Politecnico di Torino

Master's Thesis

Reverse engineering and analytic code extraction: techniques and threat analysis

Academic supervisor
Guido Masera

Student
Valerio Lanieri

February 20, 2019

"And you all know, security
Is mortals' chiefest enemy"

William Shakespeare, Macbeth (3.5, 32-33)

Abstract

Due to the pervasiveness of microprocessors, security has become a very important matter. Breaching the security of a device used for sensitive applications, such as a smart card, has devastating effects, since all the devices of the same model will be affected. This thesis aims at showing how reverse engineering can lead to such a scenario, as it can grant an attacker with enough knowledge to carry out a dangerous exploit. In particular, this thesis describes the reverse engineering of a ROM of a highly secure chip. The final goal is to extract the ROM's content, a very common objective for an attacker. The process of dumping the content of a ROM by inspecting its physical layout is called analytic ROM extraction.

The first step to reverse engineer the aforementioned ROM consisted in depackaging the microprocessor and in using failure analysis techniques to delayer it in order to expose the various metal layers and image them, using an SEM (*Scanning Electron Microscope*). The successive step consisted in combining the pictures together, realizing a multi-layer view of the ROM and starting the reverse engineering process: the single standard cells were reverse engineered and the metal connections between them were identified in order to reconstruct the netlist of the control circuitry. The observed netlist was then translated to VHDL, while the ROM bits were extracted from the pictures. The ROM was then simulated using ModelSim and, by feeding it with all the possible addresses in increasing order. The output of the ROM was sampled during the simulation in order to dump its content and complete the analytic code extraction.

In the introductory chapter the general concepts of hardware security are addressed to provide the reader with the basic information and mindset to understand the following chapters. The second chapter details the concept of reverse engineering, both from a legal and technical perspective. The third chapter describes in detail the result of the reverse engineering process, namely the architecture of the ROM. The last chapter concludes the thesis by commenting the obtained results and observing how this work would be evaluated in terms of security threat by the Common Criteria standard. The appendix, lastly, describes the evolution of the hardware security domain. This thesis has been realized during an internship at Texplained, a hardware security firm based in Sophia Antipolis. Due to the confidential nature of this work, the chip's name, its manufacturer and other sensitive data are omitted from this thesis.

Acknowledgements

I wish to thank my supervisor at Texplained, Olivier Thomas, for showing me the tricks of the trade, for always being available for explanations and for being an inexhaustible source of knowledge concerning hardware security. I also thank Tony Moor, head of the IC Inside Lab and a rockstar of IC deprocessing, for sharing his secrets with me.

Furthermore, I would not have been able to write this thesis without the help of very special persons, who gave me a crucial hand to overcome a very difficult time. One of these persons is madame Alexia Cepero of Eurecom's pedagogy office, who gave me a lot of support without thinking twice. I must also thank Martina, a dear friend and a fantastic flatmate, who stood by me tactfully but unconditionally.

Lastly, I wish to thank my parents, whose love shone like a flame in the darkness.

Contents

1 Introduction **1**

1.1 The concept of hardware security 1

1.2 Terminology 4

1.2.1 General security concepts 4

1.2.2 Cryptographic algorithms 5

1.2.3 Types of hardware attacks 8

1.2.4 Threat and security classification 11

1.2.5 Attack goals 13

1.3 Texplained 15

1.4 Goals 16

2 Reverse engineering of integrated circuits **18**

2.1 Laws on reverse engineering in the semiconductor industry 19

2.2 Failure Analysis techniques for the reverse engineering of integrated circuits 21

2.2.1 Depackaging 24

2.2.2 Deprocessing 25

2.2.3 Imaging 28

2.2.4 Analysis 31

3 ROM reverse engineering **36**

3.1 SEM pictures and ROM layers 36

3.2 ROM architecture 39

3.2.1 First address decoding stage 40

3.2.2 Second address decoding stage 41

3.2.3 Stripe and wordline selection stage 45

3.2.4 Bitline selection stage 47

3.2.5 ROM outputs 50

3.2.6 Summary 51

3.3 Bank content extraction 52

4 Conclusions **56**
4.1 Simulation and analytic code extraction 56
4.2 Follow up 57
4.3 Threat analysis 57
4.4 Further work 59

A Hardware security timeline **62**

List of Figures

1.1 Types of hardware attacks, their purpose and typical attack framework . . 2
1.2 DES scheme 6

2.1 Reverse engineering roadmap 22
2.2 Example of failure analysis: missing metal 1 pattern causing a missing connection 23
2.3 Example cross section of an integrated circuit 25
2.4 Deprocessing techniques 26
2.5 Sagitta station, integrating a CMP machine and an optical microscope . . . 27
2.6 An effective method to protect an IC from unwanted edge effects during a mechanical delayering process 28
2.7 Optical image of the substrate of a AVR ATMEGA48PB-MU microcontroller 29
2.8 Structure of a SEM 30
2.9 Gatan PECS sample coating and etching system 31
2.10 Example of multilayer superposition with Photoshop. 32
2.11 Example of annotations realized with Photoshop 33
2.12 NOR gate contained in 2.11 35

3.1 Example of SEM image with skew 37
3.2 Example of deskewed SEM image 38
3.3 ROM netlist, address store and decode section 40
3.4 *latch_mux* 41
3.5 ROM netlist, bank and bitline control section 42
3.6 ROM netlist, bank and bitline control section 43
3.7 *4NAND2B* 43
3.8 *ctrl_cell* 44
3.9 *ctrl_cell* 45
3.10 ROM netlist, wordline control section 46
3.11 *wordline_pupd* 46
3.12 ROM netlist, bitline MUX relative to the 16^{th} bit and the first bank pair . 48

3.13 ROM netlist, bitline MUX relative to the 16^{th} bit and the first bank pair . 49
3.14 *bank_cell* 50
3.15 ROM netlist, output stage for the 16^{th} bit 50
3.16 *bitline_cell* 51
3.17 Example of image processing and feature extraction using FIJI. 53

List of Tables

1.1 Types of hardware attacks 10
1.2 Common Criteria attack scenario scoring chart 14
1.3 Attack scenario score versus AVA_VAN assurance level 15

3.1 Size of the 4 banks 39
3.2 Bounding box and extracted features 54
3.3 Bank bits stitching process 55

4.1 Common criteria attack potential evaluation for analytic code extraction . . 59

A.1 Relationship between security level of several devices and time taken to hack it 68

Chapter 1

Introduction

This thesis has been realized during a 6 month internship at Texplained, a hardware security company based in Sophia Antipolis. In this chapter I address the concept of hardware security, with special attention paid to invasive attacks. I also describe Texplained, its mission and its resources. Notice that appendix A contains a basic chronology of hardware attacks. This section has been added in order to properly place this work inside the context of hardware security and give the reader a broader understanding of how hardware attacks have evolved, how the industry has answered to the various threats and the level of danger posed by the attacks described by this document. The objectives of my work are discussed as well, indicating the difficult aspects of performing the considered attacks, the modus operandi and the implications of a successful attack.

1.1 The concept of hardware security

When electronic hardware is used to protect sensitive data we fall in the domain of hardware security. The same is true if part of the hardware is used to prevent unwanted code to run on a microcontroller or even when a third party tries to steal intellectual property to evade non recurring engineering costs. Figure 1.1 shows the various types of attacks, as well as their goals. It is evident that hardware security is a broad domain with many branches, its common denominator being that it consists in preventing a third party (the attacker) from doing something he should not be allowed to do.

The main challenge posed by this field is that its foundations lie on an unstable ground. Absolute security does not exist: virtually, no countermeasure lasts forever. If the profit of hacking an electronic device overcomes the related costs, the device will be hacked, sooner or later, most likely by the party which will gain the highest benefit from it. Hardware attacks which are thought to be obsolete might make it back to the headlines. Conversely, new, sophisticated countermeasures might be bypassed in no time. If there is one lesson

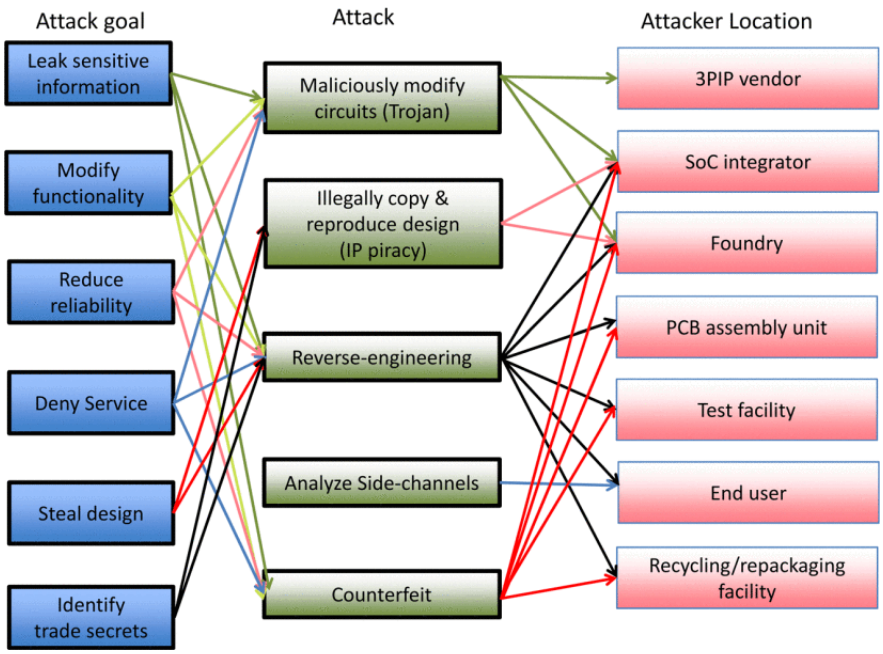


Figure 1.1: Types of hardware attacks, their purpose and typical attack framework
Source: [15] © 2014 IEEE

to keep in mind in the domain of security is that no conclusion is one hundred percent certain.

As previously stated, there is no way to grant absolute security and, even worse, no one except an attacker will know that a security breach as occurred, in most of the cases. All the possibilities must be kept open and countermeasures must be devised always thinking that the attacker is as skilled, resilient and resourceful as possible, given the considered device. For instance, in the case of a video game console, the typical worst case attacker is an experienced hacker who has previously hacked many other devices (and, possibly, older versions of the same video gaming system); in the case of a smart card used for secure authentications, the worst case attacker might be a government or a powerful multinational; and so on.

At this point it should be clear that hardware security is a never ending struggle between chip vendors and malicious third parties. It is the opinion of many scholars that chip vendors win a battle against hackers when an electronic device is replaced by a newer version without having been hacked yet. On the other hand, in the opinion of the author, attackers win the bout not by hacking a device per se but when they catch the other party unprepared. If a chip vendor is knowledgeable and aware of the possible risks it might still be possible for him to devise an emergency countermeasure for attacks which could happen, or at least to prepare a reaction by predicting worst case scenarios. In a nutshell, an attacker really has the upper hand when it succeeds in doing something which the chip

vendor was not even considering possible.

Another interesting conclusion is that the most suited persons to take care of security aspects are hackers themselves. They are used to think laterally and to spot weaknesses which a normal person would not be capable to recognize, especially at design time. It is therefore not surprising that many security courses challenge the students to bypass a security feature, be it of the software or of the hardware kind. Open mindedness and a broad knowledge on many fields are clearly the most useful skills a security expert should possess. When it comes to devise a countermeasure, however, there are no clean-cut rules: the adopted one should take into account the prototype attacker, the importance of the protected device and, therefore, the type of attacks which are most likely to be attempted. Among the numerous rules of thumb proposed by many scholars there is an axiom which still endures after more than a century:

"The system must not require secrecy and can be stolen by the enemy without causing trouble." [9]

This is the famous and ubiquitous second principle of security for cryptosystems elaborated by Auguste Kerchoffs in 1883. It proved to be so sound and insightful that it has become a cornerstone of security. This principle is in opposition with the concept of "security through obscurity", which holds that hiding the design of a secure system from the external world will contribute to its strength. This has been proven wrong in many cases, mostly due to the fact that at some point the attacker will know how the system works, either through information leaks from within the design company, through reverse engineering or even because the attack originated from the company itself.

The reason why Kerchoffs' principle is still valid is that it implies that the security of a system should never rely on its secrecy. Hence, security should always be granted by the secrecy of the keys and by sound security countermeasures, such as cryptographic algorithms. Notice that this does not necessarily mean that a security system should always be made public - after all, forcing the enemy to reverse engineer the design or to access it in some other uncomfortable way will make him waste time and resources, which are precious for both parties. On the other hand, by making a design open, scholars and "good" hackers will have a chance to take a look at it and to highlight faults that an attacker might exploit without notifying anyone. Hence, manufacturers should weigh pros and cons when considering how open should a design be.

Lastly, it is important to mark the differences between hardware and software security. The hardware and the software worlds coexist, but they respond to a very different set of rules. The latter deals with code, which has a more fluid nature than a digital circuit. A program can be updated, for instance in order to make it more resilient against a specific attack, while microelectronic hardware can not, or at least not without replacing entire

components. Conversely, hacking a program generally requires less resources than hacking hardware. While it is true that software hacking and hardware hacking do not necessarily have the same scope, and while it is true that, depending on the objective, software hacking can be more suited than hardware hacking or vice versa, it is certainly true that neglecting hardware security will benefit any possible type of attack.

Hardware influences whatever program is running on a microcontroller or a microprocessor. Hardware can provide the attacker with information through side-channels, such as power or electromagnetic emissions. There exist hardware attacks which allow to jump instructions, possibly allowing to bypass the need of performing a software hack. RAM memories can be detached and dumped to provide the attacker with normally inaccessible data. Moreover, hardware can be reverse engineered, granting the attacker virtually unlimited information and thus making it easier for him to devise an attack, be it software or hardware based. For all these reasons, hardware security has gained more and more attention in the recent years.

1.2 Terminology

As any field of expertise, hardware security has grown its own vocabulary. In this section I will explain several key words and concepts which should be known in order to understand this thesis, and before facing any scientific work involving hardware security in general. These keywords are in italic in order to make them easily identifiable.

1.2.1 General security concepts

First of all, we should clarify how the attacker devises an attack. In section 1.1 I already showed that the goals of a hardware attack can be manifold. In general, the attacker's job is to identify and exploit a *vulnerability*. A vulnerability is a weakness inherent to the device or to the software which might be used for many malicious purposes, including obtaining a cryptographic key, bypassing a security check or dumping a protected memory. For example, an accessible clock bus represents a vulnerability, as the attacker might induce clock glitches to make the processor jump vital instructions which grant security. The presence of a debug circuit, even if disconnected, represents a vulnerability as well, as the attacker might find a way to restore the connection (e.g. by using a FIB, $FocusedIonBeam$) to access powerful debug functions. It might be argued that hardware security consists in searching for vulnerabilities in order to eliminate them or, more frequently, make them harder to exploit.

Very often the goal of the attacker is to obtain a *cryptographic key*. These keys are sequences of bits which are very hard to guess, as they are used to implement services which are crucial to security. Some of the most popular ones are *data confidentiality*,

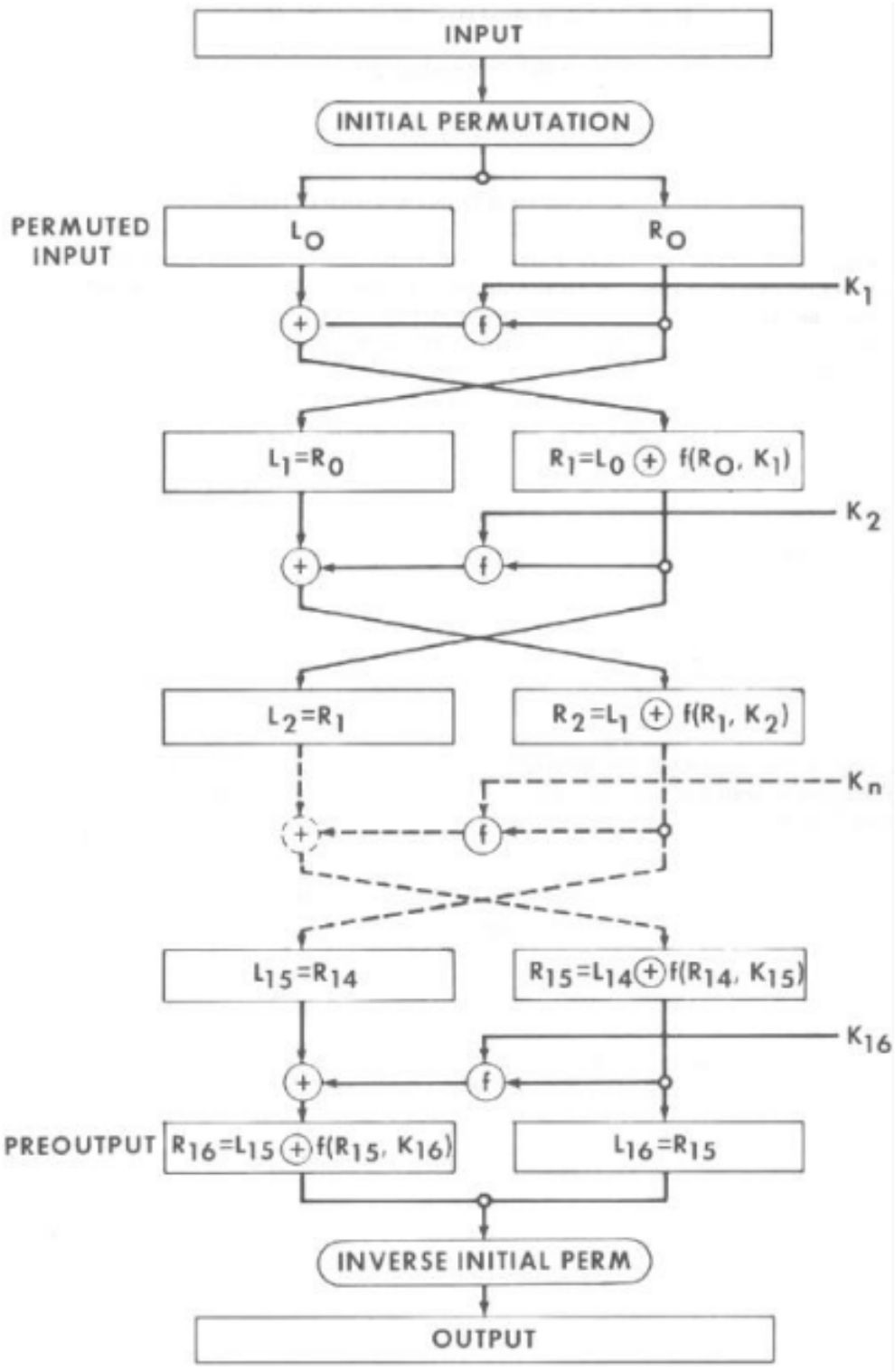
data integrity and *authentication*. These services fall under the umbrella of *cryptography*, possibly the major player in the security field. Many cryptographic algorithms exist, with different properties, advantages and disadvantages. The literature on the subject is endless and the scope of this thesis covers only a small portion of this domain, hence the discussion will be limited to some hints. The interested reader should refer to proper documentation to learn more on the subject.

To achieve data confidentiality, *data encryption* is employed. This technique is thousands of years old and has evolved greatly. It consists of combining a *cryptographic algorithm* with a key and a message, called *plaintext*, to output a *ciphertext*, namely an unintelligible (or unusable) version of the input. Notice that the term "plaintext" is used as a portmanteau for any sequence of symbols that contains information: a message, machine code, sensitive data, etc. In our case, obviously, both the plaintext and the ciphertext are made of bits. The plaintext can be recovered from the ciphertext by employing another algorithm which reverses the operations performed during encryption. This algorithm needs a cryptographic key as well, hence the reason why keys are sought after by attackers.

Cryptography may also grant data integrity, namely ensuring that this data can not be altered by an attacker, which is especially useful when it comes to code. In this case, *cryptographic hash functions* are employed. These functions take as input the plaintext and, sometimes, a cryptographic key to output a *hash*. Data integrity can be granted by comparing the hash obtained by feeding some code to the function with the hash obtained by using the original code as input: if the two hashes do not match, data alteration is detected and the device will take the necessary precautions (e.g. halting code execution). Cryptographic hash functions employing a key are used to provide authentication services. Consider, for instance, how the Windows operating system handles user authentication. In a nutshell, when creating a user account, the password is not stored in memory. Instead, it goes through a keyed hash function and the resulting hash is saved. Then, at login time, the input password will be used to produce a hash: if it matches with the stored one, access is granted. All the examples provided up until know give just a small overview on general security basics and techniques. As previously stated, there is plenty of literature on the subject which explains all these topics in a more detailed manner.

1.2.2 Cryptographic algorithms

There are two main classes of cryptographic algorithms: symmetric and asymmetric. Symmetric cryptography consists in using a single secret key, K, which is used both to encrypt and decrypt data. The encryption algorithm therefore makes use of a function which is necessarily invertible. Perhaps the most well known symmetric cryptography algorithm is *DES* (Data Encryption Standard), although far from being the most secure. Figure 1.2



Source: [5]

Figure 1.2: DES scheme

shows the basic block diagram of the DES algorithm. The input data undergoes an initial permutation and several blocks which use various operations (XOR, additional permutations, S-boxes) in combination with part of the input and a certain subkey, k_n, generated at each round by the *key schedule* algorithm, starting with the secret key K. All these operations are summed up by the f function.

$$L_{n+1} = R_n$$

$$R_{n+1} = L_n \oplus f(R_n, k_n)$$

Thanks to the properties of the XOR operation, the plaintext can be obtained by simply inputting the ciphertext in a reversed scheme, using a reversed key schedule (i.e. from the 16th to the 1st):

$$R_n = L_{n+1}$$

$$L_n = L_n \oplus f(R_n, k_n) \oplus f(R_n, k_n) = R_{n+1} \oplus f(R_n, k_n)$$

Asymmetric cryptography consists in using two different keys: a public one, K_p, and a private one, K_s. In a nutshell, a public key is used to encrypt data such that it can only be decrypted by the owner of the private key. Asymmetric cryptography can also be used for identification purposes: a signature can be encrypted using a private key, such that the receiver of a message may identify the sender by decrypting the signature with the corresponding public key. The most popular asymmetric cryptography scheme is RSA, whose name comes from the initials of the its three inventors, Rivest, Shamir and Adleman. The two keys can be computed as follows:

1. Pick two very large prime numbers, p and q;
2. Compute $R = pq$;
3. Compute $\Phi(R) = (p-1)(q-1)$;
4. Choose a small prime K_p such that it is smaller than $\Phi(R)$;
5. Compute d such that $K_s K_p \equiv 1 \ \ mod \ \ \Phi(R)$;

This scheme works thanks to Euler's theorem, which states that, as long as a is invertible mod R:

$$a^{\Phi(R)} \equiv 1 \ \ mod \ \ R$$

Given the fifth point in the list we have that:

$$K_s K_p = K_p K_s = n\Phi(R) + 1$$

Therefore:

$$(P^{K_p})^{K_s} \ mod \ R = (P^{K_s})^{K_p} \ mod \ R = P^{n\Phi(R)+1} \ mod \ R =$$

$$= (P^{n\Phi(R)} \ mod \ R)(P \ mod \ R) \ mod \ R = P \ mod \ R$$

Notice that the condition of invertibility mod R for an arbitrary plaintext, necessary in order to apply Euler's theorem, can be demonstrated using the Chinese Remainder Theorem. The original paper [14] can be used as a reference for this purpose. These two types of cryptographic systems have their advantages and disadvantages, which also include performance loss due to computational complexity. For this reason, the specific needs of each implementation dictate which is the most suitable cryptographic algorithm.

1.2.3 Types of hardware attacks

The aforementioned concepts are part of the core knowledge of security in general, but none of them is specific to hardware security. This field focuses on the physical properties of a microcontroller or a microprocessor: its layout, its connections, the clock distribution, power consumption, and so on. Hardware hackers analyze and, possibly, alter these aspects to accomplish their goals. Over the years three main categories of hardware attacks have emerged: *non-invasive*, *semi-invasive* and *invasive* attacks, based on the degree of physical manipulation necessary to perform a hack. Non-invasive attacks are those attacks which do not require to physically alter the circuit, e.g. without having to depackage it or to destroy the passivation layer.

They are the cheapest to perform, hence the threat they represent is that they are easily reproducible and that they can be attempted by anyone with limited resources. On the other hand, they usually give the attacker less room for manoeuvre, hence it might be very hard to perform an attack with non-invasive techniques if the device is particularly secure. Semi-invasive attacks were defined under the need to bridge the gap between non-invasive and invasive attacks [17]. For these attacks it is necessary to depackage the chip, thus they are harder and more expensive to perform than non-invasive attacks. However, they do not involve to establish any contact with the metal surface, hence they are simpler with respect to invasive attacks.

These attacks are performed when it is necessary to get closer to the integrated circuit without having to alter its layout: the metallization layers are not modified and the passivation layer is left untouched. Invasive attacks, on the other hand, require the physical

alteration of the internal components of an integrated circuit. They require expensive devices and skilled professionals to operate them, i.e. being the most demanding type of attack. They allow to perform the most efficient and dangerous attack techniques, such as *microprobing* and chip modification. Moreover, reverse engineering a chip through invasive techniques might allow the attacker to devise cheaper, simpler and yet extremely harmful non-invasive or semi-invasive attacks.

Fault injection and *side-channel attacks* are among the most well known and studied hardware attacks. These typologies can fit either in the non-invasive or semi-invasive category, based on the degree of alteration needed to perform the specific attack. For example, side-channel attacks involving electromagnetic radiations are best performed after chip decapsulation. Fault injection attacks consist in modifying the way the circuit behaves by playing with physical parameters, such as the clock or the supply voltage. *Clock glitching* is a widely known fault injection technique. It involves modifying the clock signal by manually injecting a charge in the clock bus, e.g. via a laser, causing a glitch to ensue in the normally periodic clock signal. This technique is often used to make a microprocessor jump instructions. In some cases, these instructions perform a password check before executing some privileged instruction or accessing secure data, hence skipping them will make the knowledge of the password completely unnecessary.

Side-channel attacks exploit information leaks originating as a side effect of the circuit operation. These information leaks are obviously unwanted by the circuit manufacturer and might come from unexpected sources. Notable sources used for side-channel attacks are power consumption or execution time, but research has proven that anything which is a consequence of code execution may be used (consider for instance the role of cache memories in the infamous Meltdown [12] and Spectre [10] attacks). The goal of side-channel attacks is usually to obtain a cryptographic key or secure data. Based on the way used to access bus lines, microprobing can be considered as non-invasive, semi-invasive or invasive. It consists in contacting a bus line or a wire where sensitive data goes through, e.g. the intermediate encryption stages of some plaintext, and analyze the extracted signals to break security. Based on the chip type, depackaging and passivation destruction might be necessary to expose the wire of interest (in which case, of course, the attack falls under the invasive category).

A famous probing attack for instance consisted in feeding the CPU of a highly secure chip, which employed on the fly code data decryption, with a large number of different instructions. The author of the attack then tried to identify them by observing the CPU behaviour, with the goal of gaining the machine code for enough instructions to write an encrypted program which could dump a protected memory [11]. Reverse engineering and chip modification are always considered invasive attacks. In all these cases chip depackaging and passivation removal are necessary steps. In the case of chip modification,

Category	Attacks	Notes
Non-invasive	Side-channel attacks - Timing analysis - Power analysis - EM analysis Glitch attacks - Clock glitching - Power glitching Brute force attacks Data remanence	- Do not require sample preparation; - Relatively cheap and easy to reproduce;
Semi-invasive	UV attacks Optical probing Fault injection EM Analysis	- More powerful than non-invasive attacks; - Much cheaper than invasive attacks; - Require depackaging but not removal of the passivation layer;
Invasive	Microprobing Chip modification Memory extraction	- Possibly the most threatening attacks; - Allow the reverse engineering of the chip; - Require lengthy sample preparation; - Need specific machines; - Technological progress makes them harder;

Table 1.1: Types of hardware attacks

metallization layers are altered to erase connections or to create new ones. For instance, the attacker could restore the connection between the main circuit and some debug circuit which has been used to test the device at manufacturing time but which has then been disconnected.

These circuits usually allow to perform powerful debug functions, possibly bypassing some circuit stages, hence being capable of operating them may grant more power over the circuit. Metallizations are altered by resorting to *focused ion beams*, complex and expensive machines which operate at sub-micron level and which require qualified and skilled personnel to operate them. Reverse engineering is addressed more thoroughly, since it is one of the main subjects of this thesis (refer to chapter 2). It involves to expose the various circuit layers, using sophisticated techniques and specific chemicals, and the usage of sub-micron microscopes to obtain pictures of the various layers and thus the knowledge of the physical layout of microelectronic circuits.

The circuit layout is observed in detail in order to understand how it works and in order to clearly identify and exploit weakness. If the goal is to dump a memory, and if no sophisticated obfuscation technique is employed, it might even be enough to simply look at the pictures of a metallization layer and identify zeros and ones in order to extract the whole code stored inside. Table 1.1 summarizes the attacks discussed in this section.

1.2.4 Threat and security classification

In this section the most widely employed classes of threat and of security levels are presented. The first list comes from a widely cited IBM paper [6] and describes the main types of attackers, their motivations, their background and their skills. The following list, taken from the same article [6], describes the protection levels attainable by a chip. Notice that these lists are intended to give an overall idea of the security framework, in real case scenarios the lines drawn by these definitions are often blurred.

Attacker types

Class I (clever outsiders)

This attacker is usually very intelligent but may have scarce knowledge of the system. They usually have access to moderately sophisticated equipment. Typically they try to take advantage of an existing weakness in the system rather than searching for a new one.

Class II (knowledgeable insiders)

This type of attacker has substantial specialized technical education and experience. It possesses a certain degree of understanding of parts of the system but potential access to most of it. This type of attacker has access to highly sophisticated tools and instruments for analysis.

Class III (funded organisations)

Highest level of threat. They are constituted by teams of specialists with related and complementary skills backed by great funding resources. These teams are capable of in-depth analysis of the system and can design sophisticated attacks, using the most advanced analysis tools. They may use Class II adversaries as part of the attack team.

Protection classes

Level ZERO

No special security features are used in the system. All parts can be freely accessed and easily investigated. Example: microcontrollers or a FPGA with an external ROM.

Level LOW

Some security features are present but they can be relatively easy defeated using simple tools, such as soldering iron and low cost analog oscilloscopes. Attacks take time but they do not involve more than 1,000 € of equipment. Example: a microcontroller with an unprotected internal memory but proprietary programming algorithm.

Level MODL

Security protects against most low cost attacks. More expensive tools are required as well as some special knowledge in order to breach security. Total equipment cost does

not exceed 3,000 €. Examples: microcontrollers weak against power analysis and power glitches.

Level MOD

Special tools and equipment are required for successful attack as well as some special skills and knowledge. Total equipment cost is up to 30,000 €. Only Class II attackers can afford this. The attack could be time-consuming. Examples: microcontrollers with protection against UV attacks, old smartcard chips.

Level MODH

Special attention is paid to design the security protection. The necessary equipment to perform an attack is available but it is expensive to buy and operate and it requires specific skills and knowledge. Total equipment cost exceeds 150,000 €. A group of Class II attackers with complementary skills may be required to work on the attack sequence. Examples: modern smartcard chips with advanced security protection, complex ASICs, secure FPGAs and CPLDs.

Level HIGH

All known attacks are defeated and research by a team of specialists is necessary to find a new vulnerability and devise an attack. Highly specialized equipment is necessary, some of which might have to be designed and built. Total cost of the attack could be over a million Euros. The success of the attack is uncertain. Only large organisations, like semiconductor manufacturers or government funded laboratories, could afford to breach the security of such a system. Examples: secure cryptographic modules in certification authority applications.

Common Criteria

The Common Criteria for Information Technology Security Evaluation is an international ISO/IEC standard (n. 15408) whose goal is to allow "comparability between the results of independent security evaluations" of IT products, which "may be implemented in hardware, firmware of software" [2]. It is currently in version 3.1, revision 5. This standard allows an IT product to be tested according standardized Evaluation Assurance Levels (EALs), which have a range from 1 to 7, with 7 being the highest degree of security testing (and the most expensive to obtain) [3]. The obtained score reflects the security level of a certain IT product, referred to as Target of Evaluation (TOE).

In this thesis we are mostly concerned by the AVA class of testing, namely the vulnerability assessment of the TOE, which may be more or less thorough depending on the EAL. The result of this testing model yields a certain AVA_VAN assurance level [4], which defines the TOE's robustness against specific attacks. An attack scenario can be assigned a certain score, based on its characteristics. This score can subsequently be compared with the AVA_VAN assurance level to see whether the component can resist the attack

or not. Table 1.2 shows how the attack potential is calculated, while table 1.3 shows the various results and compares them with the AVA_VAN levels.

The Common Criteria Recognition Arrangement is a document signed by 26 countries, including the United States of America, United Kingdom, Japan and Germany, which establishes that a Common Criteria evaluation up to level EAL 2 carried out by any of the signing parties will be mutually recognized by all the others without the need of any further validation.

1.2.5 Attack goals

Figure 1.1 introduced the goals of a hardware attack. They will be properly addressed in this section in order to complete the description of an attack scenario. Keeping in mind that the main drives behind a hardware attack are either gaining a military advantage, gaining money or gaining a reputation, the three grand categories of goals for a hardware attack are: *theft of information*, *disruption of services* and *privilege escalation*. In the first case the attacker might be interested in obtaining access to any sensitive information which may be later used to obtain a profit, such as identity theft allowing access to a bank account.

The other main objective of this first category is IP stealing, attainable via reverse engineering or by simpler means, if the attack occurs during the production phase. Consider for instance the case of overbuilding, namely physical chip manufacturers producing devices for a customer firm dealing solely in IP design: the extra devices could be sold to a competitor to allow IP theft, damaging the IP designer, or they may even be counterfeited and sold directly on the market. Electronic fingerprinting has been developed to prevent IP counterfeiting. Cloning of printer cartridges, an activity which has caused damages for millions of dollars, also falls inside this category.

Attacks having the disruption of a service as goal are usually used by competitors to damage the products of a chip manufacturer, but they may also happen in military scenarios. They consist in the disruption of a functionality offered by a certain device. In the hardware security world this scenario can occur en masse if the attack is carried out within the production line or if the attack is the result of a reverse engineering process with consequent realization of a malicious firmware capable of temporarily or permanently damaging the target device. This threat clearly highlights that systems need to have sound failsafe measures against unauthorized or non authentified firmware updates.

Privilege escalation attacks consist in obtaining more privileges than the ones intended by the manufacturer to use a device for unintended purposes or to run unwanted code. Well known examples include iOS jailbreaking, with the objective of installing software that is not authorized by Apple on iPhones or iPads, and cracking videogaming consoles, a topic more thoroughly addressed in appendix A. Another well known example of privilege

Factor	Value
Elapsed Time	
$\leq$ one day	0
$\leq$ one week	1
$\leq$ two weeks	2
$\leq$ one month	4
$\leq$ two months	7
$\leq$ three months	10
$\leq$ four months	13
$\leq$ five months	15
$\leq$ six months	17
$>$ six months	19
Expertise	
Layman	0
Proficient	3
Expert	6
Multiple experts	8
Knowledge of TOE	
Public	0
Restricted	3
Sensitive	7
Critical	11
Window of Opportunity	
Unnecessary / unlimited access	0
Easy	1
Moderate	4
Difficult	10
None	Attack path is unexploitable
Equipment	
Standard	0
Specialised	4
Bespoke	7
Multiple bespoke	9

Table 1.2: Common Criteria attack scenario scoring chart
Source: [4]

Values	Attack potential required to exploit scenario	Meets assurance components
0-9	Basic	-
10-13	Enhanced-Basic	AVA_VAN.1 AVA_VAN.2
14-19	Moderate	AVA_VAN.1 AVA_VAN.2 AVA_VAN.3
20-24	High	AVA_VAN.1 AVA_VAN.2 AVA_VAN.3 AVA_VAN.4
≥ 24	Beyond High	AVA_VAN.1 AVA_VAN.2 AVA_VAN.3 AVA_VAN.4 AVA_VAN.5

Table 1.3: Attack scenario score versus AVA_VAN assurance level
Source: [4]

escalation is the case of hacking or cloning of smartcards used for Pay-Tv, allowing to watch media content without having to pay a subscription. This phenomenon caused a considerable economical damage and it is was one of the major events driving companies to focus on hardware security.

1.3 Texplained

As previously stated, this thesis was realized during a 6 month internship at *Texplained*, a French company based in Sophia Antipolis founded in 2013. Texplained offers security audits on microchips and training on hardware security. Security evaluation is carried out by performing in-depth analysis over the targeted chip thanks to IC Inside Lab, a subsidiary company. Chipjuice, a software which assists the reverse engineering of a microchip, is one of the main assets of this company, as welll as an extensive knowledge of integrated circuit analysis.

The IC Inside Lab is an independent service laboratory specializing in the physical preparation, analysis and imaging of semiconductor devices and other electronic components. It allows services ranging from the simple mechanical cross-section of a PCB to a full layer-by-layer deprocessing and imaging of an integrated circuit. Its instrumentation allows chip depackaging and semiconductor delayering and imaging thanks to the usage of chemicals and specialized equipment, such as optical microscopes, a SEM (*Scanning Electron Microscope*) and deprocessing machines. Section 2.2 describes these operations

more thoroughly.

Chipjuice, currently developed by Texplained itself, is capable of reconstructing the netlist of a microchip starting from the images of the various layers. This allows to quickly obtain a description of the chip using a Hardware Description Language and then simulate its behaviour without having to resort to highly invasive techniques, such as microprobing. Thanks to Chipjuice, the reverse engineering process is considerably sped up.

The know-how which has allowed Texplained to gain more and more reputation in the hardware security market comes from decades of previous experience in both anti piracy research and imaging of semiconductor devices. This expertise is in fact the main drive behind Texplained and the development of Chipjuice as it directs the company's efforts into automatizing and improving the integrated circuit analysis process. Chapter 2 describes more in depth the methodologies employed to reverse engineer an integrated circuit, with particular emphasis to the strategies adopted by Texplained, and thus by me, during the realization of this work.

1.4 Goals

The goal of my work at Texplained, as well as the subject of this thesis, was to carry out the *analytic code extraction* of the ROM of a highly secure chip. Analytic code extraction is a relatively recent invasive attack which allows to dump the content of a ROM thanks to integrated circuit imaging, reverse engineering and simulation using a *Hardware Description Language* (HDL). The secondary goal of this work is to make an assessment on the feasibility of such an attack and on the risks posed by it. The step by step procedure consists in:

1. Depackaging, delayering and scanning the various layers of the chip using the IC Inside lab equipment and expertise;
2. Stitching together the various images to produce an overall picture of each layer;
3. Stacking the aligned images to produce a multi layer view of the ROM;
4. Carrying out the actual reverse engineering of the ROM addressing and output generation circuit by inspecting standard cells and following the connections created by metal layers;
5. Extract the bits contained within the ROM via an image processing tool;
6. Producing a VHDL description of the ROM control circuit and its content;
7. Compiling and simulating the VHDL project;

8. Extracting the ROM content by feeding the ROM addressing bus with all the possible addresses in increasing order;

While I was performing this work, my colleagues were analyzing other sections of the chip in order to identify the decryption circuit used to decrypt the data stored in the ROM, as it was assumed to be encrypted. The idea behind the overall work is to extract the binaries contained in the ROM in order to assess whether there are weaknesses allowing to devise a hardware attack. As this chip is widely used for highly secure applications, the consequences of a positive response would be serious, implying that a class II or III attacker could be able to do considerable damage in all the areas where this chip is employed.

Chapter 2

Reverse engineering of integrated circuits

Reverse engineering is the process of understanding how a certain system works. In particular, reverse engineering of integrated circuits narrows down to understanding how the various components of a system are built, which logic function they implement and what are the physical means that enable their functioning. In the security field, reverse engineering is used by pirates to find vulnerabilities, or to find ways to create some through circuit modification. Moreover, reverse engineering can be used by malicious third parties for IP infringement. In the framework of Texplained, on the other hand, reverse engineering is used to assess the robustness of a microchip against hardware attacks or to prove IP theft by inspecting the layout of suspicious devices.

This chapter describes the legislative take on this subject and the technological means allowing the reverse engineering of an integrated circuits, namely decapsulation, deprocessing and imaging through microscopy. The obtained images of the integrated circuit layout are then stitched together and analyzed. One of the goals of Texplained is to automatize the reverse engineering process. For this purpose, the company has developed Chipjuice, a software which assists the user through the whole reverse engineering process, from feature extraction (such as vias and metal connections) to netlist generation of the analyzed circuit. Since it was almost not used during the realization of this thesis, Chipjuice will not be described in depth. Moreover, due to a non disclosure agreement, the specific know-how employed by Texplained to treat the chip concerned by this work cannot be addressed. However, the principles treated in this chapter are common to any reverse engineering process targeting an integrated circuit, including the one in question.

2.1 Laws on reverse engineering in the semiconductor industry

Between the 1970s and the 1980s the development cycles in the semiconductor industry was rather standard: a large manufacturing company would produce a new device after a lengthy and expensive research process, hence the cost of a product which just entered the market was high in order to let the company get back all the money invested in research and testing. The prices would then decrease thanks to process improvements in the manufacturing of a certain chip and also due to the fact that secondary firms would be capable of producing equivalent chips and sell them at a smaller price. The pioneering firms had to come up with a new, innovative product to survive. The modern market life cycle of a chip has more or less remained unchanged, although specific aspects have evolved, such as improvements in the manufacturing process, increase in the manufacturing costs (related both to recurring and non recurring engineering) and the rising threat posed by piracy and IP infringement in particular, since it consents to bypass all the non recurring engineering costs.

At the beginning, patents were not used to protect proprietary designs. Leading manufacturing companies could easily survive on the market since they were the only ones to possess the necessary know-how to have a large yield in the production of a certain chip. The layout, on the other hand, especially at the beginning, was not only not protected by law, but was also impossible to keep secret, as components were large enough to be clearly visible on a PCB. Many reasons contributed to the fact that patents were not used, including a general distrust held by the American courts toward patents and also the fact that the American government sought to keep the supply of any semiconductor appliance as available as possible, hence the last thing it wanted was to discourage secondary companies from producing.

The situation changed when the life cycle of chips started to be severely disrupted by increasing manufacturing costs, and also since what used to be trade secrets became more and more widespread, hence even secondary companies could ensure good yields while not having to spend as much as the leading ones in research and development processes. Therefore, companies such as Intel began looking for help from a legal perspective. After a long series of ineffective attempts, which mostly tried to adapt existing patent legislation to fit the needs of the semiconductor industry, the Congress resolved to draw a sui generis legislation to protect semiconductor chip design. In 1984 the *Semiconductor Chip Protection Act (SCPA)* was developed. In particular, the reverse engineering provision states:

> It is not an infringement of the exclusive rights of the owner of a mask work for -

> (1) a person to reproduce the mask work solely for the purpose of teaching, analyzing, or evaluating the concepts or techniques embodied in the mask work or the circuitry, logic flow, or organization of components used in the mask work; or
>
> (2) a person who performs the analysis or evaluation described in paragraph (1) to incorporate the results of such conduct in an original mask work which is made to be distributed.

In a nutshell, the law allows reverse engineering if it is intended to foster the semiconductor manufacturing techniques and/or if it leads to a new and improved design. The SCPA recognizes reverse engineering as a mean through which the semiconductor industry can improve the current state of the art. Another goal of the lawmakers was to prevent the existence of a monopoly in the semiconductor industry by enabling constructive competition. On the other hand, the SCPA made it illegal to produce carbon copies of the reversed chips, attaching the forward engineering requirement to allow the printing of chips based on the analysis of proprietary ones. Although the efficacy of the SCPA is debated, it surely made it harder to copy any digital circuit layout, since forward engineering implies a considerable cost, hence reverse engineering with the goal of IP infringement became economically unfeasible, if the pirates wanted to remain within the boundaries of the law.

After the approval of the SCPA, many aspects of the semiconductor market changed. Nowadays, reverse engineering by itself has become much harder than it used to be due to process improvement leading to the shrinking of chips size (i.e. due to Moore's law), implying the need of more expensive machines and more time necessary. Moreover, nowadays the manufacturing process is not centralized any longer but it is split among the various companies: some take care of designing the chip, some others take care of the physical fabrication. This lead to a considerable change also in the way manufacturers seek the support of the law to be able to earn a profit. For instance, since some companies base their profit only on the design of the chip, they adopt an aggressive patenting strategy, remarking the importance of the SCPA, although detractors point out its lack of effectiveness and obsolescence.

Moreover, the SCPA influenced an agreement approved by the World Trade Organization in 1994, the *Agreement on Trade-Related Aspects of Intellectual Property Rights (TRIPS)*. This agreement includes a section concerning the intellectual property represented by the layout of integrated circuits (article 35) and regards reverse engineering in a manner which was influenced by the SCPA. The legislation of the European Union, which, in a sense, is more closely connected to this thesis, covers the topic of integrated circuits layout with the *Council Directive 87/54/EEC of 16 December 1986 on the legal protection of topographies of semiconductor products*. In particular it says:

> The topography of a semiconductor product shall be protected in so far as it satisfies the conditions that it is the result of its creator's own intellectual effort and is not commonplace in the semiconductor industry. Where the topography of a semiconductor product consists of elements that are commonplace in the semiconductor industry, it shall be protected only to the extent that the combination of such elements, taken as a whole, fulfils the abovementioned conditions.

Although this directive was emanated not that far with respect to the SCPA it is noticeable that European legislators followed their own approach in defining the laws protecting the IP of semiconductors. Moreover, no mention is being made with respect to reverse engineering and in general the norm is never specific regarding the means through which the IP defined by an integrated circuit may be stolen. This in a sense may have been the consequence of the fact that the European Union was less pressed by the matter of semiconductor cloning and IP infringement because it held a smaller interest in that market with respect to the USA. On top of that, European directives are then applied in each state by the emanation of a national law. These two aspects make so that, in general, laws concerning this subject in Europe tend to be heterogeneous and, thus, possibly more vague and less effective.

In the light of all these remarks, the services provided by Texplained are not backed by a clear-cut legislation. First of all, besides the TRIPS, there is no homogeneity in the way international laws regard the protection of integrated circuits. Moreover, besides the safeguard of intellectual property, there are no laws concerning reverse engineering as a mean to break the security of an integrated circuit or as a mean to inspect its resilience against hardware attacks. The bottom line of this analysis, as showed by the history of the SCPA, is that laws are very much influenced by changes in the economy. As of now, security is just beginning to be considered an important aspect in the semiconductor industry, as it may cause the loss of huge capitals. The expectation is that governments will soon start to follow the concerns of the industry and will begin to direct their focus on legislation concerning hardware security - possibly integrating previous laws ruling software security - and on the usage of reverse engineering. Please notice that the contents of this section have been largely extracted from the book *The Law and Economics of Reverse Engineering* [16].

2.2 Failure Analysis techniques for the reverse engineering of integrated circuits

We may consider the core of the reverse engineering process as the observation and analysis of the layout of an integrated circuit in order to understand how it works. As a consequence, it is necessary to obtain the pictures of the integrated circuit's layout. This

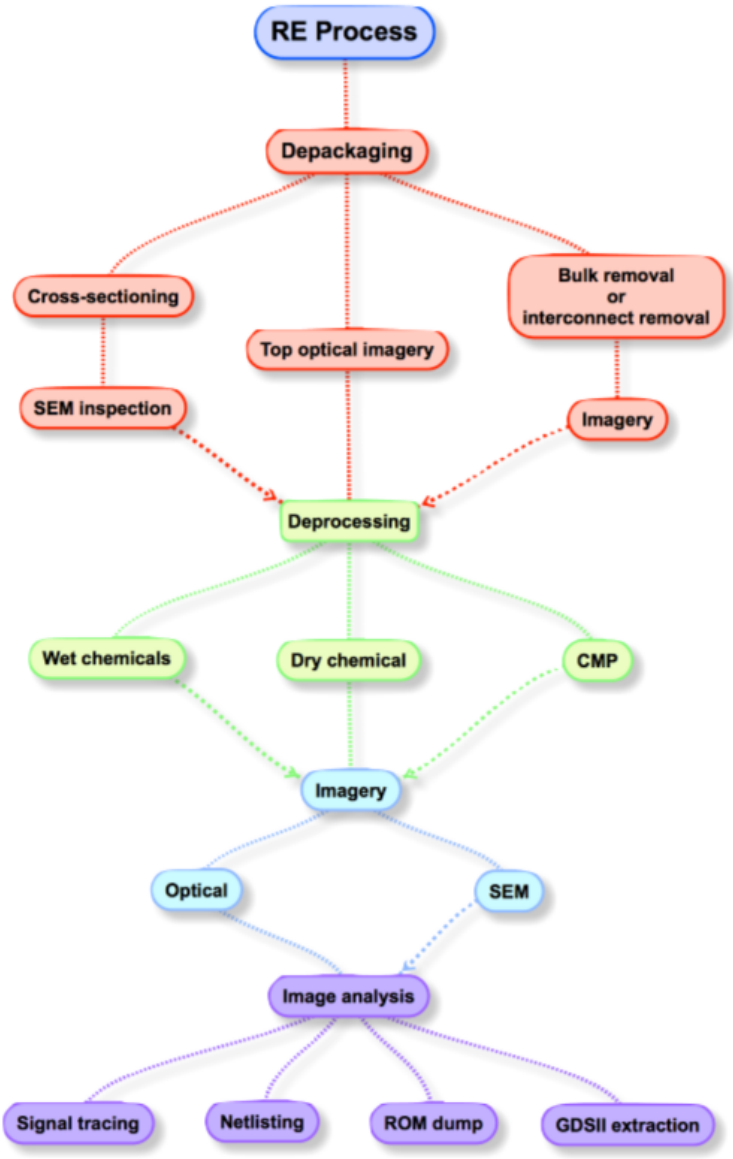


Figure 2.1: Reverse engineering roadmap
Source: Texplained

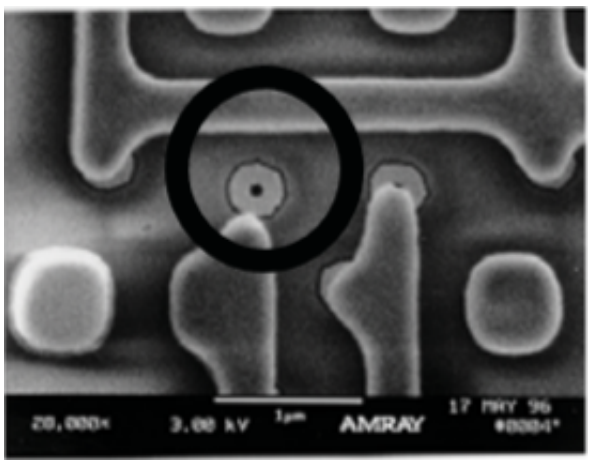


Figure 2.2: Example of failure analysis: missing metal 1 pattern causing a missing connection

Source: Intel Corporation

task is complex not only due to the submicron size of the concerned components but also since modern integrated circuits are constituted by different layers, made of different material and having different purposes. The bottom layer contains the standard cells used to implement logic functions or analog tasks, while the other layers contain the metal wires connecting the cells contained in the first layer and distributing the connections to the supply voltage and the ground.

Luckily, the expertise and technologies employed in the domain of failure analysis of integrated circuits provide all the necessary tools to obtain a layout. The real goal of failure analysis, as the name implies, is to investigate the causes of any any malfunctioning ensuing in a finished microchip. Failure analysis engineers inspect the surface of an integrated circuit to identify these causes, for instance a missing metal connection (cf. fig. 2.2) or, vice versa, an unwanted one. For this purpose, this branch has devised all sorts of mechanical, chemical and chemical and mechanical strategies to access the various layers of the chip and obtain high resolution images. It goes without saying that the exact same techniques may be used in a reverse engineering context, albeit at times in a less orthodox way.

Once the chip has been successfully deprocessed and the pictures of the various layers have been obtained, the next steps consist in putting them together and translating their features into a usable format, such as a HDL or GDSII (a binary format used to represent physical IC layouts that is the current industry standard). Clearly, the last two steps must be executed in software to be accomplished in a reasonable amount of time. Figure 2.1 sums up the process followed during any IC reverse engineering task, including procedures normally employed in failure analysis. The following sections describe all these stages.

The main references for the treated subjects are [1] and the Texplained training material.

2.2.1 Depackaging

The depackaging phase consists in removing the case containing the integrated circuit. Packaging come in different materials. In this section only plastic packaging removal is going to be treated, since it is the most common type for integrated circuits. The most commonly used plastics are amino-hardened novolac (i.e. low stress molding materials) epoxides. The conventional way to tear a hole into this type of package is through fuming nitric acid, sometimes mixed with 10% concentrated sulfuric acid. It goes without saying that the application of such an acid is to be performed under the strictest safety measures, as it can be extremely harmful to the human skin. Chemical splash safety goggles must be worn at all times while applying these acids. Also, such an action must always be performed inside a fume hood, possibly with an active exhaust fan.

There are various techniques to tear a hole locally on a package, all of which involve special care both because the involved acid is very dangerous for the operator and also since it might dissolve more than the package alone, e.g. attacking the aluminium on the surface of the IC. Since the goal is to perform a visual inspection, however, the package is most commonly removed entirely through dry chemical processes. This operation is simpler with respect to a local opening, since the action of the employed acid does not have to be focused on a specific area. Fuming nitric acid is employed in this case as well. It is first heated in a half filled covered glass beaker (containing up to 100 ml), for instance employing an aluminium container placed over a stainless steel collecting bath. Such a bath is designed to collect any acid which, boiling over, flows out of the beaker. The IC is immersed in the acid, having a stable temperature of 60±5 deg C, using e.g. a small stainless steel basket. The reaction time depends on various factors, such as acid temperature and package thickness, but it usually amounts to 3 to 7 minutes.

The exposed IC is then rinsed and dried in order to halt the acid's action. It is first sprayed with acetone, then immersed in a stainless steel basket containing fresh acetone, then in deionized water and lastly in acetone. Each immersion lasts from 5 to 10 seconds. Eventual residues may be removed through ultrasonic cleaning, which consists in immersing the IC in a certain solution (deionized water or fresh acetone, depending on the residue to be removed) and using an apparatus which agitates the liquid by emitting frequencies between 20 and 400 kHz for around 20 seconds. Drying is executed using a light stream of warm air to avoid the risk of blowing the chip away. Eventual remainders of material may be removed using tweezers, with the aid of a stereo microscope.

After decapsulation it is likely to encounter a plastic coating covering the chip. The purpose of the coating is to protect the chip surface from mechanical or chemical damages and from α rays, capable of flipping memory bits. Since the state of hardening of the

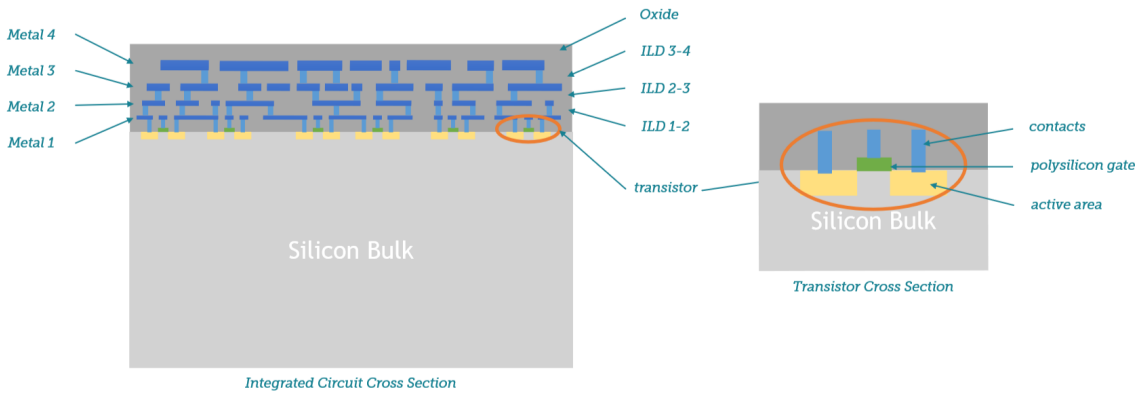


Figure 2.3: Example cross section of an integrated circuit
Source: Texplained

employed material influences the process to be followed, there is no definite procedure to comply with in this case. The normal procedures consist of chemical destruction, soaking away using a solvent or even using ultrasound in the most hardened instances. The aftertreatment is identical to the one employed after depackaging proper. The first depackaged chip of a certain kind is then "sacrificed" to obtain more information on the following steps. It is polished sideways using diamond abrasive plates, with the goal of exposing the side section of the chip, which is then imaged using a SEM.

Figure 2.3 shows a possible result of this operation, a cross section showing 4 metal layers. Notice that ILD stands for Inter Layer Dielectric, namely the insulating material which separates two adjacent layers. By seeing the cross section of an IC it is possible to identify the number of metal levels and their characteristics, such as material and thickness. This allows to plan ahead the following delayering operations, which depend on the characteristics of the considered layer. The following goal, addressed in section 2.2.2, is to remove the surface of the chip until a layer is exposed in order to image it. The process is repeated until all the layers have been imaged.

2.2.2 Deprocessing

Deprocessing consists in selectively removing materials on the surface of a chip to expose the layer below. The main risk during this phase is that the procedure either does not expose entirely the targeted layer (under etch) or is too aggressive and causes the unwanted visibility of one or more additional layers below (over etch). There are different techniques employed during this phase, which are used also during the manufacture of the IC: wet chemical etching, dry chemical etching and Chemical Mechanical Polishing (CMP). Each of these techniques has its pros and cons and is more or less suitable depending on the physical composition of the targeted layer. Moreover, since each IC has its own set of

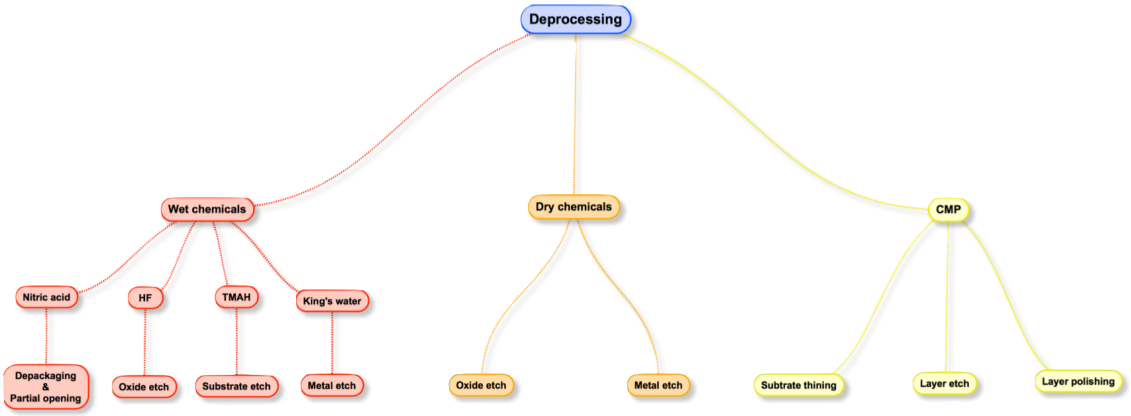


Figure 2.4: Deprocessing techniques
Source: Texplained

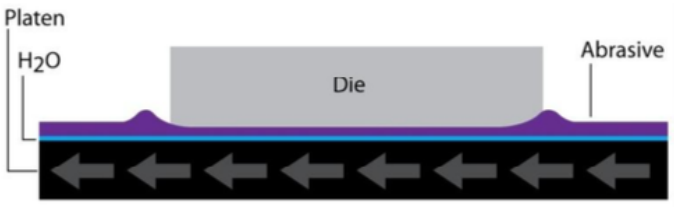
characteristics, there is no precise methodology to be followed. The common case scenario consists in using a combination of the available techniques and having to go through several trial and error steps, meaning that some samples may have to be sacrificed to test a deprocessing technique on a certain layer.

After sample depackaging, the passivation layer and the metal layer below can be easily removed using the wet and dry chemical etching processes described in section 2.2.1. The problem is that, following to these processes, the structural heterogeneity of the the metal layer will cause very deep patterns in the ILD below. This is true for any of the following metal layers, and the resulting surface profile is very hard to further deprocess using only chemical methods. For this reason, the general consensus is that the best approach to deprocess ICs is through purely mechanical techniques [19] or through Chemical Mechanical Polishing (CMP).

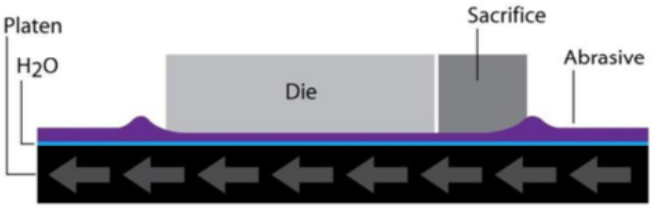
Initially, the backside of the chip is polished to make the front side as planar as possible, since the goal is to obtain a 2-D image of each layer. The basic idea is to glue the sample on a carrier which is used to hold it still while a diamond abrasive plate removes the desired metal or ILD layer. There exist different techniques to glue the sample to the carrier, such as using clear wax heated at approximately 130 °C, and to perform the mechanical abrasion. In the latter, in particular, the thickness of the diamond plate determines the resolution of the process. The main issue of mechanical techniques, however, is that they are hardly reproducible if they are performed manually, in which case a high degree of expertise is also required. This is due to the fact that each IC is manufactured in a different fashion, hence each one needs a different approach for successful delayering. Therefore, an automated machine such as the one in figure 2.5 is rather used for this purpose.

The advantage of automated mechanical deprocessing is that the various parameters allowing an optimal outcome, such as sample tilt angle and plate speed, can be applied

Figure 2.5: Sagitta station, integrating a CMP machine and an optical microscope
Source: Texplained



(a) Over etching of the leading edge is mostly caused by the bowed structure of the abrasive plate



(b) A sacrificial silicon layer can be used to protect the sample from unwanted edge effects

Figure 2.6: An effective method to protect an IC from unwanted edge effects during a mechanical delayering process

Source: [19]

using a dedicated software. Moreover, many of these machines include a microscope which aids the Failure Analysis engineer during the sample inspection in order to choose the optimal polishing parameters with a high precision. Thanks to such an instrument it is possible to easily obtain samples which can be successfully imaged. Another issue posed by mechanical delayering techniques is that the leading edge of the sample is more exposed to the abrasive plate, causing unwanted edge effects to ensue. The solution proposed by [19] is to add a sacrificial layer of silicium in front of the leading edge of the sample. The sacrificial layer will sustain the impact of the bowed abrasive, thus protecting the leading edge of the sample from unwanted over etching (figure 2.6).

2.2.3 Imaging

Optical microscopes may be used during process review, for instance coupled with clippers in order to remove eventual residual material after sample depackaging, or in order to obtain an overview of a IC, allowing to characterize its structure. Figure 2.7, for instance, shows the substrate of a microcontroller produced by Atmel. Thanks to pictures such as this one it is possible to identify and coarsely classify the cores, the memories and the analog components. However, Scanning Electron Microscopes (SEM) represent the current standard technology as far as IC reverse engineering is concerned. Images produced with an optical microscope do not have a sufficient resolution to appreciate standard cells, connections and vias. A SEM emits a highly focused electron beam over a surface and

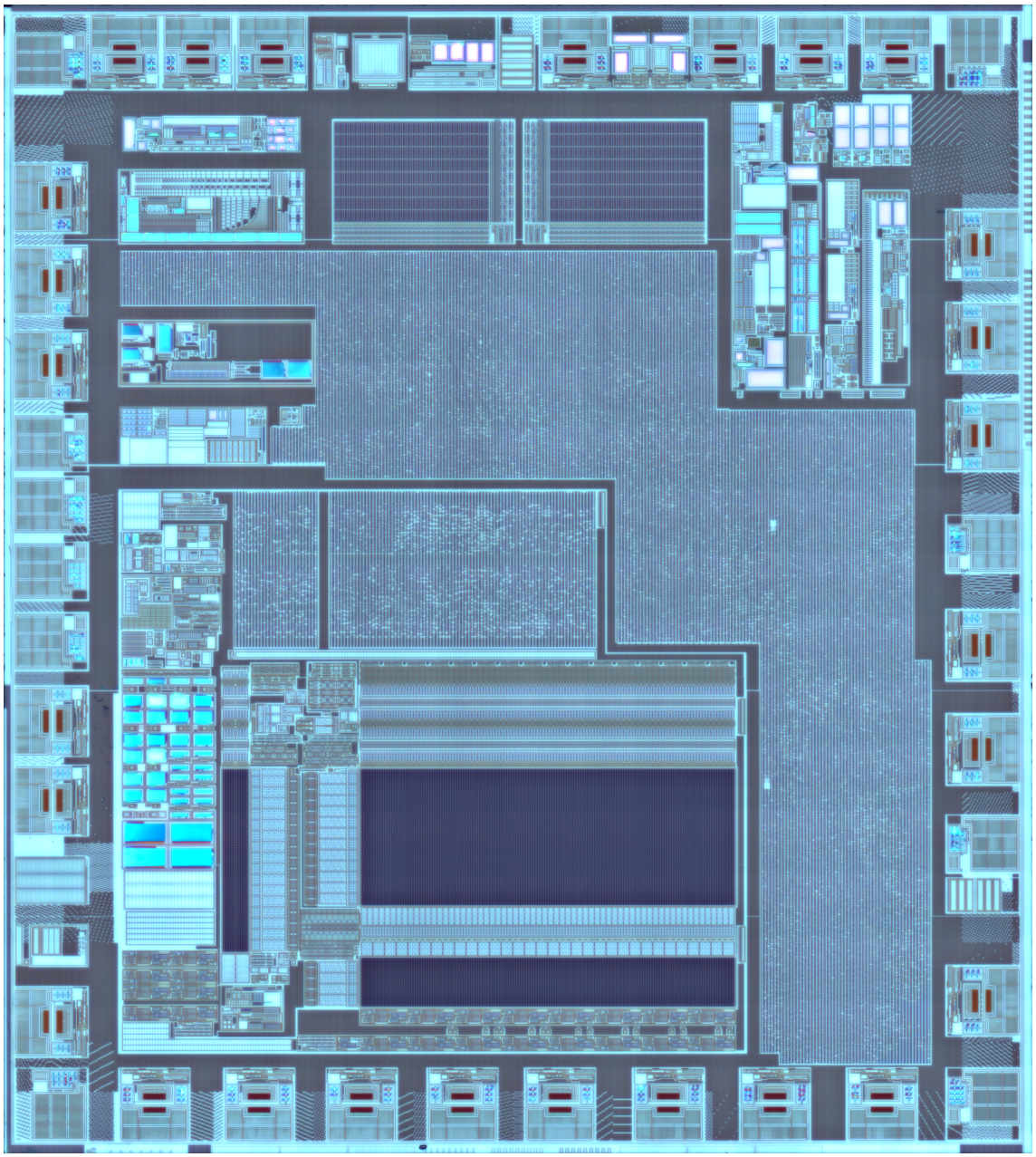


Figure 2.7: Optical image of the substrate of a AVR ATMEGA48PB-MU microcontroller
Source: Texplained

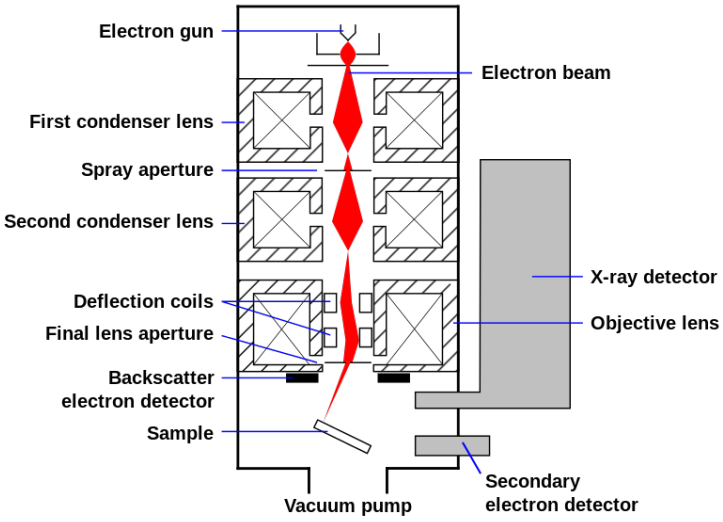


Figure 2.8: Structure of a SEM
Source: Wikipedia

then captures the reflected electrons. These electrons generate a set of signals which allow to rebuild the shape of the surface they collided with. Without going into excessive detail, the resolution of an imaging instrument is directly proportional to the wavelength of the medium used to acquire a certain image. The reason why SEMs allow much better resolution than optical microscopes is that electrons have a much shorter wavelength than white light, which is the medium through which an optical microscope obtains an image.

A SEM is made of an electron gun, a column containing electromagnetic lenses, an electron detector and a sample chamber. The electron gun is used to fire a beam of electrons which is highly focused by the electromagnetic lenses along its path. The electron beam is directed toward the surface of the sample, placed on a stage inside the sample chamber. The interaction between the electron beam and the sample's surface causes the emission of secondary electrons. Part of these electrons are captured by the electron detector, which uses them as a signal to reconstruct a digital image of the area where the focused beam hit the sample. By redirecting the beam it is possible to image the whole surface of a sample. During operation, the column and the sample chamber of a SEM are kept in a vacuum by a combination of pumps. The most obvious reason is that eventual particles would prevent the electrons from reaching the sample's surface, thus hindering the imaging process. Moreover, since a tungsten filament is commonly used to produce the electron beam, the absence of oxygen molecules prevents it from burning out when it reaches the operational temperature.

In order to allow proper imaging, samples are usually prepared before imaging by covering them with a conductive coating, using materials such as chromium, gold or graphite.

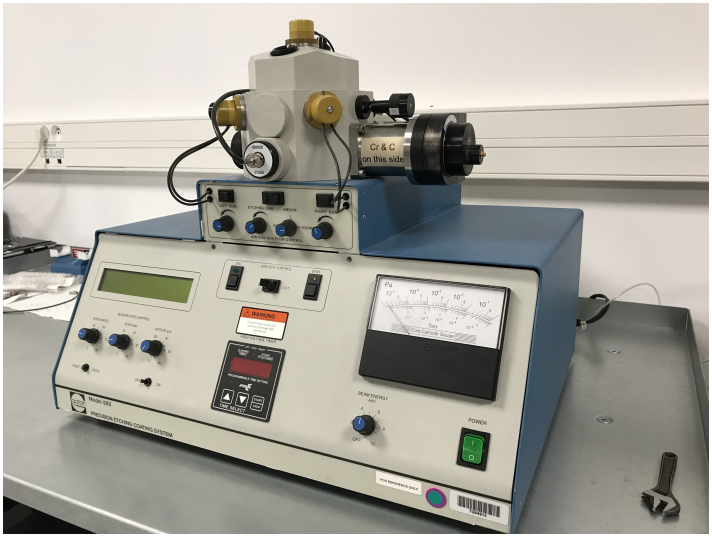
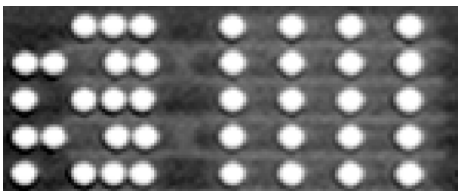


Figure 2.9: Gatan PECS sample coating and etching system
Source: Texplained

The reason is that the surface of an insulator or a semiconductor collects charge when it is hit by the electron beam, thus creating noticeable artifacts. The coating is carried out by employing a Precision Etching and Coating System (PECS) such as the one in figure 2.9, which employs three ion guns and a vacuum chamber to deposit the selected material on the the sample's surface. During the imaging process, a computer equipped with a dedicated software handles the control of the SEM and the storage of the images. Since it is impractical to take a single, high resolution picture showing the whole surface, many pictures are taken for each layer. Luckily, SEMs and their operating software allow to take sets of pictures which have a fixed degree of overlap, thus easing the successive image stitching process.

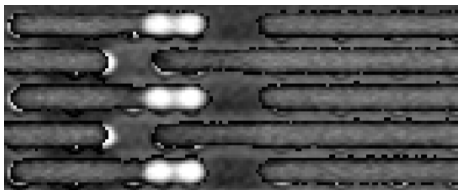
2.2.4 Analysis

The proper IC reverse engineering process begins after all the necessary images have been obtained, stitched together and juxtaposed. The pictures need to cover the entirety of the region to be reverse engineered and they must cover all the layers containing metal wires used to route inputs, outputs and control signals. The topmost layers are usually not necessary, since they usually distribute the power supply and the ground alone. For space optimization, connections between standard cells are carried out across multiple metal layers, with vias used to establish a connection between two metal wires lying on two successive layers. For this reason, the pictures of the various layers need to be juxtaposed to allow to follow a connection from a pin to another. Figure 2.10 shows the juxtaposition of two layers for a small chunk of the layout as an example.



(a) Polysilicon layer chunk

(b) Metal 1 layer chunk



(c) Resulting image

Figure 2.10: Example of multilayer superposition with Photoshop.

When all the multilayer pictures are ready, the standard cells can be identified and reversed, as well as the connections between them. Due to the size of the aforementioned task, it is at the very least necessary to use an appropriate tool, such as Chipjuice, Texplained's proprietary software, which allows to identify vias, wires and standard cells from each picture, put together data coming from the various layers and produce a VHDL description of the analyzed IC. Image processing software such as Photoshop can be used to produce annotated multilayer views, i.e. highlighting standard cell pins and control signals across multiple layers. Consider figure 2.11 as an example. The figure shows a standard cell. The input and output pins are named using letters and the red highlights represent overlying metal 1 wires connected to the power supply. Similarly, the blue highlights represent overlying metal 1 wires connected to ground. Lastly, the orange highlights represent any other connection.

Usually, the polysilicon layer and the metal layer provide sufficient information to reverse engineer a standard cell. The outer metal layers are used to route nets across significant distances, for instance the wordline control signals or the bits reaching the output of the ROM. Concerning the connections to the power supply or to ground, the rule of thumb to identify them is to individuate a set of p-MOS and n-MOS, easily distinguishable thanks to the size difference, as the area of a p-MOS is usually larger in a CMOS configuration to compensate the reduced mobility of the majority carriers and thus equalize as much as possible the rising and falling times of the pull-up and pull-down networks. Keeping this in mind, it suffices to identify multiple p-MOS whose drain is connected to the same network to identify a connection to the power supply. Analogously, multiple n-MOS having the source connected to the same net allow to identify a connection to ground. Moreover, wires distributing power supply and ground tend to be larger than the

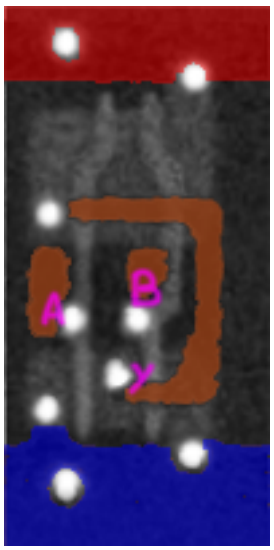


Figure 2.11: Example of annotations realized with Photoshop

other ones, since they must have a very low resistance. Notice that wires distributing the power supply and the connection to ground are usually immediately recognizable in the core of a microprocessor, since they usually run across the IC, separated by "columns" of standard cells.

The reverse engineering process is completed when all the necessary standard cells have been reversed and all the connections between them have been identified. At this point the circuit, or part of it, can be simulated through a hardware description language and a simulator. It is also possible to derive a description of the layout of the IC in GDSII. What comes next clearly depends on the goal of the reverse engineer, be it inspecting searching for a vulnerability or studying the circuit for forward engineering purposes. The goal of this thesis, as stated in section 1.4, is to obtain the code contained in the ROM of a secure chip, hence the reverse engineering process concerns all the standard cells and connections which determine the translation of the addressing bits into an output.

Reverse engineering of standard cells

An essential part of the reverse engineering process of a IC is the understanding of the logic function implemented by a standard cell. In this section the reverse engineering of a simple standard cell will be addressed to show the basic methodology. Figure 2.11 shows two superimposed layers, polysilicon and metal 1. The picture contains a simple standard cell, with added colors and letters used to annotate it (cf. section 3.1). For confidentiality reasons, the employed image does not come from the reverse engineered ROM but was taken from Texplained's training material.

The white dots are vias, connecting the two layers. The darker areas represent an

insulating SiO_2 substrate, while the less dark zones represent doped active areas. Finally, the lighter stripes are polysilicon lines. A polysilicon line separating an active zone in two parts constitutes a transistor. The pictures do not allow to appreciate the difference between a p-doped or an n-doped active area, so in principle it is not possible to tell a p-MOS from a n-MOS. However, consider that habitually p-MOS have a larger size due to the reduced mobility of their majority carriers. This fact comes to our help, since we may clearly see that the upper transistors are larger. Therefore, unless, for instance, some form of layout obfuscation is in place, they are certainly of the p-MOS type. This fact alone gives us plenty of additional information, since any long wire connecting the drain of multiple p-MOS is surely a connection to the supply voltage. Conversely, a wire connecting multiple n-MOS sources represents a connection to the ground.

The top and the bottom of figure 2.11 represent a part of these connections, which lie on metal 1. The other wires, colored in orange, also lie on metal 1. The first step to carry out the actual reverse engineering is to identify inputs and outputs. The inputs are clearly the two vias lying on top of the two polysilicon lines, since in CMOS technology the inputs always enter the gate of a transistor. In the picture they are labeled as A and B. Since the vias at the border of the image are clearly used to connect the transistors either to the supply voltage or to ground, the output must go through the two remaining vias, which are connected through the C-shaped metal connection. The output has been labeled as Y.

Let us follow A first. It enters a p-MOS (top left) and a n-MOS (bottom left). The bottom n-MOS has the source connected to ground (left with respect to the polysilicon stripe) and the drain connected to the output (right with respect to the polysilicon stripe). The top p-MOS has the source connected to the output and the drain apparently connected to nothing. In fact, the drain is realized by the same doped region which constitutes the source of the rightmost p-MOS, whose input is B. Therefore, what we are seeing are two p-MOS connected in series.

Following B, we see that it enters a n-MOS (bottom right) connected to ground and to the output Y, just like the other n-MOS. B also enters the just described p-MOS (top right), whose drain is connected to the supply voltage. By drawing the described netlist we obtain the picture in figure 2.12, clearly a NOR gate. Even if this cell is very simple with respect to others encountered in the ROM, such as sequential logic cells or more complex combinatorial logic gates, the employed methodology is always the same: identify the input and output pins, follow the inputs along the polysilicon connections and draw the transistors encountered along the path, connecting their sources and drains according to the identified connections.

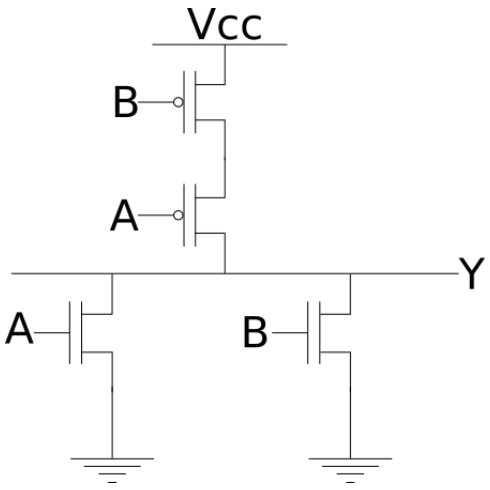


Figure 2.12: NOR gate contained in 2.11

Chapter 3

ROM reverse engineering

This chapter is the core of the thesis, as it describes what I actually have done to carry out the reverse engineering of the ROM and the extraction of its memory content. Section 3.1 describes the pictures obtained via SEM and used to perform the reverse engineering of the ROM. Section 3.2 describes the ROM functioning according to my analysis, namely the reverse engineering of the standard cells and the observation of the metal connections. Section 3.3 briefly describes how it was possible to extract the bit content of the ROM.

Chapter 4 finally presents the simulation of the ROM, showing how an address is translated into the selection of specific bits and how the output of the ROM is actually produced. As a reminder, the overall procedure through which it is possible to extract the entire content of a ROM by simulating its behavior is called analytic code extraction. This is an evolution of the *linear code extraction* technique, which consists in performing invasive probing, feeding the memory with all the possible addresses and storing the output of the ROM using additional probes. Clearly, the second technique is much more complicated, expensive and time consuming. The simulation can be performed using any programming language or HDL and a simulation environment, such as ModelSim. The latter possibility was chosen, using VHDL to describe the ROM architecture.

3.1 SEM pictures and ROM layers

The following layers of the ROM were obtained via SEM scans:

- Polysilicon;
- Metal 1;
- Metal 2;
- Metal 3;

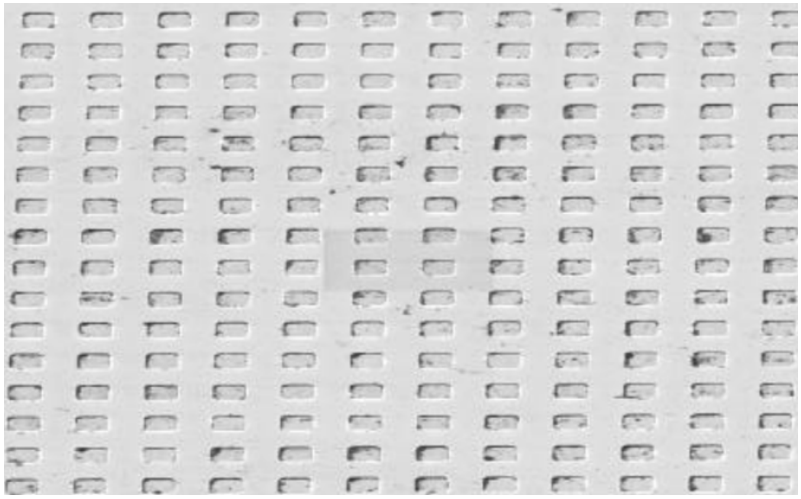


Figure 3.1: Example of SEM image with skew
Source: [8]

- Metal 4;

The Polysilicon layer contains diffusions, active areas and, clearly, polysilicon stripes, namely the actual digital standard cells realized in CMOS technology plus any other electronic component, such as capacitors or analog circuits. The other layers serve no other purpose than establishing connections between the various components and the links to ground and to the power supply. The remaining metal layers concerned only power supply lines, hence it was not necessary to employ the relative scans to derive useful connections. The pictures provided by the SEM have a small but noticeable skew due to errors ensuing during perspective projection, caused by a variety of sources (i.e. scan generator error and external magnetic fields, cf. [8]). It was thus necessary to deskew the images before using them.

Luckily, Photoshop, which was extensively used during the reverse engineering process, allows batch processing of multiple images, hence, since the skew introduced by the SEM is constant, the same deskewing action was carried out in batch for all the pictures. In order to have an appropriate resolution, the various areas were not scanned in their entirety: they were separated into many different pictures having a 20% overlap with respect to the neighboring ones. The individual pictures had to be stitched together to realize the view of an entire layer. This was done via Chipjuice, which includes a tool capable of doing so.

With all the layers ready it was possible to produce .tif files showing a multilayer view of the ROM. Due to pictures misalignment and errors introduced by the stitching procedure, and also in order to avoid the creation of a very heavy file, the ROM was split into several areas. This greatly simplified the layer alignment process and the realization of multilayer views of the ROM. For this purpose, layers had to be resized in order to

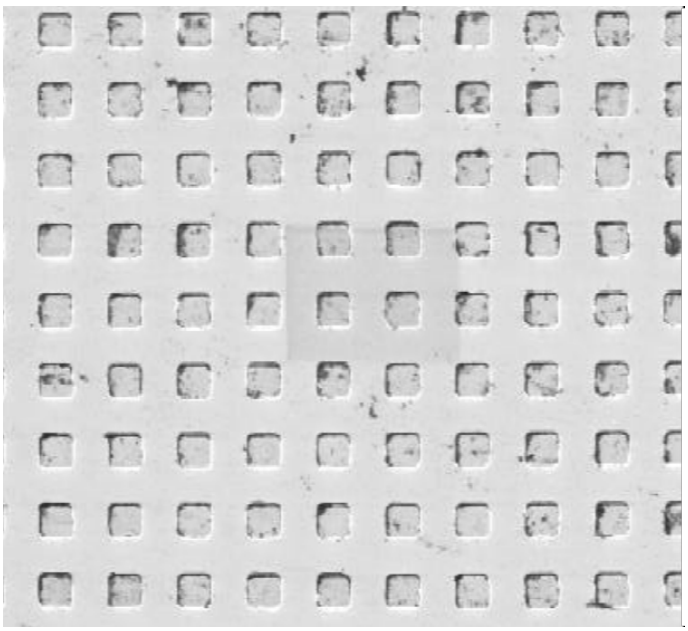


Figure 3.2: Example of deskewed SEM image
Source: [8]

properly align them, i.e. in order to place vias and metal connections over the correct zones. Photoshop allows to modify the opacity of each layer, hence it was possible to view the layer below when placing and resizing the one above.

Even when all the multilayer views of the various ROM sections were ready there still were several issues to tackle. The vias on metal 2 were only slightly visible or not visible at all due to an imperfect delayering process, hence sometimes they had to be guessed by observing the layers above and below. The repetition of a pattern in the layout allowed to make a guess on where a via might be, even when it was not actually visible. The pictures of metal 3 were of poor quality in many areas due to delayering and imaging issues, making it harder to properly understand connections and vias placement. Lastly, the polysilicon layer did not always allow to tell the difference between actual polysilicon and underlying active areas. Although most of the times these two areas are easily distinguishable, this was not always the case, adding the need to guess the layout of a standard cell according to rules of thumb, e.g. the type of the surrounding cells and the neighboring connections.

The pictures showing the actual ROM bits were not stitched together, they were just deskewed. Since the bit value is given by the absence or a presence of a metal connection, it was simply necessary to process the layer which contained these connections (or lack thereof). In this case, as in any ordinary ROM, the concerned layer is metal 1. Each picture was processed using a Jython script exploiting FIJI libraries. FIJI is an image processing package derived from ImageJ, while Jython is a blend of Python and Java.

Bank	Stripes	Wordlines	Bits	Memory locations
0, 3	16	128	278528	8192
1, 2	32	256	557056	16384

Table 3.1: Size of the 4 banks

This script allowed to extract the bit content of each picture by analyzing the feature of each image along a bitline. Additional scripts were used to stitch the obtained bits together and output several .txt files containing all the bit content of the ROM in the proper order. These aspects are more thoroughly addressed in section 3.3.

3.2 ROM architecture

The reverse engineering process began by extracting the main features of the ROM from the multilayer views described in section 3.1. Protruding wires in the upper section of metal 3 showed that the ROM has 18 inputs for addressing, making the addressing space in fact equal to $2^{18} = 262144$ memory locations. The output bits amount to 34. A picture of the entire metal 1 layer showed the presence of 4 memory banks of different sizes, as there are four homogeneous zones separated by a "skeleton" having a different and homogeneous pattern. It was assumed (correctly) that this inner circuitry performed address decoding and distributed control signals to perform bit selection.

It was also noticed, thanks to the presence of repeating patters inside each bank, that the wordlines are grouped in several stripes, 8 wordlines per stripe. It was later established that a wordline cannot be enabled unless the relative stripe is enabled as well. The ROM is clearly of type NAND, therefore the expectation was that wordlines would be enabled by pulling them down, while the remaining wordlines belonging to the same bank had to be driven high. The total number of bitlines, easily visible on metal 4, amounts to 2176, hence 2176 bitlines are selected when a wordline is selected. Out of the 2176 wordlines only 34 are connected to the output thanks to two selection stages. The first one selects one bitline out of 8 and has 272 outputs. The second stage accepts as inputs the bitlines which have been selected during the first stage and outputs 34 bits, which are connected to the ROM output bus, individuated by observing protruding wires along the upper left and upper right parts of the ROM on metal 2. Table 3.1 sums up the structure of the ROM in terms of size.

The size of the 4 banks adds up to 104.448 kB, which is therefore the total size of the ROM, while the used addressing space amounts to 26112 memory locations, 9.96% of the total addressing space. It was also established that certain standard cells exploit dynamic logic, with precharge and evaluation phases. Thanks to the control signals, the

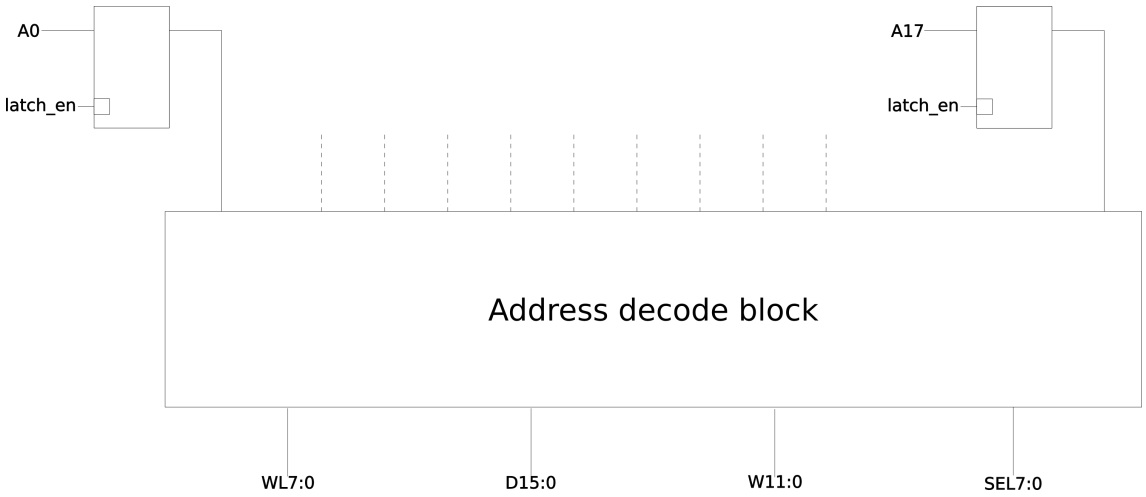


Figure 3.3: ROM netlist, address store and decode section

evaluation phase is enabled only for one bank at a time, hence it is impossible to select multiple banks. In order to tackle the reverse engineering of the ROM addressing stages, the overall area was divided into several parts. The following sections describe the result of the reverse engineering of each section.

3.2.1 First address decoding stage

The first stage, present in the middle section of the topmost part of the ROM, turns the input address bits into signals which are used to drive the stripe selection and the wordline selection (cf. section 3.2.3). Moreover, it generates signals later used to select a single bank and the output bits out of the various bitlines (cf. section 3.2.2).

The address bits are stored into latches, enabled by the *latch_en* signal. The outputs of the latches enter the decoding block, whose contents will not be described for confidentiality reasons (notice that this is not a huge loss, as it is made of rather basic logic gates). The output control signals exiting the decoding block have been named W, WL, D and SEL. They are 44 in total:

- The 12 W signals enable a single stripe for each bank. It was established that each stripe is selected when two W signals are asserted (cf. section 3.2.3). Clearly, each stripe accepts two different W signals as enable;
- Due to a repeating connection pattern, each of the 8 WL signals enables multiple wordlines, irrespective of the stripe. However, as previously stated, a wordline is enabled only if the respective stripe is enabled, hence only one wordline is actually enabled at a time;

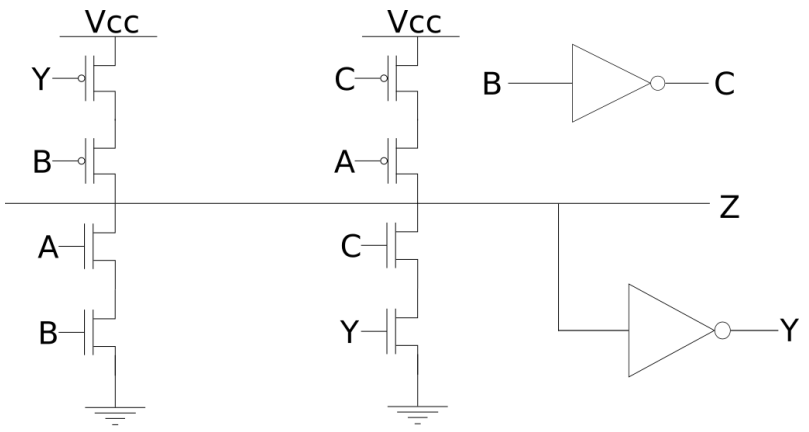


Figure 3.4: *latch_mux*

- The 16 *D* signals are further decoded and have various uses. They are described in the following section;
- The 8 *SEL* signals are further decoded and their usage will be described in the following section;

Figure 3.3 represents this section. The latches are made using the *latch_mux* cell (figure 3.4). This cell is basically a multiplexer converted into a latch by inverting its output and connecting it to one leg of the pull-up and pull-down networks. The *latch_en* signal is connected to *B*, while the input is connected to *B*. Since *C* is *B* inverted, the value of *B* determines whether the output is influenced by either the input (connected to *A*) or the current output *Z*. The output is influenced by *A* if *B* is high, hence the enable is of the active high kind.

3.2.2 Second address decoding stage

This stage is present twice within the ROM. Each instance controls two banks and the relative bitlines, possibly connecting a subset of them to the actual outputs of the ROM. The two banks controlled by each instance will be abstractedly referred to as the upper bank and the lower bank. Part of the signals described in section 3.2.1 are further decoded here. The cell named *ctrl_cell* (figures 3.8 and 3.9) accepts as inputs *ctrl_D* and three *D* signals:

- *D4*, *D5* and *D8* for the first instance (figure 3.5), controlling bank 0 (upper) and bank 1 (lower);

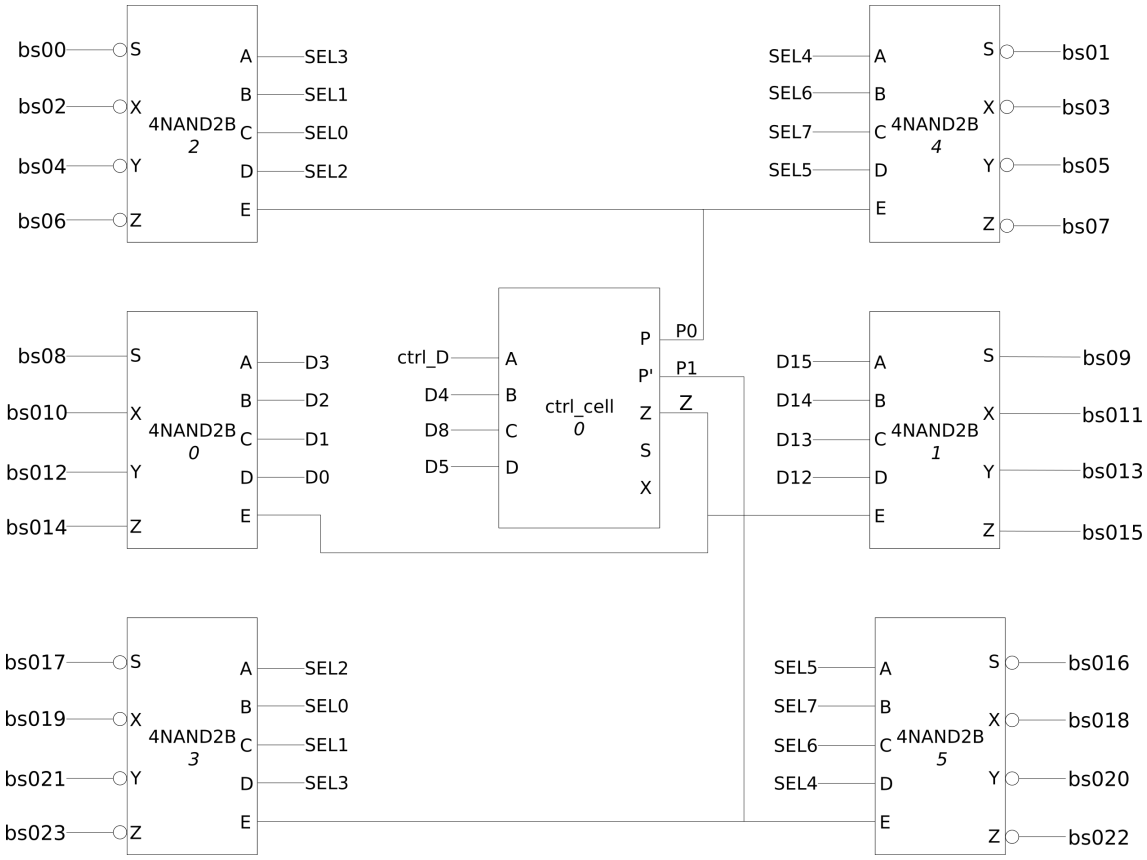


Figure 3.5: ROM netlist, bank and bitline control section

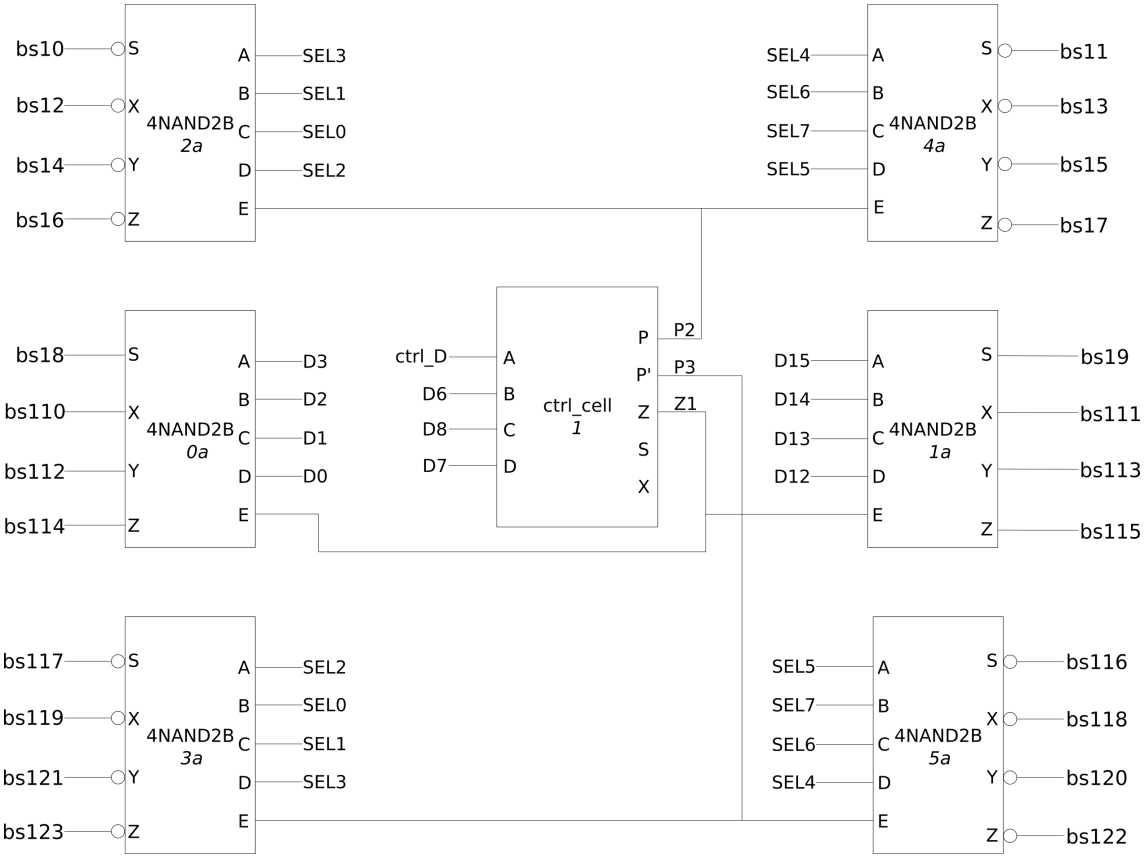


Figure 3.6: ROM netlist, bank and bitline control section

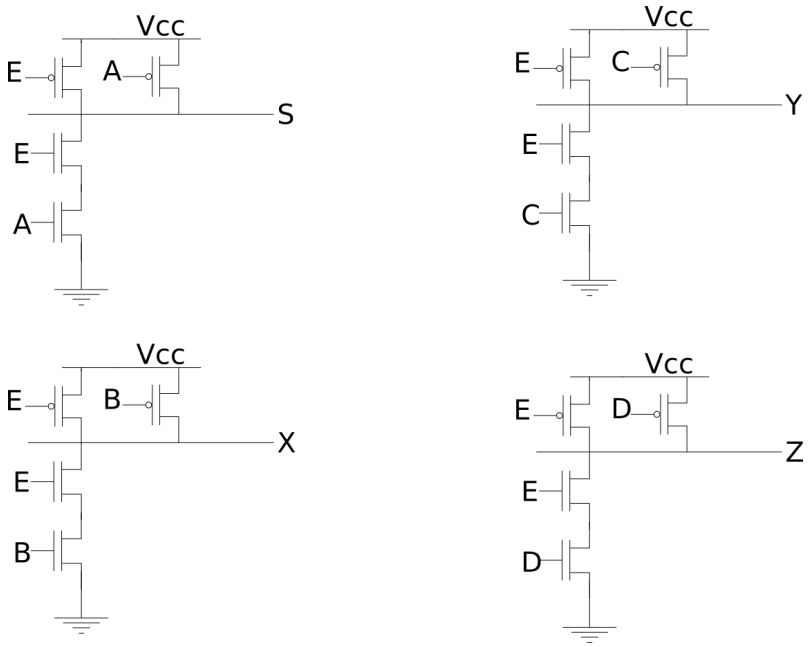


Figure 3.7: *4NAND2B*

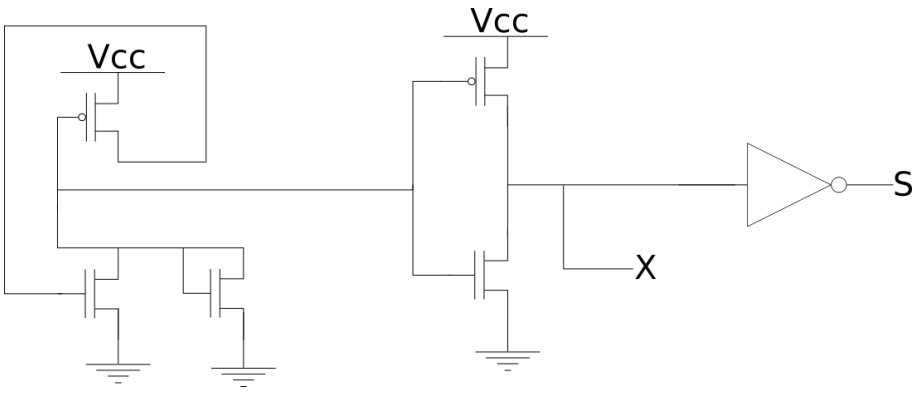


Figure 3.8: *ctrl_cell*

- *D6*, *D7* and *D8* for the second instance (figure 3.6), controlling bank 2 (upper) and bank 3 (lower);

The unused *D* signals (*D9* to *D11*) are left unconnected. Most likely several masks used to realize this ROM are also used in other configurations which have a larger number of banks, hence the reason why these three signals are unused in this case. *Ctrl_D* comes from outside of the ROM and is necessarily periodic, since it is used to generate the pull-up signal for many dynamic logic cells. This signal propagates to either *P* or *P'*, which are the pull-up signals for the two banks controlled by each stage. The three *D* signals select which out of *P* and *P'* will follow *ctrl_D*, allowing the respective bank to enter the evaluation phase when *ctrl_D* rises. An additional output signal named *Z* is the result of an OR operation between *P* and *P'*. In fact, *P* or *P'* are also used along with *Z* as enables for the *BS* signals. These signals are generated by six cells named *4NAND2B* (figure 3.7), basically NAND gates which accept as inputs one of the *SEL* or of the *D* signals and either *P*, *P'* or *Z*. The *BS* signals are used to select the 34 output bitlines for two banks. In particular:

- *BS0* to *BS7* are used to select 272 bits from the upper bank. Each one of them is enabled by the *P* signal and one of the *SEL* signals;
- *BS16* to *BS23* are used to select 272 bits from the lower bank and each of them is enabled by the *P'* signal and one of the *SEL* signals;
- *BS8* to *BS15* are used to select 34 out of the 272 bitlines selected by the other *BS* signals. They are enabled by the *Z* signal and part of the *D* signals (*D0*, *D1*, *D2*, *D3*, *D12*, *D13*, *D14* and *D15*, irrespective of the instance);

For bitline selection to occur properly, only two *BS* signals must be asserted: one out of *BS8* to *BS15* and one out of the remaining *BS* signals. The reason is explained in

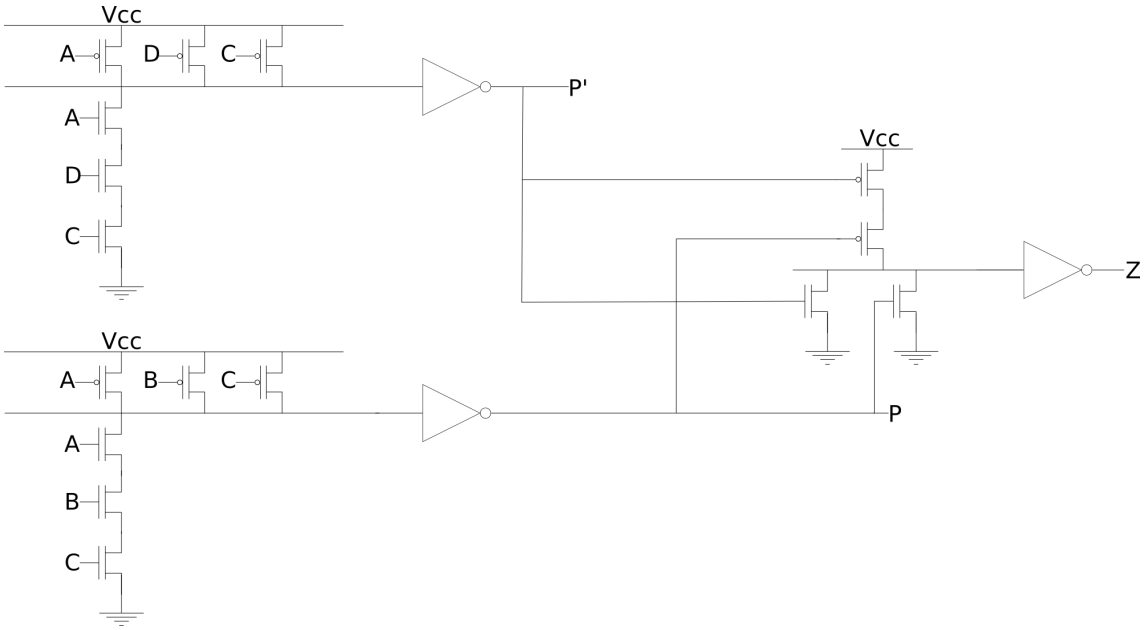


Figure 3.9: *ctrl_cell*

section 3.2.4. Lastly, notice that the cell named *ctrl_cell* outputs as well S and X, which are reference signals respectively for ground and V_{cc}. S and X are generated by a circuit which takes as inputs solely V_{cc} and ground. Their usage is unknown as they have not been followed.

3.2.3 Stripe and wordline selection stage

Within each bank, every wordline is precharged by default using the relative P signal. As previously stated, the control signals allow only one of these 4 P signals to go low, i.e. two or more banks cannot enter the evaluation phase at the same time. Another part of the address contributes to enable the discharge of the same wordline across all the stripes, while all the other wordlines maintain the high value. However, the address determines also the selection of a single stripe, so that ultimately only one wordline can actually be pulled down. Moreover, the bitlines are connected to the various stripes thanks to a pass transistor. Only the pass transistor relative to the selected stripe will be active when the precharge signal is high, thus connecting all the bitlines to a single stripe (check section 3.2.4). These two mechanisms together make so that, at each time, only one wordline out of a single stripe and out of a single bank influences the bitlines.

The only cell used in this stage is named *wordline_pupd*. In the relative figure (3.11), the S and S' signals are connected to the wordlines, with the exception of *S8* and *S0'*, which are connected to the pass transistors of two different stripes. Notice that A and A' are connected to the same input, and the same goes for B and B', C and C' and so on.

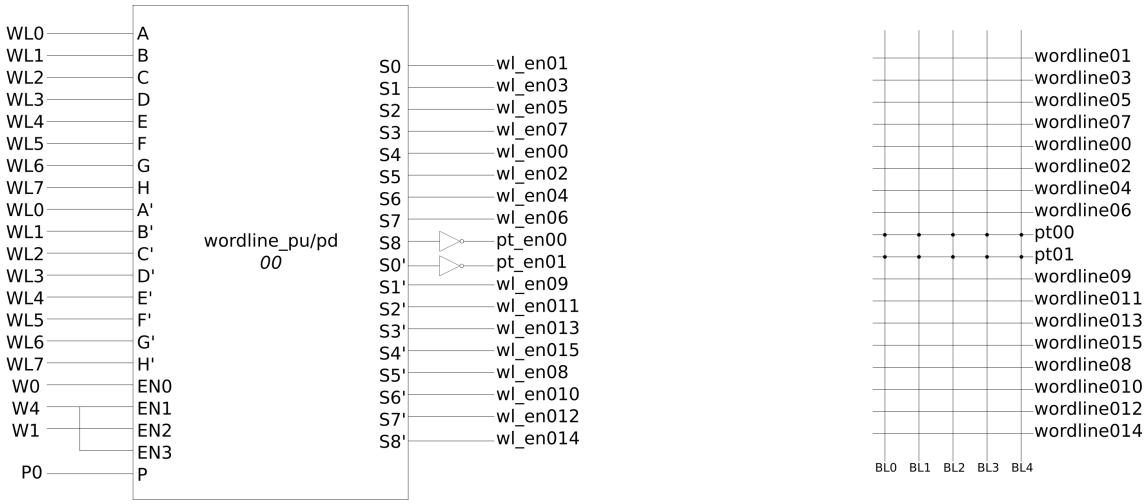


Figure 3.10: ROM netlist, wordline control section

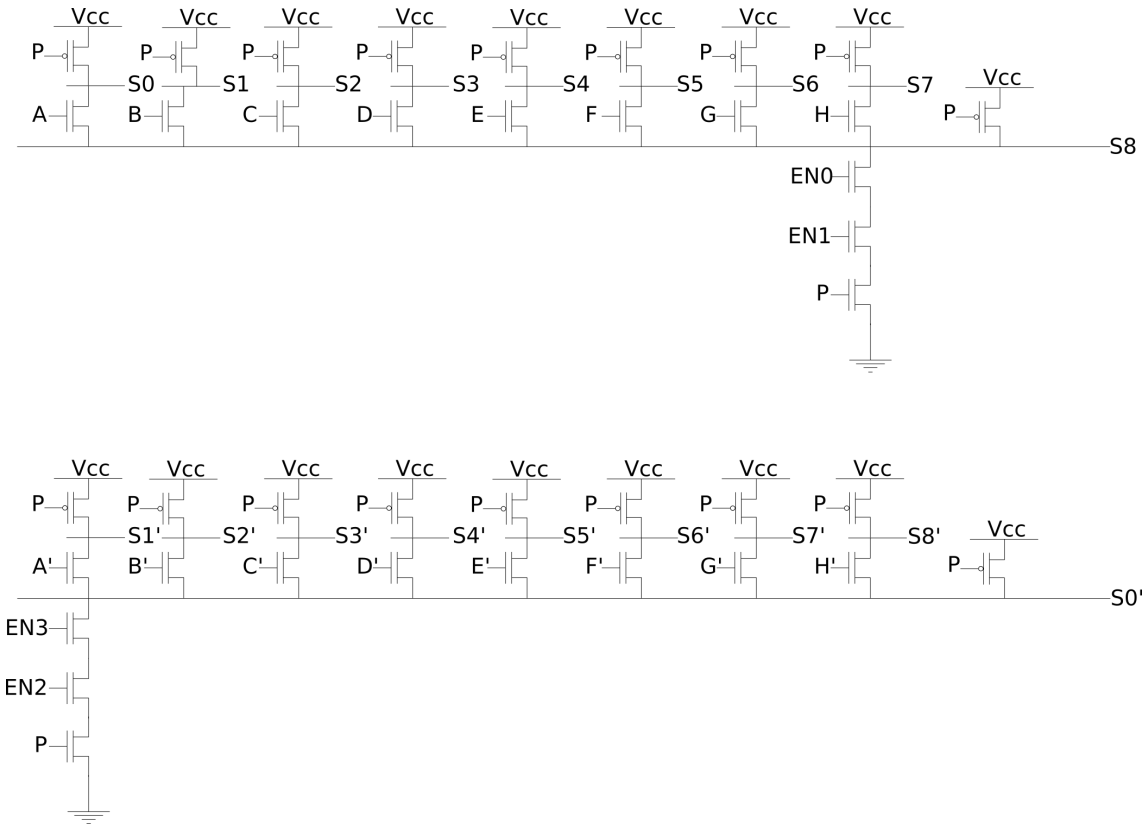


Figure 3.11: *wordline_pupd*

Each of these cells controls two different stripes. The inputs are the precharge signal (P), 16 wordline selection signals (A to H and A' to H') and 4 stripe enable signals (*EN0* to *EN3*). The wordline selection signals are pairwise connected to the same input, e.g. A and A', B and B', and so on. The number of stripes varies with the bank (cf. table 3.1). As previously stated, the wordline selection signals are named *WL* while the stripe enable signals are named *W*. Each of these cells has 16 outputs connected to an equal number of wordlines plus two outputs connected to the gates of two arrays of pass transistors (*S8* and *S0'*).

The connections between the *EN* and the *W* signals occur thanks to the vias connecting metal 2 to metal 3. As stated in section 3.1, the vias of metal 2 are not clearly visible for the first bank (bank 0). Luckily, the visible ones hinted at a pattern, which allowed to make an educated guess for the missing vias. *EN1* and *EN2* are always connected to the same *W* signal, namely one among *W4* to *W7*. In particular, the first four stripes have *EN1* and *EN2* connected to *W4*, the following four stripes have the two enable signals connected to *W5*, and so on. *EN0* is always connected to either *W0* or *W2*, which switch periodically (i.e. *W0* for the first stripe, *W2* for the third stripe, *W0* for the fifth stripe, etc.). *EN3*, *W1* and *W3* follow an analogous relationship (i.e. *EN3* is connected to *W1* in the second stripe, to *W3* in the fourth stripe, and so on).

3.2.4 Bitline selection stage

Selecting a bank, a stripe and a wordline activates 2176 bitlines from a single bank. All the bitlines are connected to several buses through pass transistors. In particular, each of these buses is connected to 16 bitlines coming from two different banks, i.e. 8 from the upper bank (A to H), 8 from the lower bank (A' to H'). The pass transistors related to the upper bank are controlled by $BS0$ to $BS7$, while $BS16$ to $BS23$ control the ones related to the lower bank. It is evident that for each pair of banks there are in total 272 buses ($\frac{2176}{8}$). The cell implementing this structure is named *bank_cell* (figure 3.14). These buses are also connected to a pull-up p-MOS and a pull-down n-MOS, both controlled by the O input: when O is low, the pull-up stage is active. This pull-up stage is assumed to have a lower driving power with respect to a 0 eventually forced by the selected bitline, following the common architecture employed for NAND ROM memories. The output of the bitline buses (Q) is then connected to an inverter and then to a NAND gate. The other input of the NAND gate is again the O signal, while the output of the NAND gate is connected to a n-MOS capable of pulling down the output, K.

Given the aforementioned observations, we may consider the O signal as an active low enable. Moreover, this input can be connected to one signal out of $BS8$ to $BS15$, which explains why they are not inverted, as opposed to the other BS signals (observe figures 3.5 and 3.6). Consider now that, as shown by figures 3.12 and 3.13, the K outputs of 8

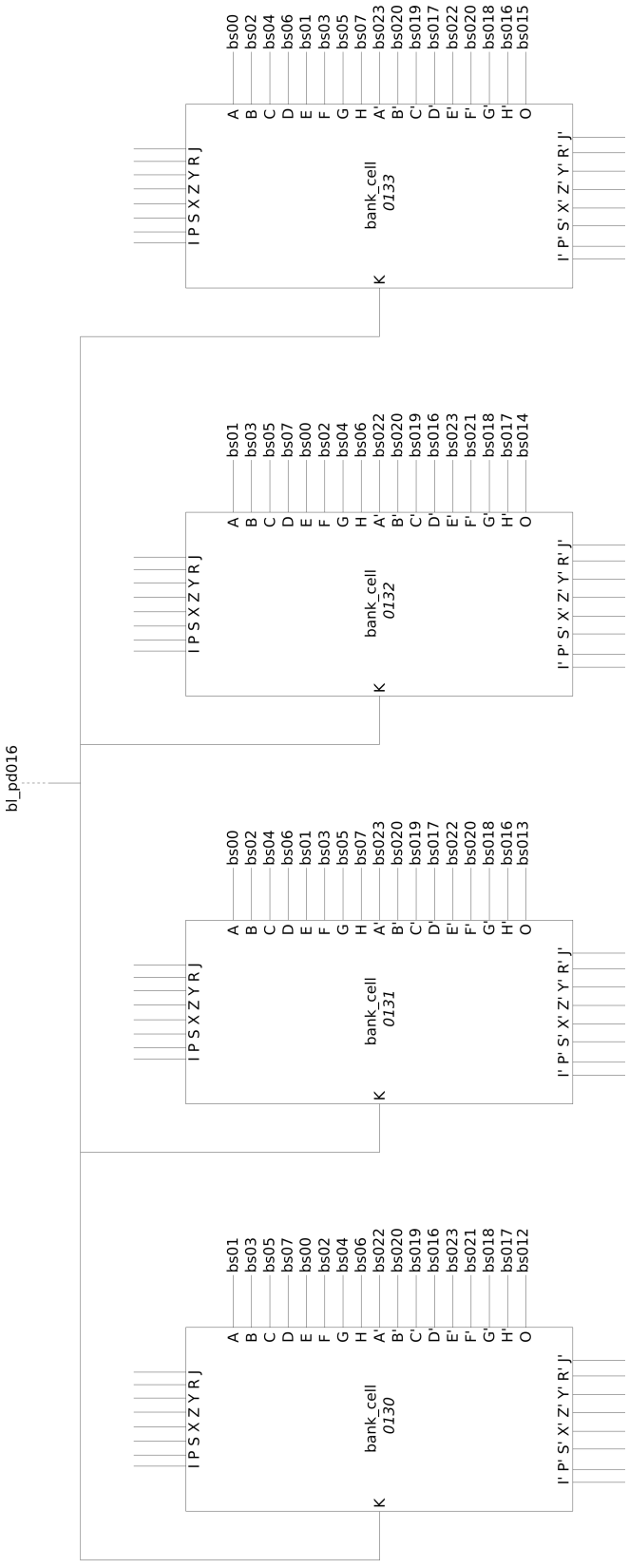


Figure 3.12: ROM netlist, bitline MUX relative to the 16^{th} bit and the first bank pair

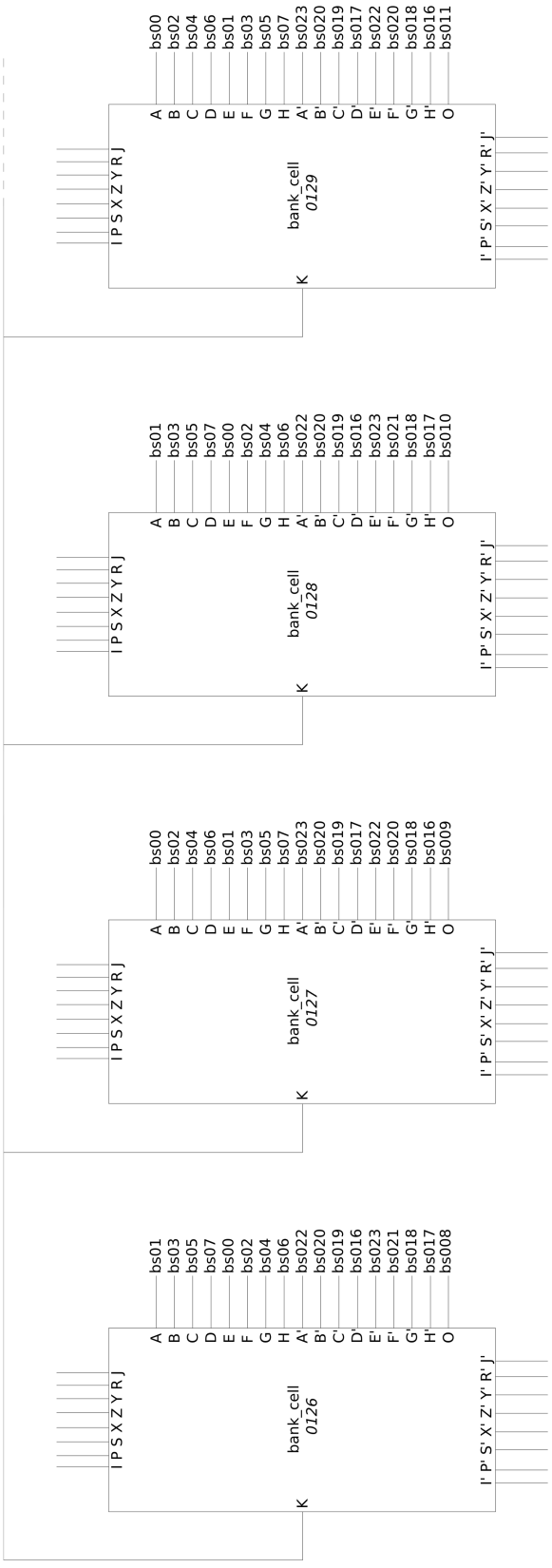


Figure 3.13: ROM netlist, bitline MUX relative to the 16^{th} bit and the first bank pair

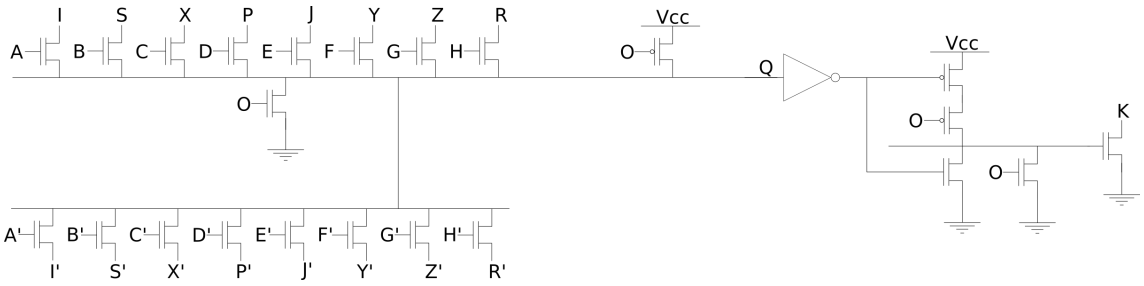


Figure 3.14: *bank_cell*

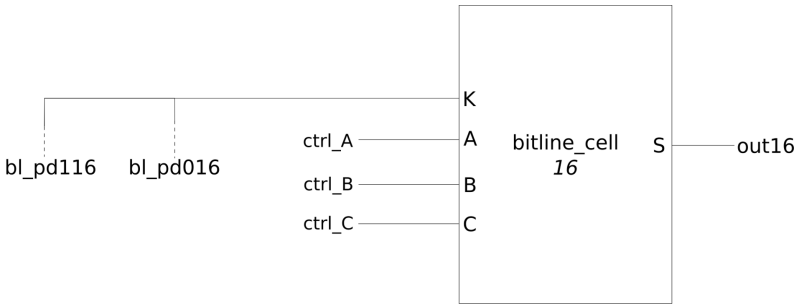


Figure 3.15: ROM netlist, output stage for the 16^{th} bit

different *bank_cell*s are connected to the same network, which is essentially a pull-down net. In particular, the figures shows the netlist and the signals corresponding to the 16^{th} output bit. It should now be clear why section 3.2.2 says that these eight BS signals are used to selectively enable only 34 *bank_cell*s, since only one of them is active at a time ($\frac{272}{8}$). Moreover, as expected, the enabled cells connect their pull-down stage to one of the 34 ROM outputs through the stage described in section 3.2.5.

There are two sets of BS signals, since each bank control stage generates them, recalling section 3.2.2. For this reason it would be more proper to add a numerical prefix to their nomenclature in order to specify the bank pair, such as in the figures of this chapter (e.g. 3.12 and 3.13). However, to keep things simple and abstract, the prefix is omitted - just recall that what is being said here applies for each of the two bank and bitline control stages.

3.2.5 ROM outputs

The ROM outputs are generated by 34 cells named *bitline_cell* (refer to figures 3.15 and 3.16). Other than the pull-down nets discussed in the previous section, these cells accept three more inputs: *ctrl_A*, *ctrl_B* and *ctrl_C*. The latter signal is connected to an inverter with a floating output, hence it serves no purpose - the same standard cell is probably used in other instances, or maybe other versions of this ROM make use of this gate. *ctrl_B* and *ctrl_A* are control signals. *ctrl_B* is basically an enable signal which, along with *ctrl_A*,

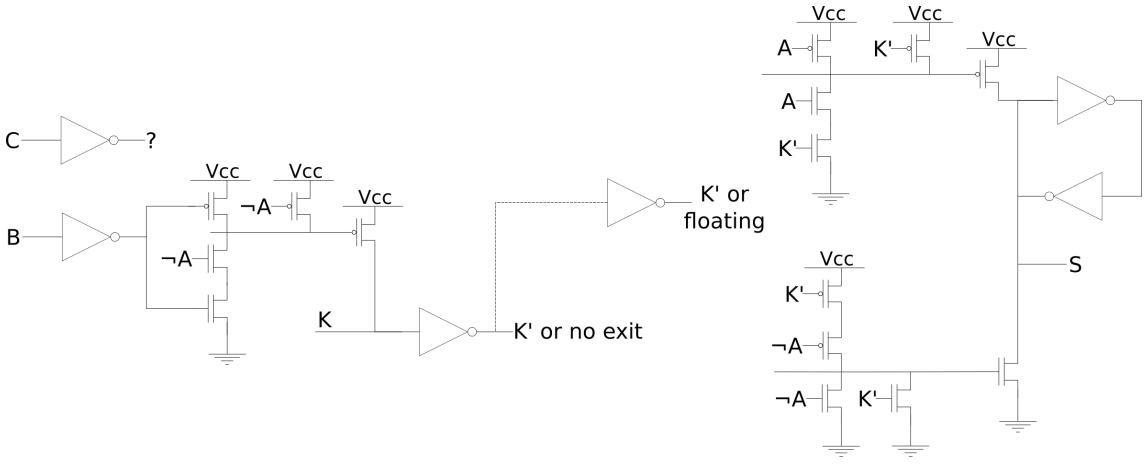


Figure 3.16: *bitline_cell*

allows the precharge of an internal bus, which is connected as well to one of the 34 enabled pull-down net.

This means that either this bus is pulled low by the pull-down net (0) or that it keeps the precharged value (1). The output of this bus is connected to one of two inverters, depending on the cell, and then to a latch having a peculiar architecture and using *ctrl_A* as enable. The output of the latch is finally connected to one of the outputs of the ROM. The output is clearly generated in two phases: during the first phase *ctrl_A* precharges the bus, then *ctrl_A* switches, entering the evaluation phase and at the same time activating the output latch, hence storing the input bit. When *ctrl_A* switches again, the bit will remain stored in the latch. The following string sums up whether the outputs are inverted or not. The leftmost bit corresponds to the least significant bit. The inverted bits are in boldface.

0 1 2 3 **4 5** 6 7 8 9 **10 11 12** 13 **14** 15 16 **17 18 19** 20 21 22 23 24 25 26 27 28 29 **30** 31 32 33

3.2.6 Summary

Depending on the number of inverters present in the *bitline_cell* stages, the bitline output might or might not be inverted. Since the *bank_cell* stage contains an inverter, the presence of a single inverter means that the output is not inverted (2 inverters in total along the path), while the presence of two inverters will cause the inversion of the stored bit (3 inverters in total). The address decoding procedure of the ROM might be summed up as follows:

1. Periodic control signals, namely *ctrl_D* and *ctrl_A*, regulate the precharge and evaluation phases of the ROM;

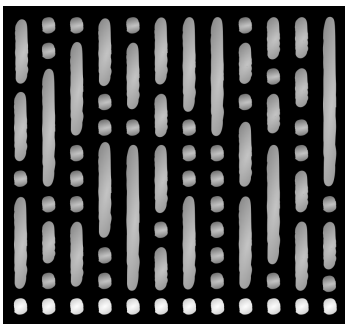
2. The address is decoded;
3. Given the address, the pull-down signal of a single bank is asserted, resulting in a single bank being evaluated;
4. During the evaluation, the address virtually determines the selection of multiple wordlines, one per stripe;
5. The address also determines the activation of a single stripe, hence a single wordline from one single bank is selected;
6. 2176 bitlines are activated via wordline selection: only 272 of them reach the input of a 8-to-1 bitline multiplexers, thanks to part of the *BS* selection signals;
7. Additional selection signals coming from the *BS* array determine which out of the eight possible bitlines filtered by the previous stage actually influences the ROM outputs;
8. Each of the 34 ROM outputs is sampled by a latch. Some outputs are inverted while some others are not;

Notice that bits belonging to the same 64 bit chunk along a wordline are unrelated. In fact, each ROM output selects one bit out of 64 belonging to the same chunk within a single wordline, thanks to the circuit described in section 3.2.4.

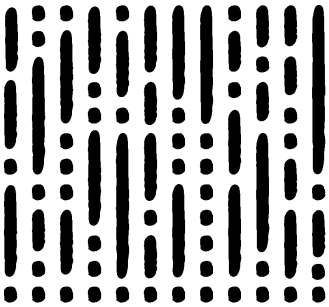
3.3 Bank content extraction

As previously stated, the bits were extracted from the features of the pictures of metal 1 that covered the four banks. A Jython script using FIJI packages was used for this purpose, with one picture at a time being processed. For confidentiality reasons, specific aspects of the employed algorithm cannot be disclosed, hence what follows is a general description of the algorithmic steps. First, the pictures are despeckled to reduce any noise. Then, each picture is turned to black and white, with pixels darker than a certain threshold turning entirely black, the others turning white. This basically allows to highlight the vertical metal wires, as the black background turns to white while the grayish metal lines turn to black.

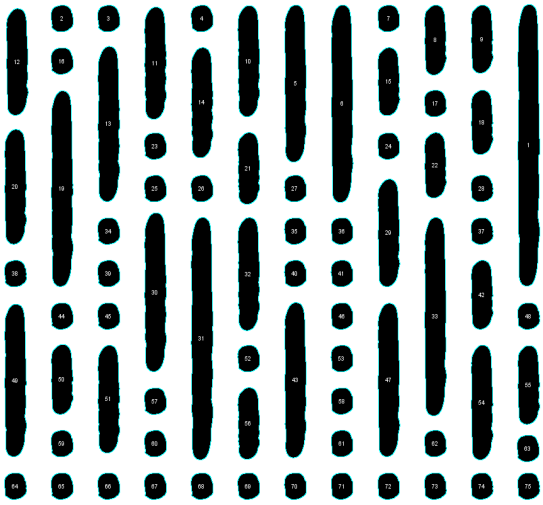
At this point, the image pixels are encoded in grayscale, hence FIJI can easily detect bonding boxes, i.e. groups of black pixels which are separated by the rest of the image by white pixels. Each bonding box contain a series of related parameters, such as position, size and circularity level. Figure 3.17 shows a sample result of the aforementioned steps, using a handmade image to simulate the processing phases just described. Table 3.2 shows



(a) Sample metal 1 picture



(b) Black and white version



(c) Bounding box identification

Figure 3.17: Example of image processing and feature extraction using FIJI.

part of the values extracted from picture c. The parameter **BX**, which is related to the position along the x-axis of the center of the bounding box, is used to determine where the bitlines are located inside the image. The average width of the bonding boxes is used to obtain the actual area in which the binary value of a bit is determined.

Thanks to these two values, the coordinates identifying the pixels which determine the presence of a 0 or of a 1 are easily identified. According to these coordinates, the picture is scanned selectively from top to bottom: if the pixels in the analyzed area are black, a metal connection is present, hence a 0 is appended to a certain wordline, otherwise a 1 is appended. Before proceeding with the aforementioned process, the wires connecting each bitline to the ground, easily recognizable thanks to the length of their bonding box, are erased to avoid adding useless zeros. On the other hand, the pass transistors on the top of each bitline introduce a series of 1s. These 1s remain in the output files generated by processing each picture, as they are useful to later align the pictures during the stitching

	Area	BX	BY	Width	Height	Circ.
1	1713	9379	322	389	4967	198
2	153	1145	333	389	467	896
3	153	1967	333	389	467	897
4	153	3611	333	389	467	903
5	942	5267	333	378	2767	326
6	1189	6089	333	389	3478	271
7	152	6912	333	378	467	909
8	410	7734	333	389	1233	595
9	399	8556	333	389	1200	603
10	662	4434	344	389	1956	426

Table 3.2: Bounding box and extracted features

process.

After processing the single images, a multitude of .txt files is obtained, containing the bits extracted by each picture. An additional script is used to stitch all these bits together, taking as input only the .txt files concerning a certain bank. As the pictures have approximately a 20% overlap, so do the extracted bits: this information is used by another script in order to find the correlated bit sequences and superpose the various .txt files. The presence of long series of 1s due to the pass transistors further help the bit alignment process. The fake wordlines generated by the presence of pass transistors are then eliminated in the output .txt file, which contains all the wordlines belonging to a certain bank. Notice that, should there be an actual wordline containing only 1s, it would not be deleted, since it would not occupy the position it would have if it was a line introduced by the presence of pass transistors. The .txt files containing the various banks are then imported into VHDL arrays, ready to be read. Considering the way the ROM was described in VHDL, the 1's are translated to 'Z', namely the character used by the *standard_logic* VHDL package to signify high impedance. Table 3.3 shows the stitching process for a small set of bits, including the removal of the fake wordline introduced by the pass transistors.

1	**0**	0	1	1	0	1	0	**1**	**0**
1	**1**	0	1	0	0	0	0	**0**	**1**
0	**1**	1	1	0	0	0	0	**0**	**1**
0	**1**	0	0	1	1	1	0	**1**	**1**
1	**1**	1	1	1	1	1	1	**1**	**1**

(a) Sample bit matrix, left chunk

1	**0**	0	0	0	1	0	0	**0**	**0**
0	**1**	0	1	1	0	1	0	**1**	**1**
0	**1**	0	0	0	1	1	1	**0**	**0**
1	**1**	1	0	0	1	0	1	**0**	**0**
1	**1**	1	1	1	1	1	1	**1**	**1**

(b) Sample bit matrix, right chunk

1	**0**	0	1	1	0	1	0	**1**	**0**	0	0	0	1	0	0	**0**	**0**
1	**1**	0	1	0	0	0	0	**0**	**1**	0	1	1	0	1	0	**1**	**1**
0	**1**	1	1	0	0	0	0	**0**	**1**	0	0	0	1	1	1	**0**	**0**
0	**1**	0	0	1	1	1	0	**1**	**1**	1	0	0	1	0	1	**0**	**0**

(c) Stitched bit matrix

Table 3.3: Bank bits stitching process

Chapter 4

Conclusions

4.1 Simulation and analytic code extraction

The VHDL simulation in ModelSim employed a testbench which generates periodic signals for *ctrl_A* and *ctrl_D*, while *ctrl_B* and *latch_en* were always stuck to 1. The address increases at each clock tick of *ctrl_A*/*ctrl_D*. The simulation showed the expected behavior for each control signal, further validating the hypothesis discussed so far. In particular:

- At each clock tick, a different *WL* signal is enabled, thus enabling a different set of wordlines. The *WL* vector of signals has a periodicity of eight clock ticks;
- The same *P* signal follows the clock for eight clock ticks, then a different *P* signal starts commuting, thus enabling a different bank. The vector of *P* signals has a periodicity of 32 clock ticks;
- The *W* vector changes every 32 clock ticks, meaning that a different stripe is enabled after all the wordlines and all the banks have been cycled;
- After the periodicity of the *W* vector has elapsed, it is the *SEL* vector of signals that commutes, thus selecting a different subset of bitlines to be sent to the outputs. Their evolution influences the *bs* signals generated by the two stages controlling the bitlines (cf. section 3.2.2);

Due to confidentiality reasons, ModelSim screens, as well as additional information on the periodicity of the last two vectors, could not be disclosed. In any case, the simulation clearly showed that the whole ROM content could be addressed by trying all the possible combinations of the address bits. Clearly, since the addressing space is much larger than the actual ROM size, many addresses activated no wordline at all. In these cases the output had a constant value which was the result of having no pull-down network influencing any precharged bit. Due to the inversion of of the outputs detailed in 3.2.4, the ROM output

in this case was stuck to 4 DF F8 5C 3F. During each simulation, the output bits were stored inside a .txt file in addressing order. This allowed to dump the content of the whole ROM memory, thus concluding the analytic code extraction procedure.

4.2 Follow up

The analytic extraction of the ROM code is not enough to be able to exploit it, since nowadays secure ROMs contain encrypted data. The next step is therefore to find the decryption circuit and reverse engineer it, so that the logic operation performed by it can be understood and performed on the encrypted bits. Usually, secure ROMs are encrypted using symmetric cryptography (refer to section 1.2.2), employing a secret key stored in the ROM itself. While I was performing the reverse engineering of the ROM, as stated in section 1.4, my colleagues in the hardware team were using Chipjuice to individuate and reverse engineer the decryption block. Unfortunately, registers and standard cells dedicated to secure operations are placed in a rather random fashion by the employed routing and placement algorithms, with the goal of hardening the chip's security against local attacks.

For this reason, it was not possible to identify a specific area of interest and delimit the reverse engineering process within. It was necessary to perform the netlist extraction of the entire processor, which proved to be challenging and resource demanding, since the chip was particularly large. However, while I am writing this section, the hardware team has completed the processing and feature analysis of all the imaged layers and is about to complete the reverse engineering of all the standard cells present in the chip.

As a consequence, we may conservatively predict that in less than a month the decryption logic will be obtained. In fact, once the whole netlist has been extracted and synthesized, all it takes is to follow the ROM outputs toward the chip's core, using an RTL viewer, and observe the logic operations performed along the path, or directly extract the related VHDL code, realize a testbench which feeds the decryption circuits with the ROM content and store the outputs. Additional time is then required to reverse engineer the extracted binaries and understand the assembly instructions implemented by each word.

4.3 Threat analysis

In less than 3 months, using specialized lab equipment, the know-how and software provided by Texplained and relatively simple FIJI scripts, it was possible to extract the encrypted content of the ROM of a highly secure chip. Moreover, as stated in section 4.2, there are very high chances that the ROM content will be decrypted in the following weeks. According to the definitions given in section 1.2.4, the chip is classifiable at least

with a MODH level, while the attacker of type represented by Texplained is of type II, being a specialized but small company with medium resources.

The availability of the ROM code of a highly secure chip translates to having extremely powerful information when it comes to mounting a hardware attack, or even a software attack. For instance, the ROM boot code contains the startup value of many registers. This knowledge alone makes glitching and probing attacks much easier, since an attacker will be able to pinpoint right away the registers and the values to modify in order to change the chip's behavior. The individuation of a bug in the boot code, albeit unlikely, since this type of program is extensively tested, would have devastating effects on the security of the chip, as it would enable powerful software attacks.

Moreover, the netlist extraction of the whole chip alone constitutes an enormous threat to the chip's security, since it will enable an attacker with a considerable amount of information usable to reduce the guess work and research necessary to mount an attack. This aspect will not be detailed, since it is not in the scope of this thesis, but it should be kept in mind nonetheless. Consider, moreover, that if the goal of an attacker was to infringe IP to make clones of an embedded system, the goal would already be reached without the necessity of any further work. A coarse estimate for the cost of the work detailed by this thesis amounts to ca. 30,000 €, considering the required materials, machines and expertise. Given the security applications of this chip, the prospective attackers would be of the type 3 kind, hence this figure is absolutely not prohibitive. Consider that this relatively low price is the consequence of technological improvements and the evolution of the market. For example, before it would have been much more expensive to have access to a SEM capable of imaging the chip concerned by this thesis. With more sophisticated technologies hitting the market, older ones become cheaper and may nonetheless be more than sufficient for reverse engineering purposes.

Nowadays it is even possible to rent a FIB for 120 € per hour, extending furthermore the accessibility to such a technology. Even the cost for the expertise is being reduced dramatically by the general knowledge increase and amount of software assisting the reverse engineering process. Chipjuice, for example, reduces by at least one order of magnitude the amount of necessary work with respect to the methodologies previously employed, such as laborious pen-and-paper approaches or others assisted by image processing tools such as Photoshop. On this regard, remember that the ROM reverse engineering process was done via Photoshop. While the amount of work was definitively not prohibitive, consider that it will be even reduced when appropriate software tools will be developed or added to Chipjuice itself.

In conclusion, this thesis should contributed to the proof that nowadays the reverse engineering process has become considerably less time and resource consuming with respect to before. The fact that the market, technologies and expertise are in constant evolution

Factor	Overview	Score
Elapsed Time	$\leq$ three months	10
Expertise	Expert	6
Knowledge of TOE	Critical	11
Window of opportunity	Unlimited access	0
Equipment	Bespoke	7
Total	28	

Table 4.1: Common criteria attack potential evaluation for analytic code extraction
Source: Texplained

makes reverse engineering and invasive attacks a tangible threat, amplified by the fact that IoT devices are becoming more and more pervasive. In the author's opinion, the current approach on the subject needs an update. The industry and the lawmakers must revise their evaluation criteria when it comes to secure chips. Other than the facts detailed in section 2.1, consider that the Common Criteria standard (cf. section 1.2.4) would give a score of 28 to the analytic code extraction procedure detailed by this thesis, which is enough to consider it as a residual threat. This is due to the the involvement of the reverse engineering procedure and the allotted time, which have the highest weight in the final score (cf. table 4.1).

It is evident that the current Common Criteria standard cannot consider any invasive attacks involving reverse engineering as a serious possibility, confining them as available only to extremely powerful attackers. With my work at Texplained I demonstrated that this is not the case. In fact, I managed to carry out the reverse engineering procedure while still being an undergraduate student, therefore not equipped with highly specific knowledge nor previous reverse engineering experience. All I needed to know was the functioning of NAND ROM memories, CMOS logic, dynamic logic and the basics of microlectronics and integrated digital circuits layout. The hope is that, in the future, security evaluation standards, and by extension lawmakers and the semiconductor industry as a whole, will consider reverse engineering as a serious threat to secure chips, as it enables very powerful invasive attacks. After all, firms such as Texplained demonstrate with their own existence that multiple chips by different manufacturers are weak against this approach.

4.4 Further work

This thesis enables two types of further work. On one hand, the development of a software capable of automatizing the reverse engineering of any ROM, regardless of the manufacturer, the type and the size, so to allow scholars and players in the semiconductor industry to make quick security assessments on this subject. This clearly requires the knowledge

of the architecture of many different ROM memories in order to create the most fitting abstract model. On the other hand, the next step in terms of research is the design of countermeasures against reverse engineering.

Since the reverse engineering process is carried out using pictures of the various layers, the most effective countermeasures seem to be either to make the delayering and the imaging processes more difficult or to obfuscate the chip's layout, such that its inspection will not easily give away information to a reverse engineer. Many ideas have already been proposed in the latter sense, such as camouflage gates [7], realized either by changing the polarity of the dopants inside selected active regions or by inserting dummy vias. A camouflage gate implements a different functionality with respect to what it would seem by simply observing its layout, so that the netlist obtained at the end of the reverse engineering process will be flawed by the presence of many standard cells which implement the wrong functionality.

I could not find any literature on the first direction. The only way to hinder the imaging process seems to be counteracting the way a SEM manages to obtain pictures. The shrinking of the technology node in principle helps in this direction, since the needed resolution might exceed the possibilities of an old machine, but this is definitely not a realistic countermeasure, given that current SEMs have a resolution better than 1 nm and that the industry manufacturing node will reach, in the most optimistic case, a 5 nm node in 2020, according to the roadmap of IBM and Samsung. Analogously, the hindering of the actual SEM imaging process has seemingly never been explored, most likely due to its dubious effectiveness. The basic idea is that the electron beam used by the SEM be disturbed by some passive or active countermeasure. An active countermeasure seems unlikely, since it would need power to operate. When a chip is being imaged it is surely not connected to any power supply, hence the only solution would be to use a battery. Besides the fact that this would imply that the active noise generator has to be extremely low power, a battery can generally easily be identified and, thus, removed, nullifying the effectiveness of such a countermeasure.

This brings us to passive imaging countermeasures, namely coating with a material that disturbs the signal produced by the electron beam, such as a substance which traps the emitted ions, causing the generation of unusable images. Such a coating has to be very difficult to remove without damaging the rest of the chip. Unfortunately, due to the lack of available documentation on the subject, I cannot vouch for the feasibility of such a countermeasure. The last aspect to be treated, namely techniques to harden the delayering process, basically poses the same difficulties of passive imaging countermeasures. A good starting point would be to refer to current delayering techniques, find their weaknesses and try to exploit them to harden the chip's resilience against reverse engineering. Several publications describe delayering techniques in detail, such as [18]. It must be kept in mind,

however, that currently even the hardest delayering scenarios can be overcome with the proper skills and equipment.

In conclusion, unless groundbreaking technological improvements occur, it should be assumed that any countermeasure can be broken. The industry should focus on this issue, never resting easy but always deploying the proper amount of resources and research on improving security measures. Security must be assured by design, such that even a thorough reverse engineering process will not lessen the security of a microprocessor.

Appendix A

Hardware security timeline

In this section I describe the most important events in the history of hardware security, presenting how technologies have evolved and which types of vulnerabilities were exploited over the years. Then, I proceed by discussing in deeper detail how hardware security was implemented by the manufacturers from the beginnings to the present day. I conclude this section by presenting and commenting a table, taken from pagetable.com, which compares the time taken to hack different electronic systems such as game consoles and tablets.

Following is a tentative timeline, put together after some research. It is surely incomplete, but I am confident that it will allow the reader to have a glimpse on how the field has changed due to emerging technologies and the discovery of vulnerabilities and bugs. Notice that the first entries might be seen as not very related to the topic, for instance the ones concerning phone phreaking, but I added them because they still concern security breaches involving electronic hardware, or electro-mechanical, like the major events concerning cryptography breaches during World War II. On the other hand, major events concerning software security (such as Kevin Mitnick's exploits and the Stuxnet and ILOVEYOU computer worms) were left out since I felt the necessity to clearly separate the two fields. Moreover, while it is easy to find sources about software security history, it is not the same when it comes to hardware security, mostly due to the fact that only recently this field is considered as being separated from software security. Lastly, the reason why topics concerning software security were left out is that, in my opinion, they substantially have little to no connection with the hardware security domain.

- **1903** Nevil Maskelyne hacks into a public demonstration of a wireless telegraphy communication technology invented by Guglielmo Marconi and claimed as secure [13];
- **1932** Marian Rejewski, Henryk Zygalski and Jerzy Różycki crack the Enigma machine decryption algorithm;

- **1939** Alan Turing, Gordon Welchman and Harold Keen develops the Bombe, an electro-mechanical device used to perform a brute force key search on Enigma;
- **1956** MI5 exploits the clicking sound produced by the enciphering machine used in the Egyptian Embassy in London to decrypt ciphered messages;
- **1957** Joe Engressia (nicknamed "Joybubbles") discovered that whistling at 2600 Hz would interfere with AT&T's automated telephone systems. This event is regarded as the beginning of phone phreaking;
- **1971** Joe Engressia, John T. Draper (nicknamed "Captain Crunch") and the phone phreaking community become famous thanks to an article published on Esquire Magazine. The article describes how phone phreakers were capable to make free phone calls and other telephone hacks through blue boxes, electronic devices capable of producing in-band signaling audio tones;
- **1973** Single event upsets, consisting in the unwanted flip of bits caused by a single ionizing particle, are described for the first time in a paper published by Edward Smith, Al Holman and Dan Binder;
- **1977** The DES (Data Encryption Standard) symmetric block cipher is adopted by NSA;
- **1977** Ron Rivest, Adi Shamir and Len Adleman, MIT students, introduce the RSA asymmetric encryption standard. An equivalent system was developed by Clifford Cocks in 1973 but it remained classified until 1997;
- **1977** Code enciphering proposed and patented by Robert Best with the goal of securing the communication between the CPU and external memory through cryptographic functions;
- **1996** Paul Kocher publishes a paper on timing analysis, a side-channel attack which can lead to the disclosure of cryptographic keys;
- **1996** Dan Boneh, Richard DeMillo and Richard Lipton show in a famous paper how accidental faults can be exploited to attack RSA and retrieve the encryption keys;
- **1998** Markus Kuhn manages to attack the bus-encryption of the super secure DS5002FP microprocessor and gain unauthorized access to the deciphered software stored in its encrypted memory;
- **1999** Differential power analysis, another side-channel attack based on power consumption, is reported by Paul Kocher;

- **1999** Introduction of the first chip-based credit card in the US, the "Blue" smart card, by American Express;
- **2002** AES (Advanced Encryption Standard) supersedes DES;
- **2013** Adi Shamir and other researchers find out that high-pitched noises emitted by computers during operation can be exploited as a side-channel to disclose security-related computations and extract RSA decryption keys;
- **2018** Meltdown, Spectre and other vulnerabilities related to branch prediction are disclosed by Google's Project Zero, Paul Kocher and other researchers;

This summary of events shows once more that hardware security faces challenges coming from multiple sources and which have evolved over time. What is also evident is that the subject now interests more and more people, with the introduction of electronics to the mass industry, and that modern attacks are not necessarily more complex than older ones. Quite the opposite, the fact that electronic systems have entered the mass industry, and the need to maximize profits, has made so that electronic systems have spread to a wider and wider audience without any strong concern regarding security, until recent times. The first major events always involved very specialized and resourceful individuals or groups, mostly because electronic systems were used in the military domain only.

With the introduction of electronic devices to the mass industry, the variety of hardware attacks simply exploded, and even more so when microcontrollers started to be used to implement security services. This occurred because the goals of the attackers incredibly increased in quantity and variety: in the very beginning, the main goal was to gain a military advantage, while modern day goals include IP theft, jailbreaking and ATM skimming, or simply making a name in the hacking community. Moreover, the fact that components are necessarily identical due non recurring engineering costs makes any successful attack very dangerous, as it will be possible to reproduce it on any other device of the same kind. This fact has gained a higher weight nowadays, as in the beginning only a few devices would be produced out of a design and they could be accessed only by authorized personnel, while the mass industry has made electronic systems spread over a large portion of the world population.

The sophistication of attacks has increased while the need of skills and resources has not necessarily followed this trend. In fact, it is true that many modern attacks require expensive personnel and equipment, but many others are very low cost and need only time and patience, such as most side-channel attacks. It is true that security has greatly improved over the years, but, due to the pervasiveness of digital technology, the available literature on the subject and the inherent difficulties of granting high security standards for mass produced devices, the recent years may arguably be regarded as the golden age for

hardware hacking. Manufacturers need to defend themselves against multiple threats, as the devices they want to secure are obviously highly available, meaning that every possible attacker will start working in this sense, including attackers in the production line itself. What is worse, in some cases, a wanna-be hacker does not need a very thorough knowledge to perform an attack.

It can be argued that hardware security properly begun in the 1970s, with the birth of embedded systems. In the beginning, as this field was basically unexplored, the only protection against cloning of such devices was the law and the cost of the attack. The components of early embedded systems, such as the ROM and the CPU, were separated and could be easily recognized by simply observing the printed circuit board (PCB), making cloning very easy. Obscurity was employed from the very beginning of circuit manufacturing, with the usage of application specific integrated circuits (ASICs) or by relying on the secrecy of the proprietary software algorithm, and it is still being employed nowadays. As previously stated in section 1.1, this approach offers little to no security at all and may even be counterproductive.

With the evolution of microcontrollers, in the late 1970s, the life of attackers was made more difficult thanks to circuit integration: microcontrollers started featuring an internal memory and I/O interfaces, and even basic security protection against unauthorized memory access. However, it was still necessary to communicate with external non volatile memories (NVM), as these components could still not be integrated within the same chip, hence attackers could still easily access them. Manufacturers started placing EEPROMs (Electrically Erasable Programmable ROM) next to the microcontroller inside the same plastic package, making things more difficult for attackers as it was thus necessary to first decapsulate the chip and then either microprobe the data chip or connect it to a custom test circuit. These techniques are expensive, hence cutting out low-budget attackers, who could subsequently only resort to software hacking to perform an attack.

Then, hardware security fuses started being introduced, placed externally with respect to the memory to be protected. For instance, they could be used to completely erase a memory if unauthorized access was detected, and their implementation did not even imply the redesign of the microcontroller. However, invasive attacks could be performed, possibly even more easily as the fuse is clearly recognizable. One of the possible techniques consists in disconnecting the sense circuit from the fuse with a FIB or a laser cutter, for example. Even non-invasive attacks were possible, as certain combinations of external signals could lead to the misreading of the state of the fuse, hence preventing data removal. The evolution of this concept was to insert the fuse in the memory itself, disabling data access if the fuse is set. This made things much more difficult, as the fuse would now be much harder to locate and tamper with, being often fabricated with the same technology of the memory.

Subsequently, part of the main memory was used to implement security, either by storing some data to a security register, whose correct content would allow the normal usage of the chip, or by using a password, which is an approach more similar to the modern technologies. If the security register would contain incorrect data, the functionalities of the chip would be disabled. These strategies have non-expensive workarounds, however, such as side-channel attacks. Even power glitching is a viable solution, if the security register is sampled at startup: the attacker might be able to tamper the value which is actually stored, bypassing the security measures. Top metal sensor meshes began to appear as well, with the goal of hardening microchips against invasive attacks. All the wires along these meshes are constantly monitored: if an anomaly is detected, the contents of a secure EEPROM are immediately erased. A less expensive counterpart to this approach is to use a fake top layer mesh, which makes optical attacks and microprobing much harder.

Manufacturers also started removing completely the standard programming interface in programmable smartcards: a bootstrap loader would either erase or deactivate itself after the upload of user code. After this phase, it is the responsibility of the user's code to allow access to data and code stored within the device. The next step was then to introduce bus encryption between the chip's EEPROM and the CPU, with the goal of preventing an attacker from performing microprobing attacks. This countermeasure can in principle be bypassed via non-invasive attacks, since the CPU will access the decrypted data. Moreover, Markus Khun demonstrated in a rather well-known paper that low-cost attacks could be carried out against the DS5002FP, a microchip employing bus encryption [11].

ASIC-like logic design also started to be implemented widely in smart cards. It consists in employing "glue logic" to prevent attackers from gaining access to information by probing certain nodes, since it gets much more difficult to individuate and probe the physical areas implementing a data bus. The microprobing of microcontrollers employing glue logic requires expensive equipment and experienced engineers. Another approach is used when the chip is built from different blocks. Each block uses glue logic, but the interconnections between the blocks occur via regular buses, hence invasive and semi-invasive attacks are easier with respect to the previous approach, although the fact that design blocks are not recognizable makes semi-invasive attacks much harder. Glue logic design, and in general employing full custom ASIC approaches, never eliminate the threat posed by non-invasive attacks. However, they are more difficult, as ASICs have better performances with respect to a general purpose counterpart. Clearly, external memories are still vulnerable when employing this approach, since they cannot be implemented with the same glue logic of the control block.

The reduction of chip size and technological improvements also inherently contributed to increase the difficulty of reverse engineering a chip. Modern deep submicron semicon-

ductor chips demand the usage of expensive and complex machines. This prevents many potential attackers from performing invasive attacks due to lack of expertise or materials. For instance, successive layers would follow the shape of the layer below, hence watching a microchip with a microscope would allow the viewer to observe the topmost layer and part of the underlying ones. With the introduction of chemical-mechanical polishing, for instance, this is not true any longer, hence the shape of the layers above is not influenced by the layers below. This hardens the inspection, which nowadays demands the usage of acids and technologies capable of exposing the various layers. However, the appearance on the market of second hand technology from previous generations allowed attackers to overcome economic issues and employ fault analysis techniques to perform chip delayering (cf. section 2.2). This summary on the evolution of hardware security techniques is mostly taken by a technical report realized by Sergei Skorobogatov [17].

Table A.1 concludes this section. It summarizes the timeline of videogaming console hacking. There are several lessons to be learned from it. The most important one, possibly, is that the main goal of hacking is almost never piracy per se: in most of cases it is jailbreaking, namely being capable of running any kind of software or OS on a certain video gaming console. The actual pirates simply wait for a device to be hacked and then exploit the techniques exposed by hackers to carry out IP theft, counterfeiting or other more damaging activities. This means that restricting freedom of usage of a gaming console has more disadvantages than benefits, as this will give the community of hackers a very good reason to put time and effort into hacking it. In fact, the reason why PS3 took so long to be hacked is that the community was completely uninterested in doing it, as it already gave the freedom to install any OS. In fact, the hacking occurred after a late update forbade this possibility. This does not necessarily mean that it is the only reason why it suddenly happened, as it is clear that piracy was one of the main goals, but the restrictions imposed by the update surely contributed to speed up the process.

Device	**Year**	**Security**	**Hacked**	**Goal**	**Effect**
PS2	1999	?	?	Piracy	–
Dbox2	2000	signed kernel	3 months	Linux	Pay TV decoding
GameCube	2001	encrypted boot	12 months	Homebrew	Piracy
Xbox	2001	encrypted and signed bootup, signed executables	4 months	Linux Homebrew	Piracy
iPod	2001	checksum	<12 months	Linux	–
DS	2004	signed and encrypted executables	6 months	Homebrew	Piracy
PSP	2004	signed bootup and executables	2 months	Homebrew	Piracy
Xbox 360	2005	encrypted and signed bootup, encrypted and signed executables, encrypted RAM, hypervisor, eFuses	12 months	Linux Homebrew	Leaked keys
PS3	2006	encrypted and signed bootup, encrypted and signed executables, hypervisor, eFuses, isolated SPU	4 years	Piracy Homebrew	–
Wii	2006	encrypted bootup	1 month	Linux	Piracy
AppleTV	2007	signed bootloader	2 weeks	Linux	Front row piracy
iPhone	2007	signed and encrypted bootup and executables	11 days	Homebrew SIM-Lock	Piracy
iPad	2010	signed and encrypted bootup and executables	1 day	Homebrew	Piracy

Table A.1: Relationship between security level of several devices and time taken to hack it

Source: pagetable.com

References

[1] Friedrich Beck. *Integrated Circuit Failure Analysis: A Guide to Preparation Techniques*. Wiley, 1998. ISBN: 0-471-97401-3.

[2] *Common Criteria for Information Technology Security Evaluation*. .Introduction and general model. Apr. 2017. URL: https://www.commoncriteriaportal.org/files/ccfiles/CCPART1V3.1R5.pdf.

[3] *Common Criteria for Information Technology Security Evaluation*. .Security assurance components. Apr. 2017. URL: https://www.commoncriteriaportal.org/files/ccfiles/CCPART3V3.1R5.pdf.

[4] *Common Criteria for Information Technology Security Evaluation*. .Evaluation methodology. Apr. 2017. URL: https://www.commoncriteriaportal.org/files/ccfiles/CEMV3.1R5.pdf.

[5] "DATA ENCRYPTION STANDARD (DES)". In: *Federal Information Processing Standards Publications* (Oct. 1999).

[6] D. G. Abraham et al. "Transaction Security System". In: 30 (Feb. 1991), pp. 206 –229.

[7] Christian Kison Georg T. Becker Marc Fyrbiak. "Hardware Obfuscation". In: (Apr. 2017).

[8] D. P. Casasent J. P. Kapur. "Geometric Correction of SEM Images". In: *Hybrid Image and Signal Processing VII* Proc. SPIE 4044 (July 2000), pp. 6 –8. DOI: 10.1117/12.391928.

[9] Auguste Kerckhoffs. "La Cryptographie Militaire." French. In: *Journal des Sciences Militaires* IX (Jan. 1883), pp. 5–38.

[10] Paul Kocher et al. "Spectre Attacks: Exploiting Speculative Execution". In: *40th IEEE Symposium on Security and Privacy (S&P'19)*. 2019.

[11] M. G. Kuhn. "Cipher instruction search attack on the bus-encryption security microcontroller DS5002FP". In: *IEEE Transactions on Computers* 47.10 (Oct. 1998), pp. 1153–1157. ISSN: 0018-9340. DOI: 10.1109/12.729797.

[12] Moritz Lipp et al. “Meltdown: Reading Kernel Memory from User Space”. In: *27th USENIX Security Symposium (USENIX Security 18)*. 2018.

[13] Paul Marks. “”Dot-dash-diss: The gentleman hacker’s 1903 lulz””. In: *New Scientist* (Dec. 2011). URL: https://www.newscientist.com/article/mg21228440.700-dotdashdiss-the-gentleman-hackers-1903-lulz.html.

[14] L. Adleman R.L. Rivest A. Shamir. “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”. In: *Communications of the ACM* (Feb. 1978), pp. 120–126. DOI: 10.1145/359340.359342.

[15] M. Rostami, F. Koushanfar, and R. Karri. “A Primer on Hardware Security: Models, Methods, and Metrics”. In: *Proceedings of the IEEE* 102.8 (Aug. 2014), pp. 1283–1295. ISSN: 0018-9219.

[16] Pamela Samuelson. “The Law and Economics of Reverse Engineering”. 111 Yale L.J. 1575: Berkley Law, 2001.

[17] Sergei P. Skorobogatov. *Semi-invasive attacks – A new approach to hardware security analysis*. Tech. rep. UCAM-CL-TR-630. University of Cambridge, Computer Laboratory, Apr. 2005. URL: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-630.pdf.

[18] Mark Tehranipoor Swarup Bhunia. *Hardware Security: A Hands-On Learning Approach*. Morgan Kaufmann, Oct. 2018, p. 254.

[19] Efrat Raz-Moyal Tony Moor Ely Malyanker. “Single Die ’Hands-Free’ Layer-by-Layer Mechanical Deprocessing for Failure Analysis or Reverse Engineering”. In: (Nov. 2008).