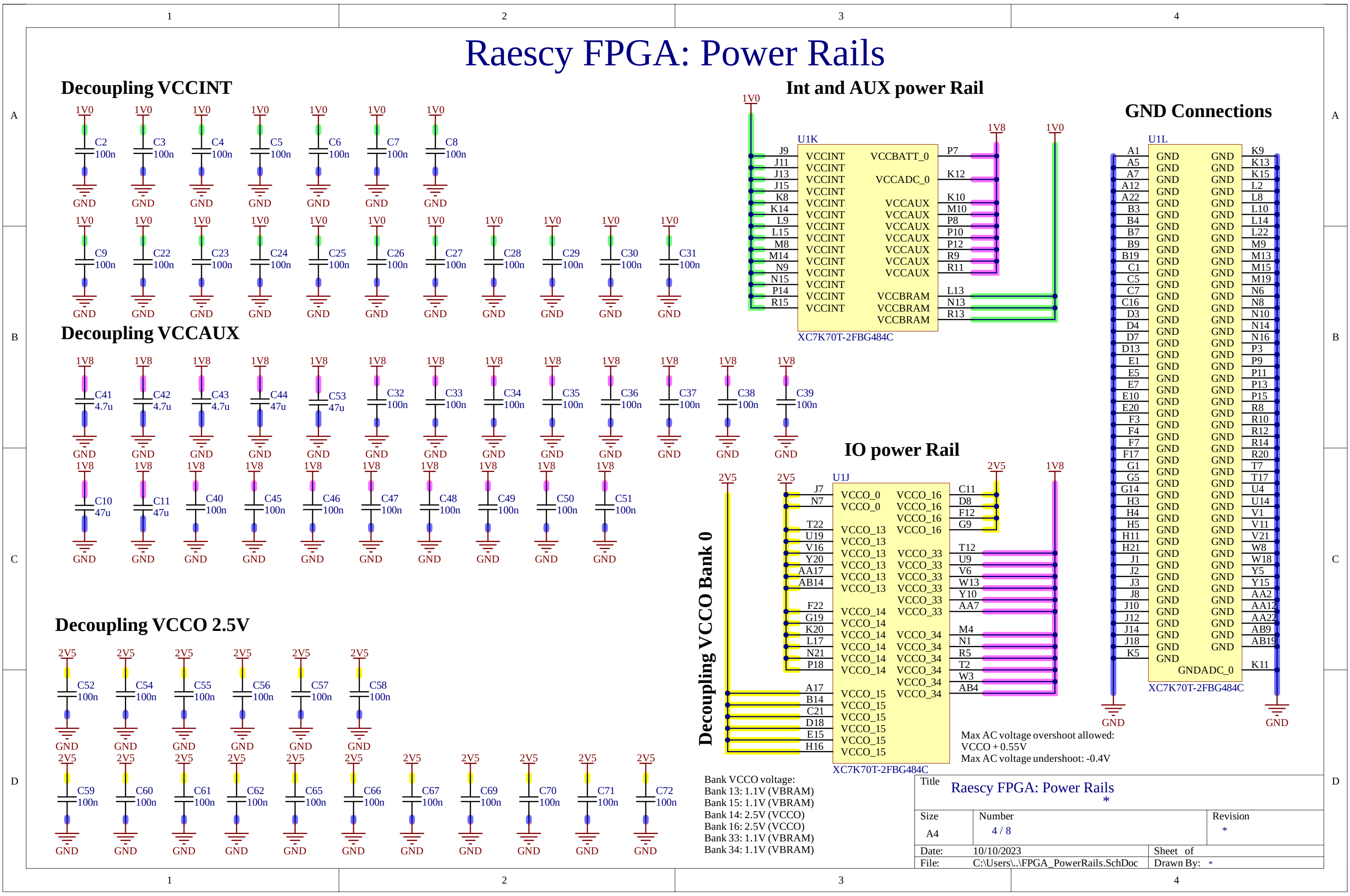
Current is intended to be measured by a Riscure current probe & amplifier with a bandwidth of 1MHz to 1GHz, which uses a Tektronix CT1 sensing probe. The probe is a current transformer with a highpass π filter to capture higher frequencies, and a low frequency impedance of of 60 mΩ + 10 µH < 200kHz

Probes and bulk capacitance should be located as close to the FPGA as possible

Capacitance after the probes must be minized. Initially no bulk capacitance will be populated, and should only be added if stability issues with the FPGA arrises.

Title: Raescy FPGA: Power	TU Delft	
Size: A4	Number: 3 / 8	Revision: *
Date: 10/10/2023	Sheet of	
File: C:\Users\..\FPGA_Power.SchDoc	Drawn By: Laura Muntenaar	

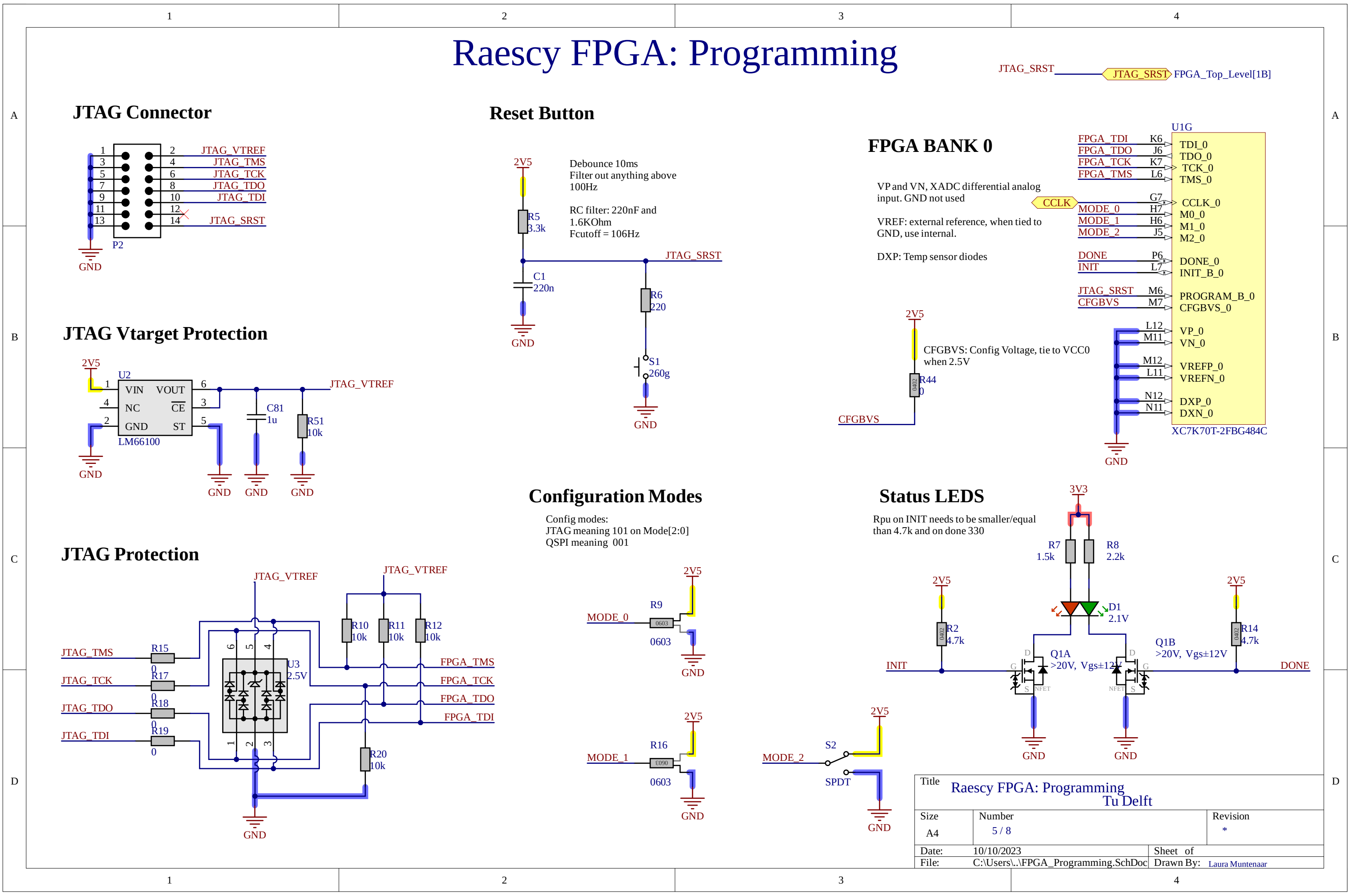
Raescy FPGA: Power Rails

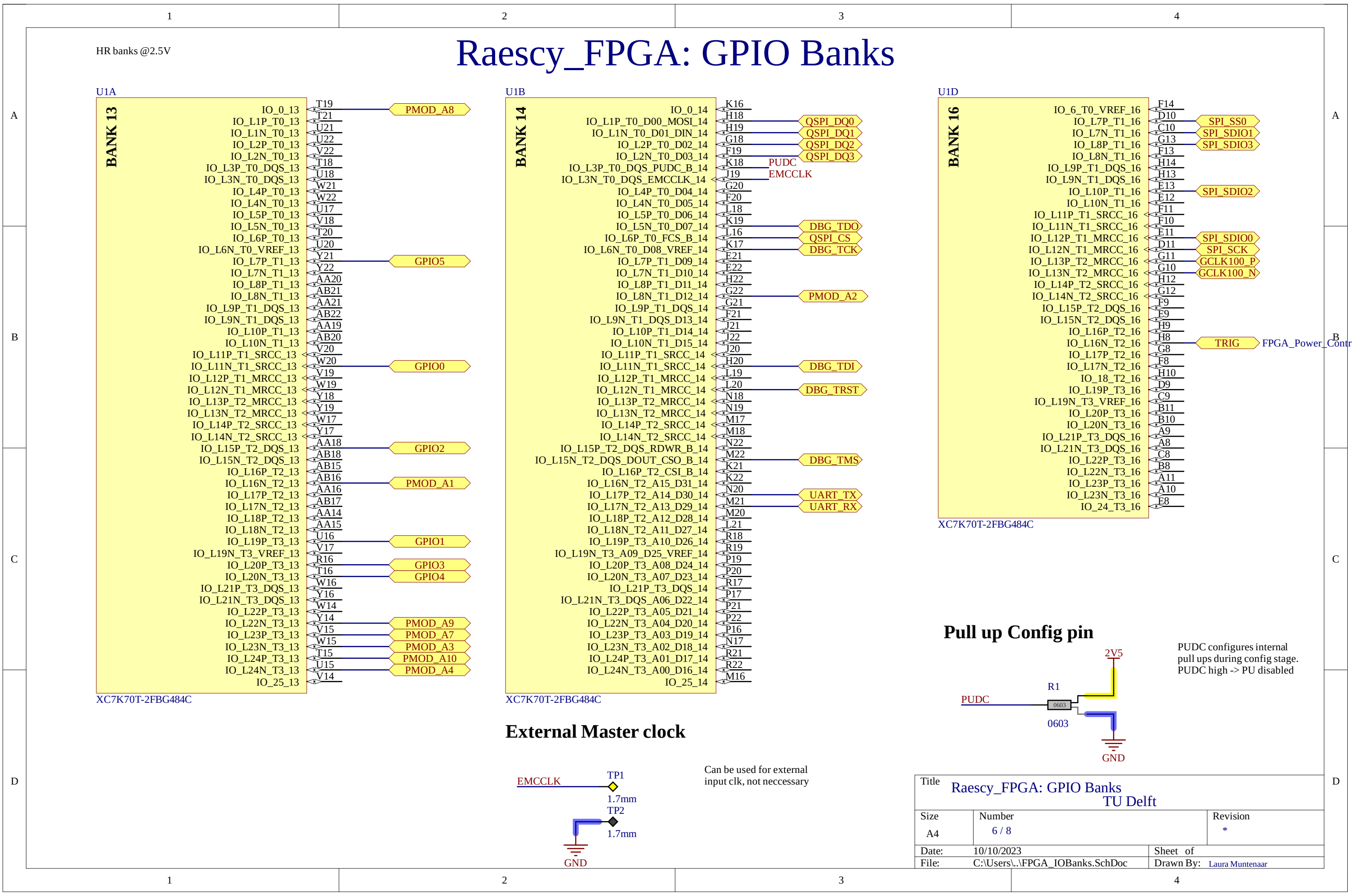

Decoupling VCCINT
1V0
C2 100n C3 100n C4 100n C5 100n C6 100n C7 100n C8 100n
C9 100n C22 100n C23 100n C24 100n C25 100n C26 100n C27 100n C28 100n C29 100n C30 100n C31 100n
Decoupling VCCAUX
1V8
C41 4.7u C42 4.7u C43 4.7u C44 47u C53 47u C32 100n C33 100n C34 100n C35 100n C36 100n C37 100n C38 100n C39 100n
C10 47u C11 47u C40 100n C45 100n C46 100n C47 100n C48 100n C49 100n C50 100n C51 100n
Decoupling VCCO 2.5V
2V5
C52 100n C54 100n C55 100n C56 100n C57 100n C58 100n
C59 100n C60 100n C61 100n C62 100n C65 100n C66 100n C67 100n C69 100n C70 100n C71 100n C72 100n
GND
Int and AUX power Rail
U1K XC7K70T-2FBG484C
J9 J11 J13 J15 K8 K14 L9 L15 M8 M14 N9 N15 P14 R15 VCCINT
VCCBATT_0 P7
VCCADC_0 K12
VCCAUX K10 M10 P8 P10 P12 R9 R11
VCCBRAM L13 N13 R13
GND Connections
U1L XC7K70T-2FBG484C
GND GNDADC_0
IO power Rail
U1J XC7K70T-2FBG484C
VCCO_0 J7 N7
VCCO_16 C11 D8 F12 G9
VCCO_13 T22 U19 V16 Y20 AA17 AB14
VCCO_33 T12 U9 V6 W13 Y10 AA7
VCCO_14 F22 G19 K20 L17 N21 P18
VCCO_34 M4 N1 R5 T2 W3 AB4
VCCO_15 A17 B14 C21 D18 E15 H16
Decoupling VCCO Bank 0

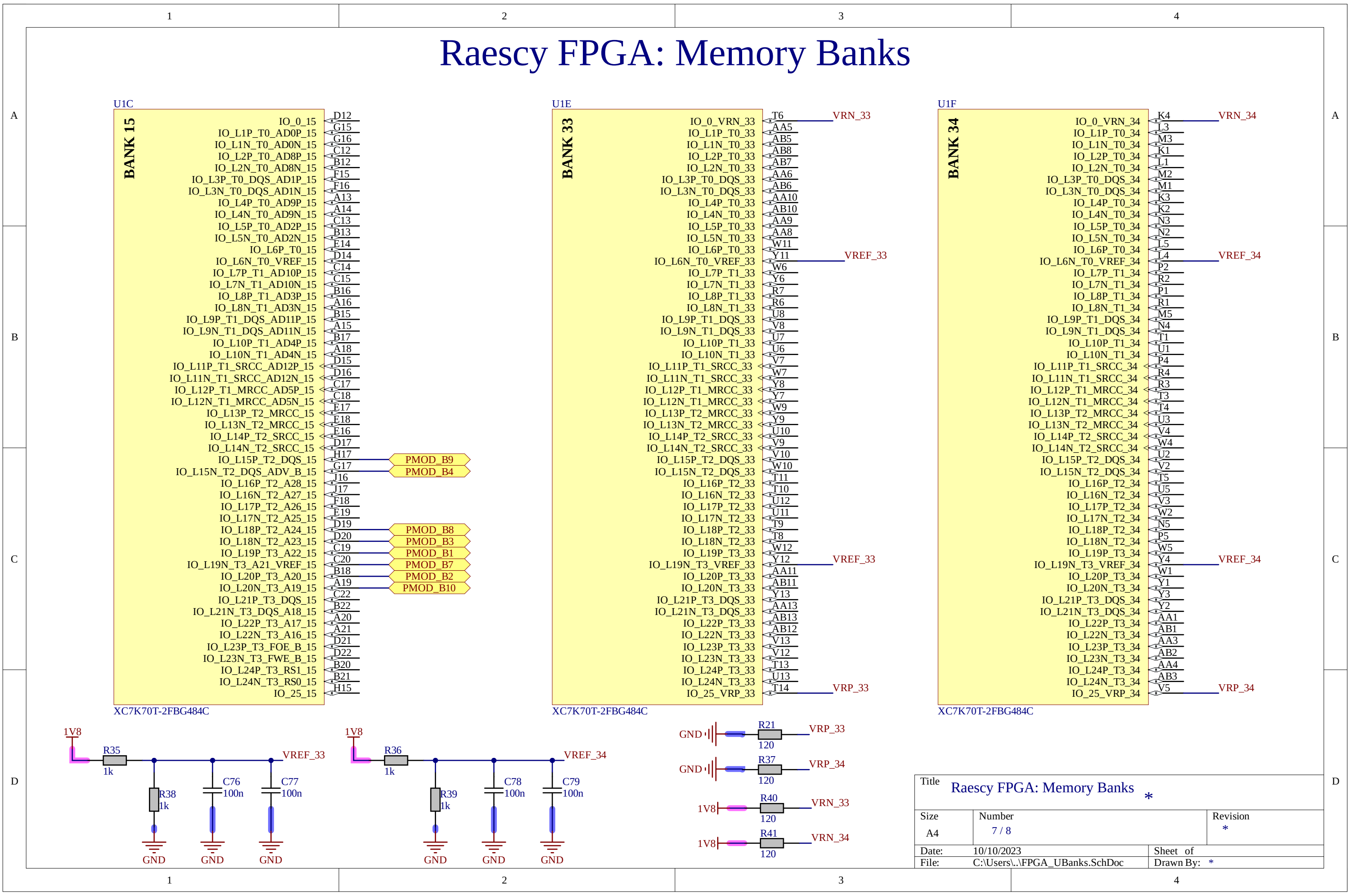
Max AC voltage overshoot allowed:
VCCO + 0.55V
Max AC voltage undershoot: -0.4V

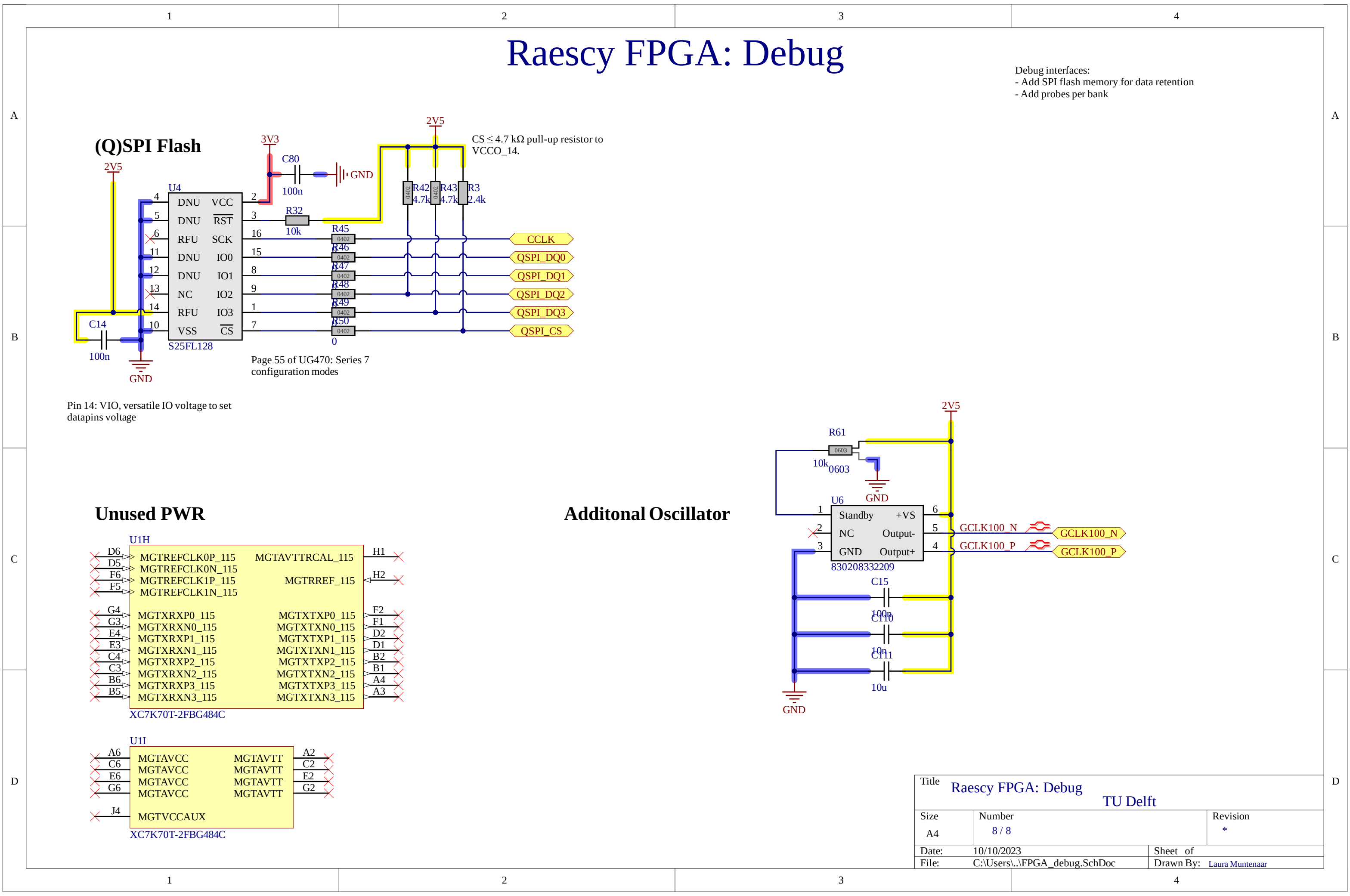
Bank VCCO voltage:
Bank 13: 1.1V (VBRAM)
Bank 15: 1.1V (VBRAM)
Bank 14: 2.5V (VCCO)
Bank 16: 2.5V (VCCO)
Bank 33: 1.1V (VBRAM)
Bank 34: 1.1V (VBRAM)

Title: Raescy FPGA: Power Rails	*	
Size: A4	Number: 4 / 8	Revision: *
Date: 10/10/2023	Sheet of	
File: C:\Users\..\FPGA_PowerRails.SchDoc	Drawn By: *	


Raescy FPGA: Programming
JTAG Connector
P2
JTAG_VTREF
JTAG_TMS
JTAG_TCK
JTAG_TDO
JTAG_TDI
JTAG_SRST
GND
Reset Button
2V5
Debounce 10ms
Filter out anything above 100Hz
RC filter: 220nF and 1.6KOhm
Fcutoff = 106Hz
R5 3.3k
C1 220n
R6 220
S1 260g
JTAG_SRST
FPGA_Top_Level[1B]
FPGA BANK 0
VP and VN, XADC differential analog input. GND not used
VREF: external reference, when tied to GND, use internal.
DXP: Temp sensor diodes
CFGBVS: Config Voltage, tie to VCC0 when 2.5V
R44 0
CFGBVS
U1G
FPGA_TDI K6 TDI_0
FPGA_TDO J6 TDO_0
FPGA_TCK K7 TCK_0
FPGA_TMS L6 TMS_0
CCLK G7 CCLK_0
MODE_0 H7 M0_0
MODE_1 H6 M1_0
MODE_2 J5 M2_0
DONE P6 DONE_0
INIT L7 INIT_B_0
JTAG_SRST M6 PROGRAM_B_0
CFGBVS M7 CFGBVS_0
L12 VP_0
M11 VN_0
M12 VREFP_0
L11 VREFN_0
N12 DXP_0
N11 DXN_0
XC7K70T-2FBG484C
JTAG Vtarget Protection
U2 LM66100
VIN VOUT NC CE GND ST
C81 1u
R51 10k
JTAG Protection
R15 0
R17 0
R18 0
R19 0
U3 2.5V
R10 10k
R11 10k
R12 10k
R20 10k
FPGA_TMS
FPGA_TCK
FPGA_TDO
FPGA_TDI
Configuration Modes
Config modes:
JTAG meaning 101 on Mode[2:0]
QSPI meaning 001
R9 0603
MODE_0
R16 0603
MODE_1
MODE_2
S2 SPDT
Status LEDS
3V3
Rpu on INIT needs to be smaller/equal than 4.7k and on done 330
R7 1.5k
R8 2.2k
D1 2.1V
R2 4.7k
R14 4.7k
Q1A >20V, Vgs±12V
Q1B >20V, Vgs±12V
INIT
DONE
Title Raescy FPGA: Programming
Tu Delft
Size A4
Number 5 / 8
Revision *
Date: 10/10/2023
File: C:\Users\..\FPGA_Programming.SchDoc
Sheet of
Drawn By: Laura Muntenaar


Raescy_FPGA: GPIO Banks
HR banks @2.5V
U1A
BANK 13
IO_0_13 T19 PMOD_A8
IO_L6N_T0_VREF_13 Y21 GPIO5
IO_L11N_T1_SRCC_13 W20 GPIO0
IO_L15P_T2_DQS_13 AA18 GPIO2
IO_L16N_T2_13 AB16 PMOD_A1
IO_L19P_T3_13 U16 GPIO1
IO_L20P_T3_13 R16 GPIO3
IO_L20N_T3_13 T16 GPIO4
IO_L22N_T3_13 Y14 PMOD_A9
IO_L23P_T3_13 V15 PMOD_A7
IO_L23N_T3_13 W15 PMOD_A3
IO_L24P_T3_13 T15 PMOD_A10
IO_L24N_T3_13 U15 PMOD_A4
IO_25_13 V14
XC7K70T-2FBG484C
U1B
BANK 14
IO_L1P_T0_D00_MOSI_14 H18 QSPI_DQ0
IO_L1N_T0_D01_DIN_14 H19 QSPI_DQ1
IO_L2P_T0_D02_14 G18 QSPI_DQ2
IO_L2N_T0_D03_14 F19 QSPI_DQ3
IO_L3P_T0_DQS_PUDC_B_14 K18 PUDC
IO_L3N_T0_DQS_EMCCLK_14 J19 EMCCLK
IO_L5N_T0_D07_14 K19 DBG_TDO
IO_L6P_T0_FCS_B_14 L16 QSPI_CS
IO_L6N_T0_D08_VREF_14 K17 DBG_TCK
IO_L8N_T1_D12_14 G22 PMOD_A2
IO_L11N_T1_SRCC_14 H20 DBG_TDI
IO_L12N_T1_MRCC_14 L20 DBG_TRST
IO_L15N_T2_DQS_DOUT_CSO_B_14 M22 DBG_TMS
IO_L17P_T2_A14_D30_14 N20 UART_TX
IO_L17N_T2_A13_D29_14 M21 UART_RX
IO_25_14 M16
XC7K70T-2FBG484C
U1D
BANK 16
IO_L7P_T1_16 D10 SPI_SS0
IO_L7N_T1_16 C10 SPI_SDIO1
IO_L8P_T1_16 G13 SPI_SDIO3
IO_L10P_T1_16 E13 SPI_SDIO2
IO_L12P_T1_MRCC_16 E11 SPI_SDIO0
IO_L12N_T1_MRCC_16 D11 SPI_SCK
IO_L13P_T2_MRCC_16 G11 GCLK100_P
IO_L13N_T2_MRCC_16 G10 GCLK100_N
IO_L16N_T2_16 G8 TRIG FPGA_Power_Control
IO_24_T3_16 E8
XC7K70T-2FBG484C
Pull up Config pin
2V5
R1 0603
PUDC
GND
PUDC configures internal pull ups during config stage.
PUDC high -> PU disabled
External Master clock
EMCCLK
TP1 1.7mm
TP2 1.7mm
Can be used for external input clk, not neccessary
Title Raescy_FPGA: GPIO Banks
TU Delft
Size A4
Number 6 / 8
Revision *
Date: 10/10/2023
File: C:\Users\..\FPGA_IOBanks.SchDoc
Sheet of
Drawn By: Laura Muntenaar



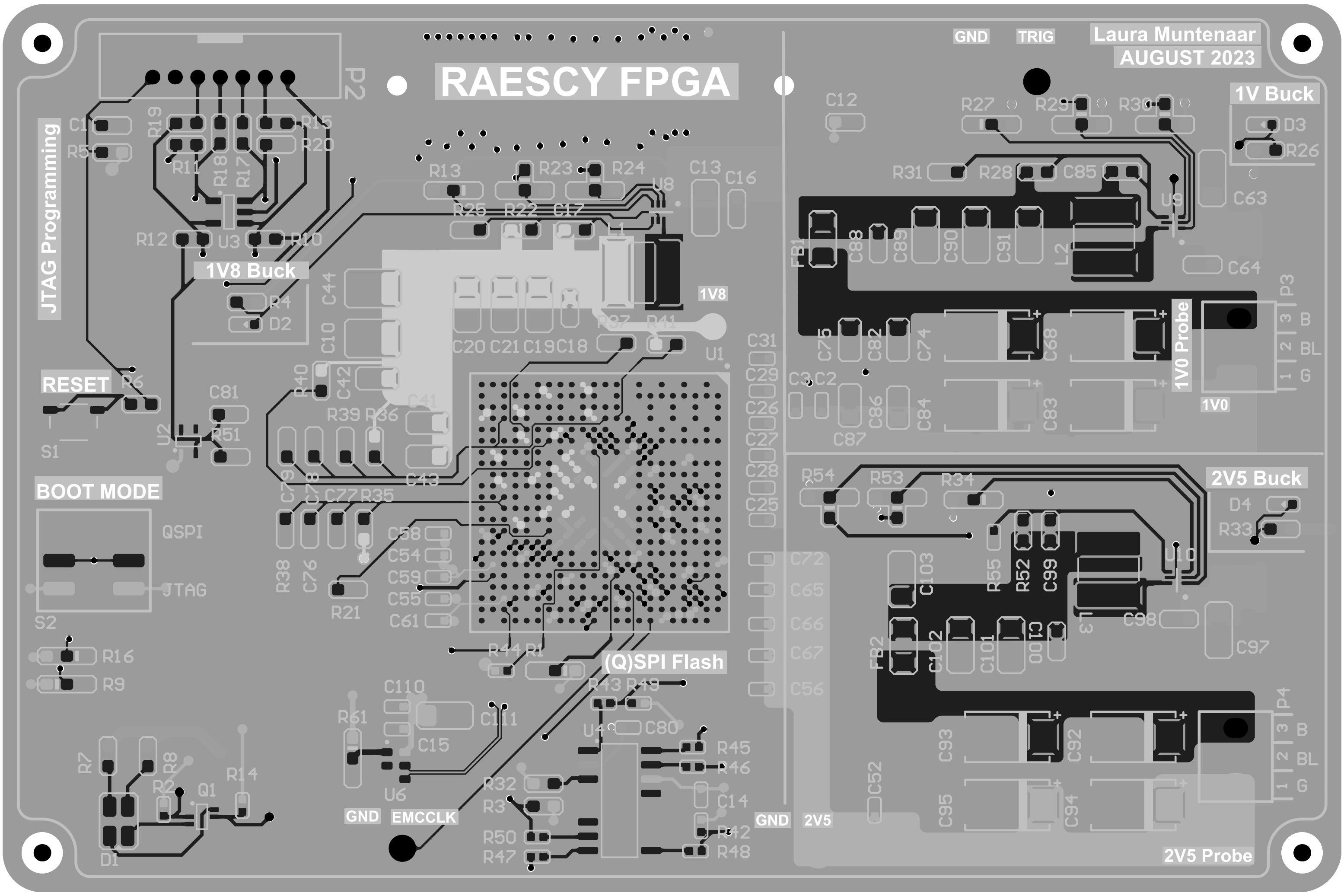


Debug interfaces:
- Add SPI flash memory for data retention
- Add probes per bank

CS ≤ 4.7 kΩ pull-up resistor to VCCO_14.

Page 55 of UG470: Series 7 configuration modes

Pin 14: VIO, versatile IO voltage to set datapins voltage



Board Stack Report

B

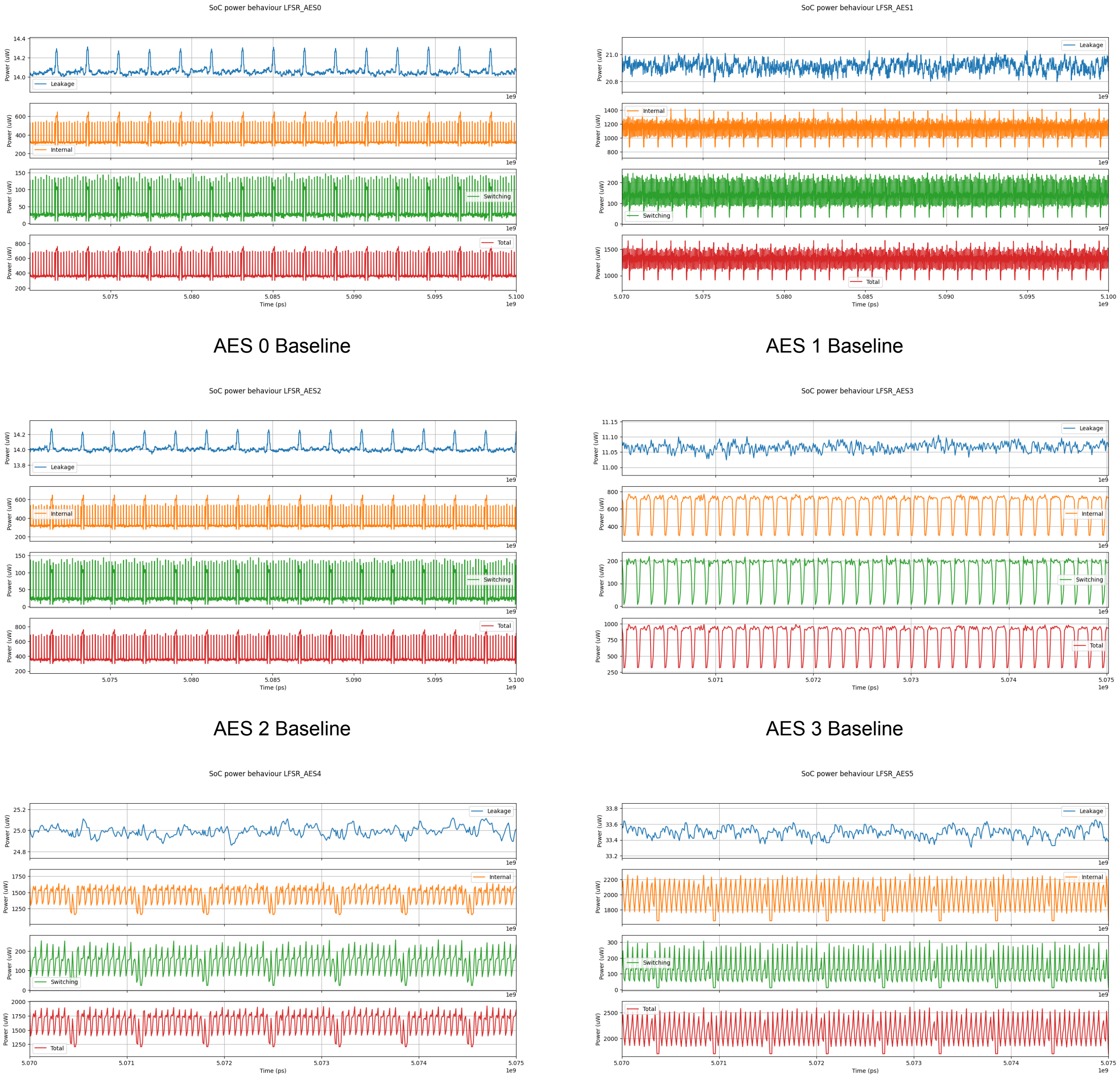
Register map of Peripherals

Pheripheral	Registername	Description	Address	Reset	Bit definitions					
SPI			0x1A10	0						
	STATUS_REG		0x1A10 0000	0	SPI Rd [0]	SPI wr [1]	SPI qrd [2]	SPI qwr [3]	SPI Csreg [11:8]	
	CONFIG_REG		0x1A10 0004	0	CLKDIV [7:0]					
	SPICMD		0x1A10 0008	0	CMD[31:0]					
	SPIADR		0x1A10 000C	0	ADRESS [31:0]					
	SPILEN		0x1A10 0010	0	CMDLEN[5:0]	ADDRLEN[11:8]	DATALEN[31:16]			
	SPIDUM		0x1A10 0014	0	DUMMYRD[0:15]	DUMMYWR[31:16]				
	TXFIFO		0x1A10 0018	0	TX[31:0]					
	RXFIFO		0x1A10 001C	0	RX[31:0]					
	INTCNF		0x1A10 0020	0						EN[31]
UART			0x1B10							
	DATA_REG	Read write register based on HWRITE	0x1B10 0000	0	RX[7:0]	TX[7:0]	status[7:0]			
	CONFIG_REG		0x1B10 0004	0	Baud_rate[0]					
TIMER			0x1C10							
	LOAD_REG	Load timing period	0x1C10 0000	0	LOAD[31:0]					
	VALUE_REG	Read current value timer	0x1C10 0004	0						
	CONTROL_REG	Select which timer, freerun or period and enable	0x1C10 0008	0	ENABLE[0]	MODE[1]	TMRSELECT[2]			
	CLEAR_REG	Clear timers	0x1C10 000C	0	CLEAREN[0]	CLRCLK16[1]				
LFSR+AES0			0x1D10							
	STATUS_REG		0x1D10 0000	0	AES_IRQ					
	CONFIG_REG		0x1D10 0004	0	START[2]	ENC/DEC[3]	LFSR_RST[4]	RESTART[5]	TRIG[6]	
	KEY_REG[31:0]		0x1D10 0008	0	KEY_REG[31:0]					
	KEY_REG[63:32]		0x1D10 000C	0	KEY_REG[63:32]					
	KEY_REG[95:64]		0x1D10 0010	0	KEY_REG[95:64]					
	KEY_REG[127:96]		0x1D10 0014	0	KEY_REG[127:96]					
	PL_REGISTER[31:0]		0x1D10 0018	0	PL_REGISTER[31:0]					
	PL_REGISTER[63:32]		0x1D10 001C	0	PL_REGISTER[63:32]					
	PL_REGISTER[95:64]		0x1D10 0020	0	PL_REGISTER[95:64]					
	PL_REGISTER[127:96]		0x1D10 0024	0	PL_REGISTER[127:96]					
	KEY_REG2[31:0]		0x1D10 0028	0	KEY_REG2[31:0]					
	KEY_REG2[63:32]		0x1D10 002C	0	KEY_REG2[63:32]					
	KEY_REG2[95:64]		0x1D10 0030	0	KEY_REG2[95:64]					
	KEY_REG2[127:96]		0x1D10 0034	0	KEY_REG2[127:96]					
	PL2_REGISTER[31:0]		0x1D10 0038	0	PL2_REGISTER[31:0]					
	PL2_REGISTER[63:32]		0x1D10 003C	0	PL2_REGISTER[63:32]					
	PL2_REGISTER[95:64]		0x1D10 0040	0	PL2_REGISTER[95:64]					
	PL2_REGISTER[127:96]		0x1D10 0044	0	PL2_REGISTER[127:96]					
	COUNT		0x1D10 0048	0	COUNT[31:0]					
	OUT_REGISTER[31:0]		0x1D10 0048	0	OUT_REGISTER[31:0]					
	OUT_REGISTER[63:32]		0x1D10 004C	0	OUT_REGISTER[63:32]					
	OUT_REGISTER[95:64]		0x1D10 0050	0	OUT_REGISTER[95:64]					
	OUT_REGISTER[127:96]		0x1D10 0054	0	OUT_REGISTER[127:96]					
LFSR+AES1			0x1E10							
	STATUS_REG		0x1E10 0000	0	AES_IRQ					
	CONFIG_REG		0x1E10 0004	0	START[2]	ENC/DEC[3]	LFSR_RST[4]	RESTART[5]	TRIG[6]	
	KEY_REG[31:0]		0x1E10 0008	0	KEY_REG[31:0]					
	KEY_REG[63:32]		0x1E10 000C	0	KEY_REG[63:32]					
	KEY_REG[95:64]		0x1E10 0010	0	KEY_REG[95:64]					
	KEY_REG[127:96]		0x1E10 0014	0	KEY_REG[127:96]					
	PL_REGISTER[31:0]		0x1E10 0018	0	PL_REGISTER[31:0]					
	PL_REGISTER[63:32]		0x1E10 001C	0	PL_REGISTER[63:32]					
	PL_REGISTER[95:64]		0x1E10 0020	0	PL_REGISTER[95:64]					
	PL_REGISTER[127:96]		0x1E10 0024	0	PL_REGISTER[127:96]					
	KEY_REG2[31:0]		0x1E10 0028	0	KEY_REG2[31:0]					
	KEY_REG2[63:32]		0x1E10 002C	0	KEY_REG2[63:32]					
	KEY_REG2[95:64]		0x1E10 0030	0	KEY_REG2[95:64]					
	KEY_REG2[127:96]		0x1E10 0034	0	KEY_REG2[127:96]					
	PL2_REGISTER[31:0]		0x1E10 0038	0	PL2_REGISTER[31:0]					
	PL2_REGISTER[63:32]		0x1E10 003C	0	PL2_REGISTER[63:32]					
	PL2_REGISTER[95:64]		0x1E10 0040	0	PL2_REGISTER[95:64]					
	PL2_REGISTER[127:96]		0x1E10 0044	0	PL2_REGISTER[127:96]					
	COUNT		0x1E10 0048	0	COUNT[31:0]					
	OUT_REGISTER[31:0]		0x1E10 0048	0	OUT_REGISTER[31:0]					
	OUT_REGISTER[63:32]		0x1E10 004C	0	OUT_REGISTER[63:32]					
	OUT_REGISTER[95:64]		0x1E10 0050	0	OUT_REGISTER[95:64]					
	OUT_REGISTER[127:96]		0x1E10 0054	0	OUT_REGISTER[127:96]					
LFSR+AES2			0x1F10							
	STATUS_REG		0x1F10 0000	0	AES_IRQ					
	CONFIG_REG		0x1F10 0004	0	START[2]	ENC/DEC[3]	LFSR_RST[4]	RESTART[5]	TRIG[6]	
	KEY_REG[31:0]		0x1F10 0008	0	KEY_REG[31:0]					
	KEY_REG[63:32]		0x1F10 000C	0	KEY_REG[63:32]					
	KEY_REG[95:64]		0x1F10 0010	0	KEY_REG[95:64]					
	KEY_REG[127:96]		0x1F10 0014	0	KEY_REG[127:96]					
	PL_REGISTER[31:0]		0x1F10 0018	0	PL_REGISTER[31:0]					
	PL_REGISTER[63:32]		0x1F10 001C	0	PL_REGISTER[63:32]					
	PL_REGISTER[95:64]		0x1F10 0020	0	PL_REGISTER[95:64]					
	PL_REGISTER[127:96]		0x1F10 0024	0	PL_REGISTER[127:96]					
	KEY_REG2[31:0]		0x1F10 0028	0	KEY_REG2[31:0]					
	KEY_REG2[63:32]		0x1F10 002C	0	KEY_REG2[63:32]					
	KEY_REG2[95:64]		0x1F10 0030	0	KEY_REG2[95:64]					
	KEY_REG2[127:96]		0x1F10 0034	0	KEY_REG2[127:96]					
	PL2_REGISTER[31:0]		0x1F10 0038	0	PL2_REGISTER[31:0]					
	PL2_REGISTER[63:32]		0x1F10 003C	0	PL2_REGISTER[63:32]					

	PL2_REGISTER[95:64]	0x1F10 0040	0	PL2_REGISTER[95:64]				
	PL2_REGISTER[127:96]	0x1F10 0044	0	PL2_REGISTER[127:96]				
	COUNT	0x1F10 0048	0	COUNT[31:0]				
	OUT_REGISTER[31:0]	0x1F10 0048	0	OUT_REGISTER[31:0]				
	OUT_REGISTER[63:32]	0x1F10 004C	0	OUT_REGISTER[63:32]				
	OUT_REGISTER[95:64]	0x1F10 0050	0	OUT_REGISTER[95:64]				
	OUT_REGISTER[127:96]	0x1F10 0054	0	OUT_REGISTER[127:96]				
LFSR+AES3		0x2010						
	STATUS_REG	0x2010 0000	0	AES_IRQ				
	CONFIG_REG	0x2010 0004	0	START[2]	ENC/DEC[3]	LFSR_RST[4]	RESTART[5]	TRIG[6]
	KEY_REG[31:0]	0x2010 0008	0	KEY_REG[31:0]				
	KEY_REG[63:32]	0x2010 000C	0	KEY_REG[63:32]				
	KEY_REG[95:64]	0x2010 0010	0	KEY_REG[95:64]				
	KEY_REG[127:96]	0x2010 0014	0	KEY_REG[127:96]				
	PL_REGISTER[31:0]	0x2010 0018	0	PL_REGISTER[31:0]				
	PL_REGISTER[63:32]	0x2010 001C	0	PL_REGISTER[63:32]				
	PL_REGISTER[95:64]	0x2010 0020	0	PL_REGISTER[95:64]				
	PL_REGISTER[127:96]	0x2010 0024	0	PL_REGISTER[127:96]				
	COUNT	0x2010 0028	0	COUNT[31:0]				
	OUT_REGISTER[31:0]	0x2010 0028	0	OUT_REGISTER[31:0]				
	OUT_REGISTER[63:32]	0x2010 002C	0	OUT_REGISTER[63:32]				
	OUT_REGISTER[95:64]	0x2010 0030	0	OUT_REGISTER[95:64]				
	OUT_REGISTER[127:96]	0x2010 0034	0	OUT_REGISTER[127:96]				
LFSR+AES4		0x2A10						
	STATUS_REG	0x2A10 0000	0	AES_IRQ				
	CONFIG_REG	0x2A10 0004	0	START[2]	ENC/DEC[3]	LFSR_RST[4]	RESTART[5]	TRIG[6]
	KEY_REG[31:0]	0x2A10 0008	0	KEY_REG[31:0]				
	KEY_REG[63:32]	0x2A10 000C	0	KEY_REG[63:32]				
	KEY_REG[95:64]	0x2A10 0010	0	KEY_REG[95:64]				
	KEY_REG[127:96]	0x2A10 0014	0	KEY_REG[127:96]				
	PL_REGISTER[31:0]	0x2A10 0018	0	PL_REGISTER[31:0]				
	PL_REGISTER[63:32]	0x2A10 001C	0	PL_REGISTER[63:32]				
	PL_REGISTER[95:64]	0x2A10 0020	0	PL_REGISTER[95:64]				
	PL_REGISTER[127:96]	0x2A10 0024	0	PL_REGISTER[127:96]				
	KEY_REG2[31:0]	0x2A10 0028	0	KEY_REG2[31:0]				
	KEY_REG2[63:32]	0x2A10 002C	0	KEY_REG2[63:32]				
	KEY_REG2[95:64]	0x2A10 0030	0	KEY_REG2[95:64]				
	KEY_REG2[127:96]	0x2A10 0034	0	KEY_REG2[127:96]				
	PL2_REGISTER[31:0]	0x2A10 0038	0	PL2_REGISTER[31:0]				
	PL2_REGISTER[63:32]	0x2A10 003C	0	PL2_REGISTER[63:32]				
	PL2_REGISTER[95:64]	0x2A10 0040	0	PL2_REGISTER[95:64]				
	PL2_REGISTER[127:96]	0x2A10 0044	0	PL2_REGISTER[127:96]				
	COUNT	0x2A10 0048	0	COUNT[31:0]				
	OUT_REGISTER[31:0]	0x2A10 0048	0	OUT_REGISTER[31:0]				
	OUT_REGISTER[63:32]	0x2A10 004C	0	OUT_REGISTER[63:32]				
	OUT_REGISTER[95:64]	0x2A10 0050	0	OUT_REGISTER[95:64]				
	OUT_REGISTER[127:96]	0x2A10 0054	0	OUT_REGISTER[127:96]				
LFSR+AES5		0x2B10						
	STATUS_REG	0x2B10 0000	0	AES_IRQ				
	CONFIG_REG	0x2B10 0004	0	START[2]	ENC/DEC[3]	LFSR_RST[4]	RESTART[5]	TRIG[6]
	KEY_REG[31:0]	0x2B10 0008	0	KEY_REG[31:0]				
	KEY_REG[63:32]	0x2B10 000C	0	KEY_REG[63:32]				
	KEY_REG[95:64]	0x2B10 0010	0	KEY_REG[95:64]				
	KEY_REG[127:96]	0x2B10 0014	0	KEY_REG[127:96]				
	PL_REGISTER[31:0]	0x2B10 0018	0	PL_REGISTER[31:0]				
	PL_REGISTER[63:32]	0x2B10 001C	0	PL_REGISTER[63:32]				
	PL_REGISTER[95:64]	0x2B10 0020	0	PL_REGISTER[95:64]				
	PL_REGISTER[127:96]	0x2B10 0024	0	PL_REGISTER[127:96]				
	KEY_REG2[31:0]	0x2B10 0028	0	KEY_REG2[31:0]				
	KEY_REG2[63:32]	0x2B10 002C	0	KEY_REG2[63:32]				
	KEY_REG2[95:64]	0x2B10 0030	0	KEY_REG2[95:64]				
	KEY_REG2[127:96]	0x2B10 0034	0	KEY_REG2[127:96]				
	PL2_REGISTER[31:0]	0x2B10 0038	0	PL2_REGISTER[31:0]				
	PL2_REGISTER[63:32]	0x2B10 003C	0	PL2_REGISTER[63:32]				
	PL2_REGISTER[95:64]	0x2B10 0040	0	PL2_REGISTER[95:64]				
	PL2_REGISTER[127:96]	0x2B10 0044	0	PL2_REGISTER[127:96]				
	COUNT	0x2B10 0048	0	COUNT[31:0]				
	OUT_REGISTER[31:0]	0x2B10 0048	0	OUT_REGISTER[31:0]				
	OUT_REGISTER[63:32]	0x2B10 004C	0	OUT_REGISTER[63:32]				
	OUT_REGISTER[95:64]	0x2B10 0050	0	OUT_REGISTER[95:64]				
	OUT_REGISTER[127:96]	0x2B10 0054	0	OUT_REGISTER[127:96]				

C

Power side channel results of all implemented AES engines



AES 0 Baseline

AES 1 Baseline

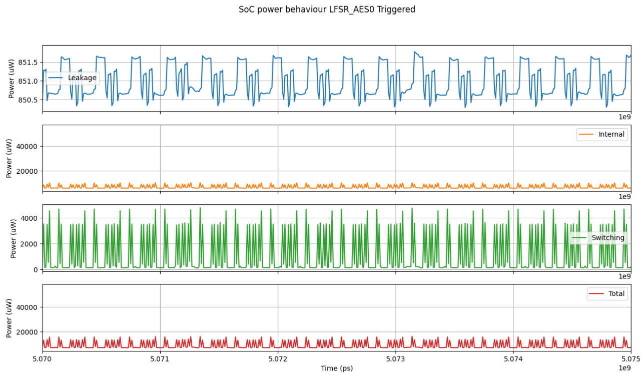
AES 2 Baseline

AES 3 Baseline

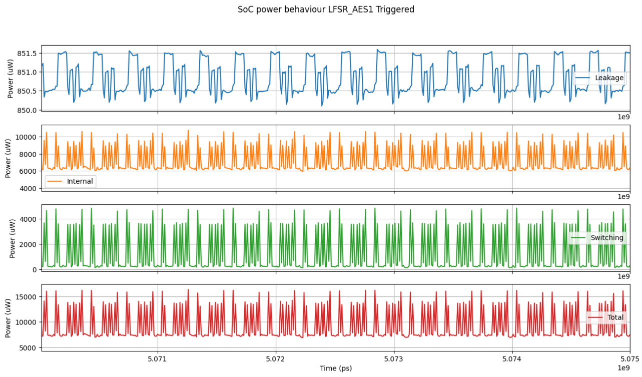
AES 4 Baseline

AES 5 Baseline

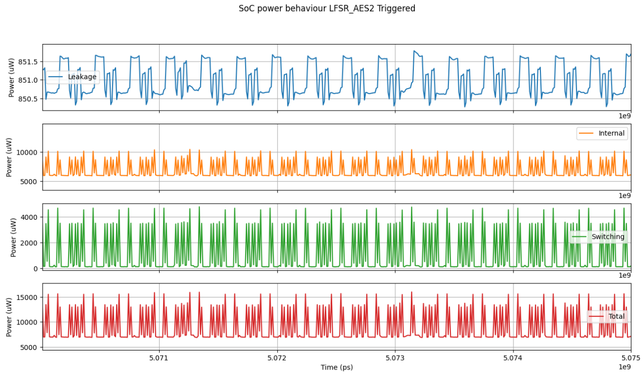
Figure C.1: Baseline power side channel behaviour



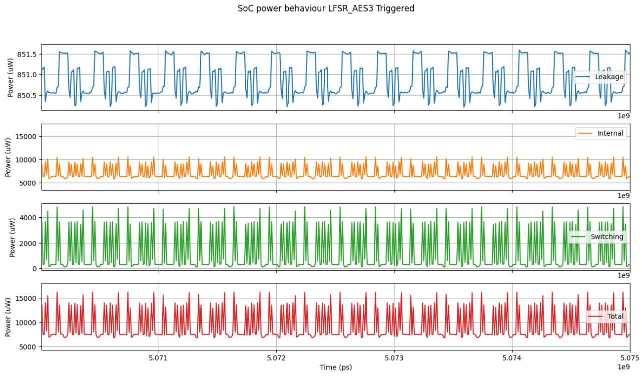
AES 0 Tirggered



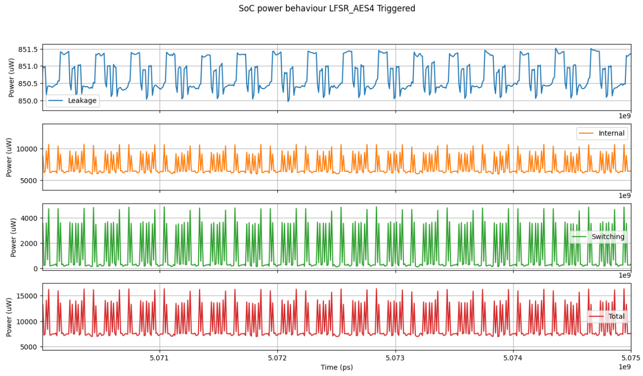
AES 1 Tirggered



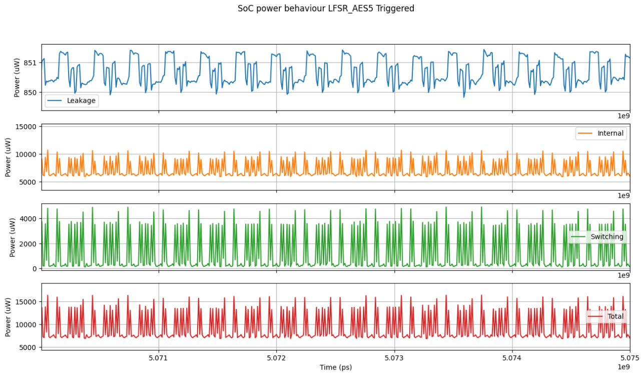
AES 2 Tirggered



AES 3 Tirggered

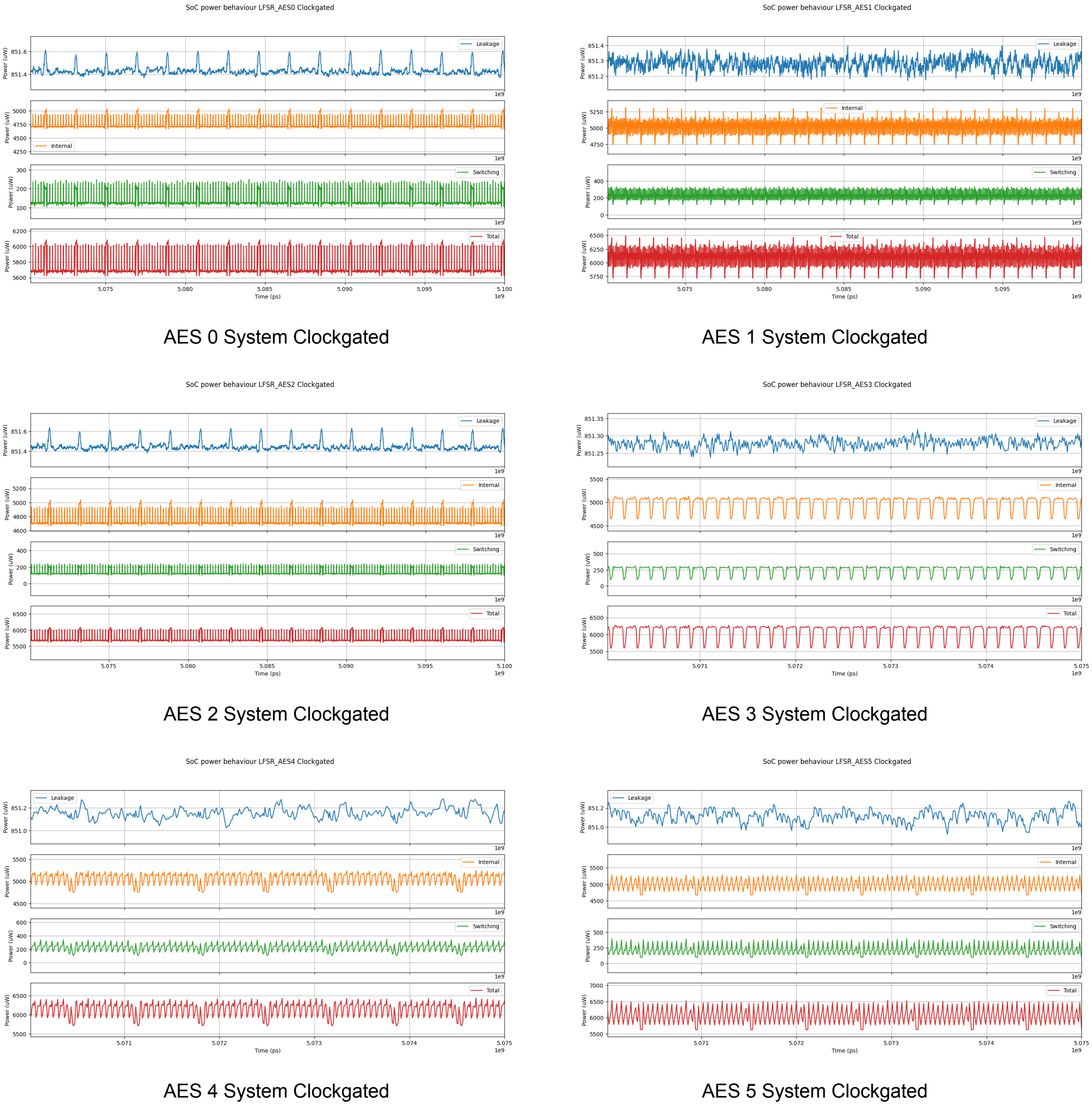


AES 4 Tirggered



AES 5 Tirggered

Figure C.2: Triggered power side channel behaviour



AES 0 System Clockgated

AES 1 System Clockgated

AES 2 System Clockgated

AES 3 System Clockgated

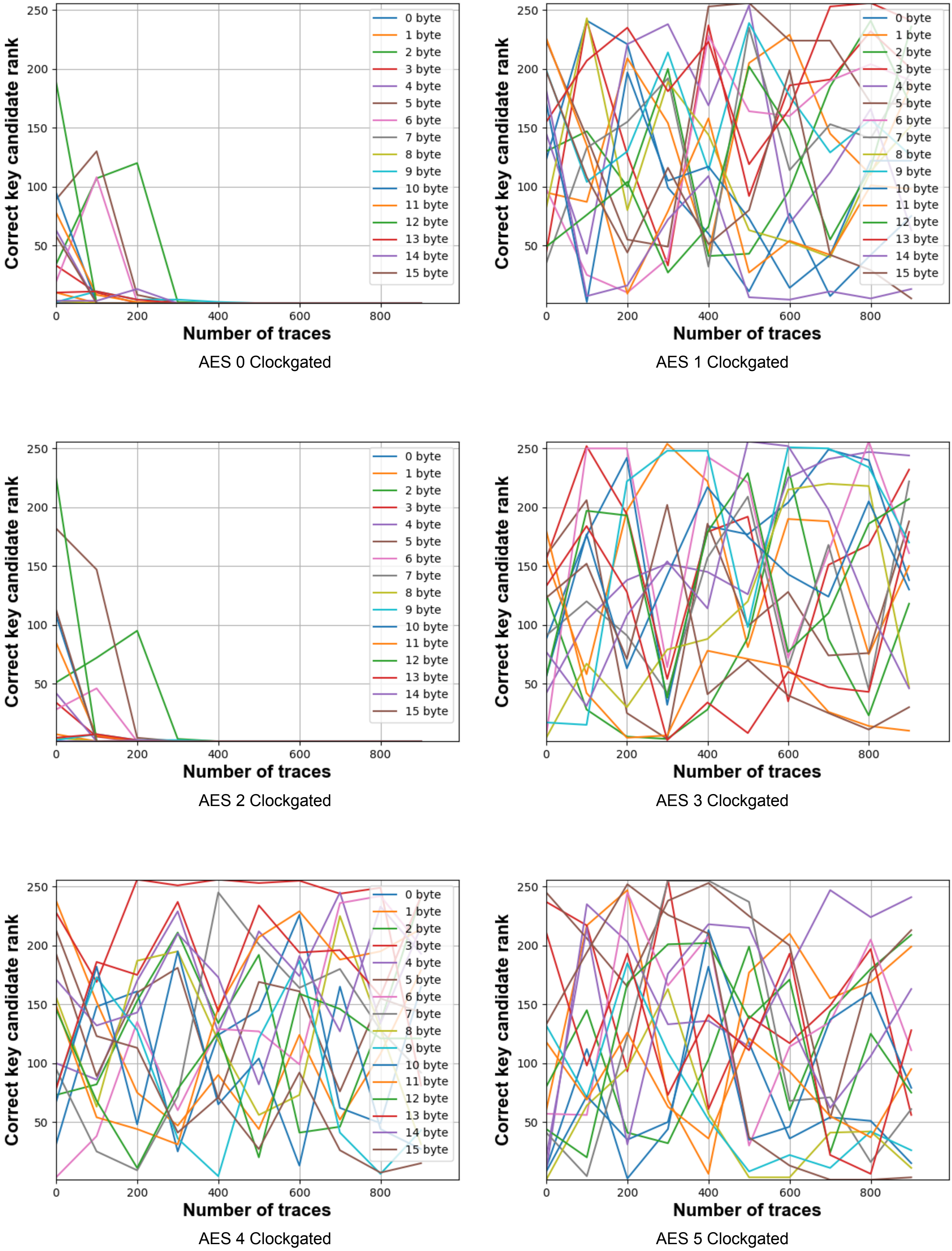
AES 4 System Clockgated

AES 5 System Clockgated

Figure C.3: System Clockgated power side channel behaviour

D

Correlation power analysis results of all implemented AES engines



AES 0 Clockgated

AES 1 Clockgated

AES 2 Clockgated

AES 3 Clockgated

AES 4 Clockgated

AES 5 Clockgated

Figure D.1: CPA on clockgated engines behaviour