TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Sicherheit in der Informationstechnik

Hardening Digital Circuits against Invasive Attacks with On-Chip Delay Measurements

Michael Weiner

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs

genehmigten Dissertation.

Vorsitzender:		apl. Prof. Dr.-Ing. habil. Helmut Gräb
Prüfer der Dissertation:	1.	Prof. Dr.-Ing. Georg Sigl
	2.	Assoc. Prof. Salvador Manich Bou, Ph.D.

Die Dissertation wurde am 15.06.2020 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 02.11.2020 angenommen.

"You cannot force ideas. Successful ideas are the result of slow growth. Ideas do not reach perfection in a day, no matter how much study is put upon them."
Alexander Graham Bell

Abstract

What do electronic passports, digital locking systems and car theft protection have in common? The answer is simple: their security is based on hardened embedded systems implementing cryptographic algorithms that promise to ensure confidentiality, integrity and/or availability of the protected system.

The level of security relies on many factors. The mathematical strength of state-of-the-art cryptographic algorithms with adequate key sizes is generally assumed to be good enough for the next years. It has therefore become interesting for attackers to target actual implementations rather than algorithms. An adversary can either exploit weaknesses in software, or he can conduct physical attacks: this means to attack the actual hardware.

Physical attacks either aim at carefully observing the attack target while it is performing a critical operation, or at disturbing it in a targeted way. This allows an attacker to extract secrets, such as intellectual property or cryptographic keys, without permission.

An elaborate, but especially powerful type of physical attacks is called microprobing. It means attaching microscopic needles, which are actually made for semiconductor testing and fault analysis, to the internal connections on a microprocessor chip die.

While there exist "secure elements", i.e. microprocessors that are specially designed to resist physical attacks, the protection against microprobing is difficult, because it directly attacks the inner workings of the chip. Therefore, protection against microprobing is an area of research with rising importance.

This thesis proposes two circuits that can detect the physical effects of microprobing, namely the additional delay introduced to a line by the added capacitive load of a probe needle. The first detection circuit is called Low Area Probing Detector (LAPD), and is optimized for a small area footprint. The second detector is called Calibratable Lightweight Invasive Attack Detector (CaLIAD); it proposes calibration to improve the sensitivity. A comparison with previous publications shows that the area of probing detectors is significantly reduced, while the sensitivity is on a comparable level, or, depending on the configuration, even better.

Finally, the thesis discusses challenges faced when integrating such a detector into a microprocessor, and proposes solutions, for example, to minimize the area overhead when protecting on-chip parallel buses.

Keywords: Physical Attacks, Microprobing, Invasive Attacks, Attack Sensors, Microprocessor Bus Systems.

Kurzfassung

Was haben elektronische Ausweisdokumente, digitale Schließanlagen und Autodiebstahlschutz gemeinsam? Die Antwort ist einfach: deren Sicherheit basiert auf gehärteten eingebetteten Systemen, die kryptografische Verfahren implementieren. Damit soll die Vertraulichkeit, Integrität und/oder die Verfügbarkeit des geschützten Systems sichergestellt sein.

Das Sicherheitsniveau hängt von vielen Faktoren ab. Die mathematische Stärke aktueller kryptografischer Verfahren wird bei adäquater Schlüssellänge als ausreichend für die nächsten Jahre betrachtet. Deshalb ist die Implementierung von Algorithmen anstelle der Algorithmen selbst in das Ziel von Angreifern gerückt. Ein Angreifer kann Schwachstellen in Software ausnutzen, oder er kann physische Angriffe durchführen: dies bedeutet, die Angriffe direkt gegen die Hardware zu richten.

Physische Angriffe zielen entweder darauf ab, kritische Operationen auf dem Angriffsziel genau zu beobachten, oder die Operation gezielt zu stören. Dies erlaubt es einem Angreifer, Geheimnisse wie z.B. geistiges Eigentum oder kryptografische Schlüssel unberechtigt zu extrahieren.

Obwohl es "Secure Elements" gibt, d.h. Mikroprozessoren, die speziell gegen physische Angriffe gehärtet sind, ist der Schutz gegen Microprobing schwierig, da hier die internen Funktionen des Chips selbst Ziel des Angriffs sind. Daher ist der Schutz gegen Microprobing ein Forschungsgebiet mit wachsender Wichtigkeit.

Diese Dissertation stellt zwei Schaltungen vor, die die physischen Effekte von Microprobing erkennen können, nämlich die Verlangsamung einer Leiung durch eine Probing-Nadel, die als zusätzliche kapazitive Last wirkt. Die erste Schaltung heißt Low Area Probing Detector (LAPD) und ist für eine kleine Chipfläche optimiert. Der zweite Detektor heißt Calibratable Lightweight Probing Detector (CaLIAD); es verwendet Kalibrierung, um die Sensitivität zu erhöhen. Ein Vergleich mit vorherigen Veröffentlichungen zeigt, dass die Fläche der Probing-Detektoren erheblich verkleinert werden konnte, während die Sensitivität je nach Konfiguration vergleichbar oder sogar besser ist.

Schließlich werden in dieser Arbeit Fragestellungen behandelt, die bei der Integration solcher Detektoren in Mikroprozessoren entstehen, zum Beispiel, wie man den Flächenbedarf klein hält, wenn man parallele chipinterne Busse schützen möchte.

Schlüsselworte: Physische Angriffe, Microprobing, Invasive Angriffe, Attack Sensors, Mikroprozessor-Bussysteme.

Acknowledgments

I would like to thank Prof. Dr.-Ing. Georg Sigl for supervising this thesis, and for all the assistance, advice and guidance during the course of this work. You did not only give me the opportunity to work on this thesis, but also made it possible that I can work on other research topics outside the main focus of the institute.

I would also like to thank Prof. Dr. Salvador Manich from the UPC Barcelona for the co-supervision. Thank you, Salvador, for introducing me to the area of analog circuit design in general, and to the research area of microprobing detection in particular. Thanks for all the long discussions, both in person and remotely. This work would not have been possible without you.

Salvador, also thank you for making the numerous stays in Barcelona possible, and for all the things you did to make me feel comfortable there. The time in Barcelona did not only help me advance in academic matters, it certainly also contributed to my development as a person.

I am especially grateful for the support and friendship I received from Salvador and his family (especially Carmen and Ramon) in personal matters during hard times.

Carmen and Ramon, thank you for providing accommodation and a family-like environment during my numerous stays in Barcelona!

I am also thankful to my employers, SimonsVoss and later BMW, for offering work conditions that facilitated writing this thesis, despite the fact that the timeline needed to be adjusted.

To all my colleagues: Thank you for all the fruitful discussions, both technical and personal, and for the friendships that arose. And a belated "sorry" for my numerous bad puns!

Nacho, thank you for the accommodation and the support when I started writing this thesis.

Finally, I would like to thank my family and friends for all their support.

Parts of this work were funded by the Spanish government projects TEC2010-18384 and TEC2013-J41209-P, and by the German Federal Ministry of Education and Research (BMBF) through the project SIBASE (01S13020A).

Contents

1 Introduction

Semiconductors have been used in security applications for more than 30 years. Public telephones were among their first applications, where they served as payment cards. In pay TV, they were required to decrypt video signals. As such security relevant semiconductors were most frequently embedded into plastic cards, the term "Smart Card" was coined for such cards with an embedded semiconductor.

Nowadays, the use cases of such systems are much more widely spread: contactless cards are used for various types of payment systems, secure microcontrollers control access to paid extra features of expensive products such as cars or test equipment, electronic keys serve as access control tokens, and smart cards are designed to store sensitive health information or are used for online identification with national identity cards.

Industry associations like the fido Alliance [fid] were founded to extend password based authentication, which is prone to password theft by phishing or malware, with hardware based security tokens.

As secure microcontrollers are now available in a broad variety of form factors, the term "Secure Element" has become prevalent for such controllers.

1.1 Types of Information Security Threats

Before considering the security of embedded systems in particular, it seems useful to get a broader overview of security in information processing systems. "Security" most commonly refers to confidentiality, integrity and/or availability that needs protection.

Confidentiality is, for example, the major design goal for pay TV providers: their goal is sharing their broadcast content only with the desired recipients, i.e. paying customers, while leaving the data unusable for others.

Besides confidentiality, online banking is a prominent example that highly depends on integrity: it ensures that the transaction details such as recepient or amount of money are not altered. This is why banks often use two independent devices to confirm the details of a transaction, such that the customer can detect changes in a transaction on the second device if the first device is compromised.

The importance of availability can be seen in recent cases where the Information Technology (IT) systems of hospitals were attacked. While this mostly targets the computers

in administration nowadays [BBC], there is no big step towards attacking embedded medical systems on which the lives of patients depend.

1.2 Classification of Attacks

Adversaries use different types of weaknesses for successful attacks. The *human factor* is one of them – unaware persons may be victim of attacks like phishing or social engineering, and thus open the door for further attacks. Whilst almost 20 years old, "The Art of Deception" gives quite vivid examples how social engineering can work in practice [MS02].

Another attack vector are weaknesses in *technical concepts*, such as cryptographic algorithms or protocols. While there exist sufficiently secure cryptographic algorithms nowadays, it is often seen that legacy systems or backwards compatibility is a gateway for attackers. There exists a large amount of cryptographic protocols, many of them are application specific and do not pass through a scientific review before being used in the field. This leads to a potentially high number of vulnerable cases. Just to name one example, the EMV standard used in card payments was prone to an attacks on the integrity of transactions [MDAB10, BCM+14]. In general, security problems with technical concepts can be challenging to solve once implemented and widely deployed, because conceptual updates often break compatibility with existing implementations.

Luckily, the conceptual level is not only a source of security problems. A good security design is also made at this level. Considerations for a good design may be, for example, the distribution and management of keys in a cryptographic protocol; keys that are not known by an entity can not be extracted by an attacker by any means. Another example for a good design goal is making security critical decisions in a secure environment, such as a secure backend, wherever possible.

There exists a broad variety of *implementation weaknesses*, especially, but not limited to, in software implementations. A typical example for flaws in this category are buffer overflows. Even though the technical advancement has already outdated many technical details, the book "The Art of Exploitation" can still be considered a good introduction to binary exploitation techniques in general [Eri08].

All previously described types of attacks exploit explicit mistakes that were made. However, an attacker can also make use of *intrinsic effects* on hardware level, for example the fact that the current consumption is influenced by the processed data, including secrets. This effect is called side channel leakage and can be exploited as, for example, discussed by Mangard et al. [MOP08]. Another attack vector – to be precise, one that motivated the work on this thesis – is the plain fact that the secrets need to be processed by a microprocessor; commercially available test equipment allows observing this even on a microscopic level.

1.3 Physical Attacks against Smart Cards

As described earlier, good security starts at the conceptual level. Ideally, hardware given out to customers does not even possess the knowledge or power to cause critical damage. Bank transactions, for example, need to be approved by a backend when a certain threshold is exceeded; Subscriber Identity Module (SIM) card duplicates in mobile networks can be detected by the operator.

There are many use cases, though, where these protection concepts cannot be applied easily. For example, pay-TV systems working over satellite have no backend connection that would allow to detect clones of cards.

It is therefore no surprise that attacks on the hardware level nourish an increasing demand for secure elements in embedded systems. However, what makes secure elements secure, in fact?

Three decades ago, when the first Smart Cards appeared, so did attacks against them: In the simplest case, their purpose could have been preventing to debit balance from phone cards, while more sophisticated attacks already aimed at full dumps to reveal algorithms and keys of cryptographic primitives. The methods used were quite simple in the beginning: Debiting balance could be prevented by disconnecting the programming voltage; ROM dumps were possible, for example, using glitching [AK96].

In the meantime, a circle of novel attacks and countermeasures have significantly improved the attack resistance of today's security microcontrollers: Glitch detectors as well as temperature and light sensors were added to detect fault attacks. When side channel attacks came up, massive efforts were spent on modeling and reducing the leakage at different abstraction layers. Today, the most sophisticated attacks of this kind appear to be localized electromagnetic attacks [HMH+12]; recent publications [HHM+14, HHM+15] have presented detectors of these attacks.

In 2010, Tarnovsky was able to carry out a full memory dump of a smart card controller by microprobing the bus [Tar10a]. This was successful in spite of its protective mesh.

Attacks like these are possible as only few countermeasures focus on the intrinsic effects caused by microprobing [MWS12, WHH+15]. The given thesis aims at filling this gap by providing circuit concepts and integration proposals aiming at a comprehensive detection of probing inside integrated circuits such as microcontrollers.

1.4 Contributions

Four main contributions are provided in this thesis.

The **Low Area Probing Detector** can detect microprobing attacks by evaluating race conditions between the delay introduced by the capacitive load of a probe and a delay

element. On a 65 nm technology, it can detect microprobes of 40 fF or less using only digital components, and its circuitry is extremely small compared to other solutions. It is marginally able to detect the best known microprobes; however, it cannot compensate manufacturing variations. The LAPD has been published in [WMS14, WMRMS18].

A drawback of the LAPD is the need to fine-tune its transistor dimensions to achieve the desired sensitivity. For this reason, the **Calibratable Lightweight Invasive Attack Detector** is presented. It allows to compensate manufacturing variations by means of calibration, and outperforms the LAPD in terms of sensitivity while avoiding the need to optimize the transistor dimensions. In its basic form, it also only consists of digital circuit components. An optional optimization can use reference capacitances to further increase the sensitivity. Its ability to make up for process and mismatch variations allows the detection margin to be reduced to 18 fF or less, depending on the mode of operation. The CaLIAD has been published in [WWL+19].

Both LAPD and CaLIAD in its plain form are presented as models for two lines, whereas the detectors shall be applicable to multi-line on-chip bus systems. This thesis presents concepts for **system integration**; this includes, for example, a discussion of where to place detectors from a security perspective, as well as a lightweight circuit to merge the results of multiple neighboring line comparisons into one result.

The last contribution of this thesis is **the analog simulation framework Simulation Automation Library for Verification and Analysis of Design Operating Regions (SALVADOR)** that can manage several tens of thousands of simulations. It consists of multiple components, one to parametrize netlists, another to manage the parallel execution of simulations, and a third to collect all results in a unified way for further post-processing. The development of this tool was motivated by the need for such simulations, especially during the work on the LAPD and the CaLIAD, where existing tools came to their limits. The lightweight yet flexible design, and the open source, may be helpful to other researchers to reproducibly process large data sets, also outside of the area of analog simulations. This made it appear appropriate to be presented as one of the main contributions, despite the lack of scientific novelty by itself. SALVADOR has been published in [WMB16].

1.5 Outline

Chapter 2 reviews the state of the art in invasive attacks and countermeasures. The LAPD is then presented in chapter 3. The CaLIAD, which achieves improved sensitivity by means of calibration, is introduced in chapter 4. Chapter 5 describes the challenges and provides conceptual solutions to integrated probing detectors into bus systems. The simulation framework SALVADOR is presented in chapter 6. Finally, chapter 7 concludes this thesis and discusses future work.

The appendix in chapter A presents derivations of electrical models related to microprobing.

2 Introduction to Invasive Attacks

Attacks on computer systems that target its hardware implementation, rather than algorithmic or software flaws, are called physical attacks. This section aims at giving an overview of the theory and practice of such attacks, especially those that are called *invasive*.

Section 2.1 provides a classification of attacks and gives examples for each type. The approach to attack real-life systems is discussed in section 2.2. Section 2.3 then presents countermeasures against invasive attacks; eventually, an electrical model of an invasive attack is given in section 2.4.

2.1 Classification of Physical Attacks

For the case of physical attacks against integrated circuits, Skorobogatov introduced the case distinction between non-invasive, semi-invasive and invasive attacks [Sko05]. Non-invasive attacks do not require decapsulation of the attacked integrated circuit, semi-invasive attacks need access to the chip surface but leave the passivation layer intact, and invasive attacks tamper with internal signals of an integrated circuit.

2.1.1 Non-Invasive Attacks

Non-invasive attack methods usually refer to operating devices outside its specification in order to trigger exploitable effects, and to most types of side-channel analysis. In spite of the relatively low effort required by an attacker, they are quite powerful against unprotected circuits and systems.

An attacker can provide a supply voltage, temperature or clock outside the specification. A prominent example for this type are cold-boot attacks [HSH+09] that allow attackers to read out the contents of Dynamic Random Access Memory (DRAM) modules after switching off their power; the bit error rate could be significantly reduced by cooling down the modules to about $-50\,°\mathrm{C}$ before powering off. Undervolting can be used, for example, to reduce the entropy of certain hardware-based random number generators [AK96].

Glitching is used to inject targeted faults into the control flow or the data by means of applying short voltage pulses to the supply net at a desired time instance, or by momentarily increasing the clock frequency beyond the specification. Electromagnetic pulses are another method of glitching [DDRT12, MDH+13]. Faults in the control flow often aim at conditional branch instructions, for example to bypass a Personal Identification Number (PIN) verification, to avoid leaving a loop that dumps memory contents to the outside [AK96], or to reduce the number of rounds executed in iterative cryptographic algorithms [AK98, WMT+13] to extract secret keys using basic cryptanalysis. Secret keys of known algorithms, or unknown actual algorithms with a known basic structure, can also be recovered by introducing faults into the data [BDL97, BS97].

Countermeasures against fault attacks can be classified into approaches that detect faults on a generic level, and detectors for specific attack types. The former adds redundancy, either in hardware [JL10, Inf12] or in software, for example by inserting and testing for checkpoints in the code, or by duplicating data [TMS+13]. Countermeasures against specific attack types include voltage and clock frequency detectors as well as using an internal clock, or dummy cycles that make it more difficult to time glitches; such countermeasures are already widely deployed in the field.

Prominent side-channel attacks are timing attacks [Koc96], including more subtle variants such as cache timing attacks [Ber05]. They exploit the fact that the runtime of an algorithm depends on a secret. Countermeasures include careful examination of the generated assembly code as well as improved cache designs [WL07].

Power analysis attacks measure the power consumption over time; variants use electromagnetic radiation as the source of information instead [QS01, AARR03]. They are classified into Simple Power Analysis (SPA) and Differential Power Analysis (DPA) [KJJ99]: Simple Power Analysis implies that secrets are visible directly from the power trace; this can be the case for the exponents of an RSA exponentiation. Differential Power Analysis makes secrets visible by classifying captured traces based on hypotheses of intermediate values that depend on secrets. The same approach is possible using the global electromagnetic radiation as a leakage source [QS01, GMO01]. As a countermeasure, masking aims at randomizing the processed data to prevent attackers from setting up hypotheses. Hiding tries to make the power consumption independent from the processed data by either equalizing it using specialized logic styles [PKZM07], or by adding noise.

2.1.2 Semi-Invasive Attacks

Semi-invasive attacks depend upon opening up the chip package, but leave the passivation layer intact as they do not need electrical contact. A classical example for these attack types use light pulses to inject faults: Low cost attackers employ flash units of cameras and expose the complete chip surface to the fault. A simple countermeasure against this type of attack are light detectors facing the top layer of the protected device.

More sophisticated adversaries take advantage of microscopes equipped with variable-power laser cutters to focus the pulses to the region of interest. Localized electromagnetic

attacks [HMH+12] are an example for semi-invasive attacks. They use inductive probes with a very close distance to the chip surface to exploit location dependent side channel leakage, which is used to circumvent conventional side-channel attack countermeasures. Such attacks can be detected by other types of detectors, as for example presented by Homma et al. [HHM+14, HHM+15].

Andrew Huang could remove the lock bits that prevent code readout from a microcontroller by exposing it to ultraviolet light in an Erasable Programmable Read Only Memory (EPROM) eraser. He had to cover the Flash area with electrical tape to prevent the code from being erased; furthermore, the controller needed to be tilted within the EPROM eraser to prevent the top layer metal covering the fuses from blocking the UV light [bH, SDK+13]. Nail polish can also be used instead of electrical tape. In general, however, this type of attack is only possible if an erased fuse corresponds to the unlocked state and if the lock bits are far enough away from the code and/or data memory. Considering these limitations during the design allows thwarting this type of attack [Zon16].

Eventually, Photonic Emission Analysis [SNK+12, KNSS13] uses the effect that switching transistors emit photons to reveal data processed by an attacked device. While this is semi-invasive by the definition of not tampering with the passivation layer, it requires significantly more effort than the other semi-invasive attacks, such as thinning the backside of the IC.

2.1.3 Invasive Attacks

Attacks that tamper with the passivation layer are called invasive. The most commonly known use case are attack methods that need electrical contact; examples include Focused Ion Beam (FIB) editing and attaching microprobes to lines containing secrets. The process of taking high-resolution chip pictures for the purpose of reverse engineering is also invasive, as it usually involves removing several metal layers, e.g. by means of polishing.

Despite the fact that invasive attacks are quite powerful, they are also economic due to the existence of a second-hand market of laboratory equipment and the availability of FIB laboratories paid per hour. Invasive attacks have not only been analyzed in academia; instead, hackers have successfully performed complete memory dumps of hardened devices [Tar10a].

2.2 Real-World Probing Attacks

This section gives an overview of probing attacks that have successfully been carried out against real-life targets. The knowledge about how attacks work is crucial to design

countermeasures to be effective; an improper design of countermeasures may not only be useless, it may even simplify attacks. Kömmerling, who used to be a pay-TV smart card hacker [Whi12], and Kuhn [KK99] mention an example of such a "countermeasure": It calculated the checksum of a memory region at each start-up; while this was designed to detect tampering with the memory contents, it turned out to be useful for an adversary aiming for the contents of this memory region – probing the data bus during checksum calculation would automatically reveal the data.

2.2.1 Reverse Engineering

Before actually probing a chip, an attacker needs to reverse-engineer it to identify the interesting probing targets. The authors of [KK99] describe the general workflow: At first, the package needs to be removed; then, a high-resolution chip image is created by stitching images captured with a digital camera connected to a microscope. Reverse-engineering custom hardware, such as cryptographic accelerators of unknown ciphers, may be assisted with the free software named degate [deg]. In the case of microcontrollers, bus lines – i.e. lines "that cross clearly visible module boundaries" such as Random Access Memory (RAM), Read Only Memory (ROM), or the Arithmetic Logic Unit (ALU) [KK99] – are a main target of interest, but also lead to other promising attack targets, such as instruction latches of the Central Processing Unit (CPU) core, which reveal the sequence of executed instructions to an attacker [Tar10b]. While optical microscopes were earlier sufficient for complete recovery of regions of interest, technologies from 130 nm or below usually require electron microscope or FIB imaging according to Tarnovsky [Tar10a]; Nohl sees the limit of optical imaging to be at 180 nm [Noh11]. In general, finding areas of interest may be assisted by commercially available databases such as the TechInsights Library, which claims to have "accurate semiconductor technical data and analysis" of "24.000+ dies" [Tec].

2.2.2 Hardware Setup of Probing Attacks

As soon as the targets have been identified, preparatory steps may be needed before actually probing the target lines. FIB processing may be required to uncover signals of interest, especially if meshes are used [Tar10a]. Eventually, microprobing is performed to listen, or to induce faults in order to, for example, repeat loops that dump memory beyond the intended iteration count [Tar08a]. An attacker uses a probe station together with micropositioners and microprobes for this purpose. A probe station consists of a microscope with high magnification and a flat surface for the micropositioners surrounding the stage with the device under attack.

The micropositioners are attached with a magnetic or vacuum fixation, and they are used to mount the actual microprobe; micrometer screws allow adjusting its position and placing it onto the line of interest. *Passive* microprobes consist of a tungsten tip

that makes electrical contact to the tested line as well as an electrical connection to the measurement device (e.g. oscilloscope or logic analyzer). In addition, *active* microprobes contain an amplifier circuitry that significantly reduces distortion of the probed lines. This is especially helpful to probe internal signals such as bus lines as their drivers are not designed to be strong enough to drive the wiring between tungsten tip and measurement device. However, their influence on the circuit is still visible; microprobe manufacturers usually model the parasitics of the tungsten tip and amplifier circuitry as a capacitive load and a leakage current.

The microprobe model that was found to have the smallest parasitics is called "Picoprobe 18C/19C" by GGB, Inc. [GGBa]. It exhibits an input capacitance of 20 fF if the specified transition time constraints are fulfilled, and a leakage current of 10 fA. While simulations have shown the effect of the leakage current to be negligible, the additional capacitive load on a bus line can be detected by precise timing measurements.

The tungsten wire connecting to the probe has a thickness in the range of 50 µm; its tip is sharpened to down to 0.1 µm [GGBa] by means of chemical etching. The small dimensions make the positioning of the probe fragile; in some occasions, for example when probing deeply buried lines from the front side or when the exposed surface of the target wire is small, one can use a FIB to make an L-shaped metal to allow a stable mechanical and electrical contact. Figure 2.1 shows this for a sample case of two lines.

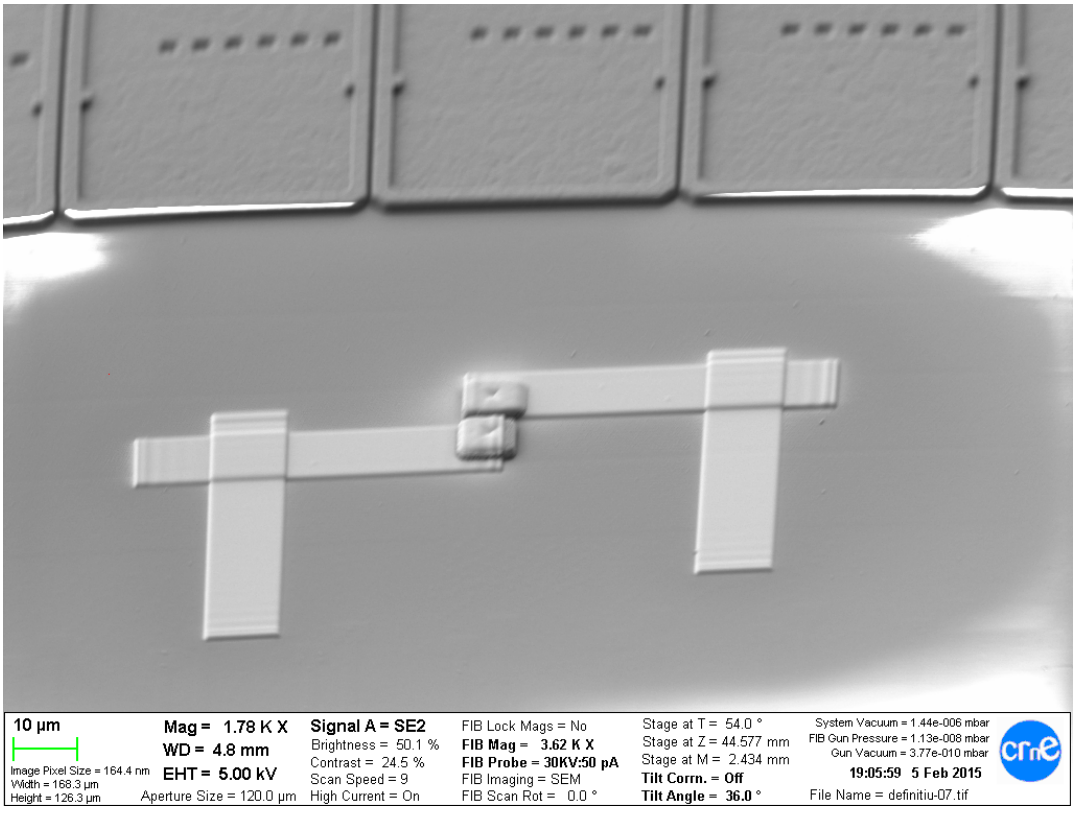


Figure 2.1: Supporting L-shape deposits to probe two lines.

2.2.3 Linear Code Extraction

When the attack objective is a memory dump, a technique called *Linear Code Extraction* can be used. In its simplest form, it uses two probe needles: one that sits on a bus line, and another one that prevents the CPU core from performing branches. To do this, an attacker desires to force the core to repeatedly execute any instruction that does not branch and does not access memory, such that the instruction fetch logic linearly walks over the address region of interest, and the actual instructions and operands, and in case of a Von-Neumann architecture also data, become visible to the probe on the bus one after another.

Forcing to repeat an instruction can be done by tampering with the clock input or clock enable line of the instruction latch. Alternatively, some architectures like ARM indicate in a single bit of an instruction whether a branch shall be executed – this bit can be overridden as well [MN12]. These steps need to be repeated as many times as there are bus lines, such that the results can be eventually merged.

The attacker needs to know the point in time at which he needs to tamper with the instruction latch such that he sees the memory region of interest on the bus. There may be cases in which finding this point is challenging as well, i.e. when an Memory Management Unit (MMU) needs to be configured before the region of interest is reached.

2.3 Countermeasures

As a first line of defense against invasive attacks, circuit camouflaging is proposed to increase the effort of reverse-engineering netlists from high resolution chip images [RSSK13]. This means implementing certain gates in such a way that gates of different type (e.g. NOR or NAND) have the same appearance, such that the logic function cannot be determined from the optical image. However, on the one hand, this does not prevent attackers from observing the interconnection architecture, e.g. from searching for sets of long lines to find buses. On the other hand, the authors of [EMGT15] show how camouflaged circuits can be attacked efficiently using Boolean satisfiability problem (SAT) solvers, and claims that IC camouflaging shall be considered by IC designers with "strong caution".

Protection techniques against microprobing in smartcard controllers can be classified into three categories: one can either devalue the outcome of probing, e.g. by using bus encryption or masking, one can obstruct physical access to target lines, or one can detect inherent effects of probes.

In the field of practical attacks against real-world chips, talks given at hacker conferences, especially by Christopher Tarnovsky [Tar08b, Tar08a, Tar10a, Tar10b, Tar19a, Tar19b], give a deep insight about the mechanics of attacking smartcard processors and the evolution of countermeasures. Tarnovsky became known as a smartcard hacker in the

early 1990ies; later on, he worked for a Pay TV company before he founded his own company, Flylogic, which he eventually sold. In his talks, Tarnovsky talks very openly about his methods and attack targets.

2.3.1 Masking Techniques

According to Tarnovsky [Tar10b], MC68HC05SC24 type processors had the internal bus available on unbonded pads in the early 1990ies, such that no probe station was necessary at all for successfully dumping the chip. A very simple obfuscation method fixed this issue by permuting the bits of the opcode.

A controller from Infineon, which was mentioned to be used in pay TV systems, XORed three instruction bits with the three least significant address bits to mask the executed instruction. OKIDATA 8051-type processors added fake tracks that were not electrically connected to anything, but were possibly aimed at distracting reverse engineers or at hiding important signals underneath.

In academia, the authors of [ISW03] apply Multi-Party Computation techniques to mask signals; however, the circuit complexity increases by $O(n^2)$ in the general case for protecting against probing n lines simultaneously. The authors themselves put the practicability of their approach in question. Furthermore, protection against fault injection would require additional complexity. A more lightweight masking scheme is proposed by Infineon [GG14]. It performs linear transformations on bus lines, which would need an attacker to probe many lines to compute the value of one single bit. As this does not prevent recombination of sequentially captured traces of single lines, they propose to introduce a distinction between uncritical and critical data; in the latter case, a subset of the bus lines shall transfer a random mask rather than using the full bus width for the payload.

Another interesting masking approach was developed by Infineon. According to Infineon design engineers, the growing number of specialized attack types raised the requirement to implement countermeasures on a more generic level rather than detecting specific attacks [JL10]. The new concept is called IntegrityGuard and is implemented in the SLE78 and SLC52G product families. The public documentation [Inf12] promises a "fully encrypted data path, including encrypted calculation in the CPU itself" ensuring that "no more plain data is left on the chip". Furthermore, two CPU cores "constantly check each other to establish whether the other unit is functioning correctly"; this aims to detect fault injection at a generic level instead of sensing for known sources of error, e.g. as this is done with light detectors. This is one of the most comprehensive approaches found on security controllers, and as of now, no public information about successful attacks was found.

On the other hand, Christopher Tarnovsky calls this architecture "über-hyped" in the Flylogic blog [Tar13] and accuses Infineon of having "sacrificed physical security" compared to the "good old trustable" predecessor SLE66P. Compared to previous blog posts,

which contain high-resolution images and detailed weakness explanations of other chips, this entry remains vague with respect to technical details.

In general, it can be considered difficult to avoid single points of failure with masking or encryption schemes. For example, even if memories and buses are encrypted, instructions have to be available in the clear at the instruction decoder. At least in the past, all successful attacks could be performed by probing the bus with one needle that walks along the different lines; additional needles were only required to inject faults and/or to capture the instruction latch clock. This was also true for one of the highest-ranked attack targets in the public: The Infineon SLE66PE could be dumped by only using two needles [Tar10a], one for inducing momentary faults, and another for the data bus, probing one line after another.

2.3.2 Obstruction of Physical Access

Obstructing access to target lines can be done in a passive way, e.g. by metal fillings or passive shields, or by active shields. Passive shields can be removed by FIB machines without any negative effects on functionality. Active shields usually drive test patterns through a mesh on the top layer and verify that the patterns reach the other ends of the mesh lines.

In a controller manufactured by Taiwan Semiconductor Manufacturing Company (TSMC), a grid of passive filler metal was added at the top layer [Tar10b]. Drilling into this grid with a FIB does not affect operation; furthermore, it can even serve as a coordinate system to help locating regions of interest. If it was intended as a countermeasure, it seems to have been designed improperly and simplifies the work of an attacker.

The first active mesh appeared on a controller from the ST16 family by STMicroelectronics [Tar10b]. This active part of that mesh was single line tied to V_{DD}. When damaged, the CPU core would halt, but an attacker could make the chip functional again by restoring the connection from this line to V_{DD}.

More complex meshes, such as, for example, used in Atmel/Microchip CryptoCompanion or in the Infineon SLE66PE [Tar10a], can be reverse-engineered by a technique called *voltage contrasting* with an electron microscope image and a FIB: disconnecting the driver of one line makes this line floating; on the resulting image, the line turns black. This allows to reverse-engineer convoluted interconnection structures without following vias.

Cioranesco et al. suggested to use cryptographic Pseudo-Random Number Generators (PRNGs) to provide a large number of unpredictable test signals [CDG+14]. However, this comes with an increased hardware cost, and it can likely be circumvented by adding bypass lines on top of the passivation layer using a FIB. Infineon patented a capacitive coupling detector that can be used to detect FIB-editing of adjacent mesh lines that are

expected to have a large mutual capacitance [LT04]. A shield that measures the capacitive coupling of the mesh lines is presented by Wan et al. [WHH+15]. This approach, however, needs large reference capacitors.

Other approaches try to bury security critical signals underneath other functional, but non-critical lines. Shi et al. present an algorithm to determine the exposure of critical lines [SAFT16]. Wang et al. introduce a workflow to generate shield nets from productive signals that do not carry critical information [WSN+19]. Still, this does not appear as the overall solution: zero exposure of target lines is hard to reach, especially if designers want to avoid multiple layout iterations, which is critical for fast time-to-market. Also, bypassing cut lines above the top layer is still feasible for an attacker.

2.3.3 Probing Detection

All of the described countermeasures do not protect against probing attacks from the backside. This vulnerability can be avoided if the inherent effects of invasive attacks such as probing are detected, as it can be done by observing the capacitive load of a probe. That way, probing can be detected no matter whether extensive FIB editing was used to uncover target lines, or whether a probe was connected on the back side. The only approach that detects such attacks and that has been evaluated with respect to process, voltage and temperature variation is the Probe Attempt Detector by Manich et al. [MWS12].

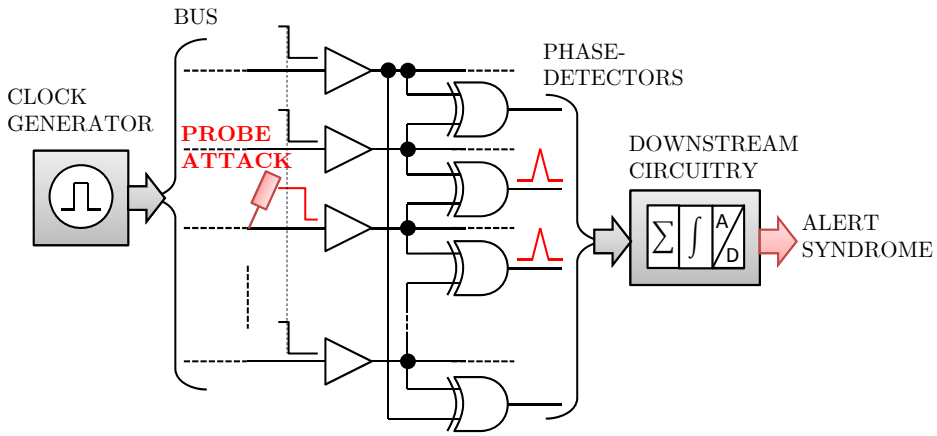


Figure 2.2: Probe Attempt Detector (PAD) overview.

In figure 2.2, an overview of the detector is shown. When the Probe Attempt Detector (PAD) is running, a periodic signal is sent simultaneously through all protected lines. Before starting, these lines are disconnected from their regular function. At the outputs, XOR gates compare the state of the lines and if transitions arrive with different propagation delays, they generate pulses of a width proportional to the delay difference. A downstream circuitry adds all these pulses, integrates over time and generates a digital

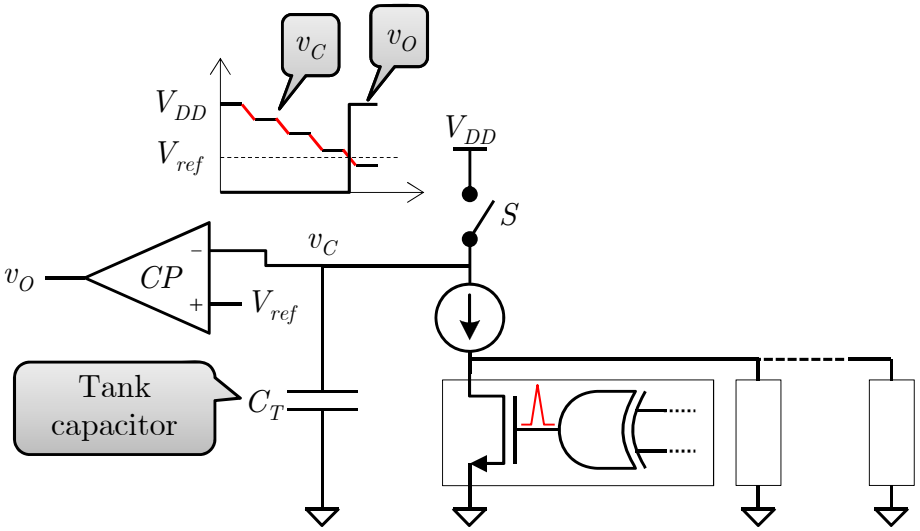


Figure 2.3: Probe Attempt Detector (PAD) detector circuit.

alert syndrome. Because of the differential mode, the response of the PAD does not depend on the number of buffers inserted in the bus lines.

Figure 2.3 illustrates a simplified model of the PAD downstream circuitry. A tank capacitor C_T with the initial charge $C_T V_{DD}$ is gradually discharged by the pulses coming from the XOR gates. When the pulses arrive, they switch on n-channel Metal Oxide Semiconductor (nMOS) transistors which in turn extract some charge from C_T through a current source; therefore, the amount of charge discharged from the capacitor is proportional to the 'active' time of the nMOS transistors. Initially, when the detector starts, C_T is charged to the maximum voltage V_{DD} through switch S. Then, the switch is opened and the XOR gates start comparing signals coming from the bus until C_T is discharged below a given threshold. If the arrival times of the XOR inputs are mutually delayed by a probe, the XOR gates generate pulses accordingly which in turn gradually discharge the capacitor. A comparator CP raises its output when the voltage v_C goes below the threshold V_{ref}. A probing attack alert is activated when this signal is raised at a lower number of clock pulses than normal.

The necessity of a large tank capacitor that needs to be charged and discharged still comes with an area, power and timing overhead that prevents it from being used in ultra low resource applications. Also, it does not allow its implementation in programmable logic platforms like Field Programmable Gate Arrays (FPGAs).

2.4 Electrical Model of Probes

It has been assumed that attaching a probe to a line can be modeled as a lumped capacitance C_A connected to it. The validity of this simple model may not be inherently

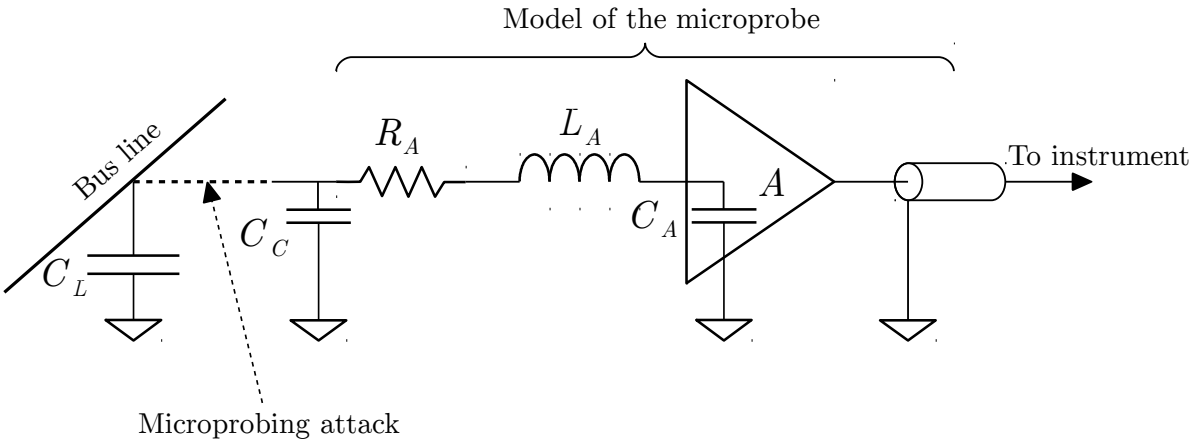


Figure 2.4: Electrical model of a microprobe attack.

evident, considering the order of magnitudes of the different components existing in the attack.

Therefore, this section presents an an extended electrical model of the microprobe AC behavior connected to a bus line. It is depicted in figure 2.4. After the attack, the parasitic components added to the bus line having an intrinsic capacitance of C_L are constituted by: C_C (contact capacitance), R_A (contact resistance of the microprobe), L_A (tip inductance) and C_A (amplifier capacitance). If the microprobe is of an active type, it will have an amplifier (A) close to the tip which will decouple impedances for the rest of the instrumentation.

The contact capacitance C_C is produced during the preparation of the attack. L shape platinum landing pads are deposited on the surface of the chip whose objectives are twofold: stabilize the contact of the microprobe and ease the contact to the bus line through the drilled vias. From our experience, we know that values lower than 10 fF may be expected.

The contact resistance R_A of the microprobe is highly dependent on the landing stability. The adversary aims to have a contact as stable as possible which maximizes the signal to noise ratio of the microprobe. Under this condition, values lower than 1 Ω are expected in tungsten tips [ZZM99]. The inductance of the tip L_A is caused by the piece of wire transmitting the signal to the amplifier. For short wires, about 2 mm in air, typical values of 100 nH can be obtained [CBB+03]. The amplifier capacitance C_A is provided in data-sheets of microprobes and is highly dependent on amplifier technologies. Minimum values of 20 fF have been found for the technology [GGBa].

After the attack, the bus line is loaded with two admittances which induce current in the bus drivers and therefore generate the corresponding delays in the transmission of the signals.

$$\begin{aligned} Y_{BL} &= Y_L + \Delta Y \\ \Delta Y &= Y_C + Y_A \end{aligned} \tag{2.1}$$

Y_{BL} is the total admittance of the bus line and ΔY is the added contribution due to the attack. A circuit analysis shows that the obtained admittances are

$$\begin{aligned} Y_C &= j\omega C_C \\ Y_A &= \frac{j\omega C_A}{\chi} \\ \Delta Y &= j\omega(C_C + \frac{C_A}{\chi}) \\ \chi &= (1 - \omega^2 L_A C_A) + j\omega C_A R_A \end{aligned} \tag{2.2}$$

The term χ is a complex dimensionless number that modifies the reactive admittance of C_A as a function of L_A, R_A and the signal frequency ω. For low frequencies its real part tends to 1 and the imaginary part tends to 0, so that the only significant contribution of the microprobe becomes C_A, and R_A and L_A can be neglected since $\chi \rightsquigarrow 1$. In effect, the microprobe is a second order filter with a given resonant frequency $\omega_c = \sqrt{L_A C_A}^{-1}$. If the excitation frequency is far below the resonant one, $\omega \ll \omega_c$ (as in our case), the gain tends to unity and the behavior becomes entirely reactive, depending only on the capacitor C_A.

For a probe band-width of 350 MHz [GGBa], and considering the maximum values assumed before for the components, the estimation of χ is

$$\chi = 0.9903 + j0.00004398 \simeq 1$$

and therefore it can be assumed that

$$\Delta Y \simeq j\omega(C_C + C_A) > j\omega C_A$$

in which the right term of the inequality is the simplified admittance considered here. Therefore, the assumption that the load contribution of the attack is due only to C_A, is a conservative strategy that assures the reliable detection of the microprobing attack. Therefore, a real attack will produce a larger load on the bus line and thus a delay larger than modeled here.

2.5 Conclusion

This chapter has summarized the state of the art in invasive attacks. It started with an overview of physical attacks to explain how invasive attacks can be classified in this area, and then gives examples of attacks against devices in the field in academia and in the hacker scene. It is shown here that microprobing serves as an important tool for real-world attacks having the goal of, for example, extracting firmware from secure microcontrollers.

The following section focused on countermeasures. On the one hand, one can observe a movement from countermeasures against specific attacks, such as fault injection by light pulses, to more generic countermeasures such as error detection schemes that are unaware of the source of error. On the other hand, detectors that are specific to intrinsic effects of probing are considered to gain importance in order to thwart the power of microprobing attacks. The Probe Attempt Detector, one of the first microprobing detectors, is described in more detail; it has a good detection performance, but depends on large analog circuit components. Finally, the electrical model of a microprobe is given. It shows that the parasitic properties of such a probe can be expressed as a lumped capacitance C_A.

This leads to the question whether probing detectors can be shrunk in size by avoiding large analog capacitors. A solution is discussed in chapter 3; however, the solution discussed there is not capable of compensating manufacturing variations, and therefore, the sensitivity leaves space for improvement. A calibratable digital sensor circuit is presented as an approach to combine a high sensitivity with the avoidance of large analog components; this is presented in chapter 4. Lastly, another open question is how probing detectors can be integrated into actual bus systems. This topic is addressed in 5.

3 The Low Area Probing Detector

The PAD [MWS12] has been published as a detector of the intrinsic effects of microprobing. It was the first circuit discussed in academia that is conceptually able to detect backside probing attacks; however, it needs a large tank capacitor, which has a considerable area usage, and which may not be available in purely digital technologies. That may be a limitation in bulk products such as SIM cards, where the price is a limiting factor.

Reducing the area overhead of microprobing detectors may open the market for such low cost commodity products. This section presents the concept of a Low Area Probing Detector (LAPD) that only consists of a few gates and therefore keeps the area and power overhead low. A demonstration of its reliability is given here with respect to process variations and varying environmental conditions. Furthermore, a small-scale optimization is carried out to increase the reliability beyond its initial limits.

Section 3.1 will give a brief description of the LAPD. The results and discussion of reliability are presented in section 3.2. Eventually, the chapter is concluded in section 3.3.

Parts of this chapter have been published in [WMS14, WMRMS18, HWPG18]. The derivative based optimization from section 3.2.6 was carried out as a joint work with Andreas Herrmann from the Chair of Electronic Design Automation at the Technical University of Munich.

3.1 Circuit Concept

As described in section 2.4, a microprobe attached to a line on a semiconductor can be modeled as a small parasitic capacitance; this increases the rise and fall times of the transmitted signals. Considering a set of lines that are symmetric with respect to dimensions and timing, probing *one* of the lines introduces a small timing asymmetry between the probed line and the unprobed lines. The Low Area Probing Detector can measure such timing differences and raise an alarm if they are beyond normal noise or manufacturing variations. This increases the complexity of a microprobing attack: If B lines are protected by the LAPD, $B-1$ microprobes can be detected such that the adversary would need to attach the same capacitive load to all B protected lines. This is assumed to be an effective countermeasure against practical probing attacks, as

the space for micropositioners on a probe station is limited and the measurement setup becomes more and more fragile with each additional probe. Tarnovsky, for example, preferred using only two probes for a successful attack, even though this implied a significant post processing overhead [Tar10a].

The LAPD performs pair-wise comparisons, so the next two sections will focus on the case of two lines. Chapter 5 will then show how a set of B lines can be protected.

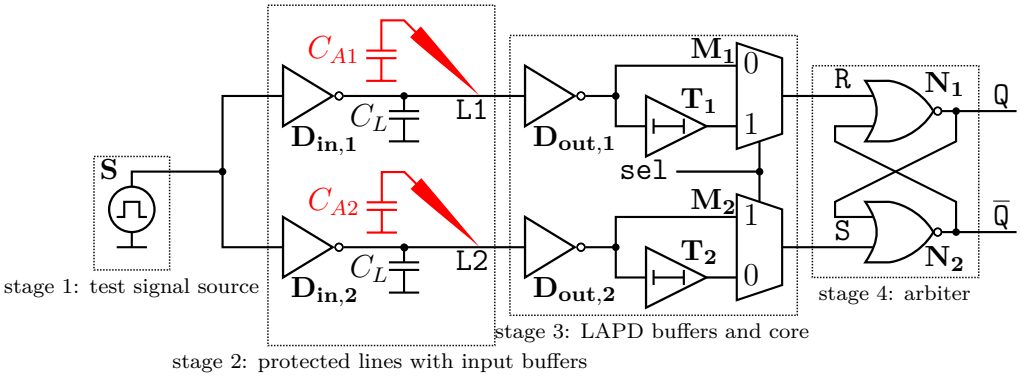


Figure 3.1: Schematic of the Low Area Probing Detector.

3.1.1 Principle of Operation

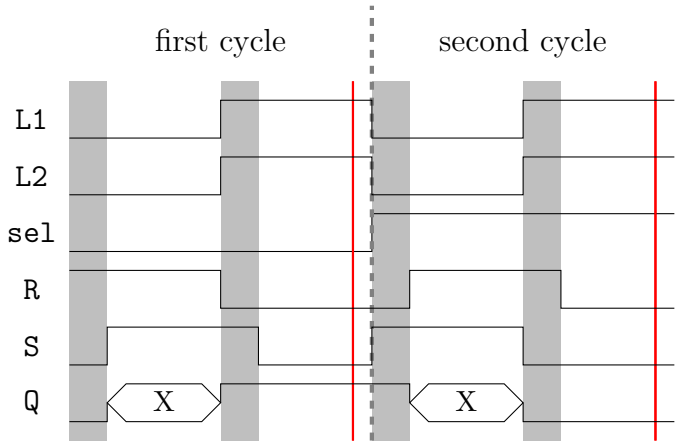
The LAPD compares the delays of two lines by alternatively introducing an intentional delay t_D to each one of the lines and then verifying that the delayed line is effectively slower than the line without intentional delay.

In figure 3.1, the full circuit is shown. Bold letters represent gate instances, typewriter letters represent line names and italic letters represent capacitances. The different stages of the LAPD are indicated by dotted squares.

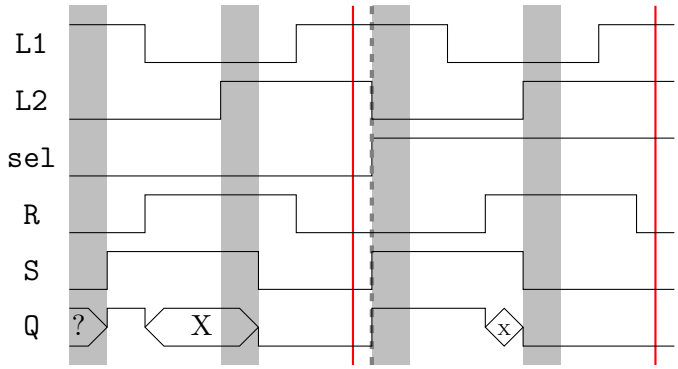
The signal source **S** in stage 1 generates test pulses that are fed to the lines under test `L1` and `L2`. In stage 3, a combination of multiplexers **M** and delay elements **T** of delay t_D allows alternately delaying one of the lines at a time through signal `sel`. Finally, the arbiter in stage 4, which consists of gates **N**, decides who "wins" the race. Under normal conditions, both lines "win" alternately; however, one line is always winning if an imbalance of more than t_D is introduced by a probe. For signal values `sel` $= 0/1$, the latch output `Q` exhibits the values Q_1/Q_2 respectively, as described in section 3.1.2.

The arbiter is implemented by a NOR RS latch. In one test cycle, both latch inputs are first set to the active state (1); after that, both inputs change to the inactive state (0). After the transition, the output is determined by the input signal that had been active for longer: If the `R` input remains active longer than the `S` input, `Q` becomes 0 and vice versa.

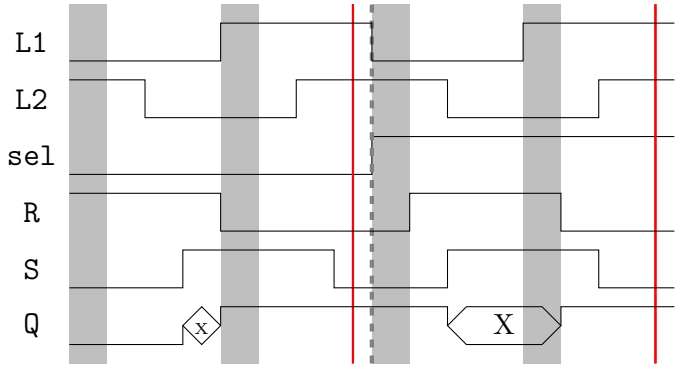
The complete LAPD timing is presented in figure 3.2. It shows two test cycles: In the first cycle, `L2` is delayed by element $\mathbf{T_2}$ while `L1` is directly passed through. This is



(a) No attack.

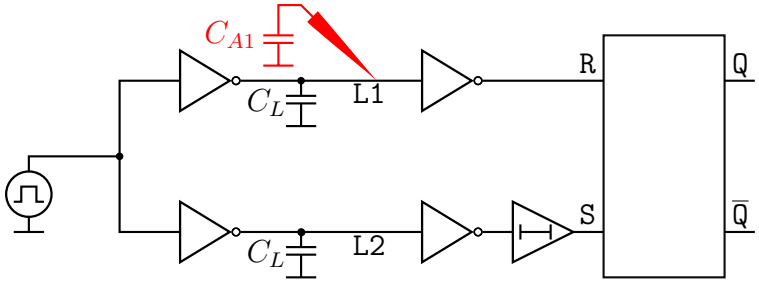


(b) L1 probed.

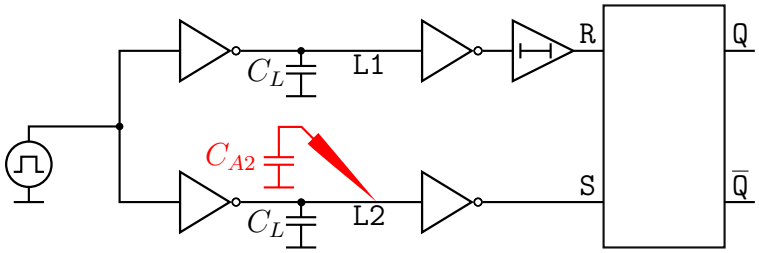


(c) L2 probed.

Figure 3.2: LAPD timing; gray bars denote intentionally introduced delay t_D; red bars represent latch output sampling time.



(a) first cycle: detection of excess of capacitance at line 1.



(b) second cycle: detection of excess of capacitance at line 2.

Figure 3.3: Simplified LAPD schematic for the two test cycles.

Table 3.1: Summary of latch outputs in the two test cycles when probing one line.

	Q_1	Q_2
no probe attached	1	0
probe at L1	0	0
probe at L2	1	1

shown in the simplified circuit diagram in figure 3.3a. This cycle can detect an excess of capacitance at L1, which is shown in the timing diagram in figure 3.2b. This diagram shows that the default latch output in the first cycles is $Q_1 = 1$, but when a probe is attached to L1, the output becomes $Q_1 = 0$.

In the second cycle, which is depicted in the simplified circuit diagram in figure 3.3b, the delay introduced by element $\mathbf{T_1}$ is applied to L1 while L2 is directly passed through. That cycle can detect an excess of capacitance at L2, which is shown in the timing diagram in figure 3.2c. Here, the default output of the latch is $Q_2 = 0$, but when a probe is attached, the output becomes $Q_2 = 1$.

The possible LAPD latch outputs are summarized in table 3.1.

Note that when both latch inputs R and S are simultaneously high, both latch outputs Q and $\overline{\text{Q}}$ are low. However, the output behavior may vary under this condition depending on the latch implementation type. Therefore, the output state of the latch is considered invalid in that period. This is denoted by X in the timing diagrams.

3.1.2 LAPD Model

In order to compare the delay between two lines it is assumed that both have an intrinsic parasitic capacitance of C_L. While the previous example assumed that only one line is attacked at a time, the LAPD is in fact sensitive to the *difference* of capacitive loads. Therefore, the model is stated more generically, assuming that *two* microprobes with parasitic capacitances of C_{A1} and C_{A2} are attached to `L1` and `L2` respectively. As a result, the effective capacitances of the lines during the attack are

$$C_1 = C_L + C_{A1} \tag{3.1}$$
$$C_2 = C_L + C_{A2} \tag{3.2}$$

Using the alpha-power model described in the appendix A.1, the delay difference between the probed and the unprobed lines is

$$\Delta t_{L1,L2} = \Omega\ (C_2 - C_1) = \Omega\ (C_{A2} - C_{A1}) \tag{3.3}$$

where Ω summarizes the technological parameters of the transistors and the supply voltage.

In a first approximation, the delay difference is proportional to the difference of attack capacitances $C_{A2} - C_{A1}$, as shown in equation (3.3). The alpha-power model approach works better for small values of C_1 and C_2. State-of-the-art semiconductor microprobes, as, for example, offered by GGB Industries, Inc. [GGBa, GGBb], have parasitic capacitances in the range of tens of femtofarads and therefore can be assumed to be small enough for the approximation. Microprobes with a larger attack capacitance may disturb regular operation of the circuit and thus not be suitable for successful microprobing attacks; furthermore, the delay function is also monotonic outside the boundaries of the small-value approximation of Equation (3.3), and therefore a reliable LAPD operation can be expected.

After the bus, the $\mathbf{D_{out}}$ inverters increase the slew rate to minimize the effects of different switching thresholds of the multiplexers $\mathbf{M}$. $\mathbf{D_{out}}$ also scales the delay difference. To model the nominal case, one can write

$$\Delta t_{\overline{L1},\overline{L2}} = k_{\mathbf{D_{out}}} \cdot \Delta t_{L1,L2} \tag{3.4}$$

where $\Delta t_{\overline{L1},\overline{L2}}$ is the delay difference observed after the $\mathbf{D_{out}}$ inverters.

After the $\mathbf{D_{out}}$ inverters, the transitions pass through $\mathbf{T}$ and $\mathbf{M}$ before they reach the RS latch, therefore the delay difference at the latch inputs can be expressed as follows:

$$\Delta t_{RS} = \Delta t_{\overline{L1},\overline{L2}} \pm t_D + (t_{\mathbf{M2}} - t_{\mathbf{M1}}) \tag{3.5}$$

t_D is the delay introduced by the delay element $\mathbf{T}$. In the two cycles shown in figure 3.2, it is alternated between the `R` and `S` inputs of the latch. In both cycles, one multiplexer is fed by $\mathbf{D_{out}}$ while the other multiplexer is driven by $\mathbf{T}$. Their outputs have slightly

different transition times, which may affect the multiplexer timing behavior as well; this is modeled by the expression $t_{\mathbf{M2}} - t_{\mathbf{M1}}$. Note that Δt_{RS} shall be interpreted as "time from `R` transition to `S` transition"; in other words, if $\Delta t_{RS} > 0$, then `S` is active longer than `R`, and if $\Delta t_{RS} < 0$, then `R` is active longer than `S`. The difference $(t_{\mathbf{M2}} - t_{\mathbf{M1}})$ models the imbalances of the multiplexers **M** due to different slew rates at the input.

Inserting (3.3) and (3.4) into (3.5), it follows

$$\Delta t_{RS} = k_{\mathbf{D_{out}}} \Omega \cdot (C_{A2} - C_{A1}) \pm t_D + t_{\mathbf{M2}} - t_{\mathbf{M1}} \tag{3.6}$$

The latch needs to have a minimum distance between the falling edges to produce a reliable output; this distance can be compared to the hold time of a flipflop. Therefore,

$$|\Delta t_{RS}| > t_H \tag{3.7}$$

holds, where t_H is the "hold time" of the latch.

Before (3.6) is inserted into (3.7), it is necessary to distinguish between two cases:

Case 1: When $C_{A2} \leq C_{A1}$ holds, the first cycle (`sel` $= 0$) is used to detect a probe. In this case, the delay element $\mathbf{T_2}$ is attached to the `S` input of the latch, which implies Δt_{RS} to be increased; therefore, t_D in equation (3.6) has a positive sign. From this, one can obtain the two inequalities

$$C_{A1} - C_{A2} < \frac{t_D + t_{\mathbf{M2}} - t_{\mathbf{M1}} - t_H}{\Omega \; k_{\mathbf{D_{out}}}} \tag{3.8}$$

$$C_{A1} - C_{A2} > \frac{t_D + t_{\mathbf{M2}} - t_{\mathbf{M1}} + t_H}{\Omega \; k_{\mathbf{D_{out}}}} \tag{3.9}$$

where inequality (3.8) refers to the case that reliably does not raise an alarm, and inequality (3.9) denotes the case that does raise an alarm reliably.

Case 2: When $C_{A2} > C_{A1}$ holds, the second cycle (`sel` $= 1$) is able to detect a probe. In that case, the delay element $\mathbf{T_1}$ is connected to the `R` input of the latch, thus causing a reduction of Δt_{RS}. Consequently, t_D in equation (3.6) has a negative sign. In that case, inequality (3.10) describes the upper bound of capacitive difference for not triggering an alarm in a reliable way; inequality (3.11) is the lower bound to trigger an alarm reliably.

$$C_{A2} - C_{A1} < \frac{t_D + t_{\mathbf{M1}} - t_{\mathbf{M2}} - t_H}{\Omega \; k_{\mathbf{D_{out}}}} \tag{3.10}$$

$$C_{A2} - C_{A1} > \frac{t_D + t_{\mathbf{M1}} - t_{\mathbf{M2}} + t_H}{\Omega \; k_{\mathbf{D_{out}}}} \tag{3.11}$$

Note that this model can only be considered as a first approximation of the performance of an LAPD implementation. Simulations provide a significantly better accuracy; it is difficult to find an accurate closed-form model of circuits that make use of analog race conditions due to the complexity of transistors. However, the model may still be helpful in cases where certain designs need to be selected out of a larger set for further analysis by simulation. The accuracy of the model is evaluated in the appendix section A.3.

Table 3.2: Qualitative advantages of the LAPD against other state-of-the-art probing protection.

alternative contermeasure	advantages of the LAPD
meshes	no additional layer protection against backside attacks
bus encryption	no latency
PAD [MWS12]	no large capacitor
general	only few gates hardware overhead

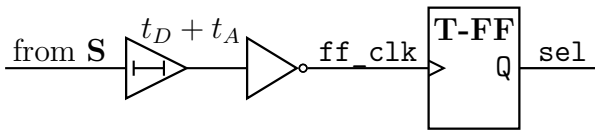


Figure 3.4: Example LAPD control logic.

3.1.3 Error Compensation

Manufacturing variations as well as varying environmental conditions lead to inter-sample variation of the threshold capacitance value that decides between "alarm" and "no alarm". In this context, the following two types of errors should be considered:

- errors upon which an alarm is raised when the circuit is in fact not being attacked. These errors are called false positives or type I errors.
- errors upon which no alarm is raised when the circuit is in fact being attacked. These errors are called false negatives or type II errors.

These types of errors will be analysed in the next section.

In this section, the concept of a Low Area Probing Detector was described. Its simple construction allows it to be implemented in a very lightweight manner. The advantages of the LAPD over other protection concepts are qualitatively summarized in table 3.2.

3.1.4 Control and Evaluation Logic

The signals required to control the LAPD, and to capture the results, can be derived from the output of the test signal source **S**. `sel` controls whether latch input `R` or `S` shall be delayed. Figure 3.4 depicts an example circuit for `sel`. It is generated by a toggle flip-flop clocked by a delayed, inverted output of the test signal generator **S**. The rising edge of the T flip-flop clock `ff_clk` shall occur after the falling edge of the delayed LAPD latch input. An additional delay t_A ensures this condition.

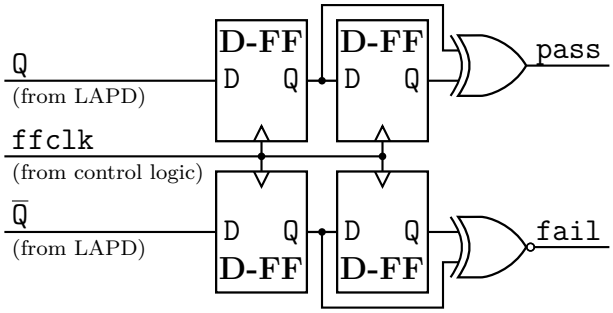


Figure 3.5: Example redundant LAPD evaluation logic.

On the output side of the latch, the evaluation logic shall provide feedback about the absence or presence of a probe. Conceptually, this is a PASS/FAIL signal where PASS means that `Q` toggles every cycle and FAIL indicates that `Q` remains at a constant value over two subsequent cycles. Implementing a single PASS/FAIL output line is dangerous, though: if an attacker would force such a line to a constant PASS, for example by the means of a second microbe, the LAPD would become obsolete.

The circuit as provided in Figure 3.5 has two redundant outputs `pass` and `fail` to avoid this single point of failure. It is fed by the signals `Q` and $\overline{\texttt{Q}}$ and uses the clock `ff_clk` coming from the control logic. As a positive side effect of the symmetry of the evaluation logic, both outputs of the LAPD latch are equally loaded, which avoids introducing a bias to the circuit.

3.2 Results

The LAPD has been implemented in a 65 nm STMicroelectronics technology with a core voltage of 1.2V. For the gates, low power standard threshold voltage transistors `psvtlp` and `nsvtlp` were used. Simulations were performed using Cadence spectre 11.1 on a machine with four AMD Opteron 6274 CPUs and 256 GB RAM.

SALVADOR was used to automate the simulation workflow in a reproducible way, and collect the results for post-processing, such as plotting the detection charts. Details of SALVADOR are explained in chapter 6.

Due to the symmetric property of the LAPD, it was considered sufficient to only simulate an attack to `L1`; this decreased the simulation runtime approximately by a factor of 2, which was especially helpful in the early exploration phase when different transistor dimensions and circuit variants were tested. Therefore, simulations were using the following assumptions:

$$C_{A1} = C_A \tag{3.12}$$
$$C_{A2} = 0 \tag{3.13}$$

3.2.1 Nominal simulation

In a first run of nominal simulations, an ambient temperature of $\vartheta = 27\,°\mathrm{C}$ was assumed. All nMOS transistors in the design had an aspect ratio $\frac{W}{L} = 10$. The intrinsic line capacitance was assumed as $C_L = 100\,\mathrm{fF}$. This corresponds to a line length of approximately 1.3 mm on the top metal layer in the used technology, assuming an adjacent `GND` line with minimum distance.

In the case that the delay elements **T** are implemented as chains of two inverters, the minimum detected attack capacitance is $C_A^* = 10.3\,\mathrm{fF}$. For the case of four inverters, the minimum value becomes $C_A^* = 23.4\,\mathrm{fF}$.

3.2.2 Effects of local variations

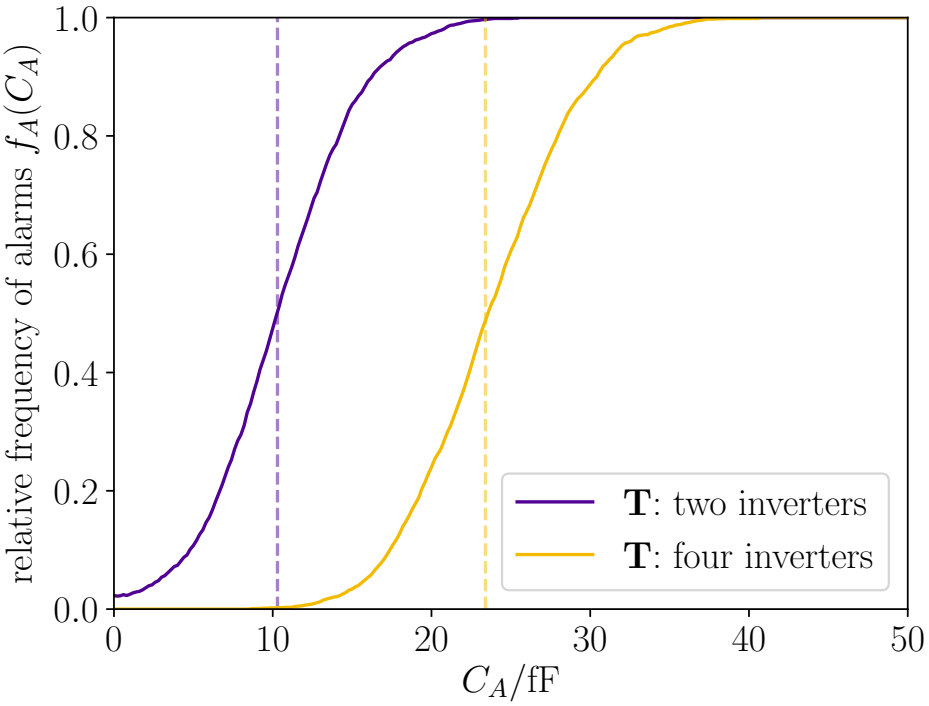


Figure 3.6: Relative alarm frequency for different implementations of delay elements **T** (nominal transition values are dashed).

As the LAPD works in differential mode, a Monte-Carlo analysis of the mismatch variations was performed using $N = 2000$ samples. Figure 3.6 shows the result for the two implementations, again with delay elements **T** consisting of two and four inverters respectively. The x axis represents the attack capacitance and the y axis denotes the relative frequency of alarms, which is defined as

$$f_A(C_A) = \frac{A}{N} \tag{3.14}$$

A is the number of Monte-Carlo instances rising an alarm, and N is the total number of Monte-Carlo simulations.

The following qualitative observations can be made from the figure:

- The two-inverter delay implementation exhibits a non-zero alarm frequency for $C_A = 0$.
- both implementations exhibit an uncertainty region ΔC_A^U in which the behavior of the circuit is not well predictable.

The uncertainty region is defined as

$$\Delta C_A^U = C_{reliableAlarm} - C_{reliableNoAlarm} \tag{3.15}$$

with the following helper variables:

$$f_A(C_{reliableAlarm}) > 1 - \epsilon \tag{3.16}$$
$$f_A(C_{reliableNoAlarm}) < \epsilon \tag{3.17}$$

By visual inspection of figure 3.6 one can estimate an uncertainty region of $\Delta C_A^U \approx 25\,\text{fF}$. Please note that state-of-the-art microprobes by the commercial supplier GGB Industries can all be detected by the four-inverter implementation: While the *Picoprobe Model 18C/19C* [GGBa] are declared to have a minimum input capacitance of 20 fF, the data sheet constrains this property to signals with a transition time lower than 3 ns, while its input capacitance is 60 fF for transition times below 1 ns. Further analysis of the simulation results shows that the maximum transition time for $C_A = 50\,\text{fF}$ is smaller than 0.6 ns. The second best microprobes with respect to input capacitance are called *Picoprobe Model 28/29* [GGBb] which exhibit $C_A = 40\,\text{fF}$ regardless of the slew rates of the probed signals.

While the two-inverter **T** implementation can marginally detect all probes according to their specification, its uncertainty region ΔC_A^U is still significant. The size of this region determines both the likelihood of false positives and false negatives, and hence the reliability of the circuit. As conversations with industry representatives have suggested that reliability is one of the most important design goals, it seems desirable to know how much the uncertainty region ΔC_A^U can be narrowed by optimizing the LAPD. For the sake of reliability, the four inverter implementation was chosen as a starting point as it does not show a non-zero alarm rate at $C_A = 0$.

To get a better understanding about the effects of variations, the first target of analysis were the effects of variation of each LAPD stage on the alarm threshold. In a first set of simulations, the variance of the delay difference between the two latch inputs $\text{Var}(\Delta t_{RS})$ at $C_A = 0$ was observed to quantify this variation. Δt_{RS} is a good indicator for the reliability of the circuit: its distribution can be mapped to the probability of the latch output `Q` being 0 or 1, however, Δt_{RS} is continuous, and therefore exhibits a non-zero $\text{Var}(\Delta t_{RS})$ at any sweep point of C_A, rather than only in the uncertainty region. This

Table 3.3: Variance of timing differences at latch inputs of four-inverter implementation.

	$\mathrm{Var}(\Delta t_{RS,sel=0})$
all variations enabled	$8.15 \times 10^{-23}\,\mathrm{s}^2$
w/o $\mathbf{D_{in}}$ variation	$5.32 \times 10^{-23}\,\mathrm{s}^2$
w/o $\mathbf{D_{out}}$ variation	$2.43 \times 10^{-23}\,\mathrm{s}^2$
w/o $\mathbf{M}$ variation	$8.00 \times 10^{-23}\,\mathrm{s}^2$
w/o $\mathbf{T}$ variation	$7.89 \times 10^{-23}\,\mathrm{s}^2$

allows efficiently analyzing the sensitivity of the LAPD stages by means of simulation with one single C_A value; sweeping over C_A is not required. Here, the value $C_A = 0\,\mathrm{fF}$ was used.

A technology feature allows to selectively switch off variations for single transistors – this was employed to selectively disable variations stage by stage and quantify the influence on $\mathrm{Var}(\Delta t_{RS})$. The results of both cases `sel` $= 0$ and `sel` $= 1$ were captured, but only minor differences could be observed, so the following explanations are focused on the first case `sel` $= 0$ for simplicity. The results for the four inverter implementation of the delay element $\mathbf{T}$ are shown in table 3.3. $\Delta t_{RS,sel=0}$ refers to the time from the edge at the `R` signal to the edge of the `S` signal in the first of the two test cycles (thus `sel=0`); the bold letters in the table refer to the gates in figure 3.1. Notice that disabling the variations in the buffer stage $\mathbf{D_{out}}$ significantly reduces the variance of Δt_{RS}. Therefore, this stage is assumed to have the highest influence on reliability at the selected design point. A model for this behavior is given in the appendix A.2.

3.2.3 Reliability Metric

Prior to dimensioning the LAPD, a reliability metric is introduced that allows comparing the quality of different LAPD implementations. This metric q is defined as the area between the ideal curve of an LAPD having a detection threshold C_A^* and the curve of an actual implementation.

$$q = \int_0^{C_A^*} f_A(C_A)\,dC_A + \int_{C_A^*}^{C_{max}} (1 - f_A(C_A))\,dC_A \tag{3.18}$$

with

$$f_A(C_A^*) = 0.5 \tag{3.19}$$

This approach takes all C_A sweep values that were simulated into consideration and thus minimizes numerical noise. For reasons of computational complexity, the boundary of this area is chosen to be $[0; C_{max}]$. Equation (3.19) centers the threshold C_A^* between the two integrals in equation (3.18) around the intrinsic 50% alarm frequency of a circuit. With this, the metric effectively prefers a low uncertainty range over a predefined alarm

threshold. The condition $f_A(0) < \epsilon$ was defined as an additional filter criterion to sort out false positives. Figure 3.7 illustrates the metric for the four-inverter implementation. The plot was generated using Matplotlib [Hun07].

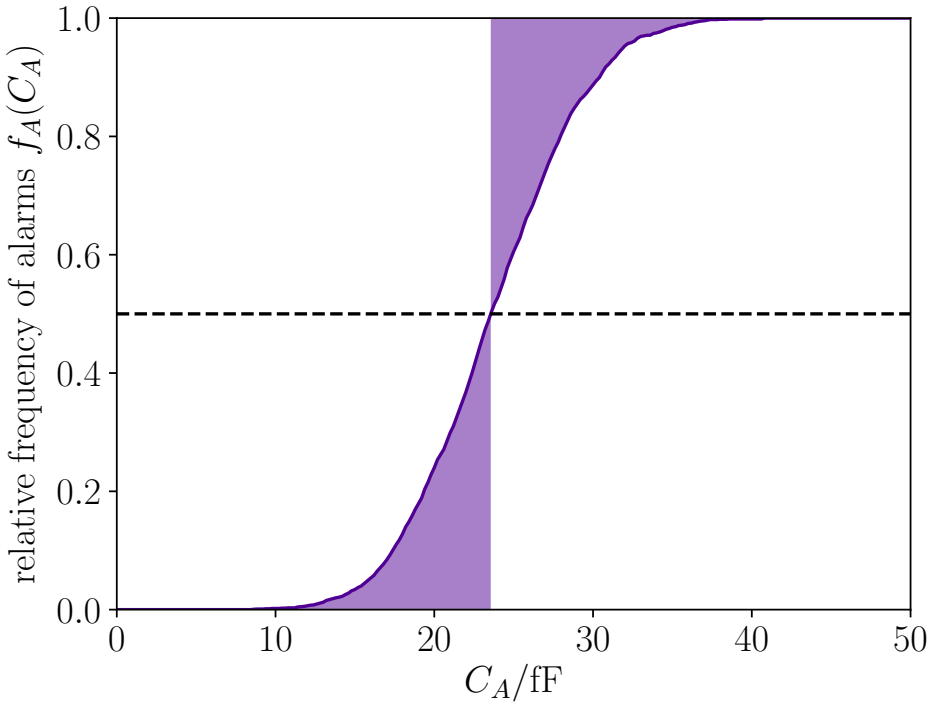


Figure 3.7: Illustration of the reliability metric of the four inverter implementation ($\frac{q}{\text{fF}} = 4.02$).

3.2.4 LAPD dimensioning

A simple optimization was performed on the four-inverter implementation to estimate the minimum C_A that can be detected reliably. As shown before, variations in stage $\mathbf{D_{out}}$ appear to have the strongest influence on the delay difference variations. For this reason, it was analyzed how much the overall reliability can be improved by fine-tuning the transistor dimensions of this stage. In a new series of simulations, a coarse sweep was performed over C_A as well as the aspect ratio and the channel length to select good candidates for a further finer analysis.

$$\frac{W}{L} \in \{10, 20, 50, 100\}$$

$$L \in \{1, 2, 5, 10\} \cdot L_{min}$$

$L_{min} = 0.06\,\mu\text{m}$ is the minimum channel length as needed to be specified in the simulator. The results of this set of simulations, for which a step size of $\Delta C_A = 5\,\text{fF}$ and 200 Monte-Carlo iterations were used at each point, is shown in table 3.4.

Table 3.4: Threshold capacitance and reliability metric for different $\mathbf{D_{out}}$ dimensions ($\Delta C_A = 5\,\mathrm{fF}$, 200 Monte-Carlo runs).

$\frac{W}{L}$	$\frac{L}{L_{min}}$	$\frac{C_A^*}{\mathrm{fF}}$	$\frac{q}{\mathrm{fF}}$
10	1	23.5	4.46
10	**2**	**23.6**	**3.58**
10	**5**	**24.1**	**3.48**
10	10	23.3	4.80
20	1	25.3	3.67
20	**2**	**24.9**	**3.32**
20	5	24.8	3.85
20	10	24.8	6.72
50	**1**	**26.6**	**3.22**
50	**2**	**26.3**	**3.38**
50	5	25.7	5.43
50	10	24.5	12.47
100	**1**	**26.9**	**3.45**
100	2	26.5	3.89
100	5	25.5	8.15
100	10	17.5	18.53

The "50% alarm capacitance" C_A^*, which is defined in equation (3.19), has been estimated by linear interpolation. q is the quality metric defined in equation (3.18). The best six rows (in bold) with respect to the reliability metric have been selected for a more detailed analysis with 2000 Monte-Carlo iterations and a step size of $\Delta C_A = 0.2\,\mathrm{fF}$.

The results of the finer analysis are shown in table 3.5; the separated row represents the initial design, the following lines show the results after optimization. The meaning of the first four columns is the same as in table 3.4, and two more columns contain additional threshold values described in the following paragraph. The best case with respect to the quality metric is highlighted in bold.

These results shall be used to estimate the real alarm probability $p_A(C_A)$ based on the absolute number of alarms A, the number of Monte-Carlo simulations N as well as the desired confidence level α that was assumed as $\alpha = 0.01$. The Wilson method [Wil27] was used to estimate the confidence intervals. Based on these, a "1% alarm thresh-

Table 3.5: Threshold capacitance and reliability metric for different $\mathbf{D_{out}}$ dimensions ($\Delta C_A = 0.2\,\text{fF}$, 2000 Monte-Carlo runs).

$\frac{W}{L}$	$\frac{L}{L_{min}}$	$\frac{C_A^*}{\text{fF}}$	$\frac{q}{\text{fF}}$	$\frac{C_{0.01}}{\text{fF}}$	$\frac{C_{0.99}}{\text{fF}}$
10	1	23.6	4.02	11.2	37.2
10	2	23.8	2.95	14.2	33.4
10	5	24.3	2.92	15.0	34.2
20	**2**	**25.3**	**2.77**	**16.4**	**34.4**
50	1	26.8	2.82	17.4	36.0
50	2	26.4	2.93	17.0	36.6
100	1	27.3	2.83	18.2	37.0

Table 3.6: Analysis of corners.

		typical	worst case	worst case at corners
initial	$C_{0.01}$	11.2 fF	7.4 fF	FF, 0 °C, 1.08 V
	$C_{0.99}$	37.2 fF	41.6 fF	SS, 85 °C, 1.08 V
optimized	$C_{0.01}$	16.4 fF	12.0 fF	FF, 0 °C, 1.08 V
	$C_{0.99}$	34.4 fF	39.4 fF	SS, 85 °C, 1.08 V

old" $C_{0.01}$ and a "99% alarm threshold" $C_{0.99}$ were estimated, such that the following inequalities hold:

$$p_A(C_{0.01}) < 0.01 \tag{3.20}$$
$$p_A(C_{0.99}) > 0.99 \tag{3.21}$$

Compared to the initial design, the quality metric of the best case has improved by more than 40 %. If the uncertainty region ΔC_A^U is defined as $\Delta C_A^U = C_{0.99} - C_{0.01}$, then the reduction of this region is also a little more than 40%. In other words, after improvement, there is 40% more margin with respect to timing jitter or C_L imbalance, and the delay elements $\mathbf{T}$ can also be tuned more effectively towards a lower C_A^* without increasing the number of false positives too much.

Figure 3.8 shows the curve of the optimized implementation next to the initial design. The reduction of the uncertainty region is also clearly visible in this figure.

3.2.5 Corners

The behavior of the LAPD after dimensioning has been analyzed with respect to process, voltage and temperature corners. The process corner points used have

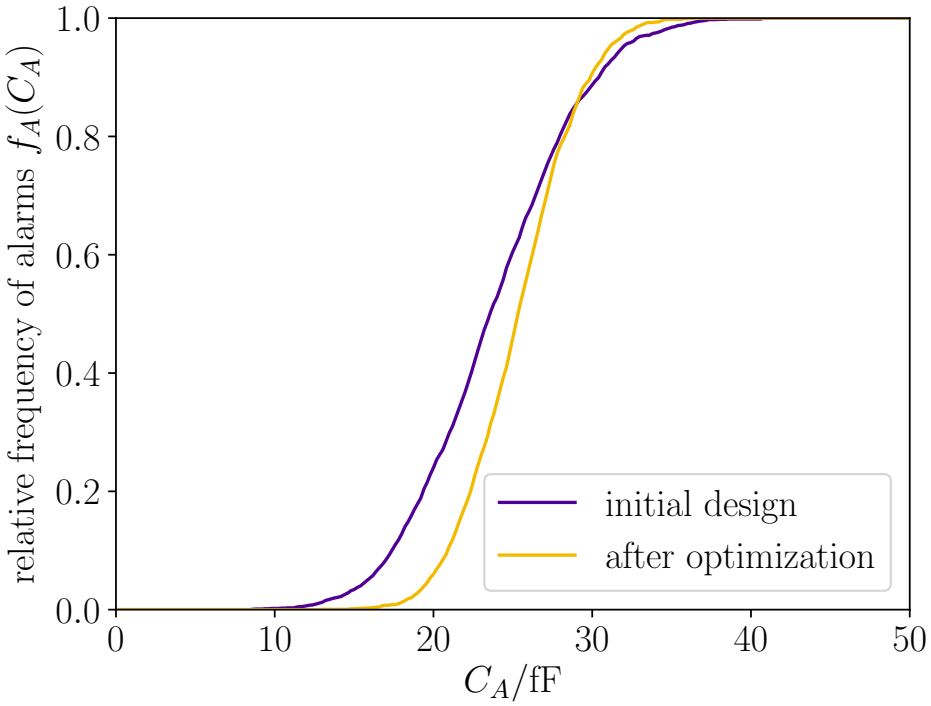


Figure 3.8: Relative frequency of alarms of the best circuit, compared to the initial design.

been `SS` (slow-slow), `TT` (typical-typical) and `FF` (fast-fast); for the temperature, $\vartheta \in \{0\,°C, 27\,°C, 85\,°C\}$ were used; for the voltage, the tested values were $V_{DD} \in \{1.08\,V, 1.2\,V, 1.32\,V\}$.

The analysis of corners was used to determine the worst-case values of $C_{0.01}$ and $C_{0.99}$. The results are shown in table 3.6. The two rows labeled "optimized" represent the corner cases of the optimized design ($\frac{W}{L} = 20, \frac{L}{L_{min}} = 2$). For reference, the values of the initial design ($\frac{W}{L} = 10, \frac{L}{L_{min}} = 1$) are also given.

One can see that the initial design fails to detect a 40 fF probe in the worst case, while the optimized design has a worst case $C_{0.99}$ slightly below this value. Also, it can be stated that the worst case $C_{0.01}$ keeps away far enough from $C_A = 0$. The worst case uncertainty region ΔC_A^U has reduced from 34.2 fF to 27.4 fF, which is an improvement by about 20 %. This shows that it is possible to achieve reliable operation of the LAPD without being obliged to use dedicated optimization tools.

3.2.6 Derivative based Optimization

In addition to the previously described manual optimization, a derivative based optimization using the electronic design automation tool WiCkeD was carried out [HWPG18]. As shown in figure 3.9, this shows significantly better results than the

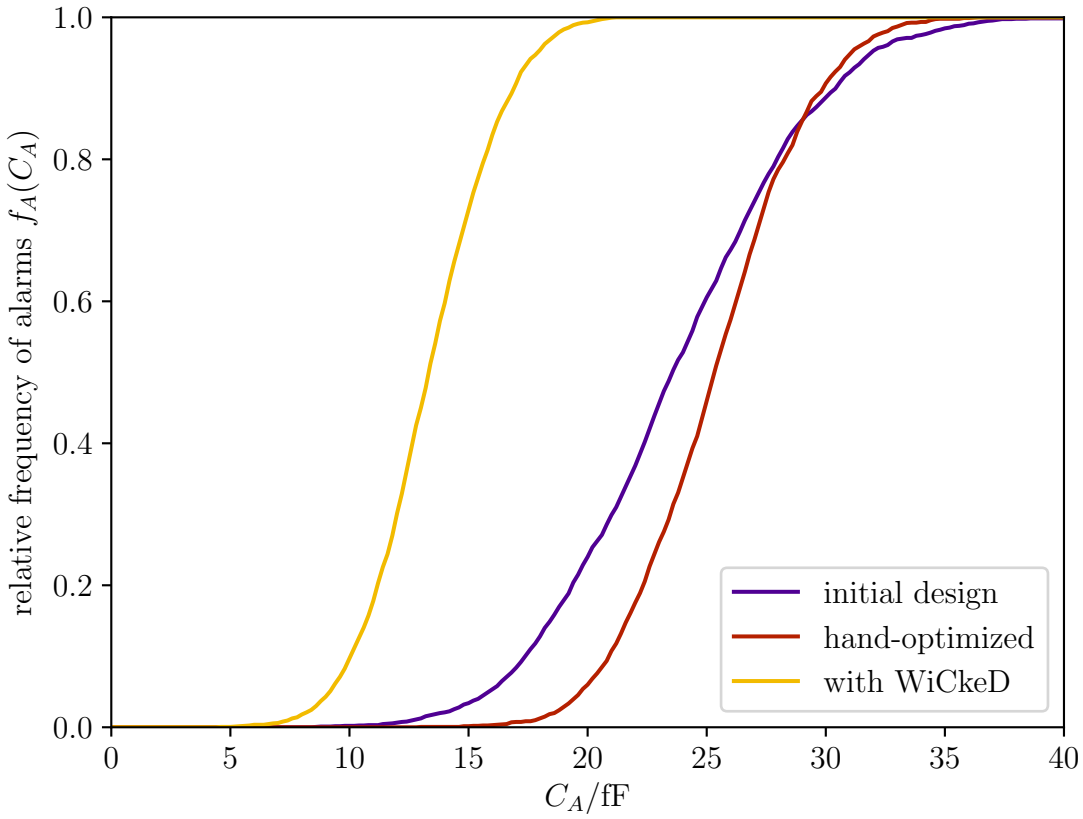


Figure 3.9: LAPD detection performance after derivative based optimization with WiCkeD.

hand optimized variant. Its nominal 1% alarm threshold is $C_{0.01} = 6.4\,\text{fF}$; the 99% alarm threshold is $C_{0.99} = 20.4\,\text{fF}$.

The methodology described here characterizes the latch using Monte-Carlo simulations in a first step to obtain a relationship between "timing difference of input signals" and "probability of the latch output being 0 or 1". In a second step, the remaining circuit shall be optimized, using the timing constraints obtained from the first step. This two-step approach is necessary as the optimization tool must be able to extract a performance metric based on *one* simulation – which is true for timing behavior, but not for latch output probabilities, as they need Monte-Carlo simulations.

Note that in the actual optimization, the latch was mistakingly included in the optimization with respect to area, such that the latch timing behavior characterized in the first step is different from the resulting latch after optimiztion. However, this does not affect the detection performance described here, as for the final evaluation, the LAPD circuit is assessed as a whole.

Table 3.7: Area, Timing and Energy Comparison of Cryptographically Secure Shields, PAD and LAPD; B denotes the number of bus lines to be protected.

	area a [GE]	cycles	energy [fJ]
CSS [CDG+14]	8081	–	$7.01 \times 10^{12}\,\mathrm{s}^{-1}$
PAD [MWS12]	549 (for $B = 2$)	50-100	n/a
LAPD	$\leq 48 \cdot B$ (for $B > 2$)	2	981 (for $B = 2$)
optimized LAPD (section 3.2.6)	$\leq 109 \cdot B$ (for $B > 2$)	2	1131 (for $B = 2$)

3.2.7 Resource Usage

A quantitative area, timing and energy consumption comparison between the Cryptographically Secure Shields (CSS) by Cioranesco et al. [CDG+14], the Probe Attempt Detector [MWS12] and the LAPD is shown in table 3.7. All three implementations are available in the same STMicroelectronics 65 nm technology (note that the description in [MWS12] uses a 180 nm implementation of the PAD, and the 65 nm implementation is currently not published). The LAPD dimensions in terms of Gate Equivalents were determined by normalizing to the sum of transistor dimensions of the smallest size standard cell NAND gate `HS65_LS_NAND2X2`. The dimensions of the CSS and the PAD, both after layout, were normalized to the layout area of the same NAND gate.

Note that the exact size figures are difficult to compare, as, for example, the CSS figures are taken from a complete Application Specific Integrated Circuit (ASIC) implementation, whereas the LAPD figures only conver the bus drivers and the detector. Therefore, the actual LAPD sizing depends on the number of lines as well as how the integration is done, which will be discussed in chapter 5. The given figures refer to the case of parallel detector instances (for details, see section 5.3.4) and can be considered as an upper bound of area consumption. In general, however, it can be seen that the LAPD is one order of magnitude smaller than the PAD and more than two orders of magnitude smaller than the CSS.

The CSS are designed to run continuously, while PAD and LAPD shall only be used prior to security critical operations. For this reason, the energy consumption of the CSS are given per second. Compared to the PAD, the LAPD is faster by a factor of 25 to 50. The energy consumption of the LAPD was simulated for one test run at $C_A = 0$ and two bus lines. Even assuming that the CSS would, for example, only run for one millisecond, its energy consumption is larger than the one of the LAPD by several orders of magnitude.

3.2.8 Error Compensation

Error probability bounds have been provided based on simulations of the aforementioned variations. However, the computational complexity of the simulations only allows to provide such bounds in the magnitude of 10^{-2}. This is still quite high when considering mass production, especially because this probability only covers a small part of the chip functionality. If all components of a chip had an error probability in this magninude, the overall error probability would be significantly worse.

Assuming statistical independence, voting schemes can be used to significantly improve the error probability, for example as proposed by Parhami [Par94]:

- local variations can be compensated by providing k LAPD instances
- timing jitter can be compensated by repeating the evaluation k times

Note that the assumption of statistical independence is not true for global variations as well as voltage and temperature variations; however, focussing on the local variations can already lead to a significant improvement due to the differential mode of operation of the LAPD.

As an example, majority voting can be used as voting scheme. In this case, k should be odd such that at least $\frac{k+1}{2}$ alarm votes are required to raise an alarm. If the alarm probability of a single LAPD instance evaluation at a certain operating point is assumed $p_A(C_A)$, the alarm probability after voting follows a binomial distribution:

$$\begin{aligned} p_A^k(C_A) &= P\left(X \geq \frac{k+1}{2}\right) \\ &= \sum_{i=\frac{k+1}{2}}^{k} \binom{k}{i} p_A(C_A)^i \left(1 - p_A(C_A)\right)^{k-i} \end{aligned} \tag{3.22}$$

This distribution shows a tendency towards its extremes:

$$\lim_{k\to\infty} p_A^k(C_A) = \begin{cases} 0 & p_A(C_A) < 0.5 \\ 0.5 & p_A(C_A) = 0.5 \\ 1 & p_A(C_A) > 0.5 \end{cases} \tag{3.23}$$

Assuming that an alarm for a specific C_A for which $p_A(C_A) < 0.5$ holds is called false positive, figure 3.10 allows to quantify the reduction of false positives. For example, voting with $k = 5$ for a single-instance alarm probability $p_A(C_A) = 10^{-2}$ leads to an overall alarm probability of $p_A^k(C_A) = 10^{-5}$. Likewise, this approach also reduces false negatives.

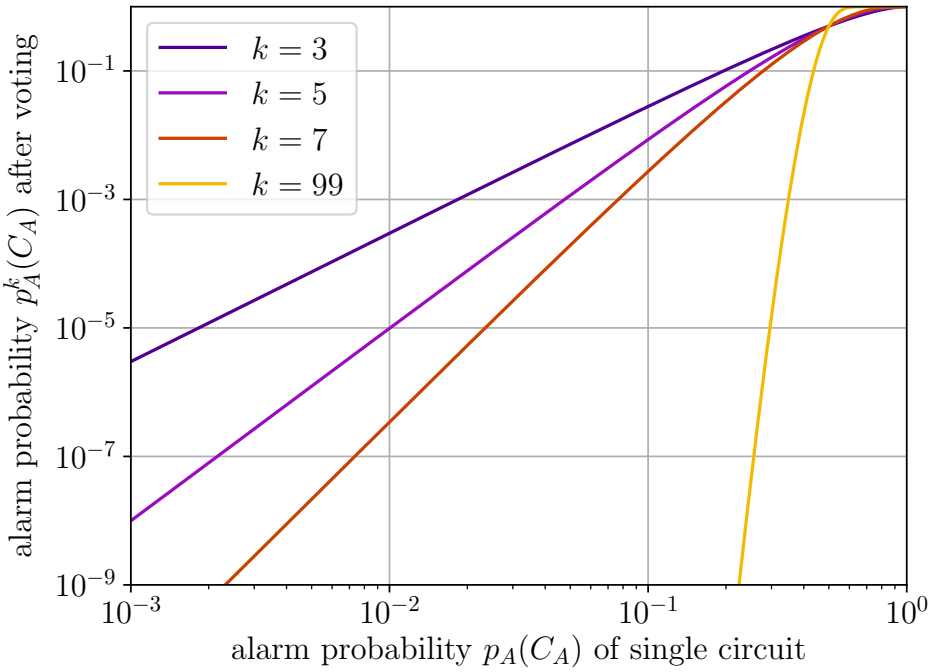


Figure 3.10: Alarm probability after majority voting.

3.3 Conclusion

This chapter has described the concept of a low area probing detector, which only consists of a few gates and has a significantly lower area than other protection mechanisms such as the PAD [MWS12] or bus encryption.

The reliability of such a detector has been analyzed with respect to local variations as well as process, voltage and temperature corners using Monte-Carlo simulations on a 65 nm technology. The results of these simulations have been used to estimate the regions of probe capacitances in which the circuit gives reliable results. These results show that an initial LAPD implementation can detect state-of-the-art commercial microprobes under typical conditions, but possibly not in worst case scenarios.

A simple optimization of the LAPD could reduce the capacity threshold for undetectable probes. With optimizing only one single stage of the LAPD, the uncertainty region could be improved by 40% under nominal conditions as well as 20% for the corners. After optimization, the previously mentioned microprobes can also be detected in the worst case scenario.

4 The Calibratable Lightweight Invasive Attack Detector

The Probe Attempt Detector and the Low Area Probing Detector are designed to detect the intrinsic effects of microprobing, and can therefore protect against attacks such as backside probing. The PAD still uses analog circuit elements, though, which are larger than digital components, and which might not be readily available in some technologies. The LAPD fills this gap with a purely digital circuit measuring race conditions between buffers and capacitively loaded lines. However, the LAPD needs the protected lines to be carefully balanced, and its transistor dimensions to be fine-tuned to work with the desired precision. Also, the inability to compensate the effects of process variation by calibration leads to a lower sensitivity than the PAD.

Yet there is no detection circuit that can be calibrated to compensate process variations and line length imbalances by calibration and that is still based on only digital components. Such a detector could be integrated into the design process of a digital Integrated Circuit (IC) much more smoothly than the LAPD. It would still be robust against process, voltage, and temperature variations as well as enough sensitivity to detect state of the art microprobes. Furthermore, it would not require fine-tuning of its transistor dimensions in order to exhibit the desired sensitivity.

In this chapter, a new invasive attack detector named CaLIAD is presented which combines the advantages of previous detectors and improves the detection sensitivity and reliability with respect to process, voltage, and temperature variations. The chapter is structured as follows: Section 4.1 describes the concept behind the CaLIAD, section 4.2 evaluates the results. An FPGA implementation, which was joint work with Emili Lupon, is presented in section 4.3. Finally, section 4.4 concludes the chapter. Parts of this chapter have been previously published in [WWL+19].

4.1 Circuit Concept

A microprobe touching a line behaves as a capacitive load, as described in section 2.4. This makes transitions less steep and consequently delays the transitions in the connected gates. When one line out of a set of symmetric lines such as a bus is probed, a timing difference between the probed and the unprobed lines can be observed. This timing difference can be measured using a Time-to-Digital Converter (TDC).

Similar to previous approaches such as the PAD [MWS12] or the LAPD described in the previous chapter, this can detect probing of at most $B - 1$ lines when protecting a bus that consists of B lines. In accordance with the explanation of practical attacks in section 2.2, this is claimed to be sufficient as it is practically difficult to probe many lines simultaneously.

4.1.1 Sub-Gate Delay Measurement Circuits

Stephan Henzler gives an overview of different types of TDCs [Hen10]; circuits that allow sub-gate delay measurements are especially interesting here because they allow a new concept of calibratable probing detectors that are presented here.

In the presented use case, one main selection criterion of the circuit concept for delay measurement is a low area. It shall also be possible to use standard logic cells for implementing such a detector, such that the standard digital design workflow can be followed. Intrinsic resilience against manufacturing variations of the delay measurement circuit is not that crucial because it is sufficient to measure the difference from a calibrated state, but it is not necessary to extract accurate absolute timings.

There are three types of sub-gate delay measurement concepts presented in [Hen10]: Local Passive Interpolation TDCs use voltage dividers between inputs and outputs of driver stages to reach below the resolution of an inverter delay. Its structure is comparatively resilient to manufacturing variations. However, it requires differential signals and stages that cause a significant area and power overhead.

Another idea to measure timings smaller than inverter delays are pulse shrinking delay lines [Hen10, RK93]. They use chains of unbalanced inverters with differing rise and fall times. A pulse applied to the input of a pulse shrinking delay line will be reduced in duration along the line before it eventually disappears completely. Its disadvantage is the need for gates with an unbalanced dimensioning of p-channel Metal Oxide Semiconductor (pMOS) and nMOS transistors that might not be available in all technology libraries.

The Vernier Delay Line (VDL) has a comparatively small area footprint but shows inferior resilience against manufacturing variations when it comes to reliability of precise time measurements. This does not pose a problem when the goal is detecting deviations from a calibrated timing behavior.

It was decided to base a probe detector on the VDL: It has a comparably low area and power overhead, and its bad performance with respect to manufacturing variations is not an issue in the use case of detecting microprobes. This effect can be compensated by storing calibration values as described later.

4.1.2 The Vernier Delay Line Principle of Operation

A block diagram of the Vernier Delay Line is shown in figure 4.1. The device under test generates two output signals `START` and `STOP` that are passed through two differently dimensioned delay chains with different propagation delays. Both signals contain one transition; at the beginning of the line, the `START` transition occurs before the `STOP` transition. However, it propagates slower through the delay line than the `STOP` signal and eventually gets passed. Arbiters connected to each tap of the delay lines determine at which of the two taps the transition comes first. An example timing is shown in figure 4.2. In this figure, `STOP` passes `START` between tap 1 and tap 2.

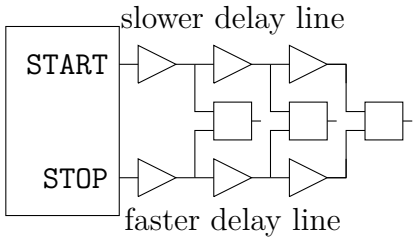


Figure 4.1: Vernier Delay Line block diagram.

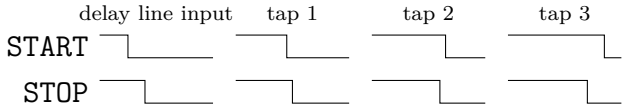


Figure 4.2: Vernier Delay Line timing.

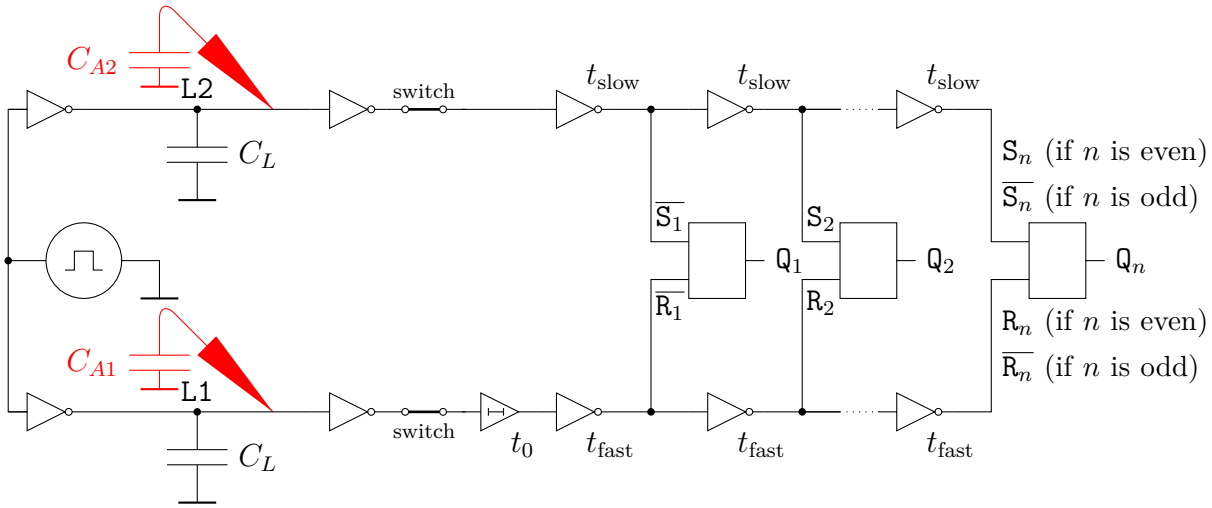


Figure 4.3: Schematic of the Calibratable Lightweight Invasive Attack Detector.

4.1.3 Delay Model

To model the delay that an attack capacitance C_A causes, the case of two lines is considered for reasons of simplicity. Chapter 5 evaluates how to apply the detection concept to buses with more symmetric lines.

Two lines `L1` and `L2` that both have an intrinsic capacitance C_L are used for the model, as depicted in figure 4.3. In addition, attack capacitances C_{A1} and C_{A2} are attached to `L1` and `L2` respectively. Equations (3.1) and (3.2) can be used also here to model the overall capacitive load of the bus lines. The timing difference of two lines is described by equation (3.3) accordingly.

4.1.4 Probe Detection Concept

The use case of a VDL is adapted by evaluating one edge of a test signal; at line `L2`, this signal is passed through a delay line with a slow incremental delay t_{slow} per stage, and at line `L1`, it is passed through a line with an initial delay t_0 and a fast incremental delay t_{fast} per stage, as shown in figure 4.3.

The two incremental delays t_{fast} and t_{slow} of the chain can be described as a common delay t_{C} and a delay difference Δt between the slow and the fast stage:

$$t_{\text{fast}} = t_{\text{C}} \tag{4.1}$$
$$t_{\text{slow}} = t_{\text{C}} + \Delta t \tag{4.2}$$

The transition at the faster line with initial delay passes the transition at the slower line between positions k and $k+1$ for which the following equations hold:

$$t_{\texttt{L1}} + t_0 + k \cdot t_{\text{fast}} > t_{\texttt{L2}} + k \cdot t_{\text{slow}} \tag{4.3}$$
$$t_{\texttt{L1}} + t_0 + (k+1) \cdot t_{\text{fast}} < t_{\texttt{L2}} + (k+1) \cdot t_{\text{slow}} \tag{4.4}$$

To insert C_{A1}, C_{A2} and Δt into the inequalities, equations (4.3) and (4.4) are solved for the delay difference $t_{\texttt{L1}} - t_{\texttt{L2}}$, which is replaced by the difference of capacitances using equation (3.3). Then, equations (3.1) and (3.2) are solved for $C_{A1} - C_{A2}$, and equations (4.1) and (4.2) are solved for Δt. This is eventually inserted into the transformed inequalities.

$$k \cdot \Delta t - t_0 < \Omega \cdot (C_{A1} - C_{A2}) < (k+1) \cdot \Delta t - t_0 \tag{4.5}$$

It will be shown later that k can be stored as a calibration value. Therefore, the concept is called “Calibratable Lightweight Invasive Attack Detector” (CaLIAD): On the one hand, calibration is possible, as it is with the Probe Attempt Detector, but which is not possible for the Low Area Probing Detector. On the other hand, it avoids large analog circuit components such as the PAD tank capacitor and only needs digital circuit elements that are comparably lightweight.

4.1.5 Circuit Realization

The schematic of the CaLIAD is shown in figure 4.3. The schematic is illustrated for a case study bus of two lines, which is the minimum number of lines required for its correct operation.

From left to right, it contains a test signal source, bus drivers, the bus lines that are under attack, output buffers, the initial delay element t_0 and eventually the VDL. One VDL chain element contains one RS latch that acts as an arbiter and two inverters: one with a slower propagation delay (t_{slow}) and a second which is slightly faster (t_{fast}).

It is recommended to introduce the ability to switch off the evaluation circuitry starting at t_0 and the first stage of t_{slow} when not in use. This is shown in the circuit diagram as "switch". Being able to switch off the circuitry saves energy on the one hand, and on the other hand it prevents attackers from probing the end of the delay to read signals from the protected lines.

Note that the VDL is tapped after each inverter rather than after each buffer which is the usual approach found in literature [Hen10]. This optimization allows to reduce the number of inverters in the VDL part by a factor of 2, but requires proper handling by

- either evaluating the rising edge of a test signal at even chain element positions k, and evaluating the falling edge of a test signal at odd chain element positions k, such that each arbiter can have high active inputs,
- or alternating between different arbiter types: NAND RS latches have low active inputs and can be used at odd chain element positions, whereas NOR RS latches have high active inputs and can be used at even chain element positions.

Both optimization alternatives were considered to be equally effective. Here, the focus was put on the latter option to keep simulation time low, as only one transition needs to be simulated instead of two.

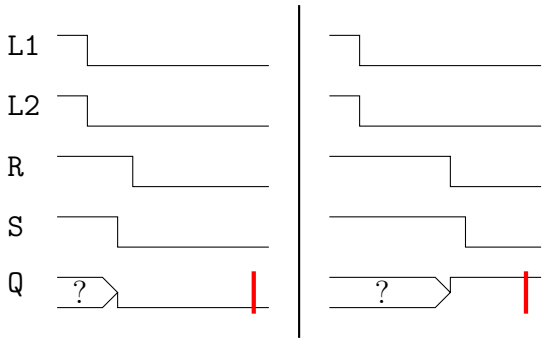
Considering equation (4.5), the outputs $\mathtt{Q}_i$ are 0 if $i \leq k$ and 1 if $i \geq k+1$. Without attack, the transition from 0 to 1 will occur at positions k_0 and k_0+1. To check for an attack, two elements are selected, one called *left* with $l \leq k_0$, and another called *right* with $r \geq k_0+1$. The following equation holds for the outputs of the selected elements.

$$(\mathtt{Q}_l, \mathtt{Q}_r) = \begin{cases} (0,1) & \text{no probe attached} \\ (0,0) & \text{probe at } \mathtt{L1} \\ (1,1) & \text{probe at } \mathtt{L2} \end{cases} \tag{4.6}$$

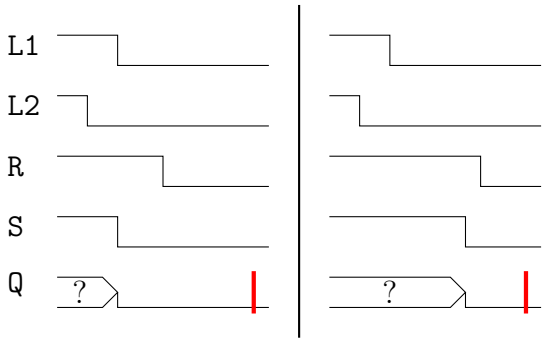
The timing of the CaLIAD is shown in figure 4.4 for the left element l and the right element r. Figure 4.4a shows the timing without attack. Figure 4.4b highlights the case of probing `L1` while figure 4.4c focuses on the effect of probing `L2`.

If more than two VDL chain elements are used, a thermometer code is obtained as a response as shown:

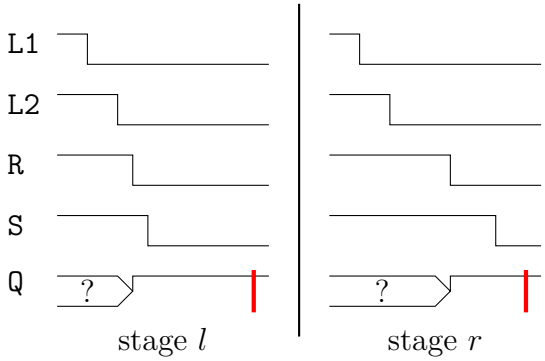
$$(\mathtt{Q}_1, \mathtt{Q}_2, \cdots, \mathtt{Q}_k, \mathtt{Q}_{k+1}, \cdots, \mathtt{Q}_{n-1}, \mathtt{Q}_n) = \mathtt{00\cdots01\cdots11} \tag{4.7}$$



(a) No attack.



(b) L1 probed.



(c) L2 probed.

Figure 4.4: CaLIAD timing; red bars represent latch output sampling time.

The position $k+1$ where the first 1 occurs will be referred to as position of first 1 (PF1).

4.1.6 Calibration

In the ideal case, i.e. assuming constant temperature and voltage as well as absence of manufacturing variations, two chain elements are sufficient, such that $l = 1$ and $r = 2$ would hold.

In reality, manufacturing variations make it difficult to predict the position of the transition from 0 to 1 in the thermometer code. When no probe is attached, i.e. $C_A = 0\,\mathrm{fF}$, the transition occurs between a position k_0 and $k_0 + 1$, where k_0 will vary from device to device. This position can be stored as a calibration value to compensate manufacturing variations. It is required to perform $B - 1$ comparisons and thus store $B - 1$ calibration values to protect a B line bus.

Depending on the sensitivity of the design, that position may shift a small number of steps when varying voltage and temperature or under the presence of noise. To compensate that, an additional safety margin m is defined to avoid false positives: If $(\mathtt{Q}_{k_0}, \mathtt{Q}_{k_0+1}) = (0, 1)$ holds for a certain calibrated position k_0, m steps to the left and to the right of that position can be skipped, and the detector can be configured such that the actual evaluation takes place at the two latch outputs l and r as follows:

$$l = k_0 - m \tag{4.8}$$

$$r = k_0 + m + 1 \tag{4.9}$$

Voltage and temperature variations cannot be compensated using this method, but they only have second order effects due to the differential nature of the circuit.

Another effect that cannot be completely compensated is a variation of the individual delay differences Δt between the slow and the fast inverters in the VDL chain, as described by the authors of [HYM+08].

The variation of Δt also implies that calibration accuracy is best at the capacitive load C_A that is used for calibration and decreases with increasing distance to the calibration capacitance. The proposal above calibrates at $C_A = 0\,\mathrm{fF}$. The C_A values at which the most accuracy is expected, though, are those that determine the transition between "no alarm" and "alarm".

Therefore, a second calibration method using a reference capacitance C_{ref} per protected line is proposed. This method shall be called two-point calibration; it is expected to further increase the detection performance compared to the previously proposed single point calibration method.

Two-point calibration is intended as a method to improve the sensitivity beyond single-point calibration, but not as a replacement; in most cases, single-point calibration is expected to be good enough. Also, two-point calibration is not applicable in all use cases, because the required reference capacitances increase the area usage; furthermore, they might help an attacker to spot the probing targets of interest.

C_{ref} is the smallest capacitance that is supposed to raise an alarm when attached to any line. To determine its value, a minimum alarm capacitance $C_{0.99}$ is defined first, for which an alarm shall be raised with an estimated alarm probability of $p_A \geq 0.99$. If it is sufficient to fulfill this condition at the voltage and temperature corner where the circuit is calibrated, one can use $C_{\mathrm{ref}} = C_{0.99}$.

Table 4.1: Two-point calibration example.

$C_{\texttt{L1}}$	$C_{\texttt{L2}}$	$\texttt{Q}_1, \cdots, \texttt{Q}_n$
$C_L + C_{\text{ref}}$	C_L	000**0**0000000000**0**11111
C_L	C_L	000**0**0000001111**1**11111
C_L	$C_L + C_{\text{ref}}$	000**1**1111111111**1**11111

As an attacker can, however, control the operating conditions, C_{ref} shall be slightly smaller to compensate second order effects. This way, the designer can ensure that an attack capacitance of $C_{0.99}$ is reliably detected at all voltage and temperature corners. The value of $C_{\text{ref}} \leq C_{0.99}$ is determined by Monte-Carlo simulations.

To perform calibration, `L1` and `L2` are alternately loaded with C_{ref}. The left calibration position k_L is determined as the PF1 when C_{ref} is connected to `L2`. Accordingly, the right calibration position k_R is determined as the position of the last 0, i.e. the immediate left neighbor of the PF1, when C_{ref} is connected to `L1`. An example is shown in table 4.1 where k_L and k_R are highlighted in bold.

The evaluation can then take place at k_L and k_R.

$$l = k_L \tag{4.10}$$

$$r = k_R \tag{4.11}$$

Notice that equation (4.6) still applies.

4.1.7 Design Considerations

The performance of CaLIAD implementations can be evaluated based on different criteria. This section gives an overview of them and concludes with general design considerations.

The minimum detectable capacitance difference $\Delta C_{A,\text{min}}$ determines the most "negative" capacitance difference, i.e. excess of attack capacitance attached to `L2` compared to `L1` that can be distinguished from range excess. When its value is applied, only the first VDL chain element is zero: $(\texttt{Q}_1, \texttt{Q}_2, \cdots, \texttt{Q}_n) = \texttt{01}\cdots\texttt{1}$. From equation (4.5), one can conclude that the following inequality must hold in this case:

$$\Delta t - t_0 < \Omega \cdot \Delta C_{A,\text{min}} < 2 \cdot \Delta t - t_0 \tag{4.12}$$

From this inequality it can be seen that the value of $\Delta C_{A,\text{min}}$ is determined by the initial delay t_0 as well as delay difference Δt between the slow and the fast inverter in the VDL chain. In the first place, this metric may not seem directly helpful – the CaLIAD only needs to give a pass/fail result and does not require to provide quantitative information about the attached probe. However, it provides a lower bound of ability to calibrate the

CaLIAD: Upon calibration, a too high value of $\Delta C_{A,\min}$ can lead to a left stage $l < 1$ that refers to a non-existing VDL chain element.

Similarly, the maximum detectable capacitance difference $\Delta C_{A,\max}$ determines the maximum excess of attack capacitance attached to `L1` compared to `L2` that can be distinguished from a range excess. When its value is applied, only the last VDL chain element is one: $(\mathtt{Q}_1, \cdots, \mathtt{Q}_{n-1}, \mathtt{Q}_n) = \mathtt{0 \cdots 01}$. One can conclude from equation (4.13) that the value of $\Delta C_{A,\max}$ is determined by t_0, Δt as well as the total number of chain elements n.

$$(n-1) \cdot \Delta t - t_0 < \Omega \cdot \Delta C_{A,\max} < n \cdot \Delta t - t_0 \tag{4.13}$$

$\Delta C_{A,\max}$ determines the upper bound of ability to calibrate, as devices with a right calibration position $r > n$ refer to a non-existing VDL stage and therefore cannot be calibrated.

Ideally, a symmetric detection range is desired, i.e. $\Delta C_{A,\max} = -\Delta C_{A,\min}$. This implies that $C_{A1} - C_{A2} = 0\,\mathrm{fF}$ corresponds to a centered transition position $(\mathtt{Q}_1, \cdots, \mathtt{Q}_{\frac{n}{2}-1}, \mathtt{Q}_{\frac{n}{2}}, \cdots, \mathtt{Q}_n) = \mathtt{0 \cdots 01 \cdots 1}$.

Another important metric is the detector sensitivity, i.e. the extent of PF1 shift depending on the shift of the attack capacitance. It is determined by Δt. Decreasing Δt increases the sensitivity, which allows to compensate manufacturing variations more accurately, but also implies longer chains, as seen in equations (4.12) and (4.13). Furthermore, circuit limitations impose a lower bound on Δt: it must be considered that the latches need a minimum time difference $\Delta t_{\mathtt{RS},\min}$ between the edges of $\mathtt{R}_i$ and $\mathtt{S}_i$ to avoid metastability and get a reliable output $\mathtt{Q}_i$. This has an implication on Δt between two VDL stages: While metastability cannot be avoided for *one* latch output, a lower bound $2 \cdot \Delta t_{\mathtt{RS},\min} < \Delta t$ shall be fulfilled to avoid that two or more latches become metastable. Otherwise, PF1 can not be uniquely determined. This effect is called bubbles. An example is shown in equation (4.14).

$$(\mathtt{Q}_{k-3}, \mathtt{Q}_{k-2}, \mathtt{Q}_{k-1}, \mathtt{Q}_k, \mathtt{Q}_{k+1}, \mathtt{Q}_{k+2}, \mathtt{Q}_{k+3}) = \mathtt{0010011} \tag{4.14}$$

Calibration would still be possible under the presence of bubbles: In the case of single-point calibration, one can use the middle between the position of the last consecutive 0 and the first consecutive 1 to estimate PF1. Obviously, m would have to be chosen such that l and r are outside the bubble region. The occurrence of bubbles indicates a limit in sensitivity. In that case, other measures would need to be taken to extend the sensitivity, e.g. an optimization of the used latches.

Other important metrics are area usage and power consumption of the CaLIAD. Aiming for good performances with respect to sensitivity and range of detectable capacitances will deteriorate the area and power performances.

4.2 Results

The CaLIAD was implemented on a 65 nm ASIC technology from STMicroelectronics. All circuit elements are based on the low power standard threshold voltage transistors `psvtlp` and `nsvtlp`. Cadence spectre 11.1.0 was used to simulate the circuit on a workstation with four AMD Opteron 6274 CPUs and 256 GB of RAM. The simulation sweeps were automated using SALVADOR, as described in section 6.2.

The simulated ambient temperature was set to a value of 27 °C; the intrinsic line capacitance was presumed as $C_L = 100\,\text{fF}$, which corresponds to an approximate line length of 1.3 mm on the top layer, assuming a parallel `GND` line with minimum distance.

4.2.1 Effects of manufacturing variations

The simulated CaLIAD chain consists of 30 chain elements. The fast inverters in the chain elements were implemented with an aspect ratio of $\frac{W}{L} = 20$, the slow inverters had an aspect ratio of $\frac{W}{L} = 8$. All other circuit elements were given a default aspect ratio of $\frac{W}{L} = 10$. The initial delay element was constructed as a chain of four inverters to have a sufficient margin at the left end of the chain.

It is expected to see a shift in PF1 depending on the attached attack capacitance difference ΔC_A. The simulations are limited to the simple case of probing either `L1` or `L2`, therefore assuming $C_A = \Delta C_A$. In order to merge the results, negative values of C_A are defined as $|C_A|$ being attached to `L2` while C_L remains constant for both lines.

$$\begin{aligned} C_A > 0 \quad &\Rightarrow C_{\texttt{L1}} = C_L + C_A \quad \wedge \; C_{\texttt{L2}} = C_L \\ C_A < 0 \quad &\Rightarrow C_{\texttt{L1}} = C_L \qquad\quad \wedge \; C_{\texttt{L2}} = C_L + |C_A| \end{aligned} \tag{4.15}$$

$N = 2000$ Monte-Carlo simulations have been executed for each sweep point $C_A \in [-40\,\text{fF}; 40\,\text{fF}]$ with increments of 1 fF. The simulations included process (per-chip) and mismatch (per-transistor) variations.

Figure 4.5 shows the distribution of the absolute PF1 in the chain as a function of the attack capacitance C_A. The larger C_A is, the further PF1 moves to the right. As described by equation (4.15), positive values of C_A represent a probe attached to `L1` and negative values refer to a probe attached to `L2`. The outlier at the bottom left of the figure is caused by large absolute C_A values that already reach the beginning of the line, i.e. $\texttt{Q}_1 = 1$. In this case, PF1 cannot move further to the left such that all further increases of absolute value of C_A accumulate in this point. With the current implementation, no bubbles have been observed in the VDL response.

The random offset of each device instance can be compensated by subtracting PF1 at 0 fF for each device instance. Figure 4.6 shows that the variation is significantly reduced by this calibration. However, one can also see that the variation increases with increasing distance to the calibration point. As explained before, this is due to the variation of

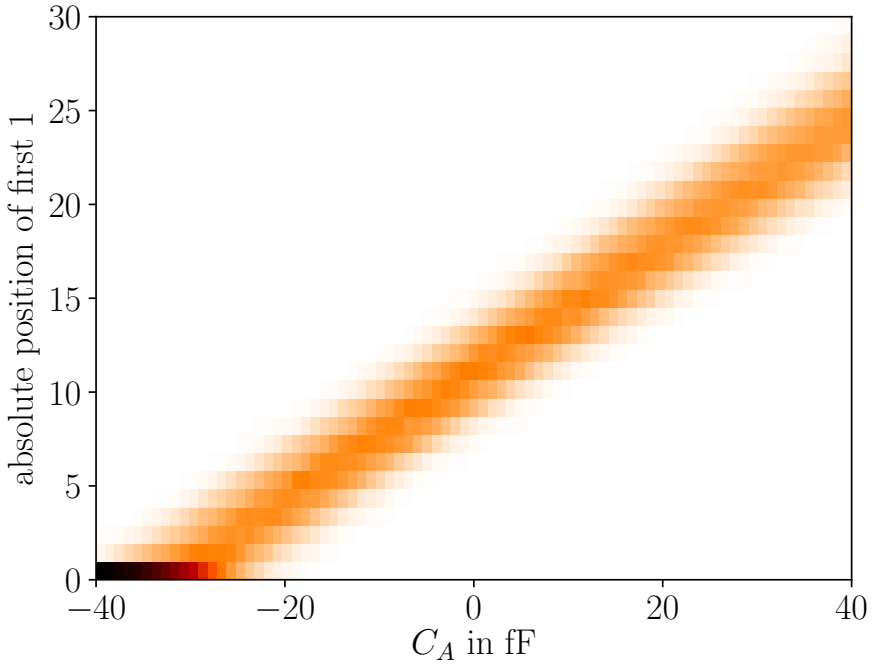


Figure 4.5: Distribution of transition positions without calibration; positive values of C_A represent a probe attached to `L1` and negative values represent a probe attached to `L2`.

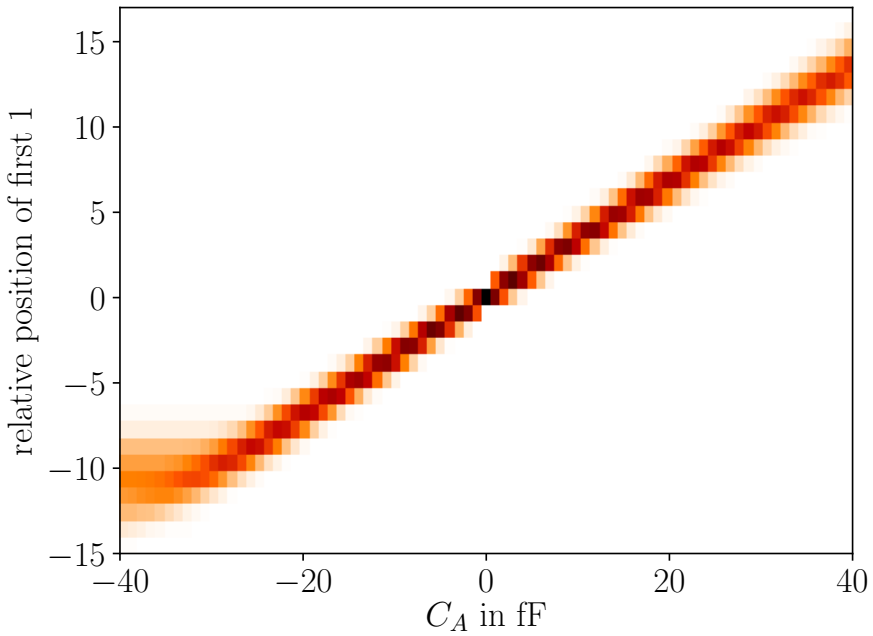


Figure 4.6: Distribution of transition positions with single-point calibration at $C_A = 0\,\mathrm{fF}$.

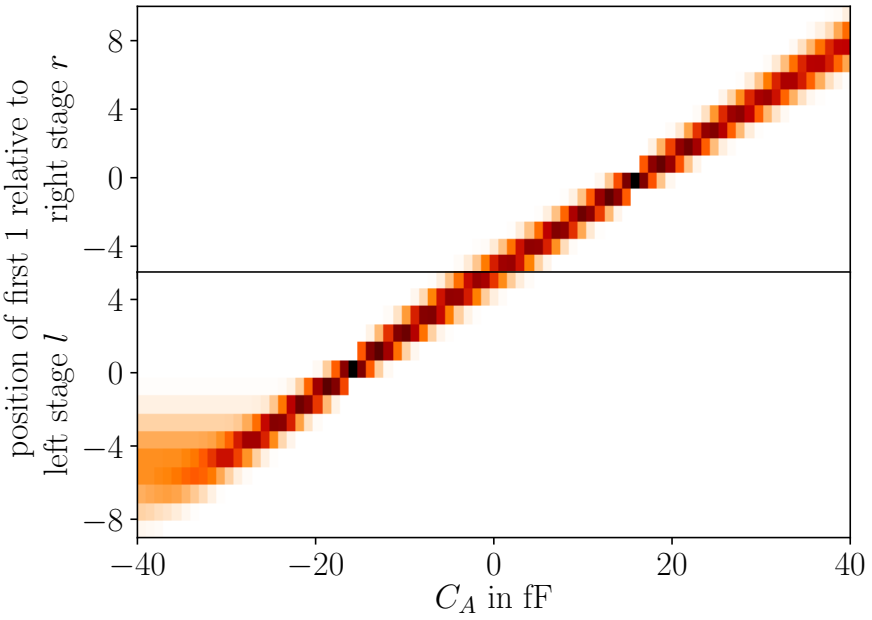


Figure 4.7: Distribution of transition positions with two-point calibration ($C_{\text{ref}} = 16\,\text{fF}$).

Δt that cannot be compensated with the single-point calibration. However, figure 4.6 suggests that using calibration at the actual points of interest – values of C_A at which the transition between "no alarm" and "alarm" is desired to occur – can further reduce the noticed variation. This effect is shown in figure 4.7. The bottom half shows PF1 relative to the left calibrated stage with index l and the top half shows PF1 relative to the right calibrated stage with index r. One can see a reduction of noise towards the extremes of C_A; more importantly, the PF1 becomes more accurate in the surrounding of the desired alarm threshold of 20 fF. This becomes visible by the fact that figure 4.7 exhibits two nodes with low variation around the desired alarm thresholds, whereas in the single-point calibration case, there is only one node at $C_A = 0$, thus further away from the point where high precision is desired.

4.2.2 Detection performance

Table 4.2 shows the detection boundaries of different CaLIAD calibration methods at the nominal corner, i.e. at a temperature of 27 °C and a supply voltage of 1.2 V. For the single-point calibration, the safety margins $m = 4$ and $m = 5$ are shown due to their proximity of the detection threshold to 20 fF, which is the smallest capacitive load of a commercial microprobe that was found [GGBa]. $C_{0.01}$ denotes the smallest capacitance with an estimated alarm probability of $p_A \leq 0.01$ at either of the lines `L1` or `L2`; accordingly, $C_{0.99}$ refers to the minimum capacitance with an estimated alarm probability of $p_A \geq 0.99$.

Table 4.2: Nominal corner detection performance of different CaLIAD calibration methods.

calibration method	$C_{0.01}$	$C_{0.99}$	ΔC	$n_{\min}$
single-point ($m = 4$)	9 fF	18 fF	9 fF	23
single-point ($m = 5$)	11 fF	21 fF	10 fF	24
two-point	11 fF	16 fF	5 fF	25
LAPD	16.4 fF	34.4 fF	18 fF	–
optimized LAPD (section 3.2.6)	6.4 fF	20.4 fF	14 fF	–

The Wilson method [Wil27] was applied to estimate the probability bounds using $N = 2000$ Monte-Carlo iterations and assuming a confidence level of $\alpha = 0.01$.

Stricter bounds would imply a significant increase of simulation time. For cases in which the estimated probability bound is not sufficient, one can use simulation methods such as Statistical Blockade [SR07, SR09] that can significantly improve the efficiency when analyzing rare events. Additionally, one can use majority voting by repeated measurements and/or circuit duplication to get significantly stricter probability bounds, as described in section 3.2.8.

$\Delta C_A^U = C_{0.99} - C_{0.01}$ is the region at which the CaLIAD output is not reliable. Smaller values of ΔC_A^U mean better performance. One can see from table 4.2 that the two-point calibration method outperforms the single-point calibration method approximately by a factor of two with respect to ΔC_A^U.

$n_{\min}$ is the minimum required chain length. It is determined such that the right VDL evaluation stage r is still within the chain length for all Monte-Carlo instances, i.e. $r \leq n_{\min}$ always holds.

The CaLIAD represents a significant improvement of detection performance compared to previous probing detectors. For example, the optimized LAPD from section 3.2.6 exhibits an uncertainty region of $\Delta C_A^U = 14\,\text{fF}$, whereas the CaLIAD has a worst case uncertainty region of 10 fF; with two-point calibration, 5 fF could be achieved.

Another advantage of the CaLIAD is its flexibility with respect to calibration: When using single-point calibration, the detection thresholds can be shifted by adjusting the safety margin m; in the case of two-point calibration, the reference capacitance C_{ref} can be chosen according to the requirements.

The value C_{ref} has been selected such that 20 fF are detected at all corners: The best microprobes with respect to input capacitance that were found are GGB Model 28/29 [GGBb] with an input capacitance of 40 fF, as well as GGB Model 18C/19C [GGBa] with an input capacitance between 20 fF and 60 fF depending of the transition time of the measured signals. Therefore, the absolute worst case attack capacitance is considered to be 20 fF.

Table 4.3: Worst case corner detection performance of different CaLIAD calibration methods.

calibration method	$C_{0.01}^{\mathrm{WCC}}$	$C_{0.99}^{\mathrm{WCC}}$	ΔC_A^U
single-point ($m = 4$)	6 fF	20 fF	14 fF
worst case at corner	0 °C, 1.08 V	0 °C, 1.08 V	
single-point ($m = 5$)	9 fF	23 fF	14 fF
worst case at corner	0 °C, 1.08 V	0 °C, 1.08 V	
two-point	9 fF	20 fF	11 fF
worst case at corner	0 °C, 1.08 V	0 °C, 1.08 V	
LAPD	12.0 fF	39.4 fF	27.4 fF

Table 4.4: Area, Timing and Energy Comparison of CSS, PAD, LAPD and CaLIAD.

	area a [GE]	cycles	energy [fJ]
CSS [CDG+14]	8081	–	$7.01 \times 10^{12}\,\mathrm{s}^{-1}$
PAD [MWS12]	549 (for $B = 2$)	50-100	n/a
LAPD	$\leq 48 \cdot B$ (for $B > 2$)	2	981 (for $B = 2$)
optimized LAPD (section 3.2.6)	$\leq 109 \cdot B$ (for $B > 2$)	2	1131 (for $B = 2$)
CaLIAD	$\leq 352 \cdot B$ (for $B > 2$)	1	1154 (for $B = 2$)

4.2.3 Corners

An analysis of voltage and temperature corners has been performed to determine the worst case corner values of $C_{0.01}^{\mathrm{WCC}}$ and $C_{0.99}^{\mathrm{WCC}}$ with respect to varying environmental conditions. The used corners were $\vartheta \in \{0\,°\mathrm{C}, 27\,°\mathrm{C}, 85\,°\mathrm{C}\}$ and $V_{\mathrm{DD}} \in \{1.08\,\mathrm{V}, 1.2\,\mathrm{V}, 1.32\,\mathrm{V}\}$. This gives results comparable to the the LAPD corners. To obtain the threshold capacitances, at first, a calibration at the nominal corner $\vartheta = 27\,°\mathrm{C}$ and $V_{\mathrm{DD}} = 1.2\,\mathrm{V}$ was conducted. Then, each corner was observed and the worse of the two capacitance values of either probing `L1` or `L2` was selected. Eventually, the worst case values of all nine corners were taken. They are shown in table 4.3.

The uncertainty region was observed to increase by at most 6 fF. Note that two-point calibration value of $C_{0.99}^{\mathrm{WCC}}$ is the desired target value that was used to determine the reference capacitance C_{ref}.

4.2.4 Resource Usage

Table 4.4 compares the area usage, timing and energy consumption of the CaLIAD with other probing detection circuits.

Table 4.5: area of the CaLIAD elements in gate equivalents.

a_{drivers}	4
a_{t_0}	8
a_{fast}	4
a_{slow}	1.6
a_{latch}	8

The CSS [CDG+14] aim at covering large parts of a chip with an active shield using Advanced Encryption Standard (AES) as a random pattern generator. Similar to the CaLIAD, the PAD [MWS12] and the LAPD detect timing imbalances of symmetric lines, but they use different measurement concepts.

The area a in GEs can be expressed as

$$a_{\text{total}} = B \cdot \left(a_{\text{drivers}} + \frac{1}{2} a_{t_0} + n \cdot (a_{\text{fast}} + a_{\text{slow}} + a_{\text{latch}}) \right) \tag{4.16}$$

where B is the number of bus lines to be protected in parallel, a_{drivers} denotes the area of the driving stages before and after the bus, a_{t_0} represents the area of the initial delay element t_0; a_{fast}, a_{slow} and a_{latch} represent the area of one CaLIAD stage element. Note that there exist alternative, more area efficient ways to protect multi-line buses; this will be discussed in chapter 5.

Each element in equation (4.16) is computed as follows:

$$a = \frac{1}{A_{\text{ref}}} \left(L^2 \cdot \frac{W}{L} \cdot t \cdot (1 + s) \right) \tag{4.17}$$

A_{ref} is the absolute area of the smallest NAND gate `HS65_LS_NAND2X2`, L is the channel length of the transistors, which is a constant of $L = 0.06\,\mu\text{m}$ in this case, $\frac{W}{L}$ is the nMOS transistor aspect ratio, which is used to tune the delay of the chain elements and has a default value of $\frac{W}{L} = 10$, t is the count of nMOS transistors of the stage under consideration, and s is a technology dependent pMOS transistor aspect ratio scale factor, which is $s = 2.2$ in this case.

Table 4.5 shows the area of the stages based on equation (4.17).

A chain length of $n = 25$ was assumed, which is the minimum length for which calibration was possible at all Monte-Carlo simulations for all described calibration methods and safety margins. Inserting the values from table 4.5 into equation (4.16) results in a total area of 352 gate equivalents. The data of CSS and PAD as well as the LAPD area are taken from section 3.2. The optimized LAPD was re-implemented to get an estimation for the power consumption; its dimensions were obtained by inserting the raw transistor dimensions into equation (4.17). While no energy consumption is available for the PAD, it is expected to be significantly higher as when compared to the CaLIAD as the PAD requires 50 to 100 cycles in which a large capacitor is charged. It

is difficult to make a timing and energy consumption comparison with the CSS as the CSS is supposed to run continuously.

The CaLIAD is second best with respect to area after the LAPD implementations. Its power consumption is comparable to the LAPD. It has a superior detection performance and a faster response time compared to PAD and LAPD (the CSS are not directly comparable in these terms).

Furthermore, as the CaLIAD offers more parameters and therefore more degrees of freedom than the LAPD, one can presume that systematic optimization can yield better results as well. First results have been published by Girardi and Graeb after the work on this thesis has been finished; they could get the number of chain elements down to $n = 8$ at 100% yield on a 130 nm technology [GG20].

4.3 FPGA Implementation

A FPGA was used as a proof-of-concept implementation platform to demonstrate its functionality on real digital hardware. Furthermore, a temperature sweep was performed in a climate chamber to analyze its reliability over temperature changes. This was a joint work with Prof. Emili Lupon from Escuela Tecnica Superior de Engineriya Industrial de Barcelona, which is part of the Universitat Politècnica de Catalunya.

4.3.1 Design

An Altera/Intel Cyclone III FPGA, model EP3C16F484C6, was used for the experiments conducted. It is implemented in a 60 nm technology [Alt15]. The majority of the 28x40 internal blocks of the FPGA are Logic Array Blocks (LABs); other blocks implement memory and multipliers or are reserved. One LAB consists of 16 Logic Elements (LEs) and a local interconnect bus. Each LE contains a four-input Lookup Table (LUT) and a flip flop.

The synthesis environment Quartus II allows placement of components down to a granularity of LEs. Routing can not be enforced, but a sparse regular architecture is a good practice to get balanced delays. Furthermore, post-routing simulations can be employed to estimate the timing behavior before testing hardware implementations.

As the CaLIAD is an asynchronous design taking advantage of timing properties, the following components must be designed with care:

- It shall be possible to emulate an attack on either `L1`, `L2`, both lines, or to disable the attack. As it is not possible to emulate a real capacitive load on an FPGA, an abstract model was chosen to be used to introduce an additional small delay to the lines for which an attack is emulated. During early simulations, it was

observed that the LUT inputs named `DATAD` exhibit an internal propagation delay in the magnitude of 60 ps, whereas another input named `DATAC` shows an internal propagation delay of approximately 120 ps across all observed LUTs without considering the output load. The difference is approximately in the same order of magnitude as the delay introduced by a probe in the ASIC technology used. This makes it possible to configure a LUT as a switchable delay element.

- The initial delay t_0 can be implemented as a combination of a coarse and a fine delay. For the coarse delay, one can make use of the inherent routing delays between cells and by forcing signals to pass through additional LUTs acting as buffers. Fine delays can be implemented by additional dummy gates increasing the load and hence the interconnect delay.
- The latches are implemented using a LUT with a feedback from the output back to an input. Note that this approach results in an inherent imbalance of feedback paths compared to the ASIC implementation. Furthermore, simulations have shown that the interconnect and/or input load of the signals `R` and `S` (and, respectively, $\overline{\texttt{R}}$ and $\overline{\texttt{S}}$) is different.
- There must be a delay difference Δt between the slow and the fast inverters of the VDL chain. This can be accomplished by using the property of different LUT inputs having different propagation delay, by different interconnect delays of different LAB columns, and by introducing different loads at the inverter outputs. Simulations have shown that the different loads of the latch inputs can be used to accomplish different delays; this option turns out to exhibit the smallest $\Delta t \approx 5\,\text{ps}$ out of all three available options.

Early experiments have shown a more symmetric timing behavior for daisy-chaining LABs in a column compared to creating chains of rows. Therefore, four columns were used to implement the VDL chain; each row implements a chain element. As there exist 28 rows in the FPGA that were used, the chain was designed to consist of 28 elements in total. The four columns are highlighted in darker blue on the right side of figure 4.8, which shows the layout of the CaLIAD implementation. From right to left, the columns implement the fast inverter chain, the latches, the slow inverter chain, and eventually a row of buffers that decouple the latches from the load of the post-processing gates. The white column between the slow inverter chain and the buffers represents hardware multipliers and is not used in this design. The dark blue area in the middle of the design implements control and read out logic that was not subject to placement constraints.

4.3.2 Experimental Setup

A state machine was implemented that emulates the four cases, i.e. no attack, an emulated attack on `L1`, an emulated attack on `L2`, and an emulated attack on both lines. 10 000 000 iterations are performed for each case, and 28 counters capture the frequency (i.e. number of events) of a “1” at each latch output. The results are sent

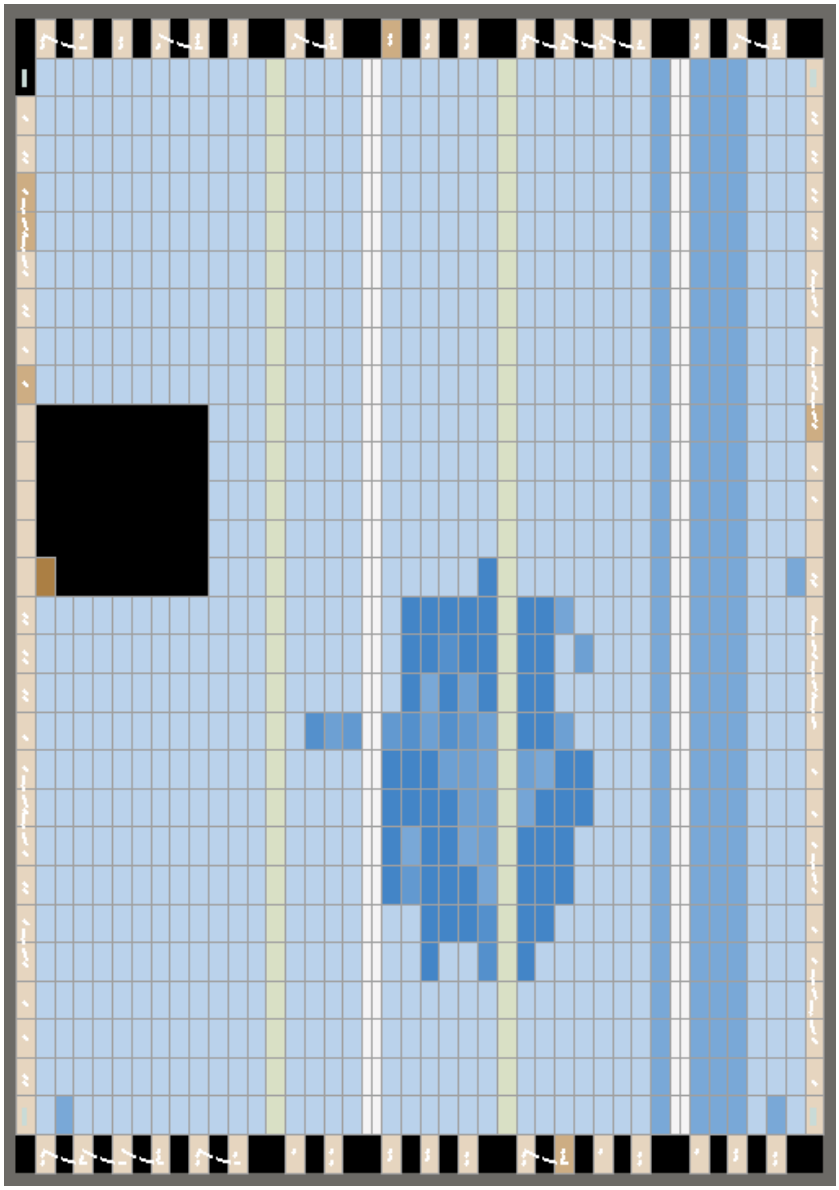


Figure 4.8: Layout of CaLIAD FPGA implementation.

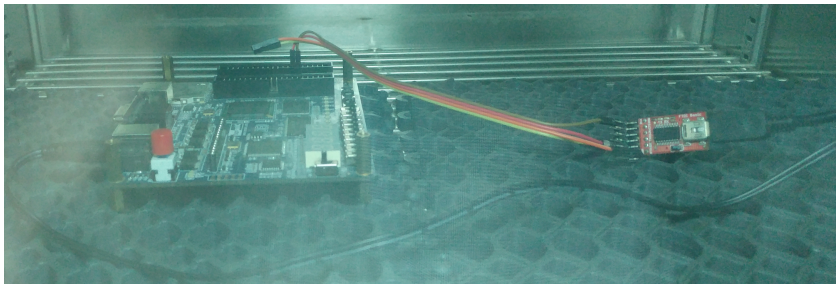


Figure 4.9: FPGA in climate chamber.

Table 4.6: Counter values captured at 20 °C.

k	emulated probe at line(s)			
	–	L1	L2	L1,L2
	count($Q_k = 1$)			
1,2,...,11	0	0	10 000 000	0
12	10 000 000	0	10 000 000	10 000 000
13	11 593	0	10 000 000	4609
14	9 999 780	0	10 000 000	9 999 310
15	1	0	10 000 000	0
16	10 000 000	0	10 000 000	10 000 000
17	9 730 190	0	10 000 000	9 379 770
18	10 000 000	0	10 000 000	10 000 000
19	9 999 710	0	10 000 000	9 998 106
20,21,...,28	10 000 000	0	10 000 000	10 000 000

back to a computer through a Univeral Asynchronous Receiver/Transmitter (UART) interface for further evaluation.

To verify the temperature stability of the design, the FPGA was placed into a climate chamber, as shown in figure 4.9. With this setup, a temperature sweep from 0 °C to 70 °C was performed.

4.3.3 Results

An initial CaLIAD implementation was tested outside the climate chamber. However, irregularities have been detected that are described as follows.

An extra LUT tuning the initial delay t_0 had to be inserted at the *slow* inverter chain in order to see a transition between zeroes and ones in the range of the VDL chain. This is presumably caused by a deviation of interconnection delays from the post place and route simulations.

Table 4.6 shows the frequency of ones for 10 000 000 iterations after the aforementioned offset correction. One can see that emulating an attack on either L1 or L2 exceeds the range of the chain: all elements are either zero or one.

The other two cases, i.e. no attack or attacking both lines simultaneously, show an almost identical behavior. Latch outputs Q_{13}, Q_{14}, Q_{15}, Q_{17} and Q_{19} are not constant – supposedly because of metastability and noise – while all other elements have a constant output of 0 or 1. Note that the observed numbers at the non-constant positions may vary significantly when the experiment is repeated. However, the positions that remained constant or non-constant did not change in all repetitions.

For the non-constant positions, a monotonic increase of counter values is expected for increasing counter indices. However, the observed sequence of counter values is not monotonic. This implies there have to be results in the shape of $(\mathtt{Q}_1, \ldots, \mathtt{Q}_{28}) =$ `0000000000011101111111111111`, i.e. bubbles. It can be considered as a result of three different effects.

First, post place and route simulations have shown a timing irregularity between row `Y14` and `Y15`. This can explain the effect that $(\mathtt{Q}_{14}, \mathtt{Q}_{15}) = \mathtt{10}$.

Second, one can observe that the counter values in the middle of the chain alternate between increasing and decreasing from one element to another. Note that the CaLIAD implementation alternates between low active NAND latches and high active NOR latches; furthermore, post place and route simulations have shown different LUT propagation delays for rising and falling edges at the input. Consequently, a connection between these two effects can be assumed.

Third, one can see that there exist situations where $(\mathtt{Q}_{12}, \mathtt{Q}_{14}) = \mathtt{10}$. Both latches are of the same type and the simulator does not point out any timing imbalances after routing. It is possible that metastability is the cause of this effect, but further investigations are required for clarification.

When performing the temperature sweep, it could be noticed that the positions with constant counter values of 0 or 10 000 000 do not change over the full temperature sweep between 0 °C and 70 °C. The only observable temperature effect was an increase in variation of counter value with increasing temperature at certain non-constant counter positions. Therefore, this design is considered temperature stable.

4.4 Conclusion

The CaLIAD is able to detect probing attacks by timing measurements through a Vernier Delay Line. Its performance has been evaluated with respect to manufacturing variations and environmental corners. It was shown that the CaLIAD is able to outperform comparable circuits such as the LAPD with respect to detection performance.

While its area usage is between the Probe Attempt Detector and the LAPD, it only consists of digital components such as the LAPD; however, it can also be calibrated like the PAD, thus exhibiting an outstanding detection performance, such as an uncertainty region as low as $\Delta C = 5\,\mathrm{fF}$ under nominal conditions and 11 fF at the worst case corner. A qualitative comparison with existing countermeasures is shown in table 4.7. The FPGA implementation has demonstrated the practical ability to detect probes, which were assumed to introduce a delay of 60 fs, on real hardware of a similar technology size than the technology used for simulations. Therefore, the use of the CaLIAD circuit concept is recommended for future probe detection designs that prefer a high sensitivity and improved reliability over a very small area.

Table 4.7: Qualitative advantages of the CaLIAD over other invasive attack countermeasures.

alternative countermeasure	CaLIAD advantages
meshes	detection of backside attacks very low circuit overhead
bus encryption	no latency
PAD [MWS12]	no analog design flow needed no capacitor
LAPD	no transistor fine-tuning required better detection margin/reliability

5 System Integration

The previous chapters have presented concepts to detect timing asymmetries in pairs of lines that are intended to be symmetric. As microprobes cause such asymmetries by their capacitive load, these detectors can be used to raise an alarm when an adversary uses them for an invasive attack.

The circuit examples described in the previous chapters were focused on protecting a pair two of lines. This chapter shows how to integrate them into parallel on-chip buses.

Section 5.1 will give an overview of buses in general. The challenges of integrating probing detectors are discussed in section 5.2. These challenges are addressed in section 5.3 that proposes implementation concepts to protect an on-chip bus. Section 5.4 presents the results of an implementation of a probing detector into a real-world bus system. Eventually, this chapter is concluded by section 5.5.

5.1 Introduction

On-chip buses provide the interface between core components of a microcontroller, such as CPU core, memories and integrated peripherals. As their interconnection infrastructure concentrates transfers of potentially confidential data to few physical lines, it is a comparably worthwhile invasive attack target.

“Classic” bus architectures are considered here, where one or more bus masters can issue read or write requests to addresses. A part of these addresses is used to resolve the addressed slave, and another part of the address is passed to the slave. After processing the request, the slave sends an acknowledge signal to the master, which is, in case of a read request, accompanied by the returned data.

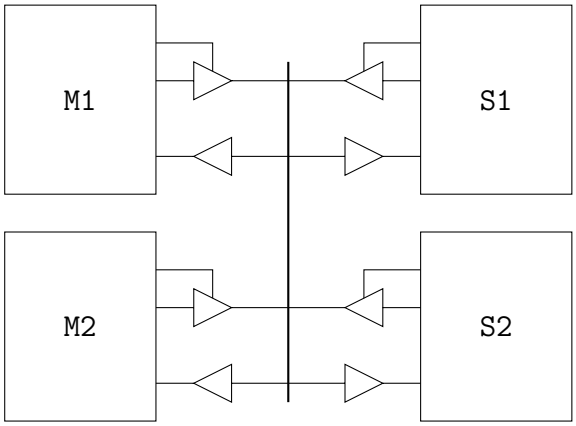
In general, the signals that make up a bus can be grouped into address, data and control signals. The terms “address bus” and “data bus” usually refer to the address or data part of a bus system rather than being an independent bus. This notation will be applied in the following chapter as well. *Masters* are bus participants that send requests to *slaves*. This is accomplished by placing an address on the address bus and activating the corresponding control signals to notify about the request. Then, if the master has requested a read operation, the addressed slave puts the data on the data bus, which is read by the master; in the case of a write operation, the master puts the data on the data bus and the corresponding slave processes it.

Buses can be classified by different properties. For example, they can be distinguished by the protocol features they support. Modern buses offer features such as bursts, pipelined transfers, split transfers or out-of-order transfers in order to reduce the number of delays on the bus, thus improving its utilization [PD08]. Implementing an improvement of bus utilization usually comes with the cost of an increased complexity of bus participants and/or the interconnect logic; consequently, the trade-off between complexity and bus performance needs to be found based on the use case. For this reason, even modern bus protocols such as Advanced Microcontroller Bus Architecture (AMBA) offer lightweight versions of bus protocols, such as ARM does with AXI4/AXI5-Lite [ARM17]. Other less commonly known protocol features target use cases differing from what standard buses can offer. Examples are streaming [ARM10] or broadcasting; the latter can be useful for cache coherency protocols [PD08].

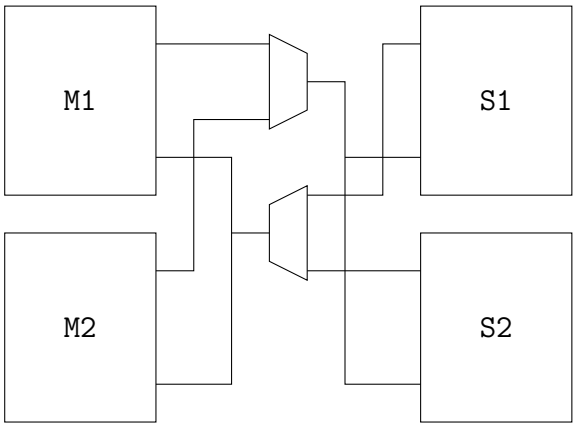
Another factor to distinguish different buses and their implementations is the interconnection topology. It determines how many bus transactions can take place simultaneously; in one extreme, a shared medium only allows one transaction at a time, whereas in the other extreme, a full bus crossbar, also called point-to-point bus, maximizes the number of parallel transfers. Besides the extremes, there are other approaches such as hierarchical topologies, partial crossbars or rings, that can be considered a trade-off [PD08]. The bus topology does not necessarily influence the interface shown to masters and slaves; for example, a shared multiplexed bus can have the same control signals for arbitration as a crossbar switch bus, as this is the case in the open source WISHBONE bus [Ope10].

The electrical properties of a bus are also important when classifying it. In particular, one can distinguish between the tri-state and multiplexed operation of a shared bus. In the tri-state implementation, all bus participants are connected to the bus lines without intermediate buffers; when a transaction occurs, only the involved parties activate the output drivers, while the drivers of all other bus participants are in the tri-state mode. In this case, the data lines can be operated bi-directionally, i.e. for read and write operations. This is shown in figure 5.1a. When implementing a shared bus using multiplexers, the data lines for the read and write direction are physically separate. Multiplexers pass through the data from one master to all slaves on the address bus as well as on the write direction of the data bus; vice versa, multiplexers on the read data bus forward the data from one slave to all masters. A simplified example of a multiplexed bus is depicted in figure 5.1b.

In earlier days of bus systems, tri-state buses were prevalent due to their simplicity and low number of transistors needed. However, the length of lines without intermediate drivers becomes challenging with decreasing transistor sizes; an increasing line length with one driver leads to a quadratic increase of delay, whereas the delay increase is only linear when intermediate buffers are used [WE93, NHEW10]. Intermediate buffers cannot be used with tri-state buses, though; furthermore, it performs worse with respect to power consumption compared to multiplexed buses, and creates challenges when debugging [NHEW10, PD08]. Therefore, multiplexed buses have become prevalent nowadays.



(a) Tri-state.



(b) Multiplexed.

Figure 5.1: different bus implementation concepts on electrical level; note that this figure only shows the data bus.

5.2 Integration Challenges

Today's bus systems are quite complex, and a 100% coverage of buses by microprobing detectors is hard to reach. Instead, it seems advisable to carefully select where exactly probing detectors shall be introduced such that the benefit exceeds the circuit overhead. An attack model, which describes the approach of an attacker, can help to find out the exact assets that need to be protected. However, such a model depends on the exact use case and environment; it shall answer simple questions like "what type and amount of data shall be protected?": is it only a small set of cryptographic keys of a known algorithm, or the complete firmware image of a microcontroller, which can take several hundreds of kilobytes or more? Can probing detectors be combined with other countermeasures, such as encryption?

As soon as the parts of the bus to be protected have been identified, the questions become more detailed. How can a bus that consists of B lines be protected by a microprobing detector from the previous chapters, which has only been presented to protect two lines? How can the detection cycles be injected into regular bus operations?

5.3 Solution

Instead of addressing the questions raised before one by one, two case studies will be used for better clarity. In the course of these studies, different variants of implementing certain details will become apparent; its benefits and drawbacks will be discussed in the course of this section.

In the first case, a *CPU core* from figure 5.2 is considered. It fetches the instructions from an encrypted bus, then passes them through an internal decryption unit, and finally stores the plain instructions in an instruction register that serves as an input to the instruction decoder and following logic. In this case, it is assumed that an attacker wants to dump the firmware using the attack technique *Linear Code Extraction* that was described in section 2.2. To do this, he prevents the CPU from performing branches by forcing the "clock enable" input of the instruction register to a logical `false` value with a probe, and he uses a second probe to capture the data on the bus iterating over all bus lines.

Another use case considers a *multiplexed bus* with several masters and several slaves. In particular, the data flow from a memory unit in the role of a bus slave to the CPU core in the role of a bus master is considered. Figure 5.3 shows this setup. In the middle, a multiplexer selects one of the inputs to be passed on to the CPU and the other masters. This bus crosses boundaries between different circuit components, and therefore exhibits significantly longer lines than there are in the CPU core case study. It can consequently be expected that the lines are placed in more upper layers, which makes them accessible more easily by front side probing. Also note the intermediate

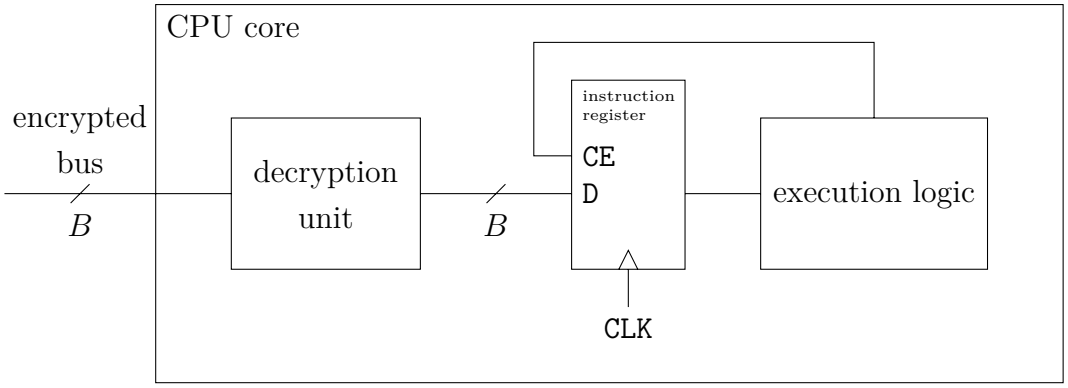


Figure 5.2: CPU core attached to an encrypted bus.

buffers both between source and multiplexer inputs, as well as between multiplexer outputs and signal destination, which are inserted for improved propagation delay.

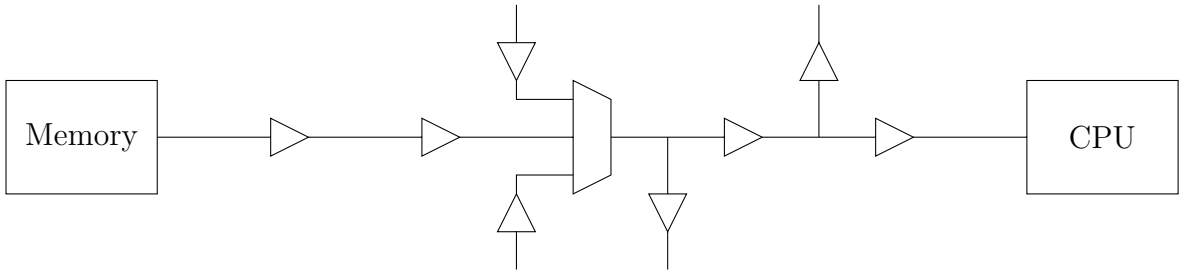


Figure 5.3: Multiplexed bus; the figure only shows the data path from slaves to masters.

Based on these use cases, the following sections will explain challenges when integrating probing detectors and discuss potential solutions.

The detectors are assumed to consist of a test signal generator that produces test patterns, and an evaluation circuitry that generates a response based on the test patterns and on the timing properties that are affected by microprobes. In most of the cases that will be described subsequently, the exact test pattern waveforms and the contents of the results do not need to be defined concretely, such that the concepts are applicable to both LAPD and CaLIAD as well as other detectors. Any deviations, i.e. integration concepts that are based on one concrete detector type, will be mentioned explicitly.

5.3.1 Considerations of Detector Placement

In the course of preparing a probing attack, an adversary needs to decide where to attach probes for a successful attack with minimal effort. This decision must be taken early in the attack process, as it may imply preparatory processing, such as removing insulating material with a laser cutter, or using a FIB to re-route wires or add landing pads for the probe tips.

Considering the CPU core case study, the attacker has three options for probe placement. Extracting the data from the encrypted bus segment, however, would require additional knowledge about the encryption function and the used key material. The output of the decryption unit seems like a good probing target, as all instructions to be executed are seen here in plain text. The output of the instruction register would in theory yield the same values; however, as the attacker has disabled its clock enable signal to prevent branches from happening, a probe would only see one instruction being executed in an infinite loop.

Consequently, a probe detection unit can be inserted between the output of the decryption unit and the input of the instruction register. A first question that can be raised here is how the test patterns can be injected into this segment. One simple option would be multiplexing between the decryption unit output and a separate test pattern generator. However, in addition to the hardware overhead, this would leave the path between the decryption unit and the multiplexer unprotected by the probe detector. Alternatively, dummy instructions can be stored, for example, in the encrypted startup code such that the decryption unit itself serves as a test pattern generator. In the case of the CaLIAD, this could be a sequence of two words that generate simultaneous falling transitions on all bus lines. A preceding instruction may instruct the control logic to disable the instruction registers for these two cycles. The values from the CaLIAD VDL chain can then be stored in special function control registers that can be evaluated afterwards.

Considering the case study of the multiplexed bus, usual multiplexer implementations only allow protecting one input per test signal generator. Evidently, this signal generator shall be inserted as close to the beginning of the line as possible, especially when intermediate buffers split the line into several segments. At some stage, however, some sort of multiplexing between signals of type *asset* and signals of type *test signal* cannot be avoided; *asset signals* contain information that an attacker wants to extract by means of microprobing, while *test signals* are used by a probe detector, but its contents are not valuable to an attacker.

While the output of a multiplexer that combines asset signals and test signals is protected by a probe detector, the asset signal at the input is usually unprotected. This conceptual problem can be mitigated when multiplexing takes place at a stage where the assets are not accumulated yet to a single set of lines; for example, the test signal generator may be integrated into a memory array, such that the internal multiplexers can switch between bit lines containing assets, which are not yet accumulated to a single bus, and test signal generators. This way, there is no segment that is unprotected and aggregates all assets at the same time.

An example of this concept is shown in figure 5.4. It shows a memory array consisting of seven rows and eight columns that is connected to the same output multiplexer than the test signal generator of a probe detector. This way, the output of the multiplexer is protected, and at the input, the assets are distributed over seven input lines, such that an attacker would need to probe all of them to dump the contents of the memory.

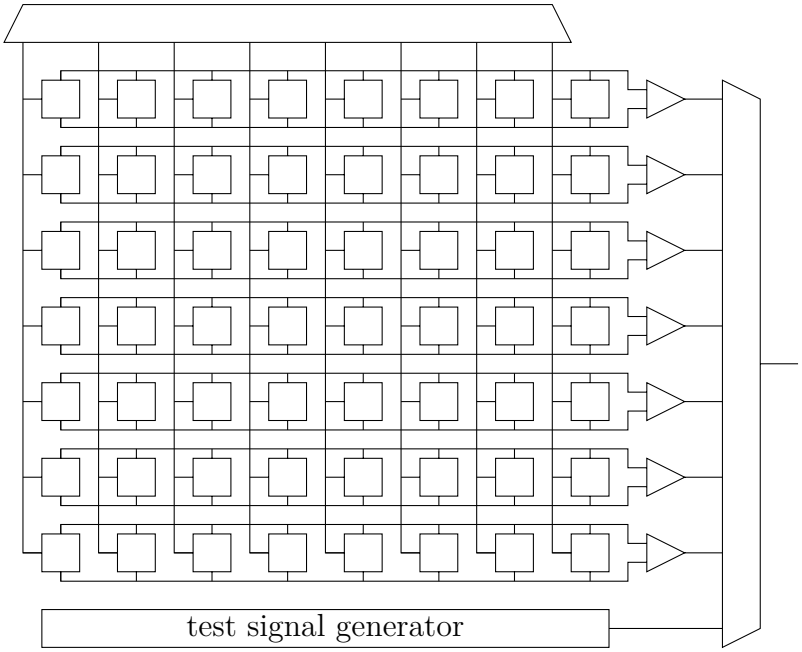


Figure 5.4: Probe detector test signal generator integrated into memory array.

As an alternative to having a dedicated test signal generator, one can consider triggering test cycles when the productive data transmitted on the bus is suitable. In the case of the CaLIAD, this is true when two neighboring bits have simultaneous falling transitions. This approach can be called *online testing* as it avoids additional bus cycles; it also reduces area overhead, and in case that the implicit test signal source also carries assets to be protected, it avoids the detector being "blind" at the multiplexer input, which would be the case for a dedicated test signal generator, as described before. It may however not always be possible to ensure that the transferred data allows testing all bus lines before a critical operation.

Branches with intermediate buffers at the multiplexer output, as for example shown in figure 5.3, prevent probe detectors from protecting the whole output tree. These limitations can be addressed by either including probe detectors in all components that are connected to the multiplexer outputs, or by electrical segmentation, i.e. disconnecting inactive bus participants from the multiplexer output, for example by using transmission gates as switches. More exotic approaches may use entire multiplexers based on transmission gates that allow interconnecting all input signals, or buffers that can be bypassed, when performing a measurement cycle.

5.3.2 Triggering and Evaluation

The evaluation must be triggered early enough to prevent successful attacks. In the CPU core use case, this means before the address space with the target code is reached, in order to ensure that the linear code extraction attack is successfully prevented. This

is especially the case when triggering the probing detection and evaluating the results in software; in that case, for manipulating the instruction register and thus stepping over the code linearly, it is sufficient for the attacker to reach the first non-branch instruction of the target code.

To prevent linear code extraction, the detector must be run before reaching the target address space. If the target CPU architecture supports an MMU, this can be accomplished by having different virtual address mapping for the startup code in ROM and the target code in flash memory such that a page fault would occur when stepping linearly through the virtual address space. Alternatively, it can be ensured that stepping over the address space halts before reaching the target address (e.g. by locating the startup code at a higher address, and preventing address wrap-around in hardware).

The action of triggering a measurement run, the evaluation of the result, and consequently also the reaction, can be implemented in hardware or in software. Software implementations can either use special extensions to the instruction set or special function registers comparable to configuring other peripherals that pause the normal circuit operation to perform a test cycle. Interrupts can be used for triggering a test cycle by hardware. As a less obvious alternative to interrupting regular operation to perform measurements, which could be called *offline testing*, one can implement circuits that detect occurrences of suitable test patterns during regular operation. In the case of the CaLIAD, such test patterns would be simultaneous falling edges on neighboring bus lines. Such an approach, which can be called *online testing*, is able to avoid interruptions of the productive use. While the data on the bus is not generally optimized for detection coverage, a dedicated sequence of data values can be placed on a bus before security critical operations to ensure that all lines are tested.

From a security perspective, the actual triggering seems of minor importance, as long as it can be ensured that certain checkpoints are only reached after a successful probing detection cycle revealing that no probe is attached. The measurement itself obviously takes place in hardware using the LAPD or CaLIAD circuits. The evaluation includes storing the measurement result for further processing and interpreting it; evaluation and measurement may also include self-tests against manipulation of the detection circuitry.

5.3.3 Security Considerations

The result evaluation and reaction are crucial from a security perspective, as the security is defeated if an attacker can prevent the execution of reactions to an attack; reaction means triggering an action, such as keeping the CPU in reset or erasing the memory, when the result yields an alarm. The security is compromised if these steps can be tricked into ignoring results that should raise an alarm. At first glance, it seems a good idea to perform the evaluation in a hardware element that is frequently used for other purposes as well; this would make malicious circuit modifications, such as, simplistically speaking, connecting an “alarm output” to ground, more difficult without leading

to other, unwanted effects. Following this argumentation, a software implementation seems preferrable. In the case of the CaLIAD, it would read the output of the VDL chain, compute the distance to a reference value stored in write-protected memory, and eventually use this distance to obtain the pass/fail result. Depending on this result, it would either continue regular program execution, or it would call a function that disables normal operation. However, note that in this case study, it is assumed that the adversary has tampered with the CPU core in a way that allows skipping the execution of instructions. Thus, the software must be written in such a way that no combination of instruction skips can lead to bypassing the alarm when a probe is detected, or when the check itself is skipped. It must therefore be protected against fault attacks either by hardware or software redundancy [BECN+06]. Techniques aiming at maintaining the integrity of the control flow that can be used are, for example, proposed by Moro et al. [BHKL12, MHER14]. Alternatively, the evaluation and reaction can be implemented in hardware. In that case, it is also advisable to keep critical signals, such as the "alarm output" line, redundant.

Similarly important is the storage location and the way of accessing the calibration values. Storing them in dedicated memory cells can facilitate attacks, as it may enable tampering with the calibration values without affecting the data the attacker aims at obtaining, for example by forcing or FIB editing attacks.

On the other hand, using regular memory for storing calibration values implies that they need to be transferred over the bus, which is the probing target anyway. The encoding of the calibration values influences how well an attacker can exploit that. If the CaLIAD calibration value is stored as a thermometrical code, only the lines in the neighborhood of the transition from 0 to 1 need to be tampered with; on the other hand, for example, a binary representation of the transition position accompanied by a checksum significantly increases the number of bits to be manipulated in order to alter the calibration value.

Another factor to keep in mind is the resistance against forcing, which can be done using microprobes or with FIB assisted circuit editing. In case of the CaLIAD, an attacker could observe the calibration value and then connect the corresponding chain elements to either $\mathtt{V_{DD}}$ or $\mathtt{GND}$ accordingly. As a countermeasure, a system designer can require a self-test before running the actual measurement; in a first test cycle, he can set the line that is connected to the "reset" inputs of the latches in the VDL chain to the active state, and set the line feeding the "set" inputs to the inactive state; he can then check whether all latch outputs are $\mathtt{Q}_i = 0$. Likewise, in a second test cycle, the line feeding the "set" inputs can be set to active, and the line driving the "reset" inputs to inactive, such that all latches are expected to output $\mathtt{Q}_i = 1$.

5.3.4 Protection of More than Two Bus Lines

Another open point when implementing bus protection using the LAPD or CaLIAD is the adaption to the number of bus lines B, as the detectors presented only can protect

two lines in their basic form. In general, probes can be detected using $B-1$ comparisons, which allow detecting $B-1$ probes. A cautious designer may also implement the comparison between line 1 and B, such that probing any line, including lines 1 and B, causes two comparisons to fail.

The comparisons can be implemented in parallel using $B-1$ parallel detector instances. This is shown in figure 5.5. The outputs of the detectors are not shown; they depend on the actual type of detector (i.e. LAPD or CaLIAD). Note that this setup only parallelizes capturing the results and still requires $B-1$ comparisons; if calibration is necessary, $B-1$ individual calibration values need to be stored.

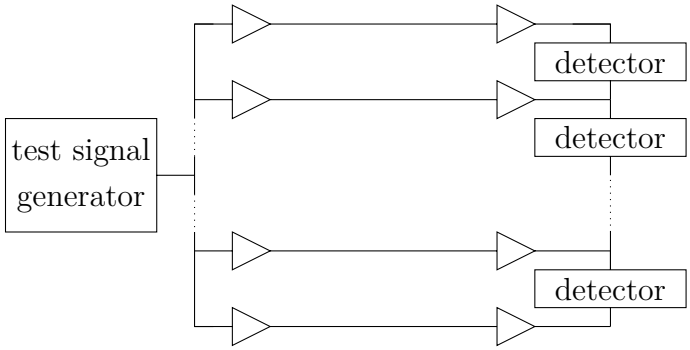


Figure 5.5: B bit bus with $B-1$ parallel detector instances.

Alternately, one detector is sufficient when the bus lines are multiplexed. In that case, two equally sized groups of lines shall be formed; for each comparison, one line is selected from each group using two multiplexers. As an example, one detector input can be multiplexed among all odd lines, and the other input can be multiplexed among all even lines. As this setup implies performing the pairwise comparisons one after another, the measurement time increases by a factor of $B-1$ (or B, depending on the number of comparisons) compared to the parallel setup, while the overhead of multiple detector instances is avoided. Note that the area required by the multiplexers can still be significant. An example of the multiplexed setup is shown in figure 5.6.

A third alternative is obtaining the timing differences between neighboring lines, then summarizing them as described later in this paragraph, and using a slightly modified detector for the actual measurement. In simple terms, the timing difference of lines can be determined by exclusive-OR gates, and the result can be summarized using an OR gate. Regular XOR gates with inputs i and j need a minimum time for which $i \oplus j = 1$ holds to reliably produce a pulse at the output. As this may prevent small-scale microprobes from being detected, Manich and Strasser proposed a Highly Time Sensitive Exclusive-OR (HTS-XOR) gate [MS13], which always produces a pulse in the case of falling transitions; the width of this pulse is sensitive to small changes of the input timing. Therefore, the HTS-XOR gate is preferrable for this use case. When a falling transition is fed simultaneously to the inputs of the bus lines, the output of the OR gate will provide a pulse with a width that reveals the maximum timing difference of

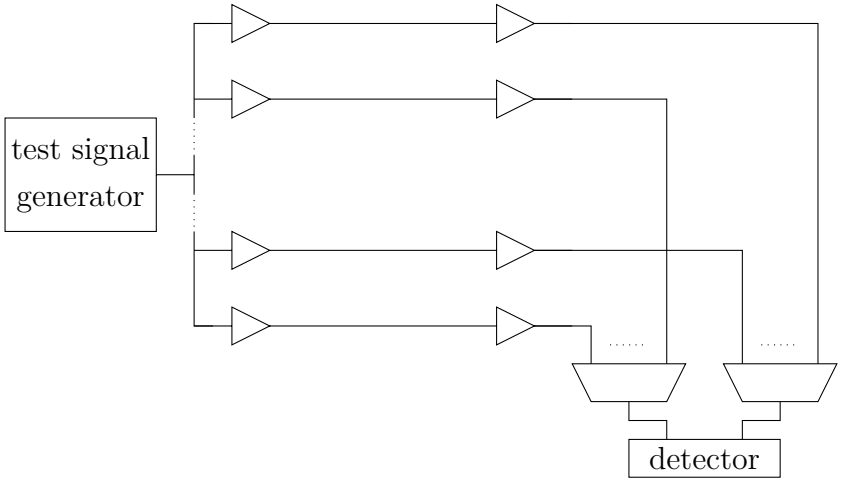


Figure 5.6: B bit bus with one multiplexed detector instance.

neighboring lines. This setup is shown in figure 5.7; note that when using the CaLIAD, which will be focussed here, the logic that generates the `START` and `STOP` signals needs to be replaced by elements that derive these signals from the OR output pulse.

Merging the pairwise line comparison results into one single pulse, whose length can be evaluated by a following detector, comes with the benefit of a low response time despite the need for only one detector instance. In the case of the modified CaLIAD, the PF1 only shifts towards the end of the chain, regardless of what line is probed. Contrast to the original CaLIAD, this setup only needs sensitivity in this direction; this allows to reduce the number of chain elements, thus saving more space. As a drawback, it is recommended to compensate systematic timing imbalances, e.g. caused by slight differences in routing, at the HTS-XOR input; otherwise, they would lead to a reduced probe sensitivity. This approach does not allow to determine the line at which probing has occurred; it can only give "pass/fail" results for the bus as a whole.

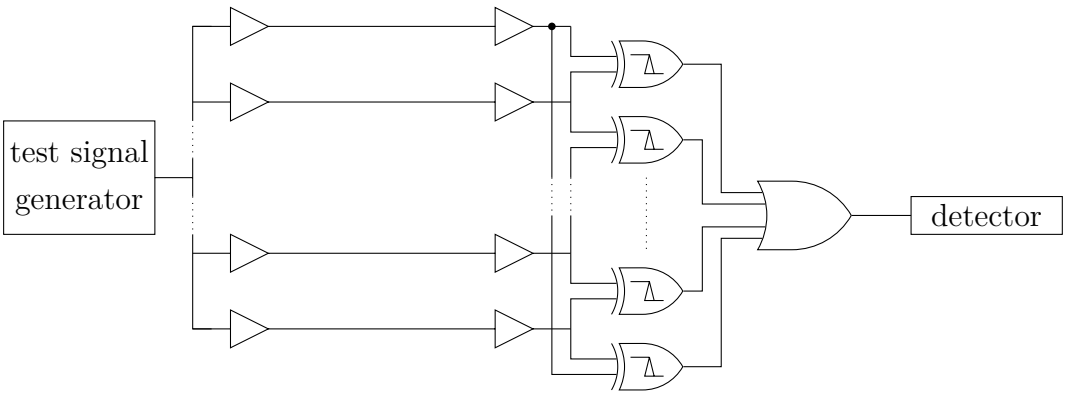


Figure 5.7: B bit bus with a signal merging logic using HTS-XOR gates performing B comparisons, and one adapted detector instance.

A proof-of-concept implementation of this approach is presented in section 5.4.2.

5.4 Results

This section presents simulations that have been performed to verify the viability of integration proposals into bus systems. The first set of simulations aims at showing how a probing detector can be integrated into a WISHBONE bus [Ope10]. Then, the combination of multiple lines using HTS-XOR gates in combination with a modified CaLIAD is observed on an electrical level. Lastly, another set of electrical simulations show how detection performance is influenced by intermediate buffers on bus lines.

For all analog simulations, a 65 nm technology from STMicroelectronics with the low power standard threshold voltage transistors `psvtlp` and `nsvtlp` was used. Cadence spectre 11.1.0 was used to simulate the circuit on a workstation with four AMD Opteron 6274 CPUs and 256 GB of RAM. The simulation sweeps were automated using SALVADOR, as described in section 6.2.

5.4.1 WISHBONE Bus Integration

This section aims at demonstrating the practical feasibility to integrate probing detectors into a multiplexed bus system using the WISHBONE bus. This bus system was chosen for its flexibility with respect to interconnection architectures, and because the availability of the VHSIC Hardware Description Language (VHDL) source code in the public domain [Wis03] facilitates modifications such as the insertion of a probe emulator. The VHDL models were simulated using GHDL. An address bus width of 16 bits, as well as a data bus width of 32 bits, were used in the sample implementation. The considered use case consists of two regular bus masters and two slaves; the data bus shall be protected in both the write and the read direction. A multiplexed interconnection topology as shown in figure 5.1b was used here.

The setup is shown in figure 5.8; it adds a detector controller and a passthrough slave to the bus system. The detector controller contains a test signal generator, actual detectors, and a finite state machine controlling these components, and the passthrough slave forwards the data from the data input to the data output. The detector controller is connected to the bus as a slave on the one hand in order to be configured and triggered by other masters; this can happen before critical operations, such as the transmission of a cryptographic key. On the other hand, the detector controller uses its master interface to send the test signal and receive the response through the passthrough slave. Here, the actual detector consists of $B = 32$ parallel CaLIAD instances to allow parallel monitoring of all bus lines. Registers after the CaLIAD latch chain capture the detector results for reading them out later. Finally, the state machine starts a detection cycle when a "`1`" is written to an internal control register by another master. Its state diagram is shown in figure 5.9. After triggering a detection cycle, the result registers must be read out and evaluated in software. In the given proof-of-concept implementation, the registers contain the "bare" CaLIAD result as a thermometrical code; note that a productive

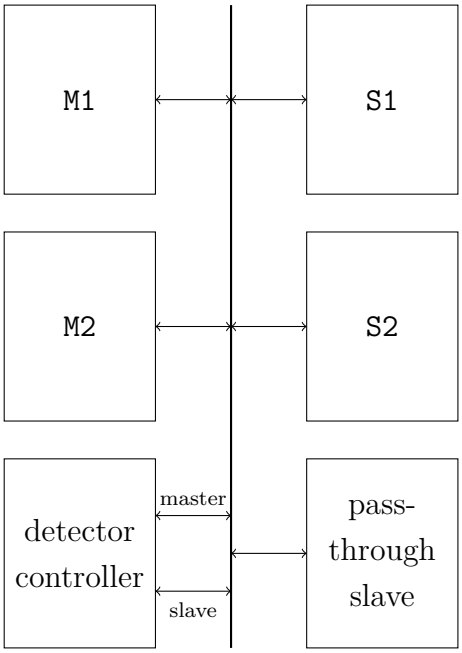


Figure 5.8: Sample bus use case with two regular masters, two regular slaves and two probe detection components.

implementation shall use a more manipulation resistant encoding, as described in section 5.3.3.

In the simulation environment, the read and write requests coming from masters `M1` and `M2` are read from a text file.

This environment can monitor the data bus after the multiplexers in both read and write directions. Additionally, a configuration logic in the passthrough slave allows forwarding the address bus input instead of the data bus input as well in order to monitor the address bus. This can be useful when secrets are contained in the addresses accessed rather than data, such as when using an AES implementation with an S-Box implemented as a lookup table. Note that the most significant bits of the address that are used to select the slave cannot be observed that way without modifying the interconnection logic; generally, this does not pose any limitations, as secret address values usually do not cross slave boundaries.

With the given implementation, one detection run requires 71 bus cycles:

- At first, the passthrough slave is configured to either pass through the address or data bus input to the data bus output. This means writing to a control register and takes one bus cycle.

- The detection run is triggered by writing to a control register through the slave interface of the detector controller. This also takes one bus cycle.

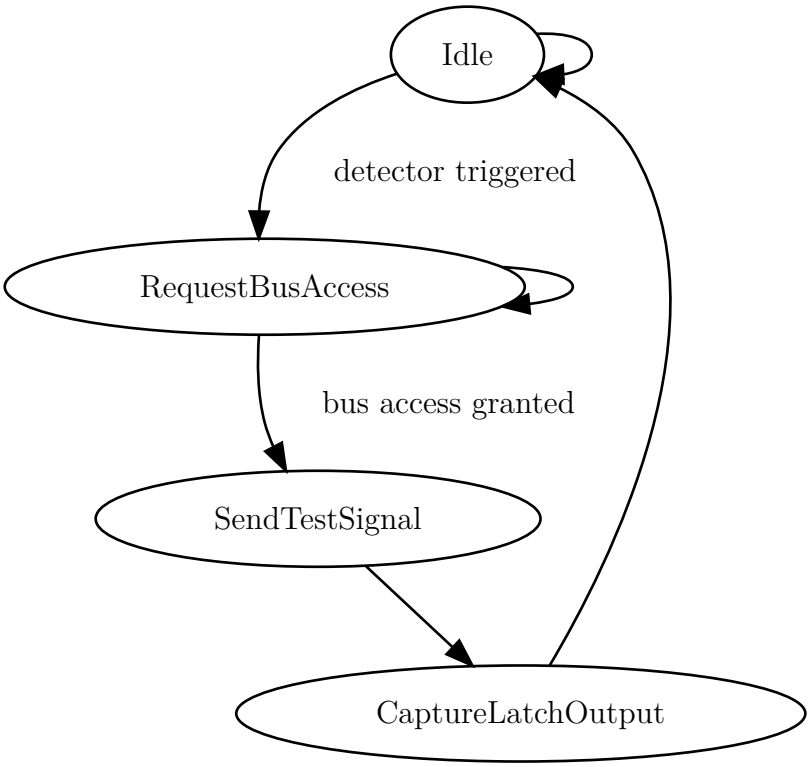


Figure 5.9: State diagram of CaLIAD detector controller.

- In the current implementation, the state machine of the detector controller needs five bus cycles to perform the detection and capture the results in the result register.

- After that, the results can be read out from the result registers. There exists one result register for each of the 32 pairs of lines; thus, this step takes 32 bus cycles.

- Finally, the reference values need to be read from memory. This is also assumed to take 32 bus cycles; however, the actual number may vary depending on the memory timing.

This does not include the comparison itself, which is expected to be done by the CPU core in software.

Note that the exact timings on a complete System-on-Chip (SoC) device are highly dependent on implementation details, such as the memory latency and the relation between CPU and bus clock. Given one completely integrated SoC implementation, deriving generically valid quantitative conclusions about performance would still be challenging without further examination; therefore, such an analysis is left for further work.

5.4.2 Multi-Line Bus Protection with HTS-XOR Gates

As shown in section 5.3.4, the results of the pair-wise line comparisons can be merged before the detector input to reduce the hardware effort needed for multiplexing or for parallel detector instances. As depicted in figure 5.7, HTS-XOR gates can be used for comparison, and an OR gate for merging the result, which is then evaluated by a modified CaLIAD.

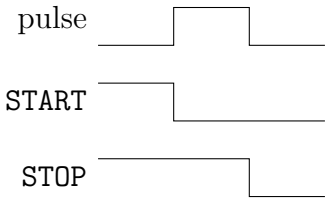


Figure 5.10: Signals START and STOP that must be derived from the pulse coming out of the OR gate that merges the pairwise line comparisons.

The modifications comprise a glue logic that converts the pulse coming out of the OR gate into two different signals START and STOP as shown in figure 5.10. These signals are driving the VDL part of the CaLIAD: START feeds the input of the first slow inverter t_{slow} shown in figure 4.3 and STOP provides the input to the first fast inverter t_{fast}.

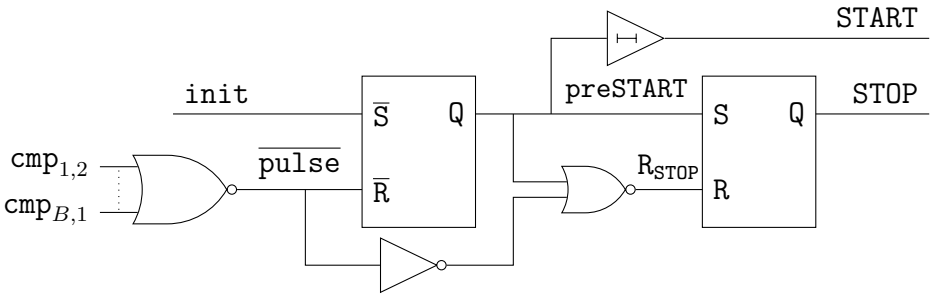


Figure 5.11: Example for glue logic between HTS-XOR gates and VDL part of CaLIAD.

A sample implementation of the glue logic is shown in figure 5.11, and the corresponding timing is depicted in figure 5.12. To reduce the number of required gates, the logic uses a NOR gate instead of an OR gate to merge the HTS-XOR outputs, as shown on the left side. Note that the number of inputs depends on the number of protected bus lines. In the example case of a 32 bit bus, the pull up network would contain up to 32 pMOS transistors connected in series; first simulations have shown that this heavily deteriorates the rise time of the pulse signal. Experiments have shown that a single pMOS transistor in saturation mode (i.e. the gate connected to the drain) exhibits a significantly better timing performance.

The latch on the left consists of two NAND gates. Its output is set to Q = 1 before the pulse using an external signal named init; the latch is used to generate the falling transition of the START signal when the pulse begins. Note that a delay element is used

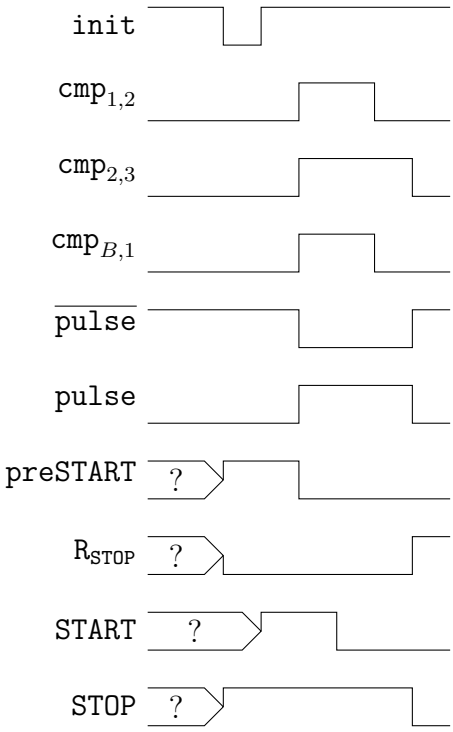


Figure 5.12: Timing of example glue logic.

to compensate the extra delay that comes from the HTS-XOR gate; this allows reducing the number of elements in the VDL chain.

The following latch consists of two NOR gates. It is used to generate the falling transition of the `STOP` signal at the end of the pulse.

Nominal simulations were performed for the use case of an 8-bit bus. The same circuit parameters as described in 4.2 were applied. For the previously described pull-up pMOS transistor of the NOR gate, an aspect ratio of $\left(\frac{W}{L}\right)_{\text{PMOS}} = 11$ was used.

All 256 possible combinations of probes being attached to the bus lines were simulated and grouped by the number of probes. $C_A = 20\,\text{fF}$ was assumed as the attack capacitance value.

The results are illustrated in figure 5.13. The x axis represents the number of probes being attached, and the blue bars denote the mean value of the shift of PF1 compared to the "golden" case without any attached probes. The black bar, which only becomes visible for the case of four attached probes, represents the minimum and maximum PF1 shift. Note that the minimum and maximum values are equal in all other cases.

The area usage of this approach in GEs can be expressed as

$$a_{\text{total}} = B \cdot a_{\text{perLine}} + a_{\text{NOR-Pullup}} + a_{\text{glueLogic}} + n \cdot a_{\text{perChainElement}} \tag{5.1}$$

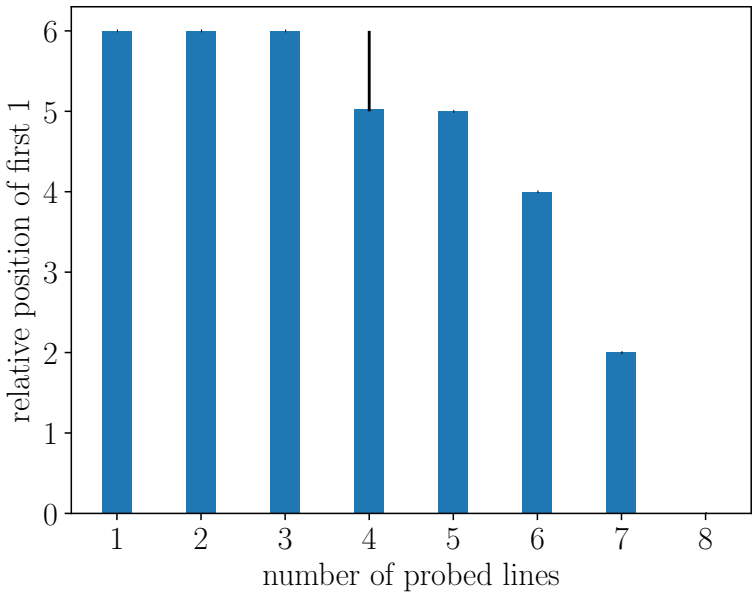


Figure 5.13: Shift of position of the first 1 depending on the number of probes attached; the black line represents minimum and maximum values.

Table 5.1: Area of the individual parts of an HTS-XOR based CaLIAD in gate equivalents.

a_{drivers}	4
$a_{\text{HTS-XOR}}$	8
$a_{\text{NOR-Pulldown}}$	0.625
$a_{\text{NOR-Pullup}}$	3.125
$a_{\text{glueLogic}}$	24
a_{fast}	4
a_{slow}	1.6
a_{latch}	8

with

$$a_{\text{perLine}} = a_{\text{drivers}} + a_{\text{HTS-XOR}} + a_{\text{NOR-Pulldown}} \tag{5.2}$$

$$a_{\text{perChainElement}} = a_{\text{fast}} + a_{\text{slow}} + a_{\text{latch}} \tag{5.3}$$

Using the data from table 5.1, and assuming a CaLIAD chain length of 25, as it was done in chapter 4, one can summarize the area as

$$a = 377.1 + 12.6 \cdot B \tag{5.4}$$

which scales significantly better than a full parallel implementation, the area of which was estimated in table 4.4.

attacked line	attacked segment 1	2	3
L1	≤ -11	≤ -11	-8
L2	10	11	7

Table 5.2: Shifts of the PF1 when loading different line segments with $C_A = 20\,\text{fF}$.

To conclude, the simulations confirm the ability of this concept to detect up to seven probes that are simultaneously attached to an 8-bit bus in the nominal case. The area estimation suggests a significantly lower area consumption compared to the stand-alone CaLIAD implementation.

5.4.3 Handling of Intermediate Buffers

This section aims at analysing the effects of segmented bus lines on the detection capabilities of probing detectors.

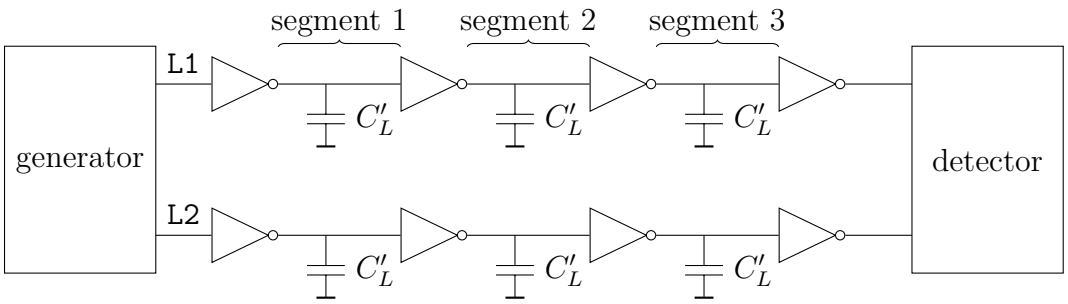


Figure 5.14: Simulated environment of a probing detector protecting a segmented bus line.

For this purpose, the CaLIAD circuit was modified to contain three bus line segments instead of one, which is shown in figure 5.14. The intrinsic capacitance of each line segment was assumed to be divided by three accordingly, i.e.

$$C'_L = \frac{1}{3} C_L = 33\,\text{fF} \tag{5.5}$$

while the transistor dimensions remained unchanged.

The simulations were run using an attack capacitance of 20 fF that was alternatingly applied to the three line segments of the two lines. Table 5.2 shows how the position of the first 1 in the VDL chain shifts compared to the case without attack. Note that without attack, the PF1 occurs at the 12th chain element; when L1 is attacked at segment 1 or 2, the detector reaches the limit of the chain. Compared to the original CaLIAD simulations presented in chapter 4, one can notice an increased sensitivity caused by the reduced intrinsic capacitance per segment.

5.5 Conclusion

Not only the design of probing detectors is important to protect an integrated circuit against invasive attacks; it is equally relevant how and where such detectors are integrated into a device to provide adequate protection while keeping the hardware and timing overhead acceptable.

By discussing different buses and interconnections as well as variants of integrating probing detectors, this chapter serves as a toolbox to system designers helping to understand the pros and cons of the available options with respect to time, area and security level.

A proof-of-concept implementation of the CaLIAD in the WISHBONE bus has given a basic example of a bus integration by explaining the necessary hardware components and timing. Furthermore, different ways to use two-line probe detectors to actually protect B bus lines were presented; in general, the implementation using HTS-XOR gates is recommended when low area and time consumption is important. The last set of simulations shows that the CaLIAD can successfully protect segmented bus lines. Compared to the original scenario, it exhibits an increased sensitivity, which can be beneficial to detect future smaller scale probes.

6 Simulation Environment

Analog simulation is an essential tool for the design space exploration of circuits that are difficult to describe on a more abstract level; typical applications include amplifier or RF circuits. However, they are also handy in the hardware security domain: Circuits such as Ring-Oscillator based Physical Unclonable Functions (RO-PUFs) make use analog properties of circuits that were designed for digital purposes. This is also true for microprobing detectors that are the focus of this thesis.

6.1 Introduction

Well-known analog simulators are spice and spectre, which are often used in combination with large Integrated Circuit (IC) design frameworks such as Cadence Virtuoso [Cad14]. These are focused on accompanying the complete chip design flow and offer tools for digital design synthesis, drawing schematics, performing layout, parasitics extraction, and analog simulations.

While the integration of tools for different IC design stages is very helpful if the actual goal is producing an IC, this work focuses on analog simulations. These were used for exploring the design space and improving the performance of microprobing detectors on the one hand; on the other hand, they were needed to generate data for the plots illustrating the performance of the simulated circuits.

The 65 nm technology library `cmos065` by STMicroelectronics was used for simulations using Cadence Virtuoso 6.1.5 with spectre 11.1.0.509.isr14 under the hood. As large batches of simulations were to be carried out, several limitations could be observed when using the Virtuoso graphical user interface.

- The configuration options for the simulation environment are distributed across different graphical dialogues. This makes configuration error prone.
- Not all relevant messages generated by spectre are passed on to Virtuoso. This includes warnings, such as floating nets, indicating errors in the simulated circuits.
- At some stages of the design, changes in hierarchical circuit components needed to be made; for example, the definition of parameters of an inverter needed to be changed when refactoring simulations. When performing a modification like this, it is desired to maintain the ability to reproduce the results of previous simulations.

With Virtuoso, this implies copying all components or views in a circuit hierarchy from the changed component up to the top-level test bench – which is not only cumbersome, but also impacts clarity, as all circuit components reside in the same namespace.

- Virtuoso allows parameters sweeps, i.e. performing a set of simulations with different combinations of values assigned to circuit parameters, as well as Monte-Carlo simulations for statistical analyses of manufacturing variations. However, Virtuoso does not allow to combine both options in the version used.
- Virtuoso does not support certain features offered by spectre, such as using parameters to select different implementations of a component, e.g. to specify the length of an inverter chain. Note that this has been solved in future versions of Virtuoso [MA15].
- The graphical user interface needs to be visible while simulations are running. It implies for long-running simulations on a remote server that third-party software such as x2go is required to maintain a virtual display (i.e. X server), as otherwise, connection losses would lead to aborting the simulations.

Considering the use case of design space exploration and/or circuit optimization, it is possible to use dedicated circuit optimizers like WiCkeD [Mun] in many cases. Its systematic approach usually outperforms hand-optimization of circuits significantly. WiCkeD is well integrated into the Virtuoso workflow and can help optimizing circuits without needing any additional tools.

However, there are certain limitations: for example, a performance figure that is subject to optimization needs to be visible from one single simulation. It is not directly possible, for example, to formulate constraints like "optimize the dimensions of a flip-flop such that the setup time $t_s < t_{s,\max}$ is fulfilled in 99% of manufactured samples", because this would require multiple Monte-Carlo simulation iterations to obtain the formulated performance value.

Hence, there are use cases of optimization where existing optimization tools are not sufficient. Furthermore, the generation of performance plots can require simulations that cannot be automated by Virtuoso, for example, when it is necessary to combine Monte-Carlo simulations and parameter sweeps.

In the course of this thesis, a framework named SALVADOR has been developed to automate large-scale simulations in a lightweight and portable manner. SALVADOR is not an optimization tool; its purpose is rather

1. instantiating netlist templates (in fact, this is context-agnostic: the templates are directories containing text files with variables, which makes the tool quite flexible),
2. managing the execution of simulations,
3. and collecting their results to have a machine-readable format for further post-processing, such as plotting charts.

While SALVADOR is not a scientific result by itself, discussions with fellow researchers have indicated that the limitations mentioned above were also faced by others, and were often answered by manual repetition of simulations or case-specific monolithic scripts that were difficult to adapt to changing use cases or environments. Therefore, publishing the framework as open source software at https://gitlab.lrz.de/michael.weiner/salvador and providing a description of the motivation and a comprehensive documentation as a part of this thesis nevertheless seems to be a helpful contribution helping the research community to increase tooling efficiency and avoid errors.

SALVADOR was designed to be as independent from the used simulation environment as possible: the generation of simulation instances only comprises inserting variables into text-file templates; the part that generates netlist instances is not aware of any specifics of the netlist description, and therefore can be used with spectre, spice, and any other simulator that uses text files as input. The result extraction is specific to spectre, as it needs to parse log and result files that are generated by the simulator; however, the modular design of SALVADOR allows an easy adaption to other use cases without the need to understand or modify the rest of the code base.

The rest of this chapter is organized as follows: Section 6.2 discusses the tools of which SALVADOR consists and proposes a workflow of how to use them. Then, section 6.3 elaborates potential future improvements and extensions to SALVADOR. Eventually, section 6.4 concludes this chapter. Parts of this work have previously been published in [WMB16].

6.2 The SALVADOR simulation framework

A lightweight simulation framework was developed to compensate the limitations described in section 6.1. It is named Simulation Automation Library for Verification and Analysis of Design Operating Regions (SALVADOR). The following design criteria were considered during its design:

- All configuration settings of one set of simulations shall be concentrated in one file.
- Combining parameter sweeps and Monte-Carlo simulations within one set of simulations shall be possible.
- The simulator statistics shall be made visible in order to detect potential mistakes in the netlist design that do not cause the simulation to fail, and to give feedback about optimization potential.
- The commands shall be run from the command line in order to support performing simulations in virtual terminals such as `screen` or `tmux` over unstable Secure Shell (ssh) connections.

- The different steps during execution (e.g. inserting parameters into netlists, running simulations, collecting results) shall be performed by independent programs, as suggested by the Unix philosophy. This shall simplify adaptions to different use cases; it shall be possible to re-use parts of the tools in different contexts, and furthermore, users shall only need little context knowledge when they need to make changes.
- The simulation environment shall not be aware of the netlist description language as long as netlists only consist of plain text files. This avoids the potentially huge implementation effort of implementing netlists parsers, and allows portability to different simulators.

SALVADOR is a set of command line tools to accomplish these goals. Figure 6.1 gives an overview of the simulation workflow and shows how the SALVADOR tools can interact with each other. Rectangular boxes represent programs belonging to either SALVADOR or to third parties, and parallelogram-shaped boxes denote data. Edges between the nodes resemble the data flow; note that the data exchanged between `combine` and `instantiate` is omitted here for reasons of simplicity. The following paragraphs will describe the data formats and explain how to use the tools of the framework. Note that the modular design also allows to re-use parts of SALVADOR in other contexts, e.g. when other file formats are desired for result storage.

To begin, it is necessary to have a netlist template containing variables, as well as a simulation parameter file that contains concrete values for these variables. The workflow up to the point of netlist creation (i.e. the output of `instantiate`) will be demonstrated using a sample circuit described below. The netlist template is a directory that only contains text files with variables following the syntax of Python format strings [Pyt]. Note that the netlist files listed below have been simplified for better readability.

As an example, the circuit in figure 6.2 will be simulated. It contains two lines with capacitive loads; the top line is loaded with $C_0 + \Delta C$, the bottom line is loaded with only C_0. The bottom line has two additional inverters connected in series to compensate the delay introduced by ΔC in the top line. An exemplary goal is characterizing the timing difference Δt that occurs at the output as a function of C_0, ΔC and the transistor dimensions. It is assumed $C_0 = 100\,\mathrm{fF}$. spectre, the simulator, uses the following variables:

- `g_driversize` describes the nMOS transistor aspect ratio of the first and last drivers of both lines,
- `g_delaysize` describes the nMOS transistor aspect ratio of the delay chain at the bottom line after the capacitor,
- `g_delaylength` is the number of inverter pairs that form a delay chain at the bottom line after the capacitor,
- and `g_deltac` represents ΔC from the circuit diagram.

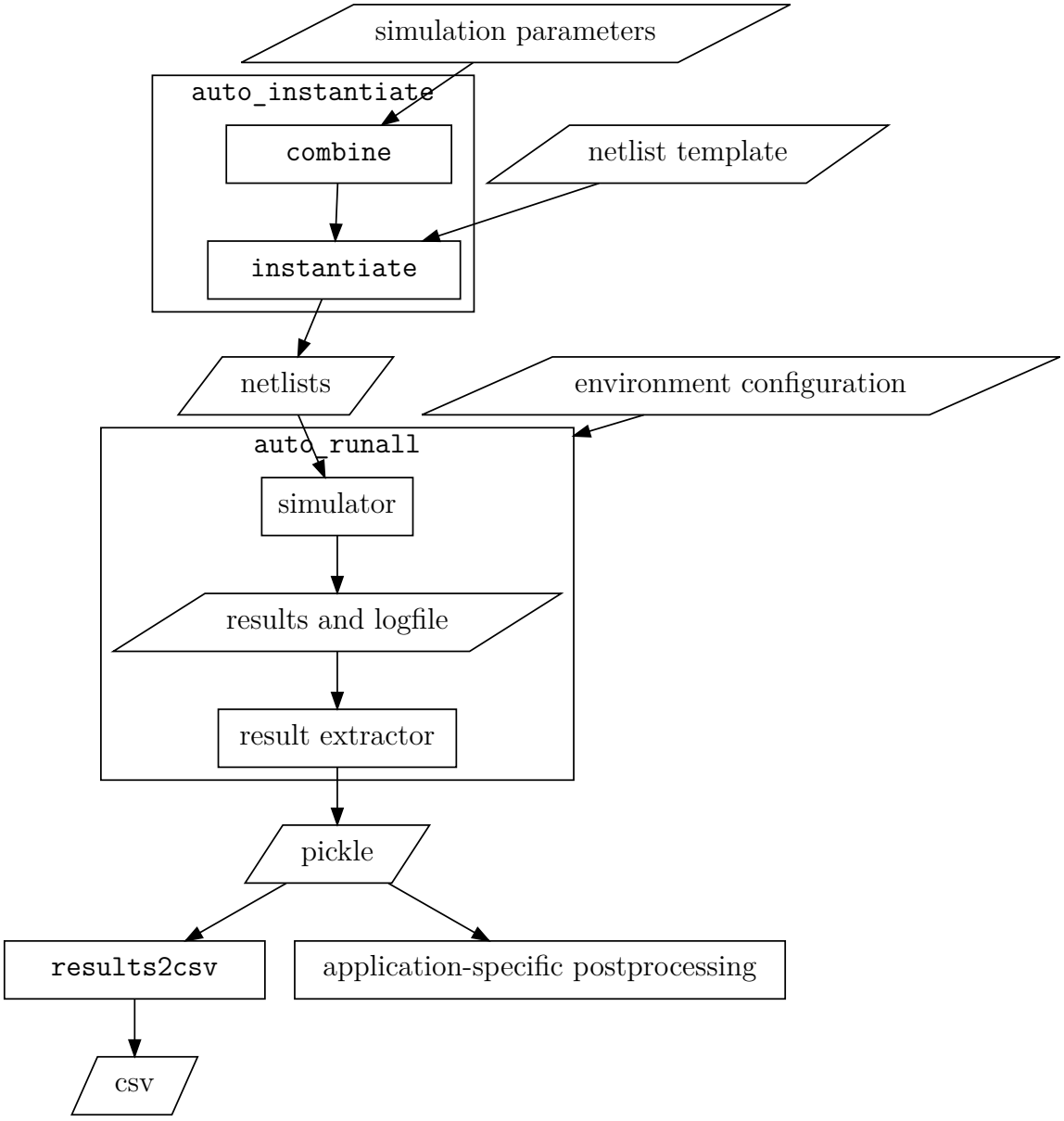


Figure 6.1: suggested SALVADOR workflow.

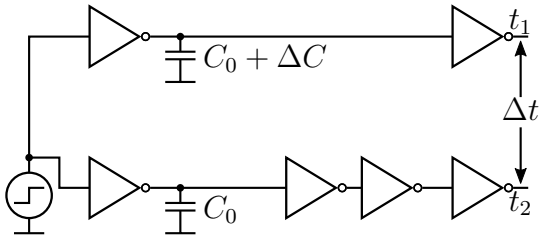


Figure 6.2: example circuit for simulation description.

It is assumed that the listing below are the file contents of `tb_demo_simplified/input.scs`, where `tb_demo_simplified` is the name of the template directory.

Note that the composition of spectre netlists that are generated by Virtuoso is elaborated in detail in the appendix in section A.5.

```
1  parameters g_driversize={fDriverSize:f}
2  parameters g_delaysize={fDelaySize:f}
3  parameters g_delaylength={dDelayLength:d}
4  parameters g_deltac={dDeltaCLine_fF:d}*1.0e-15
5
6  // test signal
7  //
8  // t/ns    0  100
9  //         | . |
10 //            _
11 // in_s    \_/ \
12 //
13 //
14 Vpulse (in_s 0) vsource dc=0 type=pulse val0=0 val1=1.2 period=100n \
15   delay=50n rise=10p fall=10p width=50n
16
17 // bus
18 I1_mid (in_s    line1_s 0 vdd!) inv nmos_aspect_ratio=g_driversize
19 I1_out (line1_s out1_s  0 vdd!) inv nmos_aspect_ratio=10
20 I2_mid (in_s    line2_s 0 vdd!) inv nmos_aspect_ratio=g_driversize
21 I2_out (l2del_s out2_s  0 vdd!) inv nmos_aspect_ratio=10
22
23 I2_delay (line2_s l2del_s 0 vdd!) inv_chain \
24   nmos_aspect_ratio=g_delaysize chain_length=2*g_delaylength
25
26 CLine1 (line1_s 0) capacitor c=100f+g_deltac
27 CLine2 (line2_s 0) capacitor c=100f
```

Lines 14-27 contain instantiations of circuit elements according to the regular spectre syntax. In the first four lines, one can see four SALVADOR parameter definitions:

1. `fDriverSize` of type `f` (float)
2. `fDelaySize` of type `f` (float)
3. `dDelayLength` of type `d` (integer)
4. `dDeltaCLine_fF` of type `d` (integer)

The simulation parameter file shown below contains values for these parameters.

```
1 {
```

```
2   # circuit parameters
3   "fDriverSize": (3,10,),
4   "fDelaySize":  (3,),
5   "dDelayLength":(1,),
6
7   # line/attack parameters
8   "dDeltaCLine_fF":  (0, 20,),
9 }
```

Its syntax is very similar to JavaScript Object Notation (JSON), but in fact it is Python code describing a hash table (more precisely, a `dict`). This way, the parameters can be annotated using Python-style comments; in contrast, JSON has no syntactic support for comments. The Python type `dict` contains list of key-value pairs in the syntax "`key1: item1, key2: item2, ...`". In this context, keys represent variable names; the associated items are sets of corresponding values to be inserted. The tool `combine` parses this data and generates the Cartesian product of all sets, i.e. all combinations of variables. In this example, four tuples would be generated: `fDriverSize`×`fDelaySize`× `dDelayLength` × `dDeltaCLine_fF` = $\{(3, 3, 1, 0), (3, 3, 1, 20), (3, 10, 1, 0), (3, 10, 1, 20)\}$

`combine` supports two types of key-value pairs:

- In the simplest case, a key corresponds to *one* template variable, as shown in the example above. In this case, keys are of type `str` containing the netlist variable names, and values are `tuple`s of a type that is consistent with the type specification in the netlist template.
- It is also possible to define keys that correspond to more than one template variable. This is useful when it is not desired to create all combinations of values for given sets of variables, but specify the list of combinations by hand. In that case, keys are `tuple`s containing the netlist variable names, and values are `tuple`s of `tuple`s containing the desired combinations.

For example, if only the combinations $\{(3, 0), (10, 20)\} \subset$ `fDriverSize` × `dDeltaCLine_fF` shall be considered instead of all four combinations, it is possible to replace lines 3 and 8 from the example with the line below.

```
1 ("fDriverSize", "dDeltaCLine_fF"): ( (3,0), (10,20) ),
```

The output of `combine` is an array of hash tables in JSON (or, in JSON terms, an array of objects), where each array element represents the values to be inserted into one netlist instance. This data is used by `instantiate` to insert values into the template variables of all files in the template directory. In the first example, `instantiate` would create four directories named `00000`, `00001`, `00002` and `00003` that contain copies of the template directory with the concrete values inserted into the template variables. In addition, a file named `params.values` that contains the inserted values in JSON format is added to each directory in order to maintain the relation between inputs and results.

`auto_instantiate` is a short shell script wrapping the calls to `combine` and `instantiate`, which are usually called together. It assumes a directory and file nomenclature that shall help keeping the simulations consistent. Note that this may change in future versions; the reader is therefore advised to read the documentation at https://gitlab.lrz.de/michael.weiner/salvador.

- A template named `${TEMPLATE_NAME}` is stored in `templates/${TEMPLATE_NAME}`. In the example above, `${TEMPLATE_NAME}` is `tb_demo_simplified`.
- `paramsets/${TEMPLATE_NAME}/${SWEEP_NAME}.tuples` is the name of the simulation parameter file.
- By default, the sweeps (i.e. simulation instances) are stored in `sweeps/${TEMPLATE_NAME}.${SWEEP_NAME}`. `auto_instantiate` also allows to specify an alternative sweep directory, which is useful, for example, for cases where large amounts of data produced by simulations shall only be stored in a temporary location.

Up to this point, the workflow is completely context free; the only constraint is that the template must be printable text. This gives a high degree of freedom with respect to the selection of the simulator as well as its configuration. The workflow up to the template instantiation can also be used in completely different contexts that are unrelated to analog simulation.

`auto_runall` is then in charge of running the simulations and collecting the results. It is written in a generic manner as well, but calls context-aware commands that are specified in a configuration file. Specifically, `auto_runall` performs three tasks:

1. Each netlist instance is expected to contain an executable program that is executed by `auto_runall`. The user can specify a number of simulations to be executed in parallel. The name of the executable program is defined in the configuration file as the variable `${RUN_SIMULATION}`; in the case of Virtuoso-generated netlists, this would be `runSimulation`. Note that simulations sometimes fail for technical reasons, e.g. when all licenses are occupied. Therefore, `auto_runall` checks for common errors using the simulator-specific command `${GET_PENDING_SIMS}`.
2. After simulations, a program defined as `${GET_RESULTS}` is called to collect the results from all netlist instances and store them in one file.
3. The netlist directories are packed into an Lempel-Ziv-Markov chain algorithm (LZMA)-compressed archive (`tar.xz`) for documentation purposes. Despite the fact this is not needed in most cases, it allows evaluations going beyond the results captured in the result file.

The following paragraphs will focus on the use case of spectre using the `montecarlo` environment, which allows to specify Open Command Environment for Analysis (OCEAN) expressions within the netlist. Note that this environment can also be used for deterministic simulations. Here, `${GET_RESULTS}` is set to `spectre_mc_getresults`, which

is included in the SALVADOR repository. It reads data as inputs from each simulation instance and creates a `tuple` consisting of three `dict`s as follows.

- The simulation parameters are read from a JSON file named `params.values` that is generated by the SALVADOR tool `instantiate`. After parsing, the first `dict` contains the exact contents of this file.
- The OCEAN expressions are read from `monteCarlo/mcparams`, the corresponding results are taken from `monteCarlo/mcdata`. The second `dict` contains the OCEAN expression names as keys, and the corresponding results as a `list` of `float` numbers. The number of Monte-Carlo iterations is equal to the number of elements in the list.
- Simulation statistics are taken from `spectre.log` and stored into the third `dict` with predefined strings as keys. This include simulation time (keys `"time_cpu"` and `"time_elapsed"`), timestamp of beginning (`"timestamp_started"`) and end of simulation (`"timestamp_stopped"`), memory usage (`"memory_peak"`) as well as the number of errors (`"errors"`), warnings (`"warnings"`) and notices (`"notices"`).

These tuples are serialized and stored into a file for further processing. The module `pickle` from the Python standard library is used for that purpose. Note that the `tuple` of each instance is serialized individually, instead of, for example, being packed into one large `list` that would contain all results. This avoids the need to accumulate all results in RAM, which would have a heavy impact on performance for large simulations. Thus, the tool that post-processes the results needs to call `pickle.load()` n times for a sweep consisting of n simulation instances. However, the calls to this function are usually made in an endless loop stopped by the `EOFError` exception, as this avoids the redundant storage of the number of instances as metadata explicitly.

The format of the result file should allow the usage with other simulators as long as they allow to define performance expressions and assign them names; only the available statistics may vary.

As a last step in the generic workflow, `results2csv` converts the results into a Comma Separated Values (CSV) file. This allows quickly reviewing the results, which is recommended to be done before further case-specific processing. The results of the example circuit are shown in figure 6.3. One line represents one simulated circuit; columns A to D denote the input parameters that were referenced in the listing of the simulation parameter file, and the resulting delay, which was described as an OCEAN expression in the netlist template, is given in column E.

6.3 Future Work

The set of features supported by SALVADOR was sufficient to perform all analog simulations for this thesis in a manner that is significantly more efficient than with Virtuoso.

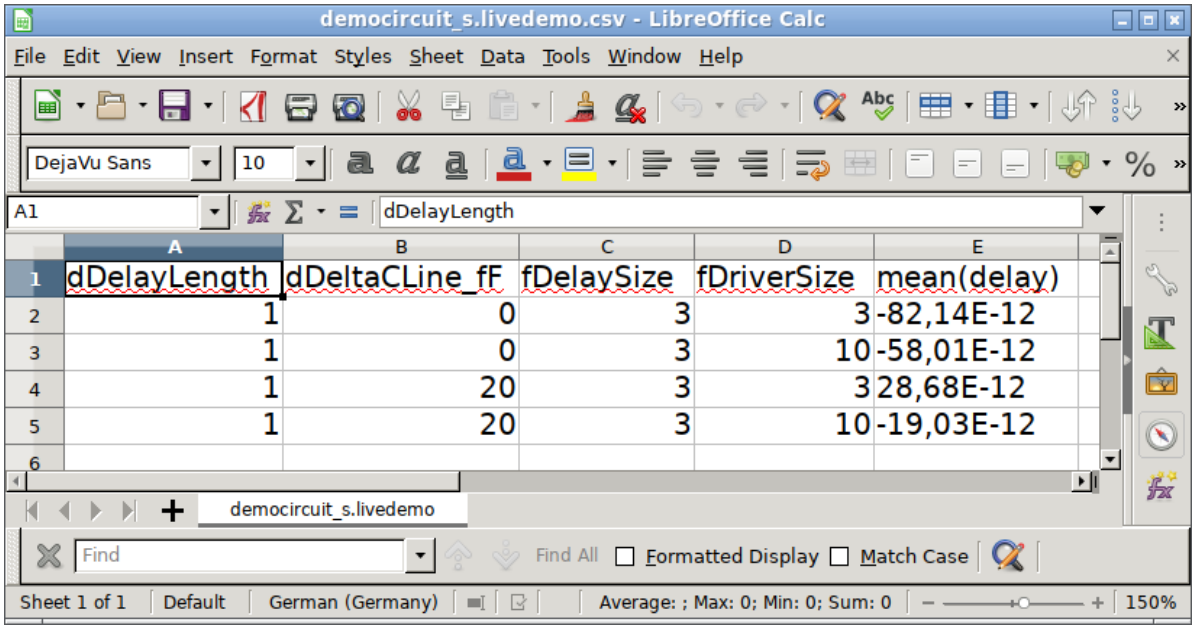


Figure 6.3: output generated by `results2csv` from example circuit.

However, there are several features that may extend the usability for third parties or for other projects.

As of now, OCEAN expressions can only be extracted by SALVADOR tools using the `montecarlo` environment of spectre. One improvement would be implementing an alternative to `spectre_mc_getresults` that evaluates the OCEAN expressions based on the raw data instead of having spectre perform this task. An open question here is where the OCEAN expressions shall be inserted into the workflow. Possible alternatives are:

- providing them as part of the simulation parameters;
- providing them as part of the netlist. This is similar to the way that is implemented now;
- introducing a new input to the result extractor. This would require an adaption of the `auto_runall` workflow.

Furthermore, the support for other simulators – most prominently, spice or one of its variants – can be implemented.

6.4 Conclusion

SALVADOR is a generic analog simulation automation framework designed to support large-scale simulation sets. Its modular design allows using SALVADOR with any simulator that supports to receive inputs as text files. The simulator-specific parts of the

workflow are currently implemented for spectre, which was used for the analog simulations in this thesis, and it is possible to add support for other environments without detailed context knowledge of the other process steps or tools used. Compared to state-of-the-art frameworks such as Cadence Virtuoso, it gives a higher degree of flexibility and avoids errors caused by manually repeating non-automatable steps. In addition to result extraction, it allows to parse the log file to detect possibly important warning messages that would otherwise not attract attention. SALVADOR is open source software that is, as of March 2020, available at https://gitlab.lrz.de/michael.weiner/salvador.

7 Conclusion and Future Work

In this thesis, mechanisms to detect microprobing were developed and evaluated with respect to their performance; furthermore, it was elaborated how the detection mechanisms can be integrated into existing integrated circuit concepts.

Certain evaluations required large sets of analog simulations. This included varying parameters in netlists, managing the parallel execution of simulations as well as summarizing the results in a way that simplifies post-processing. For this purpose, SALVADOR has been developed. It is a flexible open source framework that can parametrize netlists; the simple syntax-agnostic approach of inserting variables into text files makes this approach usable for most analog simulators such as spectre or hspice; it may even be helpful in use cases beyond analog simulation. SALVADOR then uses simple Unix shell commands to allow executing a specified number of executions in parallel. Finally, it extracts the simulation results and stores them in a format well suited for post-processing; this step is simulator-dependent, but the flexible architecture simplifies adaptions to other use cases.

The Low Area Probing Detector has been developed as an area efficient microprobing detector that compares the delay introduced by the parasitic capacitance with the delay of an inverter chain. This approach has the lowest area overhead of detectors focused on the intrinsic effects of microprobing. In the tested 65 nm technology, it is capable of detecting the smallest commercially available microprobes that were found during the work of this thesis; however, it still requires optimizing transistor dimensions for best performance.

One disadvantage of the LAPD, as compared to, for example, the PAD is its inability to compensate manufacturing variations. For this purpose, the CaLIAD has been developed. It compares the delay differences of neighboring lines using a VDL chain that produces a thermometrical code as a response; this response can be stored as a calibration value. This approach allows a significantly better detection performance compared to the LAPD and does not need time-consuming optimizations; still it exhibits a lower area than the PAD.

The two presented detection concepts in its basic form only discussed how to protect one line out of two, and without considering constraints of real bus systems. Therefore, the last chapter of this thesis addressed how to integrate detectors in a real bus system. It showed, for example, three different approaches how to protect more than two lines – in most cases, the approach using HTS-XOR gates is recommended. Furthermore,

a sample integration into the WISHBONE bus demonstrated what components and operations are necessary to perform a detection run.

To conclude, this thesis has enabled to include the detection of intrinsic effects of microprobing into existing integrated circuits, such as microcontrollers. Contrast to many other countermeasures, this also permits to detect less common attack techniques such as backside probing.

In some cases, e.g. when protecting serial buses, it is desirable to protect all B lines instead of only $B - 1$. In such a case, an additional reference line can be added; it can, for example, be dimensioned with a slightly different driver size, such that probing the reference line will change the timing behavior differently than the asset lines. Further analysis in this direction can help applying the presented protection concepts to lines other than classical buses, such as active shields, serial buses and Network-on-Chip (NoC) interconnects.

Another aspect of which a further analysis seems helpful is security of the storage of calibration values and the evaluation. It may depend on implementation details, such as the chosen encoding of calibration values, or the selection of the component that performs the actual pass/fail checking.

Lastly, but not less importantly, an ASIC implementation would be able to evaluate the performance of detection circuits far beyond the capabilities of simulations.

A Appendix

A.1 Alpha-Power Model

The delay of a Complementary Metal Oxide Semiconductor (CMOS) driver can be estimated using the alpha-power model for the transistors [SN90, BAE+99]:

$$d = \tilde{k}\frac{C\ V_{DD}}{(V_{DD} - V_t)^{\alpha}} \tag{A.1}$$

C is the capacitive load at the buffer output and V_{DD} denotes the supply voltage. V_t is the threshold voltage of the transistors, α represents the velocity saturation coefficient of the carriers and $\tilde{k}$ is called the trans-resistance that summarizes the remaining transistor parameters [BCCK07]. It is assumed that the technological parameters between nMOS and pMOS transistors are balanced with respect to the output transition times. Furthermore, it is assumed that the signals in the lines exhibit the full swing between *GND* and V_{DD} for Equation (A.1), otherwise the approximation would significantly deviate from the real behavior. This last assumption is quite reasonable since an attack will always try to disturb the observed signals as little as possible.

Summarizing the transistor parameters and the supply voltage as Ω, one gets

$$d = \Omega C \tag{A.2}$$

where Ω is the lumped technological parameter

$$\Omega = \tilde{k}\frac{V_{DD}}{(V_{DD} - V_t)^{\alpha}} \tag{A.3}$$

The delay difference of two lines with equally dimensioned transistors and independent line capacitances C_i and C_j can then be described as

$$\Delta d = d_j - d_i = \tilde{k}\frac{(C_j - C_i)V_{DD}}{(V_{DD} - V_t)^{\alpha}} = \Omega\ (C_j - C_i) \tag{A.4}$$

A.2 Dependency between Input Slew Rate and Effects of Mismatch Variations for Inverters

The LAPD principle of operation is based on the detection of the delay difference Δt_{RS} arriving at the latch inputs. The transitions arriving at the latch are delayed by chains d_1, d_2 and t_D, t_D being alternated between the two chains during the two operating cycles. Process variations alter the propagating delay of these three chains in such a way that the magnitude of Δt_{RS} becomes unstable at a certain degree and therefore less predictable.

These effects cannot be avoided completely but diminished at a certain degree as it is seen in section 3.2.4. In particular, after a first assessment it is observed that inverter $\mathbf{D_{out}}$ has a significantly larger influence on the variability than the rest of stages and therefore the optimization is further concentrated on this inverter.

To understand why this is the case, the focus can be put on the delay propagating model of a single inverter that was presented by Shoji in [Shō88], which is summarized in this section.

In Figure A.1, a simple inverter is shown with the corresponding input / output transitions. Input / output slew-rates [$\mathrm{V\,s^{-1}}$] are α_I and α_O respectively. The delay of the gates is calculated at 50% of the signal levels and is symbolized by T_{OI}. Internally, the pMOS and nMOS transistors have transconductances [$\mathrm{A\,V^{-1}}$] that are represented by b_N and b_P respectively, and have a big contribution to the switching speed of the output, together with the load capacitance.

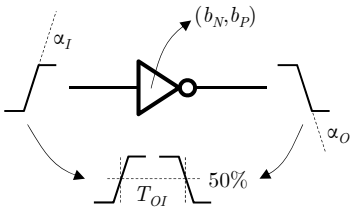


Figure A.1: Input / output slew-rates and delay of an inverter.

The following paragraphs will analyze the two possible scenarios in which the variability can disturb the propagating delay.

Inverter delay is independent of the input slew-rate Intuitively, it seems apparent that if the input slew-rate is extremely fast ($\alpha_I \to \infty$), the propagating delay ($T_{OI} \to t_\infty$) will exclusively depend on the inverter load capacitance and the (dis-)charging transistor transconductance, which is b_N for the falling output transition shown in figure A.1. In each inverter, the delay will be affected by the variations of the (dis-)charging transistor transconductance and the load capacitance dimensions (typically the input of the next

stage). These two elements will generate variability in the propagating delay (t_∞) but it will be independent of the input slew-rate variability produced by the previous stage. Therefore, the total delay variability of a chain of inverters will be the sum of the independent variabilities of each inverter stage, and it will typically become a normal random distributed variable whose variance will be the sum of the variances of each inverter delay.

This scenario is the most favorable in terms of reducing the effects of process variabilities. Minimizing tactics are fundamentally based on placing strategies and enlarging the dimensions of transistors in order to reduce the percentage of the variability over the total physical dimensions. However, this is at the cost of more area and is only partially applied until the range of the circuit tolerance is achieved.

Inverter delay is dependent of the input slew-rate When the input slew-rate α_I is significantly smaller than the output slew-rate α_O, the delay of gate T_{OI} becomes sensitive to it too. Particularly, the degree of sensitivity follows a hyperbolic function whose growing degree depends on the ratio between pMOS and nMOS transconductances. Therefore, if the previous inverter controlling the input slew-rate is considered, its variability will propagate to the next inverter through the variability in the slew-rate, and at the same time it will affect the next stage delay with a contribution much stronger than the simpler addition seen in the previous scenario.

In the LAPD circuit in figure 3.1, this effect is clearly observed in the inverter $\mathbf{D_{out}}$ because it receives the input from heavily loaded bus-lines and the output drives smaller gates like the internal delay chain **T** and the multiplexer **M**.

The reduction of the variability effects in this scenario is achieved by doing a proper balance of the pMOS and nMOS transconductances as it will be clear from the Shoji delay model presented below.

To model the delay under variable input slew rate conditions, the transconductance ratio $\beta = b_N/b_P$, the normalized input slew-rate $S_l = \alpha_I/\alpha_O$ and the normalized inverter delay $T_{inv} = T_{OI}/t_\infty$ shall be defined first.

According to Shoji [Shō88], the delay can be approximated by the following closed expressions if the transistor models are linearized and fixed to a zero threshold-voltage:

$$\begin{aligned} T_{inv} &= 1 + \frac{1}{2(1+\beta)}\frac{1}{S_l};\ S_l \geq \frac{\beta}{2(1+\beta)} \\ T_{inv} &= 2\sqrt{\frac{\beta}{2(1+\beta)}\frac{1}{S_l}} + \frac{1-\beta}{2(1+\beta)}\frac{1}{S_l};\ S_l < \frac{\beta}{2(1+\beta)} \end{aligned} \tag{A.5}$$

Interestingly, for more realistic transistor models (including non-zero threshold voltages) Shoji shows that the dependency of T_{inv} from S_l will follow the same law despite a closed expression could not be found.

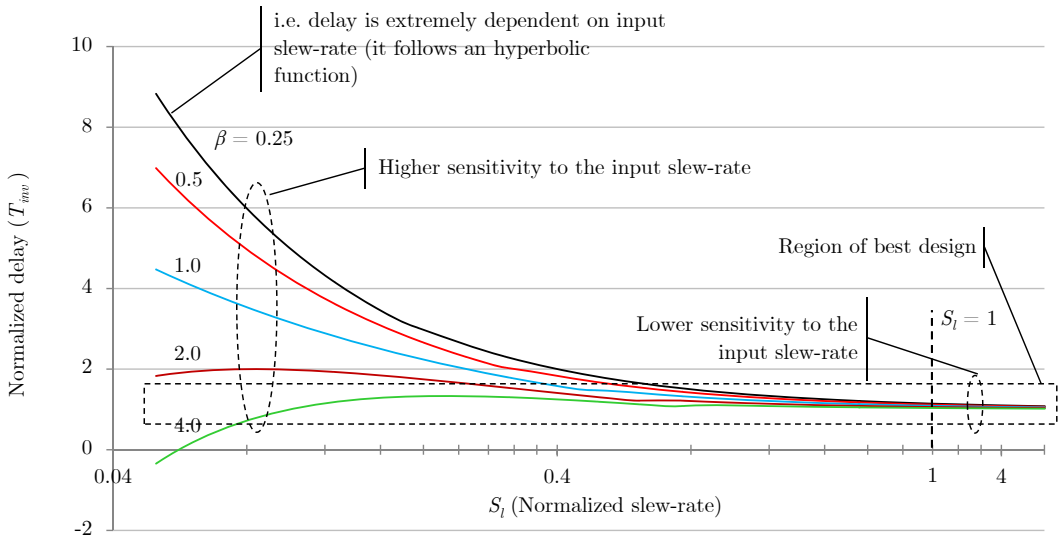


Figure A.2: Variation of the normalized inverter delay as a function of the normalized input slew-rate [Shō88].

Equation (A.5) is plotted in figure A.2. At the x-axis, the normalized input slew-rate is fixed and it has two main regions (separated by a dashed line): larger and smaller than 1. For values larger than 1, the input transition is faster than the output while for values smaller than 1, the input transition becomes slower than the output one. At the y-axis, the normalized delay is presented. When it is 1, the delay of the inverter is exactly t_∞ and is equal to the delay when the inverter input switches very fast. Each one of the curves represents a different β ratio going from 0.25 to 4.

When S_l is higher than one, all the curves closely coincide and are almost equal to 1. This shows that the delay of the inverter is almost independent of the input slew-rate S_l and that the transconductance ratio β does not have any importance with respect to the process variations.

When S_l is smaller than one, curves diverge and thus the sensitivity of the inverter delay becomes stronger to the input slew-rate S_l. This is clearly seen in the curve $\beta = 0.25$, that for a variation of S_l from 0.05 to 0.042 (a relative change of 13%) the normalized inverter delay changes from 6 to 8 approximately (a relative change of 29%). This strong dependence can be reduced by tuning the β ratio at the proper value. In the plot, a dotted rectangle indicates the **region of best design**. The transconductance ratio should be adjusted such that the normalized delay is kept inside this region.

While this process cannot be done analytically given the complexity of the transistor models, simulations can be used to find the best transconductance ratio β, i.e. like for the critical inverter $\mathbf{D_{out}}$.

A.3 LAPD model fit

This section aims at evaluating the accuracy of the LAPD model described in section 3.1.2, i.e. the inequalities (3.8) and (3.9) assuming (3.12) and (3.13), the latch consisting of gates $\mathbf{N_1}$ and $\mathbf{N_2}$ in figure 3.1 was characterized to find out t_H before analyzing the complete circuit to determine the product $k_{\mathbf{D_{out}}}\Omega$.

A sweep over Δt_{RS} at the latch inputs was performed to estimate the minimum "hold time" t_H for which the output is reliable, i.e.

$$\Delta t_{RS} < 0 \;\Rightarrow\; p(\mathtt{Q} = 0) > 0.99 \tag{A.6}$$

$$\Delta t_{RS} > 0 \;\Rightarrow\; p(\mathtt{Q} = 1) > 0.99 \tag{A.7}$$

hold. $N = 2000$ Monte-Carlo simulations were used with a Wilson score interval [Wil27] at a confidence level of $\alpha = 0.01$ to estimate the bounds of t_H and obtained a value of $t_H = 1.40\,\mathrm{ps}$. This value summarizes the mismatch variation of the latch.

To continue the analysis, equation (3.6) was solved for

$$k_{\mathrm{D_{out}}}\Omega = \frac{t_D + t_{\mathrm{M2}} - t_{\mathrm{M1}} - \Delta t_{RS}}{C'_A} \tag{A.8}$$

and then, the mean and variance of this value were determined at different C'_A sweep points by simulation of the complete LAPD circuit; the values of t_D, $t_{\mathbf{M1}}$ and $t_{\mathbf{M2}}$ were captured as well. Three sigma distances from the mean $E(k_{\mathbf{D_{out}}}\Omega)$ were then used to estimate the reliability bounds:

$$C^*_{0.01} = \frac{t_D + t_{\mathbf{M2}} - t_{\mathbf{M1}} - t_H}{E(\Omega\; k_{\mathbf{D_{out}}}) + 3\sqrt{\mathrm{Var}(\Omega\; k_{\mathbf{D_{out}}})}} \tag{A.9}$$

$$C^*_{0.99} = \frac{t_D + t_{\mathbf{M2}} - t_{\mathbf{M1}} + t_H}{E(\Omega\; k_{\mathbf{D_{out}}}) - 3\sqrt{\mathrm{Var}(\Omega\; k_{\mathbf{D_{out}}})}} \tag{A.10}$$

Table A.1 shows the approximated threshold capacitances using $N = 2000$ Monte-Carlo simulations at different sweep points. The reference values are listed in the "typical" column of table 3.6. One can see that with increasing C'_A, the difference $C^*_{0.99} - C^*_{0.01}$ shrinks. Also, the error of $C^*_{0.01}$ increases; in absolute terms, the maximum error occurs with the initial design at $C'_A = 40\,\mathrm{fF}$ is $\Delta C_{0.01} = 16.3\,\mathrm{fF} - 11.2\,\mathrm{fF} = 5.1\,\mathrm{fF}$. On the other hand, an increasing C'_A also leads to a reduction of error for the $C_{0.99}$ estimation: At $C'_A = 40\,\mathrm{fF}$, the worst case occurs at the optimized design at $\Delta C_{0.99} = 32.7\,\mathrm{fF} - 34.4\,\mathrm{fF} = -1.7\,\mathrm{fF}$. In relative terms, the approximation is more accurate for $C_{0.99}$ at higher values of C'_A than for $C_{0.01}$ at low values of C'_A. Therefore, it seems recommendable to focus on these values when using the model.

The proposed linear model appears sufficient for qualitative comparisons between different LAPD implementation variants and helps to significantly reduce the number of required simulations; for precise quantitative analyses, a more elaborate model seems advisable.

Table A.1: Approximated Threshold Capacitances.

		C'_A sweep point			
		10 fF	20 fF	30 fF	40 fF
initial design	$\frac{C^*_{0.01}}{\text{fF}}$	10.0	13.7	15.4	16.3
	$\frac{C^*_{0.99}}{\text{fF}}$	< 0	75.6	44.4	36.3
optimized design	$\frac{C^*_{0.01}}{\text{fF}}$	13.0	16.8	18.4	19.3
	$\frac{C^*_{0.99}}{\text{fF}}$	258.5	47.3	36.6	32.7

A.4 VHDL Source Code of CaLIAD Emulator

```
1  -- Slow inverter chain is implemented in column 35. 271 - 240 ps per inverter
2  -- (LH - HL fastest conditions).
3  -- Fast inverter chain is implemented in column 37. 267 - 233 ps per inverter
4  -- (LH - HL fastest conditions).
5  -- Delay difference between fast and slow inverter chains is 4 - 7 ps per inverter,
6  -- so 5.5 ps in mean (LH - HL fastest conditions).
7  -- Delay selection is implemented in column 35. Delay added when selected is 61 ps
8  -- (fastest conditions).
9  -- An interconnect delay is expected between delay selection and inverter chains.
10 -- It is expected a large interconnect delay to column 37 (fast inverter chain)
11 -- than to column 35 (slow inverter chain). The interconnect delays should be
12 -- adjusted to facilitate a difference of 77 ps (fastest conditions).
13 -- Latches are implemented in column 36. Interconnect delays from columns 35 and
14 -- 37 must be considered.
15 -- The latches' outputs drive buffers implemented in column 33, thus allowing to
16 -- postprocess them without modifying the delays in the inverter chains.
17 -- Each latch has  a 24-bit counter associated to it to count the number of times
18 -- the latch becomes set due to transmission of a pulse on the transmission lines.
19 -- An experiment cycle consist in the transmission of 10**7 pulses followed by the
20 -- transmission of the results to a PC through an asynchronous link at 19200 bauds
21 -- with the following format (88 bytes in total):
22 -- 1) A header symbol compound of three 0xFF bytes.
23 -- 2) The count for each latch, starting with latch in step 1 and ending with latch
24 --    in step 28. Each count is a 24-bit integer, splitted in three bytes. Bytes
25 --    are sent in big-endian order.
26 -- 3) The mode of the experiment cycle (1 byte).
27 --    Mode 0 = no transmission line is being tapped (balanced response).
28 --    Mode 1 = only transmission line of slow inverter chain is being tapped (set
29 --             signals are delayed, more latches will be set).
30 --    Mode 2 = only transmission line of fast inverter chain is being tapped (reset
31 --             signals are delayed, more latches will be reset).
32 --    Mode 3 = both transmission lines are being tapped (balanced response).
33 -- Experiment cycles are performed continously, incrementing the mode modulus 4 at
34 -- each new experiment cycle.
35
36 library ieee;
37 use ieee.std_logic_1164.all;
38 use ieee.numeric_std.all;
39
40 library altera_mf;
```

```
41 use altera_mf.altera_mf_components.all;
42
43 entity two_chains is
44 generic (
45   version : string := "experiment"        -- "simulation" or "experiment"
46 );
47 port (
48   clk50M  : in std_logic;
49   txd     : out std_logic
50 );
51 end entity two_chains;
52
53 architecture estructural of two_chains is
54
55 constant baudrate : integer range 0 to 2**10-1 := 10009600/19200;
56 type states is (start,measure,byte_load,byte_trans);
57 subtype count10M is integer range 0 to 2**24-1;
58 type count10M_vector is array(natural range <>) of count10M;
59 signal count5 : integer range 0 to 7 := 0;
60 signal clk10M : std_logic := '0';
61 signal state : states := start;
62 signal mode : integer range 0 to 3 := 0;
63 signal sel_delay_f,sel_delay_s : std_logic;
64 signal pulse : std_logic := '0';
65 signal count : count10M;
66 signal out_buf_1,out_buf_2 : std_logic;
67 signal out_mux_f,out_mux_s : std_logic;
68 signal out_buf_f,out_buf_s : std_logic;
69 signal out_buf_dly_s : std_logic;
70 signal out_inv_f,out_inv_s : std_logic_vector(1 to 28);
71 signal out_load_buf : std_logic_vector(1 to 2);
72 signal q : std_logic_vector(1 to 28);
73 signal out_buf_q : std_logic_vector(1 to 28);
74 signal q_count : count10M_vector(1 to 28);
75 signal byte_count_txd : integer range 0 to 87 := 0;
76 signal bit_count_txd : integer range 0 to 10 := 0;
77 signal time_count_txd : integer range 0 to baudrate-1 := 0;
78 signal bits_txd : std_logic_vector(10 downto 0) := "00000000001";
79 alias data_txd : std_logic_vector(7 downto 0) is bits_txd(8 downto 1);
80 signal txd_full : std_logic := '0';
81
82 begin
83
84   div5 : process (clk50M)
85   begin
86     if clk50M'event and clk50M = '1' then
87       if count5 /= 4 then
88         count5 <= count5+1;
89       else
90         count5 <= 0;
91       end if;
92     end if;
93   end process div5;
94
95   clk10M <= std_logic_vector(to_unsigned(count5,3))(2);
96
```

```
97   automata : process (clk10M)
98   variable maxcount : count10M := 0;
99   variable maxtime : integer range 0 to 2**10-1 := 0;
100  begin
101    if version = "simulation" then
102      maxcount := 15;
103      maxtime := 15;
104    else
105      maxcount := 9999999;
106      maxtime := baudrate-1;
107    end if;
108    if clk10M'event and clk10M = '1' then
109      pulse <= not pulse;
110      case state is
111      when start =>
112        if pulse = '1' then
113          for i in 1 to 28 loop
114            q_count(i) <= 0;
115          end loop;
116          count <= 0;
117          state <= measure;
118        end if;
119      when measure =>
120        if pulse = '0' then
121          for i in 1 to 28 loop
122            if out_buf_q(i) = '1' then
123              q_count(i) <= q_count(i)+1;
124            end if;
125          end loop;
126          if count /= maxcount then
127            count <= count+1;
128          else
129            count <= 0;
130            byte_count_txd <= 0;
131            state <= byte_load;
132          end if;
133        end if;
134      when byte_load =>
135        if txd_full = '0' then
136          bit_count_txd <= 0;
137          time_count_txd <= 0;
138          txd_full <= '1';
139          bits_txd(0) <= '0';
140          bits_txd(10 downto 9) <= "11";
141          case byte_count_txd is
142          when 0 => data_txd <= "11111111";
143          when 1 => data_txd <= "11111111";
144          when 2 => data_txd <= "11111111";
145          when 3 => data_txd <= std_logic_vector(to_unsigned(q_count(1)/2**16,8));
146          when 4 => data_txd <= std_logic_vector(to_unsigned((q_count(1)/2**8) mod 2**8,8));
147          when 5 => data_txd <= std_logic_vector(to_unsigned(q_count(1) mod 2**8,8));
148          when 6 => data_txd <= std_logic_vector(to_unsigned(q_count(2)/2**16,8));
149          when 7 => data_txd <= std_logic_vector(to_unsigned((q_count(2)/2**8) mod 2**8,8));
150          when 8 => data_txd <= std_logic_vector(to_unsigned(q_count(2) mod 2**8,8));
151          when 9 => data_txd <= std_logic_vector(to_unsigned(q_count(3)/2**16,8));
152          when 10 => data_txd <= std_logic_vector(to_unsigned((q_count(3)/2**8) mod 2**8,8));
```

```
153             when 11 => data_txd <= std_logic_vector(to_unsigned(q_count(3) mod 2**8,8));
154             when 12 => data_txd <= std_logic_vector(to_unsigned(q_count(4)/2**16,8));
155             when 13 => data_txd <= std_logic_vector(to_unsigned((q_count(4)/2**8) mod 2**8,8));
156             when 14 => data_txd <= std_logic_vector(to_unsigned(q_count(4) mod 2**8,8));
157             when 15 => data_txd <= std_logic_vector(to_unsigned(q_count(5)/2**16,8));
158             when 16 => data_txd <= std_logic_vector(to_unsigned((q_count(5)/2**8) mod 2**8,8));
159             when 17 => data_txd <= std_logic_vector(to_unsigned(q_count(5) mod 2**8,8));
160             when 18 => data_txd <= std_logic_vector(to_unsigned(q_count(6)/2**16,8));
161             when 19 => data_txd <= std_logic_vector(to_unsigned((q_count(6)/2**8) mod 2**8,8));
162             when 20 => data_txd <= std_logic_vector(to_unsigned(q_count(6) mod 2**8,8));
163             when 21 => data_txd <= std_logic_vector(to_unsigned(q_count(7)/2**16,8));
164             when 22 => data_txd <= std_logic_vector(to_unsigned((q_count(7)/2**8) mod 2**8,8));
165             when 23 => data_txd <= std_logic_vector(to_unsigned(q_count(7) mod 2**8,8));
166             when 24 => data_txd <= std_logic_vector(to_unsigned(q_count(8)/2**16,8));
167             when 25 => data_txd <= std_logic_vector(to_unsigned((q_count(8)/2**8) mod 2**8,8));
168             when 26 => data_txd <= std_logic_vector(to_unsigned(q_count(8) mod 2**8,8));
169             when 27 => data_txd <= std_logic_vector(to_unsigned(q_count(9)/2**16,8));
170             when 28 => data_txd <= std_logic_vector(to_unsigned((q_count(9)/2**8) mod 2**8,8));
171             when 29 => data_txd <= std_logic_vector(to_unsigned(q_count(9) mod 2**8,8));
172             when 30 => data_txd <= std_logic_vector(to_unsigned(q_count(10)/2**16,8));
173             when 31 => data_txd <= std_logic_vector(to_unsigned((q_count(10)/2**8) mod 2**8,8));
174             when 32 => data_txd <= std_logic_vector(to_unsigned(q_count(10) mod 2**8,8));
175             when 33 => data_txd <= std_logic_vector(to_unsigned(q_count(11)/2**16,8));
176             when 34 => data_txd <= std_logic_vector(to_unsigned((q_count(11)/2**8) mod 2**8,8));
177             when 35 => data_txd <= std_logic_vector(to_unsigned(q_count(11) mod 2**8,8));
178             when 36 => data_txd <= std_logic_vector(to_unsigned(q_count(12)/2**16,8));
179             when 37 => data_txd <= std_logic_vector(to_unsigned((q_count(12)/2**8) mod 2**8,8));
180             when 38 => data_txd <= std_logic_vector(to_unsigned(q_count(12) mod 2**8,8));
181             when 39 => data_txd <= std_logic_vector(to_unsigned(q_count(13)/2**16,8));
182             when 40 => data_txd <= std_logic_vector(to_unsigned((q_count(13)/2**8) mod 2**8,8));
183             when 41 => data_txd <= std_logic_vector(to_unsigned(q_count(13) mod 2**8,8));
184             when 42 => data_txd <= std_logic_vector(to_unsigned(q_count(14)/2**16,8));
185             when 43 => data_txd <= std_logic_vector(to_unsigned((q_count(14)/2**8) mod 2**8,8));
186             when 44 => data_txd <= std_logic_vector(to_unsigned(q_count(14) mod 2**8,8));
187             when 45 => data_txd <= std_logic_vector(to_unsigned(q_count(15)/2**16,8));
188             when 46 => data_txd <= std_logic_vector(to_unsigned((q_count(15)/2**8) mod 2**8,8));
189             when 47 => data_txd <= std_logic_vector(to_unsigned(q_count(15) mod 2**8,8));
190             when 48 => data_txd <= std_logic_vector(to_unsigned(q_count(16)/2**16,8));
191             when 49 => data_txd <= std_logic_vector(to_unsigned((q_count(16)/2**8) mod 2**8,8));
192             when 50 => data_txd <= std_logic_vector(to_unsigned(q_count(16) mod 2**8,8));
193             when 51 => data_txd <= std_logic_vector(to_unsigned(q_count(17)/2**16,8));
194             when 52 => data_txd <= std_logic_vector(to_unsigned((q_count(17)/2**8) mod 2**8,8));
195             when 53 => data_txd <= std_logic_vector(to_unsigned(q_count(17) mod 2**8,8));
196             when 54 => data_txd <= std_logic_vector(to_unsigned(q_count(18)/2**16,8));
197             when 55 => data_txd <= std_logic_vector(to_unsigned((q_count(18)/2**8) mod 2**8,8));
198             when 56 => data_txd <= std_logic_vector(to_unsigned(q_count(18) mod 2**8,8));
199             when 57 => data_txd <= std_logic_vector(to_unsigned(q_count(19)/2**16,8));
200             when 58 => data_txd <= std_logic_vector(to_unsigned((q_count(19)/2**8) mod 2**8,8));
201             when 59 => data_txd <= std_logic_vector(to_unsigned(q_count(19) mod 2**8,8));
202             when 60 => data_txd <= std_logic_vector(to_unsigned(q_count(20)/2**16,8));
203             when 61 => data_txd <= std_logic_vector(to_unsigned((q_count(20)/2**8) mod 2**8,8));
204             when 62 => data_txd <= std_logic_vector(to_unsigned(q_count(20) mod 2**8,8));
205             when 63 => data_txd <= std_logic_vector(to_unsigned(q_count(21)/2**16,8));
206             when 64 => data_txd <= std_logic_vector(to_unsigned((q_count(21)/2**8) mod 2**8,8));
207             when 65 => data_txd <= std_logic_vector(to_unsigned(q_count(21) mod 2**8,8));
208             when 66 => data_txd <= std_logic_vector(to_unsigned(q_count(22)/2**16,8));
```

```
209           when 67 => data_txd <= std_logic_vector(to_unsigned((q_count(22)/2**8) mod 2**8,8));
210           when 68 => data_txd <= std_logic_vector(to_unsigned(q_count(22) mod 2**8,8));
211           when 69 => data_txd <= std_logic_vector(to_unsigned(q_count(23)/2**16,8));
212           when 70 => data_txd <= std_logic_vector(to_unsigned((q_count(23)/2**8) mod 2**8,8));
213           when 71 => data_txd <= std_logic_vector(to_unsigned(q_count(23) mod 2**8,8));
214           when 72 => data_txd <= std_logic_vector(to_unsigned(q_count(24)/2**16,8));
215           when 73 => data_txd <= std_logic_vector(to_unsigned((q_count(24)/2**8) mod 2**8,8));
216           when 74 => data_txd <= std_logic_vector(to_unsigned(q_count(24) mod 2**8,8));
217           when 75 => data_txd <= std_logic_vector(to_unsigned(q_count(25)/2**16,8));
218           when 76 => data_txd <= std_logic_vector(to_unsigned((q_count(25)/2**8) mod 2**8,8));
219           when 77 => data_txd <= std_logic_vector(to_unsigned(q_count(25) mod 2**8,8));
220           when 78 => data_txd <= std_logic_vector(to_unsigned(q_count(26)/2**16,8));
221           when 79 => data_txd <= std_logic_vector(to_unsigned((q_count(26)/2**8) mod 2**8,8));
222           when 80 => data_txd <= std_logic_vector(to_unsigned(q_count(26) mod 2**8,8));
223           when 81 => data_txd <= std_logic_vector(to_unsigned(q_count(27)/2**16,8));
224           when 82 => data_txd <= std_logic_vector(to_unsigned((q_count(27)/2**8) mod 2**8,8));
225           when 83 => data_txd <= std_logic_vector(to_unsigned(q_count(27) mod 2**8,8));
226           when 84 => data_txd <= std_logic_vector(to_unsigned(q_count(28)/2**16,8));
227           when 85 => data_txd <= std_logic_vector(to_unsigned((q_count(28)/2**8) mod 2**8,8));
228           when 86 => data_txd <= std_logic_vector(to_unsigned(q_count(28) mod 2**8,8));
229           when others => data_txd <= std_logic_vector(to_unsigned(mode,8));
230           end case;
231           state <= byte_trans;
232         end if;
233       when byte_trans =>
234         if time_count_txd = maxtime then
235           time_count_txd <= 0;
236           bits_txd <= '1'&bits_txd(10 downto 1);
237           if bit_count_txd = 10 then
238             bit_count_txd <= 0;
239             txd_full <= '0';
240             if byte_count_txd = 87 then
241               byte_count_txd <= 0;
242               mode <= (mode+1) mod 4;
243               if out_load_buf(1) = out_load_buf(2) then          -- always true
244                 state <= start;
245               end if;
246             else
247               byte_count_txd <= byte_count_txd+1;
248               state <= byte_load;
249             end if;
250           else
251             bit_count_txd <= bit_count_txd+1;
252           end if;
253         else
254           time_count_txd <= time_count_txd+1;
255         end if;
256       end case;
257     end if;
258   end process automata;
259
260   buf_1 : lcell port map (pulse,out_buf_1);
261   buf_2 : lcell port map (pulse,out_buf_2);
262   mux_f : lcell port map (
263     (not sel_delay_f and out_buf_2) or (sel_delay_f and out_buf_1),out_mux_f);
264   mux_s : lcell port map (
```

```
265     (not sel_delay_s and out_buf_2) or (sel_delay_s and out_buf_1),out_mux_s);
266   buf_f : lcell port map (out_mux_f,out_buf_f);
267   buf_s : lcell port map (out_mux_s,out_buf_s);
268   buf_dly_s : lcell port map (out_buf_s,out_buf_dly_s);
269   load_buf_f1 : lcell port map (out_buf_f,out_load_buf(1));
270   load_buf_f2 : lcell port map (out_buf_f,out_load_buf(2));
271   inv_f_1 : lcell port map (not out_buf_f,out_inv_f(1));
272   inv_s_1 : lcell port map (not out_buf_dly_s,out_inv_s(1));
273
274   inv_chains : for i in 2 to 28 generate
275     inv_f : lcell port map (not out_inv_f(i-1),out_inv_f(i));
276     inv_s : lcell port map (not out_inv_s(i-1),out_inv_s(i));
277   end generate;
278
279   latches : for i in 1 to 28 generate
280     latch_n : if i mod 2 = 1 generate
281       latch : lcell port map (out_inv_f(i) and (not out_inv_s(i) or q(i)),q(i));
282     end generate;
283     latch_p : if i mod 2 = 0 generate
284       latch : lcell port map (not out_inv_f(i) and (out_inv_s(i) or q(i)),q(i));
285     end generate;
286     buf_q : lcell port map (q(i),out_buf_q(i));
287   end generate;
288
289   sel_delay_f <= std_logic_vector(to_unsigned(mode,2))(1);
290   sel_delay_s <= std_logic_vector(to_unsigned(mode,2))(0);
291
292   txd <= bits_txd(0);
293
294   end estructural;
```

A.5 Simulating with Cadence Virtuoso and spectre

This section explains the regular workflow of analog circuit simulation using Cadence Virtuoso, and it describes certain internals of the workflow that may be helpful for later usage with SALVADOR.

The sample circuit shown in figure 6.2 is shall be used to illustrate the workflow. Here, the focus is on the usage of Virtuoso, e.g. how parameters can be defined and how performance figures can be measured.

At first, the circuit schematic is drawn in *Virtuoso Schematic Editor*, which is shown in figure A.3. Values are also assigned to the component parameters here. Parameters can also be defined here by using the expression `pPar("`*parameter-name*`")` in the *Edit Object Properties* window. They can be used in the simulation, or, for hierarchical circuits, in the component to be used in upper hierarchy levels. In the sample circuit, `DriverSize` (aspect ratio of transistors driving the capacitors), `DelaySize` (aspect ratio of transistors emulating the delay in the lower line) and `DeltaC` were used as such parameters. Note that the transistors provided by the technology library offer a parameter named `mismatch` that allows to configure the simulation of mismatch variations that will be described later.

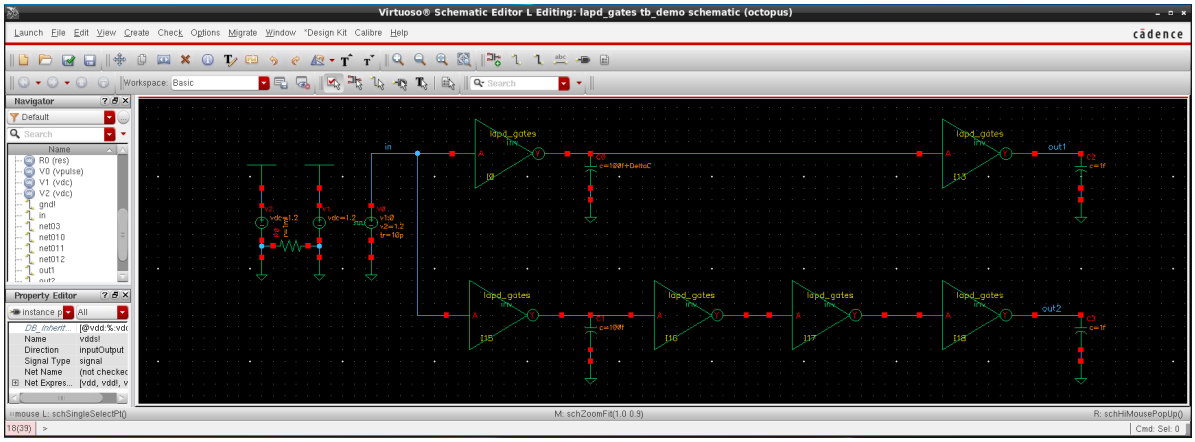


Figure A.3: Virtuoso Schematic Editor.

In the next step, the simulation environment is configured using *Virtuoso Analog Design Environment (ADE)*. This includes selection of the simulation type (e.g. transient simulations, DC simulations, AC simulations), the environment temperature to be simulated, or simulator options such as how many simulations are allowed to be run in parallel.

One can also chose between deterministic or Monte-Carlo simulations, which enable statistical variations of transistor parameters emulating the manufacturing variations based on data from the technology supplier. They can be classified into *process* or *global* variations that affect all circuit elements in the same manner and *mismatch* or *local* variations that affect each circuit element individually and independent from each other [Cad11b].

In addition to statistical data, technology suppliers usually also provide *corners* that allow to simulate corner cases (i.e. worst cases) of global variations. Using corners adds an offset to certain internal transistor parameters, as defined by the technology supplier, instead of applying statistics, in order to save simulation time. However, the notion of a "worst case" is circuit dependent: For example, using the "slow" corner of a technology may be a worst case for digital designs where the critical path needs to be kept small, whereas differential circuits such as operational amplifiers might not be affected as much by slow transistors as by a systematic mismatch of pMOS and nMOS transistors. Therefore, it is recommended to use Monte-Carlo simulations in addition to the default extreme cases unless the meaning of worst case is clearly defined for the circuit under test.

The window that allows configuring corners and Monte-Carlo simulations is shown in figure A.4. The drop-down boxes referred to under "GLOBAL VARIATIONS": "Select All" allows to chose between different corners (the most common corners are "`TT`" for typical, "`FF`" for fast, and "`SS`" for slow corners) or enable Monte-Carlo process simulations. In the latter case, the given technology allows to chose between two different distributions, one for a single fab in Crolles, France named `crolles` or one for multiple fabs named `multifab`. "LOCAL VARIATIONS": "Select All" allows to enable Monte-

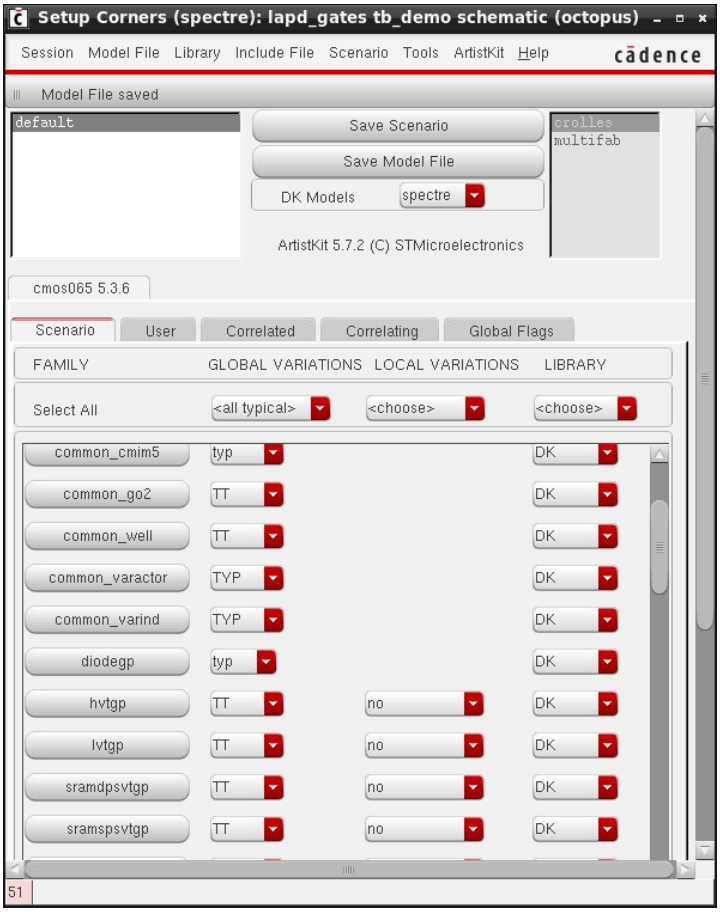


Figure A.4: corner and Monte-Carlo configuration options.

Carlo mismatch simulations. Using "Select all" for configuration is broken down to the individual configuration elements below.

ADE requires to assign concrete values to design parameters that were defined in the previous step. It is possible to assign multiple values, which are separated by a space, to each parameter. This will trigger a repetition of the simulation with all given values, which is called a *sweep*. When multiple values are given for more than one parameter, simulations will be performed for the Cartesian product of the parameters – i.e. for all combinations.

In the used version, ADE does not allow to combine Monte-Carlo simulations and parameter sweeps. Therefore, the following two simulation sets were defined:

- a sweep with `DelaySize`=3, `DriverSize` $\in \{3, 10\}$ and `DeltaC` $\in \{0\,\mathrm{fF}, 20\,\mathrm{fF}\}$, resulting in a total amount of four simulations
- a Monte-Carlo at simulation at `DelaySize`=3, `DriverSize`=3 and `DeltaC`=20 fF with 200 iterations

It is possible to define so-called OCEAN expressions that extract metrics from the circuit under test [Cad12]. In the example case, an expression was written to measure Δt.

The configuration of the design parameters and the outputs for the second simulation instance is shown in figure A.5. The OCEAN expression to measure Δt is shown as the fourth line in the "Outputs" box.

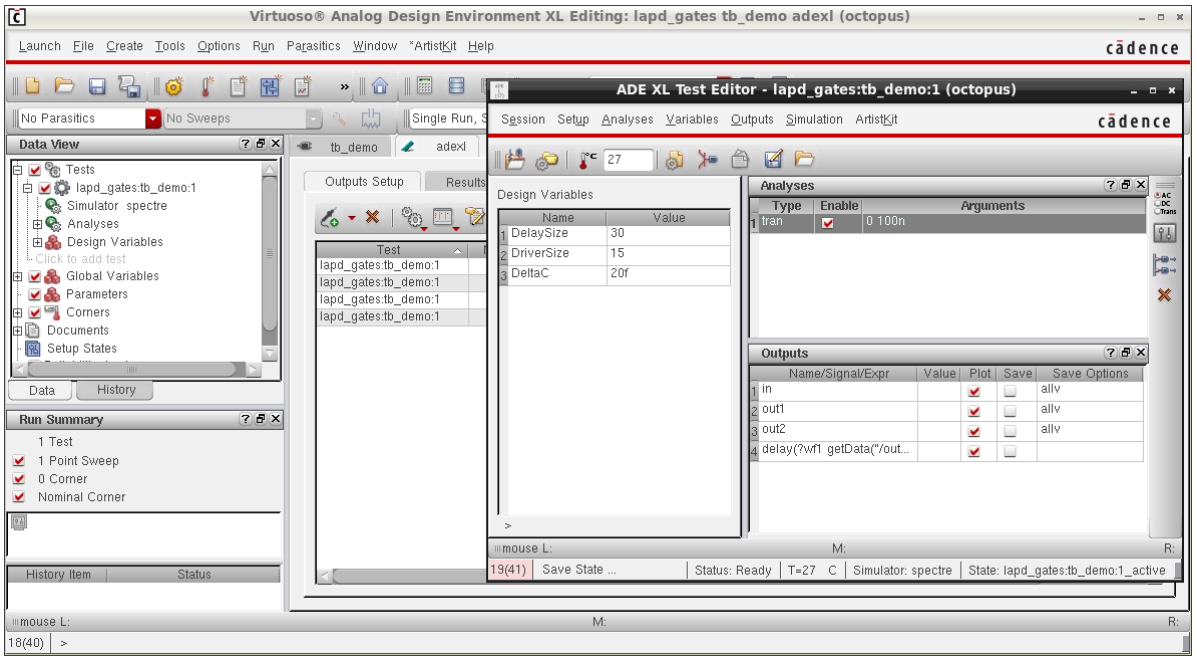


Figure A.5: configuration of simulator options in ADE.

When simulations are started, Virtuoso converts the circuit diagrams, configuration options and parameters into a textual description called netlist, which is understood by the actual simulator. The default simulator is spectre, Virtuoso however also supports other simulators such as hspice. Implementation details may vary when using a different simulator; this thesis will focus on the case of spectre.

In the simplest case, a spectre netlist is one text file with the extension "`.scs`" that instantiates circuit elements intrinsically known by spectre, such as `capacitor` or `resistor`, and defines connections between circuit element ports and *nets*. Nets represent elements that are electrically connected and therefore have the same potential; they are identified by their name and can be defined by simply using that name.

In addition to basic circuit elements, spectre knows complex abstract transistor models such as `bsim4` with more than 100 parameters [Cad11a]. These models are instantiated in models of concrete transistors offered by technology suppliers.

In the shown example case, the netlist consists of multiple files, which use `include` statements such that the contents are merged at run time. The include hierarchy observed for the used technology is shown in figure A.6.

`input.scs` contains the actual netlist including all user-defined hierarchical components. The `mismatch` parameter of the transistors that allows disabling mismatch variation for individual descriptors, which was mentioned earlier in the Schematic Editor description,

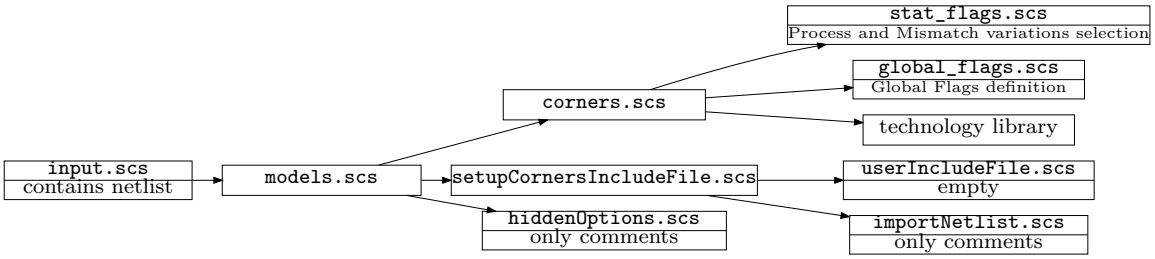


Figure A.6: include hierarchy of Virtuoso-generated spectre netlists.

can be found in the list of parameters of each transistor instance. The simulation type and specific configuration options are defined towards the end of the file. During the work of this thesis, only simulations of type `tran` (i.e. transient simulations) were used; other simulation types include `ac` or `dc`. When Monte-Carlo simulations are performed, this is wrapped in a block named `montecarlo`, which instructs spectre to use pseudo-random number generators for parameter determination according to the statistical parameters defined by the technology. One technical difference between deterministic and Monte-Carlo simulations can be observed with respect to OCEAN expressions: These are included in Monte-Carlo netlists, but they are not included for deterministic netlists. In the OCEAN expressions generated in Virtuoso, it is common to see net names starting with a a slash ("`/`"). They have a special meaning, as they refer to "schematic names" that need additional information, which is usually contained in proprietary binary files in a folder named `amap`. When writing expressions from scratch or when using Virtuoso-generated netlists as templates for simulations independent from Virtuoso, it must be ensured that only spectre-internal names are used [iA12].

Out of the remaining files, the following contain configuration options:

- `corners.scs` contains the selection of the configured corners. With few exceptions, each configuration element from shown in figure A.4 is represented by an `include` statement referencing a file from the technology library; the corner is chosen by selecting the corresponding section in the include file.
- `stat_flags.scs` configures the variations for Monte-Carlo simulations. In the technology used, this is implemented by setting configuration parameters corresponding to entries in figure A.4.
- `global_flags.scs` contains the definition of two parameters in the tab "Global Flags" of figure A.4. They were always left untouched in the course of this thesis.

Experiments with the netlists revealed the possibility to disable variation as part of the technology-dependent configuration in `stat_flags.scs` while continuing to use the spectre block `montecarlo`. The numerical results obtained in this setup were identical to the results from regular transient simulations with the occasional exception that the least significant digit shown in Virtuoso was off by one; however, this was considered negligible. This can be used to have spectre evaluate OCEAN expressions for nominal simulations on the command line.

The netlists are stored as temporary files in the following locations:

1. `~/simulation/`*library*`/`*cell*`/`*view*`/results/data/.tmpADEDir_`*user*`/`
   library`:`*cell*`:1/`*library*`_`*cell*`_`*view*`_spectre/netlist`
2. `~/simulation/`*library*`/`*cell*`/`*view*`/results/data/`*result*`/psf/`
   library`:`*cell*`:1/netlist`
3. `~/simulation/`*library*`/`*cell*`/`*view*`/results/data/`*result*`/`*number*`/`
   library`:`*cell*`:1/netlist`

library refers to the library name where the components are stored, *cell* is the component name, and *view* denotes the so-called view name that refers to the Analog Design Environment in this case, and is "`adexl`" by default. The simulation instance is named *result*; one instance is created when the user triggers the start of simulations, e.g. by clicking on the green "start" button. It has a default prefix of "`Interactive.`" for deterministic simulations or "`MonteCarlo.`" for Monte-Carlo simulations, and a number as a suffix. Simulation instances performing parameter sweeps consist of multiple simulations, which are referred to by *number*. Finally, the Unix user name of the account used to run the simulations is called *user*.

The exact contents of the netlist files will differ slightly in the three mentioned directories. For example, different parameters are inserted at different simulation *numbers*. Also, the first directory path in the list above does not contain the *result* name, so it can be assumed that it will be updated every time a simulation of *cell* in *library* at view *view* is performed by *user*, whereas the paths containing *result* stay unchanged in this case. The third paths contain a shell script named "`runSimulation`" that contains the command to call the simulator; it can be called without Virtuoso in a self-contained way.

When spectre is running, it generates the following outputs. Note that the file names can be configured on the command line or in `input.scs`.

- Notices, warnings and errors, as well as memory usage and computing time are logged to `spectre.log`.
- The result data (in the case of `tran` simulations, measured voltages and currents over time) is stored in a directory named `../psf` relative to the third mentioned netlist directory in a proprietary binary format called "psf". spectre also supports American Standard Code for Information Interchange (ASCII) based or user-defined formats; however, this may affect the simulation performance.
- spectre evaluates OCEAN expressions and stores the expressions to `monteCarlo/mcparams`, and the results to `monteCarlo/mcdata` in a human readable form if the simulations were wrapped into a `montecarlo` statement. Note that without this statement, e.g. when only using `tran`, spectre does not evaluate OCEAN expressions itself; instead, they can be extracted from the result data using the command line tool `ocean`.

After the simulations, ADE shows the results of the OCEAN expressions in the "Results" tab, as shown in figure A.7 for the parameter sweep. The transient response can also be displayed by clicking on the plot icon left of the drop-down menu showing "Replace". This loads *Virtuoso Visualization & Analysis*, which is shown in figure A.8. This tool can be called directly using the command `viva`. It is able to load spectre results in the previously mentioned psf format; this is also true for spectre results generated outside of the Virtuoso environment, i.e. when spectre was called manually on the command line.

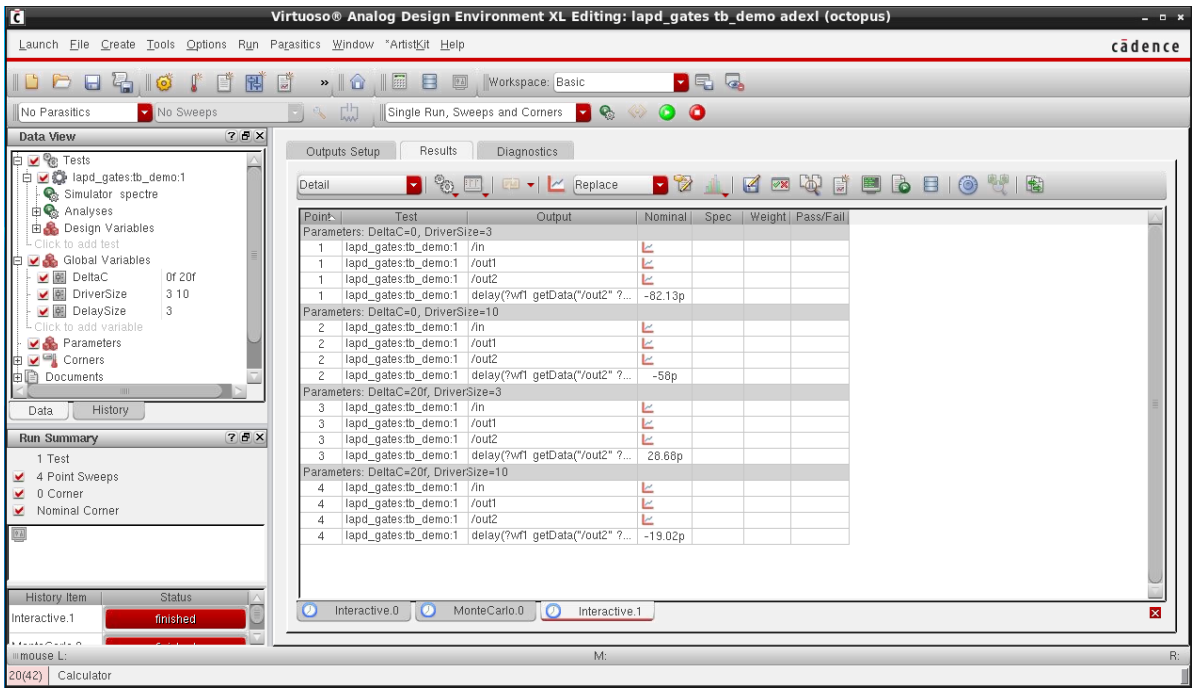


Figure A.7: OCEAN results shown for parameter sweep in ADE.

Figure A.8: transient response shown in Virtuoso Visualization & Analysis.

Bibliography

[AARR03] D. Agrawal, B. Archambeault, J. R. Rao, P. Rohatgi: *The EM side-channel(s)*, in: B. S. Kaliski, ç. K. Koç, C. Paar (eds.), *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 29–45, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[AK96] R. Anderson, M. Kuhn: *Tamper resistance: A cautionary note*, in: *Proceedings of the 2Nd Conference on Proceedings of the Second USENIX Workshop on Electronic Commerce - Volume 2*, WOEC'96, USENIX Association, Berkeley, CA, USA, 1996.

[AK98] R. Anderson, M. Kuhn: *Low cost attacks on tamper resistant devices*, in: B. Christianson, B. Crispo, M. Lomas, M. Roe (eds.), *Security Protocols*, volume 1361 of *Lecture Notes in Computer Science*, pages 125–136, Springer Berlin Heidelberg, 1998.

[Alt15] Altera/Intel: *Altera product catalog – devices: 60 nm device portfolio*, https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/pt/cyclone-iii-product-table.pdf, 2015, accessed on 03.07.2019.

[ARM10] ARM Limited: *AMBA 4 AXI4-stream protocol*, https://static.docs.arm.com/ihi0051/a/IHI0051A_amba4_axi4_stream_v1_0_protocol_spec.pdf, March 2010, accessed on 09.07.2019.

[ARM17] ARM Limited: *AMBA AXI and ACE protocol specification*, https://static.docs.arm.com/ihi0022/fb/IHI0022F_b_amba_axi_protocol_spec.pdf, December 2017, accessed on 09.07.2019.

[BAE+99] K. A. Bowman, B. L. Austin, J. C. Eble, X. Tang, J. D. Meindl: *A physical alpha-power law MOSFET model*, in: *Proceedings of the 1999 International Symposium on Low Power Electronics and Design*, ISLPED '99, pages 218–222, ACM, New York, NY, USA, 1999.

[BBC] BBC: *Rouen hospital turns to pen and paper after cyber-attack*, https://www.bbc.com/news/technology-50503841, accessed on 17.05.2020.

[BCCK07] A. Balankutty, T. C. Chih, C. Y. Chen, P. Kinget: *Mismatch characterization of ring oscillators*, in: *IEEE Custom Integrated Circuits Conference (CICC)*, pages 515–518, 2007.

[BCM+14] M. Bond, O. Choudary, S. J. Murdoch, S. Skorobogatov, R. Anderson: *Chip and skim: Cloning EMV cards with the pre-play attack*, in: *2014 IEEE Symposium on Security and Privacy*, pages 49–64, May 2014.

[BDL97] D. Boneh, R. A. DeMillo, R. J. Lipton: *On the importance of checking cryptographic protocols for faults*, in: W. Fumy (ed.), *Advances in Cryptology — EUROCRYPT '97*, pages 37–51, Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[BECN+06] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, C. Whelan: *The sorcerer's apprentice guide to fault attacks*, Proceedings of the IEEE, volume 94(2), pages 370–382, Feb 2006.

[Ber05] D. J. Bernstein: *Cache-timing attacks on AES*, 2005.

[bH] A. "bunnie" Huang: *Hacking the PIC 18F1320*, https://www.bunniestudios.com/blog/?page_id=40, accessed on 20.12.2020.

[BHKL12] P. Berthomé, K. Heydemann, X. Kauffmann-Tourkestansky, J.-F. Lalande: *High level model of control flow attacks for smart card functional security*, in: *2012 Seventh International Conference on Availability, Reliability and Security*, pages 224–229, Aug 2012.

[BS97] E. Biham, A. Shamir: *Differential fault analysis of secret key cryptosystems*, in: B. S. Kaliski (ed.), *Advances in Cryptology — CRYPTO '97*, pages 513–525, Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[Cad11a] Cadence Design Systems, Inc., *Virtuoso Simulator Circuit Components and Device Models Manual*, product version 10.1.1 edition, jun 2011.

[Cad11b] Cadence Design Systems, Inc., *Virtuoso Spectre Circuit Simulator Reference*, product version 11.1 edition, sep 2011.

[Cad12] Cadence Design Systems, Inc., *OCEAN Reference*, product version 6.1.5 edition, sep 2012.

[Cad14] Cadence Design Systems, Inc.: *Virtuoso Analog Design Environment Family*, https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/tools/custom-ic-analog-rf-design/virtuoso-analog-design-fam-ds.pdf, 2014, accessed on 03.12.2018.

[CBB+03] A. Chandrasekhar, S. Brebels, E. Beyne, W. D. Raedt, B. Nauwelaers, T. V. Bever: *RF evaluation of low-cost leadless packages and development of distributed electrical models*, in: *53rd Electronic Components and Technology Conference, 2003. Proceedings.*, pages 1550–1558, May 2003.

[CDG+14] J.-M. Cioranesco, J.-L. Danger, T. Graba, S. Guilley, Y. Mathieu, D. Naccache, X. T. Ngo: *Cryptographically secure shields*, in: *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 25–31, May 2014.

[DDRT12] A. Dehbaoui, J. Dutertre, B. Robisson, A. Tria: *Electromagnetic transient faults injection on a hardware and a software implementations of AES*, in: *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 7–15, Sep. 2012.

[deg] degate: *Reverse-engineering integrated circuits with degate*, https://degate.org/, accessed on 10.07.2019.

[EMGT15] M. El Massad, S. Garg, M. V. Tripunitara: *Integrated circuit (IC) decamouflaging: Reverse engineering camouflaged ICs within minutes*, in: *22nd Annual Network and Distributed System Security Symposium, NDSS*, 2015.

[Eri08] J. Erickson: *The Art of Exploitation*, No Starch Press, Inc., 2008.

[fid] fido Alliance: *Alliance overview*, https://fidoalliance.org/overview/, accessed on 17.05.2020.

[GG14] R. Göttfert, B. Gammel: *Busanordnung und Verfahren zum Senden von Daten über einen Bus*, July 2014, Patent DE 10 2013 100 572 A1.

[GG20] A. Girardi, H. Graeb: *Modeling and optimization of a microprobe detector for area and yield improvement*, in: *2020 33rd Symposium on Integrated Circuits and Systems Design (SBCCI)*, pages 1–6, Aug 2020.

[GGBa] GGB Industries, Inc.: *Picoprobe model 18c & picoprobe model 19c*, http://www.ggb.com/PdfIndex_files/mod18c.pdf, accessed on 07.11.2016.

[GGBb] GGB Industries, Inc.: *Picoprobe models 28 & 29*, http://www.ggb.com/PdfIndex_files/mod28.pdf, accessed on 07.11.2016.

[GMO01] K. Gandolfi, C. Mourtel, F. Olivier: *Electromagnetic analysis: Concrete results*, in: Ç. K. Koç, D. Naccache, C. Paar (eds.), *Cryptographic Hardware and Embedded Systems — CHES 2001*, pages 251–261, Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[Hen10] S. Henzler: *Time-to-Digital Converters*, volume 29, Springer Science+Business Media B.V., springer series in advanced microelectronics edition, 2010.

[HHM+14] N. Homma, Y.-i. Hayashi, N. Miura, D. Fujimoto, D. Tanaka, M. Nagata, T. Aoki: *EM Attack Is Non-invasive? - Design Methodology and Validity Verification of EM Attack Sensor*, pages 1–16, Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[HHM+15] N. Homma, Y.-i. Hayashi, N. Miura, D. Fujimoto, M. Nagata, T. Aoki: *Design methodology and validity verification for a reactive countermeasure against EM attacks*, Journal of Cryptology, pages 1–19, 2015.

[HMH+12] J. Heyszl, S. Mangard, B. Heinz, F. Stumpf, G. Sigl: *Localized Electromagnetic Analysis of Cryptographic Implementations*, pages 231–244, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[HSH+09] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, E. W. Felten: *Lest we remember: Cold-boot attacks on encryption keys*, Commun. ACM, volume 52(5), pages 91–98, May 2009.

[Hun07] J. D. Hunter: *Matplotlib: A 2D graphics environment*, Computing in Science Engineering, volume 9(3), pages 90–95, May 2007.

[HWPG18] A. Herrmann, M. Weiner, M. Pehl, H. Graeb: *Bringing analog design tools to security: Modeling and optimization of a low area probing detector*, in: *2018 15th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, pages 1–4, July 2018.

[HYM+08] T. Hashimoto, H. Yamazaki, A. Muramatsu, T. Sato, A. Inoue: *Time-to-digital converter with vernier delay mismatch compensation for high resolution on-die clock jitter measurement*, in: *2008 IEEE Symposium on VLSI Circuits*, pages 166–167, June 2008.

[iA12] "imagesensor123", Andrew Beckett: *ocean expression in monte carlo analysis*, Cadence Custom IC Design Forums, https://community.cadence.com/cadence_technology_forums/f/custom-ic-skill/21674/ocean-expression-in-monte-carlo-analysis, feb 2012, accessed on 25.03.2019.

[Inf12] Infineon Technologies AG: *Integrity guard - the newest generation of digital security technology*, http://www.infineon.com/dgdl/Infineon-Integrity_Guard_The_newest_generation_of_digital_security_technology-WP-v04_12-EN.pdf?fileId=5546d46255dd933d0155e31c46fa03fb, September 2012, accessed on 07.11.2016.

[ISW03] Y. Ishai, A. Sahai, D. Wagner: *Private Circuits: Securing Hardware against Probing Attacks*, pages 463–481, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[JL10] M. Janke, P. Laackmann: *The technical progress of hardware security*, The Silicon Trust, https://silicontrust.org/2010/09/06/the-technical-progress-of-hardware-security/, September 2010, accessed on 29.06.2019.

[KJJ99] P. Kocher, J. Jaffe, B. Jun: *Differential power analysis*, in: M. Wiener (ed.), *Advances in Cryptology — CRYPTO' 99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397, Springer Berlin Heidelberg, 1999.

[KK99] O. Kömmerling, M. G. Kuhn: *Design principles for tamper-resistant smartcard processors*, in: *Proceedings of the USENIX Workshop on*

Smartcard Technology on USENIX Workshop on Smartcard Technology, WOST'99, USENIX Association, Berkeley, CA, USA, 1999.

[KNSS13] J. Krämer, D. Nedospasov, A. Schlösser, J.-P. Seifert: *Differential photonic emission analysis*, in: E. Prouff (ed.), *Constructive Side-Channel Analysis and Secure Design*, volume 7864 of *Lecture Notes in Computer Science*, pages 1–16, Springer Berlin Heidelberg, 2013.

[Koc96] P. C. Kocher: *Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems*, in: N. Koblitz (ed.), *Advances in Cryptology — CRYPTO '96*, pages 104–113, Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[LT04] P. Laackmann, H. Taddiken: *Apparatus for protecting an integrated circuit formed in a substrate and method for protecting the circuit against reverse engineering*, September 2004, U.S. Patent 6,798,234 B2.

[MA15] Michael Weiner, Andrew Beckett: *parametrized netlist generation at scale*, Cadence Custom IC Design Forums, https://community.cadence.com/cadence_technology_forums/f/custom-ic-design/34695/parametrized-netlist-generation-at-scale, oct 2015, accessed on 25.03.2019.

[MDAB10] S. J. Murdoch, S. Drimer, R. Anderson, M. Bond: *Chip and PIN is broken*, in: *2010 IEEE Symposium on Security and Privacy*, pages 433–446, May 2010.

[MDH+13] N. Moro, A. Dehbaoui, K. Heydemann, B. Robisson, E. Encrenaz: *Electromagnetic fault injection: Towards a fault model on a 32-bit microcontroller*, in: *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 77–88, Aug 2013.

[MHER14] N. Moro, K. Heydemann, E. Encrenaz, B. Robisson: *Formal verification of a software countermeasure against instruction skip attacks*, Journal of Cryptographic Engineering, volume 4(3), pages 145–156, Sep 2014.

[MN12] P. Maier, K. Nohl: *Low-cost chip microprobing*, 29th Chaos Communication Congress (29C3), 12 2012.

[MOP08] S. Mangard, E. Oswald, T. Popp: *Power Analysis Attacks*, Springer US, 2008.

[MS02] K. Mitnick, W. L. Simon: *The Art of Deception*, John Wiley & Sons, 2002.

[MS13] S. Manich, M. Strasser: *A highly time sensitive XOR gate for probe attempt detectors*, IEEE Transactions on Circuits and Systems II: Express Briefs, volume 60(11), pages 786–790, Nov 2013.

[Mun] MunEDA GmbH, *WiCkeD Manual*, wicked 6.7 edition.

[MWS12] S. Manich, M. S. Wamser, G. Sigl: *Detection of probing attempts in secure ICs*, in: *Hardware-Oriented Security and Trust (HOST)*, pages 134–139, 2012.

[NHEW10] D. M. H. Neil H. E. Weste: *CMOS VLSI Design: A Circuits and Systems Perspective (4th Edition)*, Addison Wesley, 4th edition, 2010.

[Noh11] K. Nohl: *Reviving smart card analysis*, Chaos Communication Camp 2011, 08 2011, accessed on 10.07.2019.

[Ope10] OpenCores: *Wishbone B4 – WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores*, https://cdn.opencores.org/downloads/wbspec_b4.pdf, 2010, accessed on 09.07.2019.

[Par94] B. Parhami: *Voting algorithms*, IEEE Transactions on Reliability, volume 43(4), pages 617–629, Dec 1994.

[PD08] S. Pasricha, N. Dutt: *On-Chip Communication Architectures: System on Chip Interconnect*, Systems on Silicon, Morgan Kaufmann, 2008.

[PKZM07] T. Popp, M. Kirschbaum, T. Zefferer, S. Mangard: *Evaluation of the masked logic style MDPL on a prototype chip*, in: P. Paillier, I. Verbauwhede (eds.), *Cryptographic Hardware and Embedded Systems - CHES 2007*, pages 81–94, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[Pyt] *Python: string – common string operations – format specification mini-language*, https://docs.python.org/3/library/string.html#formatspec, accessed on 02.04.2019.

[QS01] J.-J. Quisquater, D. Samyde: *Electromagnetic analysis (EMA): Measures and counter-measures for smart cards*, in: I. Attali, T. Jensen (eds.), *Smart Card Programming and Security*, pages 200–210, Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[RK93] T. E. Rahkonen, J. T. Kostamovaara: *The use of stabilized CMOS delay lines for the digitization of short time intervals*, IEEE Journal of Solid-State Circuits, volume 28(8), pages 887–894, Aug 1993.

[RSSK13] J. Rajendran, M. Sam, O. Sinanoglu, R. Karri: *Security analysis of integrated circuit camouflaging*, in: *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 709–720, ACM, New York, NY, USA, 2013.

[SAFT16] Q. Shi, N. Asadizanjani, D. Forte, M. M. Tehranipoor: *A layout-driven framework to assess vulnerability of ICs to microprobing attacks*, in: *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 155–160, May 2016.

[SDK+13] D. Strobel, B. Driessen, T. Kasper, G. Leander, D. Oswald, F. Schellenberg, C. Paar: *Fuming acid and cryptanalysis: Handy tools for overcoming*

a digital locking and access control system, in: R. Canetti, J. Garay (eds.), *Advances in Cryptology – CRYPTO 2013*, volume 8042 of *Lecture Notes in Computer Science*, pages 147–164, Springer Berlin Heidelberg, 2013.

[Shō88] M. Shōji: *CMOS digital circuit technology*, Prentice Hall, 1988.

[Sko05] S. P. Skorobogatov: *Semi-invasive attacks – A new approach to hardware security analysis*, Technical Report UCAM-CL-TR-630, University of Cambridge, Computer Laboratory, April 2005.

[SN90] T. Sakurai, A. R. Newton: *Alpha-power law MOSFET model and its applications to CMOS inverter delay and other formulas*, IEEE Journal of Solid-State Circuits, volume 25(2), pages 584–594, Apr 1990.

[SNK+12] A. Schlösser, D. Nedospasov, J. Krämer, S. Orlic, J.-P. Seifert: *Simple photonic emission analysis of AES*, in: E. Prouff, P. Schaumont (eds.), *Cryptographic Hardware and Embedded Systems – CHES 2012*, pages 41–57, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[SR07] A. Singhee, R. A. Rutenbar: *Statistical blockade: A novel method for very fast monte carlo simulation of rare circuit events, and its application*, in: *2007 Design, Automation Test in Europe Conference Exhibition*, pages 1–6, April 2007.

[SR09] A. Singhee, R. A. Rutenbar: *Novel Algorithms for Fast Statistical Analysis of Scaled Circuits*, Springer Netherlands, 2009.

[Tar08a] C. Tarnovsky: *Introducing momentary faults within secure smartcards*, DEF CON 16, aug 2008.

[Tar08b] C. Tarnovsky: *Security failures in secure devices*, Black Hat DC, February 2008.

[Tar10a] C. Tarnovsky: *Deconstructing a 'secure' processor*, Black Hat DC, February 2010.

[Tar10b] C. Tarnovsky: *Semiconductor security awareness, today & yesterday*, Black Hat Las Vegas, jul 2010.

[Tar13] C. Tarnovsky: *Change of guard at Infineon*, Flylogic's Analytical Blog, nov 2013, accessed on 11.04.2019.

[Tar19a] C. Tarnovsky: *Exposing the deep-secure elements of smartcards*, hardwear.io USA, jun 2019.

[Tar19b] C. Tarnovsky: *Sophisticated million dollar hack to discover weaknesses in a series of smartcards affecting millions*, hardwear.io USA, jun 2019.

[Tec] TechInsights: *Techinsights library*, https://www.chipworks.com/access-reverse-engineering/overview/library-for-patent-investigations/techinsights-library, accessed on 10.07.2019.

[TMS+13] N. Theißing, D. Merli, M. Smola, F. Stumpf, G. Sigl: *Comprehensive analysis of software countermeasures against fault attacks*, in: *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '13, pages 404–409, EDA Consortium, San Jose, CA, USA, 2013.

[WE93] N. H. E. Weste, K. Eshraghian: *Principles of CMOS VLSI Design*, chapter Circuit Characterization and Performance Estimation, pages 175–259, Addison-Wesley Publishing Company, 1993.

[WHH+15] M. Wan, Z. He, S. Han, K. Dai, X. Zou: *An invasive-attack-resistant PUF based on switched-capacitor circuit*, IEEE Transactions on Circuits and Systems I: Regular Papers, volume 62(8), pages 2024–2034, Aug 2015.

[Whi12] V. White: *BBC Panorama: Murdoch's TV pirates*, https://www.bbc.co.uk/programmes/b01dlvbm, March 2012, available on YouTube, https://www.youtube.com/playlist?list=PLqpZC2iaEqGs8HGudtGjX2N6iU4Qo4Z66, accessed on 09.07.2019.

[Wil27] E. B. Wilson: *Probable inference, the law of succession, and statistical inference*, Journal of the American Statistical Association, volume 22(158), pages 209–212, 1927.

[Wis03] *The WISHBONE Service Center*, http://www.pldworld.com/_hdl/2/_ip/-silicore.net/wishbone.htm, Sep 2003, accessed on 05.12.2018.

[WL07] Z. Wang, R. B. Lee: *New cache designs for thwarting software cache-based side channel attacks*, SIGARCH Comput. Archit. News, volume 35(2), pages 494–505, June 2007.

[WMB16] M. Weiner, S. Manich Bou: *The SALVADOR simulation framework*, TRUDEVICE Workshop, nov 2016.

[WMRMS18] M. Weiner, S. Manich, R. Rodríguez-Montañés, G. Sigl: *The low area probing detector as a countermeasure against invasive attacks*, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, volume 26(2), pages 392–403, Feb 2018.

[WMS14] M. Weiner, S. Manich, G. Sigl: *A low area probing detector for power efficient security ICs*, in: *Radio Frequency Identification: Security and Privacy Issues*, volume 8651, Oxford, UK, July 2014.

[WMT+13] M. Weiner, M. Massar, E. Tews, D. Giese, W. Wieser: *Security analysis of a widely deployed locking system*, in: *ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 929–940, ACM, New York, NY, USA, 2013.

[WSN+19] H. Wang, Q. Shi, A. Nahiyan, D. Forte, M. M. Tehranipoor: *A physical design flow against front-side probing attacks by internal shielding*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2019.

[WWL+19] M. Weiner, W. Wieser, E. Lupon, G. Sigl, S. Manich: *A calibratable detector for invasive attacks*, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, volume 27(5), pages 1067–1079, May 2019.

[Zon16] A. Zonenberg: *Resetting lock bits with UV – or, semi-invasive attacks for dummies*, S4x16 Conference, jan 2016.

[ZZM99] Y. Zhang, Y. Zhang, R. B. Marcus: *Thermally actuated microprobes for a new wafer probe card*, Journal of Microelectromechanical Systems, volume 8(1), pages 43–49, March 1999.

List of Figures

List of Tables

List of Acronyms

ADE Analog Design Environment

AES Advanced Encryption Standard

ALU Arithmetic Logic Unit

AMBA Advanced Microcontroller Bus Architecture

ASCII American Standard Code for Information Interchange

ASIC Application Specific Integrated Circuit

CaLIAD Calibratable Lightweight Invasive Attack Detector

CMOS Complementary Metal Oxide Semiconductor

CPU Central Processing Unit

CSS Cryptographically Secure Shields

CSV Comma Separated Values

DPA Differential Power Analysis

DRAM Dynamic Random Access Memory

EPROM Erasable Programmable Read Only Memory

FIB Focused Ion Beam

FPGA Field Programmable Gate Array

GE Gate Equivalent

HTS-XOR Highly Time Sensitive Exclusive-OR

IC Integrated Circuit

IT Information Technology

JSON JavaScript Object Notation

LAB Logic Array Block

LAPD Low Area Probing Detector

LE Logic Element

LUT Lookup Table

LZMA Lempel-Ziv-Markov chain algorithm

MMU Memory Management Unit

nMOS n-channel Metal Oxide Semiconductor

NoC Network-on-Chip

OCEAN Open Command Environment for Analysis

PAD Probe Attempt Detector

PF1 position of first 1

PIN Personal Identification Number

pMOS p-channel Metal Oxide Semiconductor

PRNG Pseudo-Random Number Generator

RAM Random Access Memory

RO-PUF Ring-Oscillator based Physical Unclonable Function

ROM Read Only Memory

SALVADOR Simulation Automation Library for Verification and Analysis of Design Operating Regions

SAT Boolean satisfiability problem

SIM Subscriber Identity Module

SoC System-on-Chip

SPA Simple Power Analysis

ssh Secure Shell

TDC Time-to-Digital Converter

TSMC Taiwan Semiconductor Manufacturing Company

UART Univeral Asynchronous Receiver/Transmitter

VDL Vernier Delay Line

VHDL VHSIC Hardware Description Language

VHSIC Very High Speed Integrated Circuit

List of Symbols