Federal Office
for Information Security

# Security Evaluation of Hardware Design Synthesis

# Document history

| *Version* | *Date* | *Editor* | *Description* |
|---|---|---|---|
| Content | 24.04.2024 | | Final version |
| | | | |

*Table 1: Name*

Federal Office for Information Security
P.O. Box 20 03 63
53133 Bonn
E-Mail: chipsecurity@bsi.bund.de
Internet: https://www.bsi.bund.de

# Acknowledgement

This report was authored by the Fraunhofer Institute for Applied and Integrated Security (AISEC) on behalf of the German Federal Office for Information Security (BSI).

# Table of Contents

# List of Abbreviations

AES Advanced encryption standard

AIG And-inverter graph

ANSSI Agence Nationale de la Sécurité des Systémes d'Information

AOI AND-OR-Inverter

ASIC Application specific integrated circuit

BSI Bundesamt für Sicherheit in der Informationstechnik

CC Common criteria for information technology security evaluation

CMS Consolidated masking scheme

COTS Commercial off-the-shelf

CPU Central processing unit

DAG Directed acyclic graph

DFA Differential fault analysis

DOM Domain-oriented masking

EDA Electronic design automation

EM Electromagnetic

FI Fault injection

FPGA Field programmable gate array

FSM Finite state machine

GDSII Graphic design system II stream format

GE Gate equivalence

HDL Hardware description language

HPC Hardware private circuit

IC Integrated circuit

IP Intellectual property

LFSR Linear feedback shift register

LUT Look-up table

NI Non-interference

OAI OR-AND-Inverter

OEM Original equipment manufacturer

PDK Process design kit

PINI Probe-isolating non-interference

PRNG Pseudo-random number generator

ROBDD Reduced-order binary decision diagram

RTL Register transfer level

SCA Side-channel attack

SCFI State-machine control-flow hardening against fault attacks

SIFA Statistical ineffective fault analysis

SNI Strong non-interference

TI Threshold implementations

TRNG True-random number generator

TVLA Test vector leakage assessment

VHDL Very-high-speed integrated circuit hardware description language

# 1 Introduction

In the 21st century, smart electronics are prevalent in everyday life. Therefore, it is important that consumers can trust their electronic devices. In this context, the term *trust* covers many aspects of an electronic device, among others its correct functionality, safety in regard to its user and environment, and security. Within this report, we focus on the security of electronic devices, more specifically, their resilience against hardware attacks.

For embedded devices, hardware attacks must be taken into consideration, as these devices are often accessible to adversaries. Hardware attacks pose an enormous threat for electronic devices, as they affect the underlying hardware, regardless of any application software. A recent BSI study demonstrated this with an analysis of control flow manipulation and read-out protection bypass attacks on COTS microcontrollers [1].

In order to allow consumers and OEMs to judge the extent to which embedded devices withstand hardware attacks, requirements such as the common criteria for information technology security evaluation (CC) were created by various standardization organizations. During a CC evaluation, the evaluated product is analyzed by checking if its security properties and functions correspond to the protection profile that is necessary for the respective product class. Such an evaluation should be conducted as thoroughly as possible, while also taking into account the economic interests of vendors to get innovative products to market as fast as possible. This conflict of interest raises the following question: *How can the security properties and functions of a chip be analyzed thoroughly in as little time as possible?* and consequently: *What does accurately represent a chip's security properties and functions and can thus be used for verification?*

Digital chip design usually starts with high-level concepts and block designs. Afterwards, the chip's functionality is specified at register transfer level (RTL) in high-level hardware description languages (HDLs) such as (System) Verilog and VHDL. During RTL *synthesis*, this description is compiled to a technology specific netlist, which contains only standard cells of the respective process design kit (PDK). At this point, design tools can begin to estimate a design's area requirements, power consumption, and critical paths. Designers may define certain constraints in the synthesis procedure, to ensure that requirements on these performance criteria are met, or different optimization steps are applied to different modules. This concludes the *frontend* design of a chip. During the *backend* design flow, the cells in the synthesized netlist are placed and routed. Macros such as memories and analog modules can be integrated into the design. At this point, a transistor-level description of the design exists and can be verified for correct behavior.

Finally, all design descriptions are combined into a GDSII file, which is used by the foundry to produce the *integrated circuit* (IC). For more details on the inner workings of the frontend and backend design flow, the interested reader is referred to [72], where the open-source OpenLANE[1] tape-out flow is introduced.

For product vendors, it would be desirable to certify the resilience against hardware attacks of their ICs as early in the design flow as possible, e.g. by demonstrating the integration of countermeasures in the HDL description of a module. The certified module could then be re-used for a complete line of products and multiple generations. From the vendor's perspective, even evaluating the countermeasure once on one manufactured IC and then re-using the HDL description for other products would be more desirable than evaluating ICs for every product. For certification agencies, this raises the following question: *Does RTL synthesis and the backend design flow have a negative impact on countermeasures against physical adversaries?*

To this end, we provide two case studies, one on the impact of RTL synthesis on fault injection (FI) countermeasures and one on the impact of RTL synthesis on side-channel attack (SCA) countermeasures. Our investigations are based on the following methodology:

1. Integrate countermeasures into HDL designs or select designs, which already incorporate countermeasures.

[1] https://github.com/The-OpenROAD-Project/OpenLane

2. Synthesize the design to a netlist with different commercial and open-source tools and FreePDK45[2], a free PDK for research purposes.
3. Investigate the resulting netlist for the countermeasures and their effectiveness.

Our FI countermeasures are based on redundancy and fault-resilient encodings. For SCA countermeasures, we focus on the popular masking countermeasure [19] [37]. The necessary background on these techniques can be found in Chapter 2. The case studies are described in Chapter 3 and Chapter 4, respectively. Each case study starts with background on state-of-the-art literature. Subsequently, our practical evaluations and their results are laid out. Remarks on open questions (Chapter 5) and a summary of our results and their implications (Chapter 6) conclude this report.

[2] https://github.com/mflowgen/freepdk-45nm

# 2 Preliminaries

Within this background chapter, we give a short overview on hardware attacks (Section 2.1), in particular fault injection (FI) and side-channel attacks (SCA). A rough understanding of these attacks is sufficient to understand the countermeasures, which are introduced subsequently. Key concepts used in hardware design synthesis and common EDA tool vendors are introduced in Section 2.2.

## 2.1 Hardware Attacks and Countermeasures

Hardware attacks target the underlying hardware of a chip. Therefore, hardware attacks might work independent from any software running on the target, or exploit vulnerabilities caused by physical effects that occur when running the software on a chip. Such attacks are often summarized with the term side-channel attack (SCA), as *physical* side-channels are exploited. In the following, however, we consider only two types of hardware attacks: fault attacks, where an adversary inserts an active fault to compromise a target and passive side-channel attacks, where an adversary attempts to extract secret information from the power consumption and electromagnetic emanation of a device. As such, we use the term side-channel analysis only for passive attacks.

### 2.1.1 Fault Attacks

Fault attacks are active, physical attacks mostly used to extract sensitive information or bypass security features of a device by intentionally injecting faults in a design. There are several methods available to inject such faults, among others, voltage and clock glitches, electromagnetic pulses and focused laser beams. A fault induced by these methods causes several effects at the physical level such as timing violations or changes in transient voltage or current. An adversary can exploit these effects to alter the control flow of a device or access sensitive information. It is also possible to combine the principle of fault attacks with the concept of differential cryptanalysis. The combination of these two ideas is known under differential fault analysis (DFA) and is a very effective tool to attack cryptographic implementations. DFA can reduce the key space significantly by analyzing the relationship between faulty and non-faulty ciphertexts for the same input. The practicality and effectiveness of DFA and fault attacks in general depend on the underlying fault model.

**Fault Models** A fault model characterizes a fault attack and specifies the location and duration of a fault as well as the effect of the fault. The most common fault models are bit-flips and stuck-at effects. A bit-flip temporarily changes the value of one bit to its opposite value while stuck-at faults change the value of a bit to a either one or zero permanently. Additionally, a fault model includes the spatial and temporal precision of an adversary. In [59], the authors present a verification tool which is able to model the effect of injecting single or multiple transient faults as well as stuck-at faults into single gates of a netlist. Thus, their tool can depict various fault models.

**Active Security** From fault models, formal security notions such as active security were derived [28][29]. In short, a design is $k^{th}$-order active secure, if there does not exist a combination of $k$ or less bit-flips that corrupt the design. A design is corrupted, if faults produce an incorrect output, which is not detected. In recent literature, various verification tools for this security notion were proposed [68][69]. Depending on the configuration, the tool in [59] can also verify this notion.

### 2.1.2 Fault Injection Countermeasures

Fault attacks are usually not considered when developing a cipher or cryptographic algorithm. Therefore, hardware and software designers have to deploy adequate countermeasures against this kind of attack when implementing the cipher or cryptographic algorithm. Most fault injection countermeasures are detection-based countermeasures [41] [42]. There are also infection-based countermeasures [35] [64] which diffuse the effect of a fault and render the faulty ciphertext unexploitable. Such countermeasures are needed to thwart

the threat of SIFA, a class of attacks where biased faults (i.e. either setting or resetting a bit are more likely) are injected and the adversary uses only the information whether the fault had an effect or not to collect secret information. SIFA can also be prevented with correction-based countermeasures, however, they are typically found in safety use-cases, where reliability is the most important issue. In the following, we focus on detection-based countermeasures. Detection-based countermeasures aim to detect data modification caused by injected faults. On that account, detection-based countermeasures rely on redundancy and deploy methods from coding and information theory. They can be categorized based on the form of redundancy:

- Hardware redundancy relies on redundant instantiation of a module in hardware. By comparing the output of the instantiated modules, faults can be detected. More specifically, if $k$ redundant modules are instantiated, up to $k - 1$ faults can be detected.
- Information redundancy relies on error detecting codes. To be able to detect $k - 1$ faults, the minimum Hamming distance for the set of codewords must be $k$.
- Timing redundancy relies on executing a computation multiple times in a sequential manner and comparing the results. In the case of timing redundancy an attacker would need to inject the same fault(s) at every execution of the computation. Otherwise, there would be a mismatch between the outputs and the faults would be detected. A computation must be repeated $k$ times, to detect all possible $k - 1$ faults.

### 2.1.3 Side-Channel Attacks

The fact that cryptographic implementations can be broken from a chip's power consumption [48] and electromagnetic emanation [66] has been known for a long time. Typically, a side-channel attack consists of the trace acquisition, trace processing, leakage assessment, and leakage exploitation phases. During trace acquisition, power and EM traces are collected and during trace processing, signal processing methods such as alignment, filtering, averaging, etc. are used, s.t. optimal extraction of secret data from the traces is possible. During leakage assessment, specific and unspecific methods are used to determine whether secret information can be obtained from the traces. To exploit leakage, the adversary must find a way to obtain secret information of interest from the leaked information, e.g. Hamming weights or distances of certain values.

### 2.1.4 Masking

Masking is a popular countermeasure against side-channel attacks that is based on secret-sharing and attributed to the seminal works by Chari et al. [19] and Goubin and Patarin [37]. A $d^{th}$-order masking scheme splits the secret information into at least $d + 1$ shares. Assume we want to mask a single secret bit $s \in \mathbb{F}_2$. For boolean masking, this sharing is created as follows:

$d$ shares are drawn from a uniform distribution, so $s_i \leftarrow \mathcal{R} \; \forall \; i \in [0, d - 1]$, then $s_d = s_0 + s_1 + \cdots + s_{d-1}$, where addition is in $\mathbb{F}_2$, i.e. a XOR operation.

The rationale behind masking is that, if the shares $s_i$ instead of $s$ are used for computation, the direct correlation of $s$ with the target's power consumption and electromagnetic emanation is removed. In [65], a formal proof that masking increases the complexity of a side-channel attack exponentially with the number of shares is given. For linear operations in $\mathbb{F}_2$, an operation can be independently repeated for each $s_i$. For non-linear operations, information of all input shares is required to compute the output shares correctly. Implementing this in a way that side-channel attacks are still as hard as recovering $d + 1$ shares is a non-trivial task that early applications of masking to ciphers failed to solve adequately [2] [19] [22] [63].

**Probing Models** In [40], Ishai et al. undertook the first steps towards formalizing the security of masking by introducing the $d$-probing model. This assumes that an adversary has access to any $d$ probes. In this case, a probe corresponds to the value a wire in a circuit is carrying at time $t$. The circuit is called $d^{th}$-order *probing secure* if the adversary cannot recover secret information from their $d$ probes. The authors also introduced

an AND gate that guarantees the security within this model by inserting fresh randomness into the computation. Subsequent works discovered, that this simple probing model does not match the leakage of actual circuits. The authors of [51] discovered that, in certain masked designs, information is leaked due to the fact that inputs arrive asynchronously at logic gates, which leads to temporary changes in the output. This behavior is referred to as *glitch*. Additional effects that are not considered in this simple probing model include the *transition* when memory elements such as registers are overwritten [4] [23] and the *coupling* of adjacent wires [20]. Most recently it was discovered that even in the absence of glitches at the output of a gate, delay effects at an input wire might cause leakage [49]. The *d-robust probing model* [33] can extend an adversary's probes to cover these effects. A *glitch-extended probe*, for instance, includes the probed wire and all wires contributing to it up to the driving synchronization elements, e.g. registers. Similarly, the *transition-extended probe* includes the current probed value and its predecessor value, the *coupling-extended probe* includes values of the adjacent wires, and a *delay-extended probe* includes all values from all possible delay effects. If an adversary with a set containing any $d$ extended probes does only obtain information about $d$ shares, the circuit is called $d$-robust probing secure. Further, to account for the physical reality an adversary faces, that is that values are probed with a certain level of noise, *random probing models* have been proposed and evaluated [8] [15] [30].

**Threshold Implementations (TI)** In [62], a first proposal for a generic blueprint to construct masked gadgets that comply with the robust probing model was made. The authors come to the conclusion that any implementation that fulfills three requirements are correctly masked. The first requirement is the *incompleteness*, stating that every function needs to be independent of at least $d$ shares of each input variable. *Correctness*, the second property, requires that the recombination of all shares should correspond to the desired output. The third property is the *uniformity* and corresponds to the fact that the distribution of output shares should be balanced for all possible distributions of the input sharings.

**Domain-oriented Masking (DOM)** The authors of [38] proposed a more strict blueprint to implement masking. For a $d$-th order masking scheme, not at least, but exactly $d + 1$ shares are used. Each share is referred to as domain. If domains are crossed, e.g. for non-linear AND operations, intermediates are blinded with fresh randomness and synchronized by registers. This makes DOM the more strict blueprint that inherently requires fresh randomness, whereas threshold implementations allow more freedom in describing masked permutations.

## 2.2 Hardware Design

A design flow describes a set of procedures which are necessary to progress from a specification to the final physical chip. A generalized design flow is depicted in Figure 2.1.

The start of a design flow is typically a set of requirements or a specification. Based on these requirements designers will develop HDL code to model the hardware at RTL level. A synthesis tool transforms this HDL description into a gate-level netlist. The synthesis step will be explained in more detail in Section 2.2.1. The result of the synthesis step is a structural description of the design in form of a gate-level netlist. To actually manufacture the chip it is required to obtain a physical representation of the structural description. This step is called physical synthesis or layout generation and includes tasks such as floorplanning, placement of cells, routing, parasitic extraction, clock tree synthesis, as well as electromagnetic, timing and power analysis. For an extensive introduction and more details about the hardware design flow, the reader is referred to [75].

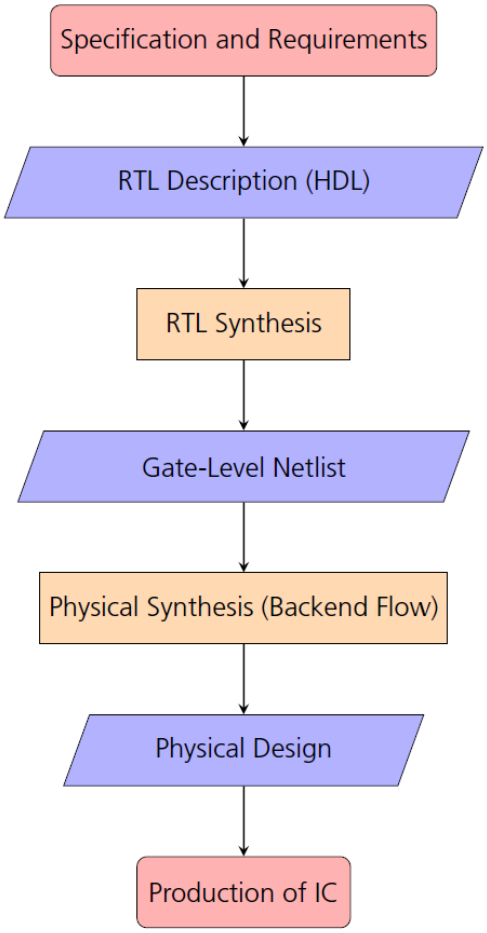



*Figure 2.1: ASIC design flow.*

### 2.2.1 Synthesis Flow

As our experiments investigate the effects of synthesis tools on the effectiveness of countermeasures, we will give an overview of the synthesis process in this section.

In short, synthesis describes the process of transforming a behavioral HDL description into a structural representation by the use of standard cells. As input, a synthesis tool takes the HDL description of a design, as well as a standard cell library. In the first step, a synthesis tool reads those inputs and parses, analyzes and elaborates (constructs hierarchies, etc.) the HDL design. Afterwards, the design environment and design constraints are set. These design constraints are additional inputs given by the designer and include certain synthesis attributes such as timing or area constraints.

In the next step the actual synthesis task is done. The behavioral HDL description is mapped to generic logic gates and optimized. Afterwards, these generic logic gates are mapped to actual standard cells defined by the provided PDK. Then, additional optimizations such as datapath optimizations or state machine decomposition are applied to reduce area and improve timing.

**Hierarchies** To implement complex designs, it is best practice to decompose one large and complex design into smaller pieces. This divide and conquer approach is repeated on each module until it is at an appropriate level of abstraction. Using hierarchies within a design has many benefits as it adds structure to the design process, enables IP-reusage and provides the means to split the tasks of a design into different portions.

However, EDA tools provide the option to flatten a hierarchical design during the synthesis process in order to enable more extensive power, performance, or area optimizations. An example for a hierarchical and

flattened design is given in Figure 2.2. The hierarchical design of a CPU shows how the datapath is decomposed recursively until the gate-level. In the flattened design all hierarchies are resolved.

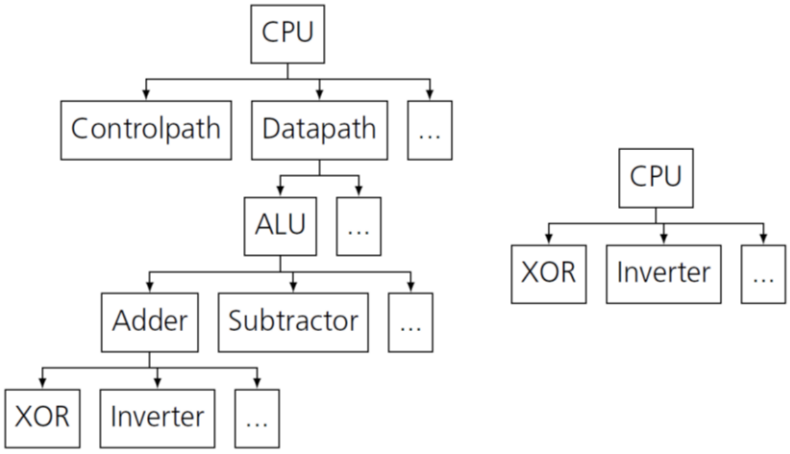



*Figure 2.2: Hierarchical (left) and flattened (right) design.*

**Standard Cells** The physical design which can be manufactured is built from so called standard cells. The idea behind standard cells is to build a whole design with a small amount of basic standardized building blocks in order to reduce development costs in comparison to full custom designs. Standard cells are basic building blocks and include basic logic elements such as inverters, NAND gates, XOR gates, flip-flops and more complicated gates. The job of a synthesis tools is to map an entire design written in HDL code onto these standard cells defined in the provided PDK. There are also more complex gates such as AOI or OAI gates. Such cells require less transistors in comparison to implementing the AOI functionality from a NAND, inverter and NOR gate. Therefore, using these gates leads to smaller area and increased speed as fewer transistors are utilized.

**Critical Path** Synchronous circuits contain combinational logic blocks, which are segmented by registers. For synchronous circuits two conditions must be met to ensure the correct behavior of the circuit: the setup and the hold time condition, which are given in Equation (2.1) and Equation (2.2), respectively.

$$T_{clk} > T_{clk2q} + T_{logic,max} + T_{setup} \quad (2.1)$$

$$T_{clk2q} + T_{logic,min} > T_{hold} \quad (2.2)$$

The setup time condition is determined by the combinational path with the longest delay $T_{logic,max}$ which is often called the critical path. The propagation delay $T_{clk2q}$ describes the time needed for data at the input to travel through the register to the output at a rising clock edge. The setup time $T_{setup}$ is the time data at the input must be stable before a rising edge in order to be read correctly. The setup time condition states that it must be ensured that the slowest logic signal must arrive at the next register before the clock edge. A design's maximum frequency ($\frac{1}{T_{clk}}$) is limited by the setup time condition and the critical path.

For the hold time condition, it must be ensured that the input of a register does not change after a rising clock edge before the hold time $T_{hold}$. This condition is due to the internal construction of registers. It does not influence the maximum frequency of a circuit and can easily be ensured by adding delay buffers to the path with minimal delay $T_{logic,min}$.

**Timing Constraints** Timing constraints are rules which dictate timing restrictions for the implementation. To be more specific, they set boundaries for the propagation time from one logic element to another. Timing constraints ensure that the setup and hold time conditions of each single flip-flop in the design are not violated for the target clock frequency.

**Driver Strength and Fan-Out** Digital gates have a capacitance which affects the propagation delay of a gate and ultimately the timing behavior of a circuit. Furthermore, the load capacitance of a gate also depends on the capacitance of subsequent gates to drive. The propagation delay of a gate increases with a higher load capacitance. The driver strength characterizes the ability of a gate to drive capacitive loads. By scaling the gate, its driver strength increases as a higher current can be drawn from the supply. Consequently, the capacitance of subsequent gates can be charged quickly. Standard cell libraries usually include multiple sizes of each gate. For example, an AND gate with two inputs may be called AND2_X1 while the same AND gate with twice the size is labeled AND2_X2. The fan-out of a gate describes the number of subsequent gates driven by the output of the gate. As explained above, a higher fan-out leads to increased load capacitance and therefore higher propagation delay. For more details about driver strength and fan-out, the reader is referred to [75].

**Retiming** Retiming [50] is a technique which changes the location of registers within a design without affecting the input/output behavior of a design. Unlike pipelining, retiming does not increase a circuit's latency. Retiming can be used to transform a given synchronous circuit into a more efficient circuit while the functional behavior is preserved.

The retiming effect can be exemplified with simple gates, e.g. a 2-input XOR. From a designer's perspective it is generally more desirable to place flip-flops on the output of the gate. If placed on the inputs, two flip-flops would be needed. An example for this is illustrated in Figure 2.3. When placing the flip-flop at the output of the circuit just one flip-flop is necessary. Retiming can also be applied to reduce the length of the critical path. This is achieved by moving the location of the flip-flop stage. While the critical path contained three logic levels before, the critical path in the retimed circuit contains only two logic levels. However, the area requirements increased as two flip-flops are necessary instead of one.

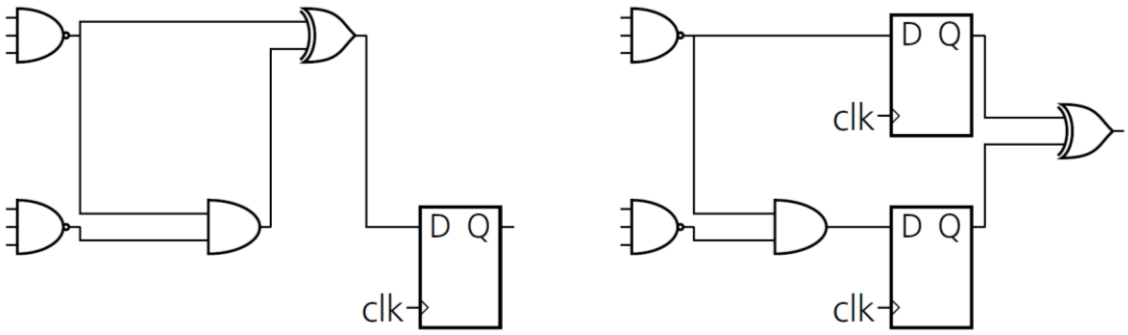



*Figure 2.3: Circuit before retiming (left) and after retiming to balance the delay of the circuit (right).*

**NAND Gate Equivalent** The NAND gate equivalent (GE) is a metric to measure the area of a circuit independent of the technology and used PDK. The area of a two-input unit NAND gate (NAND2_X1) constitutes the technology-dependent unit area. A circuit's area is then represented by dividing its technology-dependent area consumption by this unit area. This amounts to a circuit's NAND-GE.

### 2.2.2 Tooling and Vendors

There are several EDA tool vendors which offer a wide range of software, including tools for RTL synthesis, simulation, and physical design. RTL synthesis tools can further be divided into tools for FPGA or ASIC development. Table 2.1 gives a short overview over the most popular synthesis tools. However, the most popular EDA software vendors and firm market leaders are Cadence and Synopsys. Their combined market share comprises over 60% of the global market, as depicted in Figure 2.4. Therefore, we focus on ASIC synthesis tools offered by Cadence and Synopsys in the following investigations of this report. Additionally, we use the open-source tool Yosys, as it allows us to explore its inner workings and open-source hardware toolchains are getting more popular.

*Table 2.1: List of popular synthesis tools for FPGA and ASIC synthesis.*

| ***Vendor*** | ***Tool*** | ***Technology*** |
|---|---|---|
| Cadence | Genus | ASIC |
| Synopsys | Design Compiler | ASIC |
| Siemens | Oasys-RTL | ASIC |
| AMD | Vivado | FPGA |
| Intel | Quartus Prime | FPGA |
| Synopsys | Synplify | FPGA |
| Siemens | Precision-RTL | FPGA |

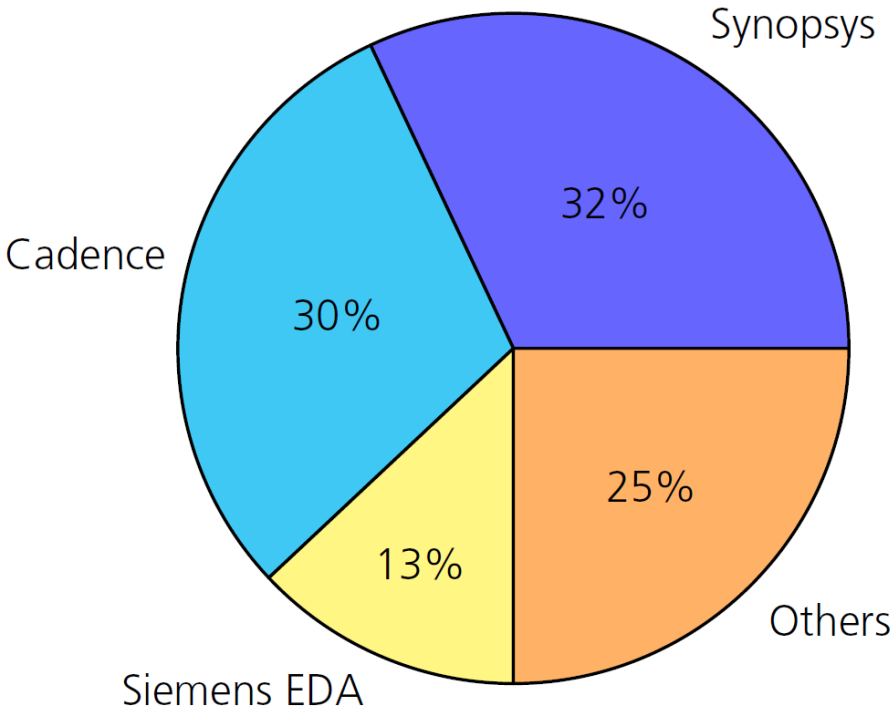



*Figure 2.4: EDA software market shares, August 2021. Source:*
*https://www.trendforce.com/presscenter/news/20220815-11338.html*

# 3 Fault Injection Countermeasures

During the synthesis step of the design flow, synthesis tools perform optimizations on the design to reduce area and improve timing. On that account, synthesis tools are able to recognize redundancy and tend to optimize redundant structures in favor of reduced area and a shorter critical path. Section 3.1 reviews the literature on fault injection countermeasures with respect to synthesis. In particular, literature on hardening FSMs is discussed. These publications already highlight certain effects during synthesis and propose solutions. Section 3.2 describes pre-silicon fault analysis and focuses on the SYNFI tool as verification framework for synthesized gate-level netlists. In Section 3.3, the SYNFI framework is used to conduct six case studies. Goal of these case studies is to analyze and evaluate different synthesis tools and settings with respect to harmful optimizations applied during RTL synthesis which could weaken countermeasures. Any effects which could occur after RTL synthesis are not considered here. However, Chapter 5 discusses the questions that are still open.

## 3.1 State-of-the-Art

As discussed in Section 2.1.2, most fault injection countermeasures are based on redundancy. Therefore, it is intuitive, that optimizations in the design flow might have negative impacts on the effectiveness of these countermeasures. There are various publications in which researchers deal with this problem. In the following, we give a short impression of such works.

Synthesis tools tend to introduce *don't care* states and transitions into FSMs to simplify the circuit and reduce spent area. A *don't care* state is a state which is not specified or does not matter for a particular combination of inputs. It means that the FSM can transition to any state for that specific input combination. These *don't care* states are then used to simplify the design of an FSM by reducing the number of required logic gates. These additional states and transitions might introduce vulnerabilities into the design which can be exploited by an attacker. For example, a security critical state within a FSM could be bypassed if states can be accessed, i.e. by means of fault injection, from unspecified *don't care* states and transitions [57]. Nahiyan et al. address this issue in [58] and propose a security aware design flow to develop secure finite state machines. In their work, they analyze vulnerabilities of FSMs by analyzing the state transitions. Their tool checks if a fault can be injected during a state transition such that a critical state can be accessed through an unauthorized state. A critical state could bypass security checks or make unauthorized changes to sensitive data. As solution to such vulnerabilities, they propose a two-fold approach. First, they propose a security-aware encoding scheme with a special focus on *don't care* states. Second, their presented FSM architecture ensures that only authorized states can access critical states.

In a similar work [60], Nasahl et al. proposed an FSM protection methodology called state-machine control-flow hardening against fault attacks (SCFI) which detects deviations from the intended control-flow. Their tool substitutes the unprotected next-state logic of an FSM with a fault-hardened next-state logic which takes the execution history as well as the FSM's control signals as additional inputs to derive the next state. For that purpose, the next-state logic is implemented via a multi-input signature register which compresses the execution history and detects control-flow deviations. Furthermore, they extend the open-source synthesis tool Yosys with their SCFI tool. As a result, their modified Yosys synthesis toolchain is capable of automatically protecting arbitrary FSMs with SCFI.

The work of Kibria et al. [43] highlights the need for comprehensive security verification in the early stages of system-on-chip (SoC) design. In particular, their focus is the control flow implemented in FSMs. The authors propose a set of security rules for FSM designs and introduce a verification framework called ARC-FSM-G to detect violations of these rules at the gate-level netlist abstraction. Their framework maps the high-level RTL description onto associated state transitions graphs extracted from a synthesized gate-level netlist. With this mapping, the gate-level state transition graphs are validated by checking the proposed security rules.

## 3.2 Pre-Silicon Fault Analysis

The previous section introduced the state-of-the-art on fault injection analysis and hardening of FSMs. The literature points out that EDA tools might introduce vulnerabilities into a design or weaken countermeasures through optimizations. Therefore, it is not sufficient to consider only an abstract RTL representation of a design alone. Rather, a representation of a design which is close to the actual physical level, e.g. gate-level netlists, must be considered when performing fault analysis.

In this context, the work by Nasahl et al. [59] enables us to analyze gate-level netlists and therefore investigate the influence of synthesis tools on fault injection hardened designs in a systematic way. They propose a formal pre-silicon fault verification framework called SYNFI. This framework takes synthesized gate-level netlists as input and verifies the effectiveness of fault injection countermeasures after synthesis. Within their work, the authors show various case studies in which they use SYNFI. Their results confirm the outcomes of the publications summarized in the previous section [43] [58] [60]. Synthesis tools and the respective optimizations can have a negative influence on the effectiveness of countermeasures. However, they do not investigate different EDA tools and settings in systematic way and thus leave space for further analysis. More concretely, we are interested in the influence of various optimization parameters, differences in tools, and best practices for secure hardware design. Therefore, we use the SYNFI framework to analyze different synthesis settings and tools with regards to potentially harmful optimizations during synthesis. In the following, we describe six case studies. Before discussing these case studies in more detail, the following gives an overview of the SYNFI framework and its workflow.

The workflow of the SYNFI framework is depicted in Figure 3.1. It requires the following inputs:

- Netlist: The netlist is synthesized independent of SYNFI with an EDA tool such as Cadence Genus, Synopsys Design Compiler or Yosys. Internally, this netlist is converted into a graph representation of the netlist.
- Cell Library: The cell library is converted into an internal library. For that purpose, SYNFI extracts the names, the boolean functions and the pins of the standard cells. With the help of this cell library the netlist is converted to a graph.
- Specification of Fault Experiment: The specification of the fault experiment contains the specification of a target circuit (within the netlist), a fault mapping and the requirements for an effective fault including the expected output behavior of a circuit as well as the condition which is fulfilled when the error detection/countermeasure is triggered. For a more detailed description of the fault experiment specification, the reader is referred to [59].

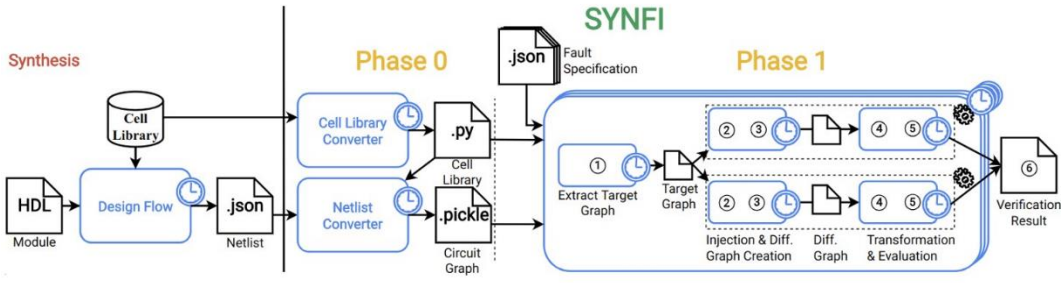



*Figure 3.1: Block diagram of the SYNFI framework from [59].*

In the pre-processing step (see Phase 0 of Figure 3.1), SYNFI takes the netlists as well as the cell library and converts the netlist to a circuit graph. Afterwards (see Phase 1 of Figure 3.1), in the first step of the SYNFI framework, the tool takes the fault experiment specification and extracts a subgraph from the circuit graph. This subgraph is called target graph and contains only the part of the netlist which is relevant for the fault experiment and specified within the fault experiment specification. In the next step, faults are injected into

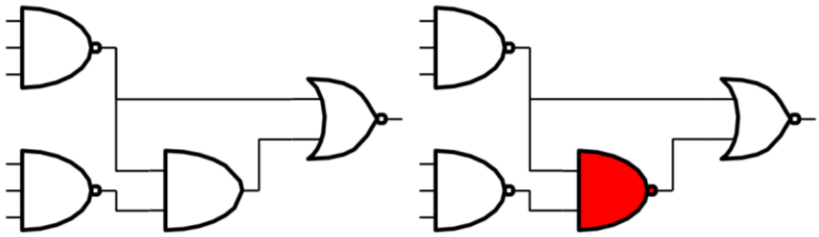


*Figure 3.2: Circuit without injected fault (left). Circuit with injected fault into AND gate (red) by mapping AND to NAND (right).*

the target graph by exchanging cells according to the fault mapping. An example for such a fault mapping is converting AND gates to NAND gates (AND=[NAND]) and is illustrated in Figure 3.2. In this example, a transient fault injected into the AND gate is modeled by replacing it with a NAND gate for one clock cycle. Furthermore, SYNFI is able to model stuck-at faults in a similar manner (e.g. AND=[0]). When providing multiple entries (e.g. AND=[NAND,0]), SYNFI is able to analyze a circuit with regards to transient and stuck-at faults. This means that a wide range of fault models (Section 2.1) can be analyzed.

If the fault does not have an effect on the input/output relationship of the target circuit the fault is called ineffective. Otherwise, the fault is called effective. When the target circuit contains a countermeasure, a fault is effective if the input/output relationship of a circuit has changed and the countermeasure is not triggered. Otherwise, the fault is an ineffective fault. The tool exhaustively injects faults into all gates and tries every possible combination when injecting simultaneous faults. In this process, SYNFI uses a SAT solver to find effective fault combinations. For a more detailed description of the SYNFI framework the reader is referred to [59].

**SYNFI workflow and limitations** To specify fault experiments with SYNFI it is necessary to examine the gate-level netlist to be analyzed. From this gate-level netlist the target circuit must be extracted by hand. Depending on a tool's naming conventions and applied optimizations it is hard and in some cases impossible to identify the target circuit's components within a netlist. Furthermore, when specifying the target circuit in the specification of the fault experiment, the target circuit has to be divided into stages (paths between flip-flops). For each gate type within the netlist, a fault mapping as explained above must be specified. Moreover, input values of the circuit's input pins, as well as expected output values must be provided. If the circuit comprises a countermeasure, it is necessary to specify what signal is set to which value when a fault is detected. A fault experiment conducted with SYNFI only takes one clock cycle into account. Therefore, analyzing a complete cryptographic algorithm running on fault-hardened hardware with SYNFI is a tedious process. Depending on the size of the circuit and the number of bit-flips, the verification procedure becomes computationally expensive. As result, SYNFI reports the number of effective faults that could be injected into the target circuit. The tool does not report the fault combinations directly. However, only slight modifications are necessary to enable SYNFI to do so.

**Fault model and assumptions** Within our case studies, we use the findings from SYNFI as metric. We make the following assumptions: the more bits must be flipped, and the lower the number of one/two/three bit-flip combinations to corrupt a design, the more resilient is the design. In the following, we use the term *one fault* equivalent to one-bit fault. It is intuitive that two precise bit-flips are harder to achieve than one precise bit-flip. The question of exactly how much harder is still largely unanswered. We revisit this question in Chapter 5.

## 3.3 Case Studies

In this section we analyze different synthesis settings and tools with regards to potentially harmful optimizations during synthesis. We use SYNFI to assess the resilience of a netlist.

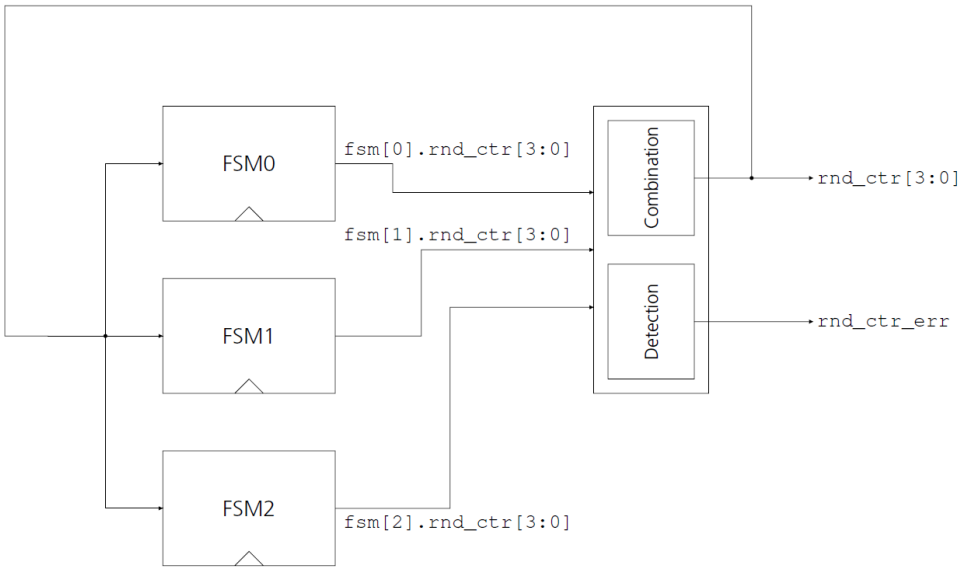



*Figure 3.3: Sub-circuit comprising three redundant FSMs. The "Combination" block combines the three round counter values from the FSMs into one value, which is fed back to the input of the FSMs. The "Detection" block implements the error detection.*

In the case studies we consider the control module of the AES core of the OpenTitan[3]. The AES core provides encryption and decryption mechanisms for the AES block cipher [61]. It is one of the most crucial components of the OpenTitan and hardened against side-channel attacks as well as fault injection. More specifically, we focus on the round counter functionality within the so called *aes_cipher_control module*. Depending on the mode of operation a certain number of rounds is performed. The round counter within the aes_cipher_control module takes care that the correct number of rounds is executed. Manipulating the round counter could weaken the cryptographic strength of AES as shown in [10]. Therefore, the round counter within the aes_cipher_control module is highly security-critical.

The round counter corresponds to a 4-bit register. Its value is controlled by a FSM, which determines when a round is completed and increments the value accordingly. To counteract fault attacks, this FSM is instantiated three times. An abstract block diagram of this circuit is depicted in Figure 3.3. Each FSM within this sub-circuit has its own 4-bit round counter, denoted by *fsm[i].rnd_ctr[3:0]* for $i = 0, 1, 2$. These three signals are combined to derive the input round counter signal rnd_ctr[3:0] for the FSMs in the next clock cycle. If the round counter values *fsm[i].rnd_ctr[3:0]* for $i = 0, 1, 2$ differ from each other, the rnd_ctr_err signal is triggered which forces the AES to stop all processing. Thus, all faults up to two bit-flips injected into the round counter logic should be detected. Injecting three faults should lead to effective faults. Note that not all combinations of three simultaneous faults lead to changes in the input/output relationship and thus are effective faults. Analyzing the effect of three simultaneous faults serves as sanity check and furthermore, provides an additional attest of the circuit's resilience. Therefore, we use SYNFI to investigate the effect of up to three simultaneous faults on the input/output relationship of the target circuit comprising the round counter and the corresponding error detection.

In all following case studies the relative number of effective faults describes the number of effective fault combinations in relation to the number of all possible fault combinations.

Note that within this analysis the aes_cipher_control module is considered standalone. Therefore, effects which would occur when integrating this module into a larger design, such as merging logic between different modules, are not captured here.

[3] https://github.com/lowRISC/opentitan

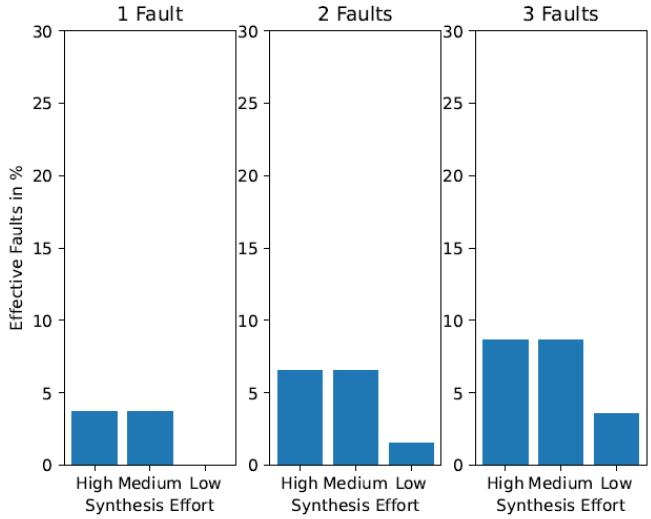
### 3.3.1 Case Study 1: Synthesis Effort

In this first case study we analyze the impact of the synthesis effort on the effectiveness of the fault detection countermeasure. On that account we synthesize the aes_cipher_control module with Cadence Genus and apply different effort levels, namely “low”, “medium” and “high”. According to the reference manual [12] these effort levels have the following effects:

- low: The design is only mapped to gates but barely any optimizations are performed.
- medium: Timing-driven structuring is applied to restructure critical paths such that delay is reduced. Additionally, redundancy identification and removal is performed.
- high: Timing-driven structuring is applied on larger structures. Additionally, redundancy identification and removal is performed more aggressively.

Figure 3.4 shows the results of the conducted experiments. Although the design is intended to detect up to two faults, our results show that there exist fault combinations which have an influence on the input/output relationship of the circuit without being detected. These fault combinations are denoted as effective faults. It is clear from Figure 3.4 that the synthesis effort - and therefore the applied optimizations - have an influence on the effectiveness of the deployed countermeasure. For the effort levels “medium” and “high” there are single faults which change the input/output relationship of the circuit without triggering the error detection. For the effort level “low”, at least two simultaneous faults are necessary to change the input/output relationship of the circuit without triggering the error detection.

While a low synthesis effort might be beneficial for the effectiveness of fault injection countermeasures, it is not a practical solution as the circuit’s area is significantly increased. To emphasize this point, Table 3.1 provides an overview of the design’s area consumption with regards to synthesis effort.




(a) Barplot with relative effective faults.

| | # Sim. Faults | Abs. # Eff. Faults | Rel. # Eff. Faults |
|---|---|---|---|
| high/medium | 1 | 6 | 3.7% |
| low | 1 | 0 | 0.0% |
| high/medium | 2 | 1,692 | 6.53% |
| low | 2 | 606 | 1.53% |
| high/medium | 3 | 177,822 | 8.67% |
| low | 3 | 135,921 | 3.49% |

(b) Table with relative and absolute effective faults.

*Figure 3.4: Number of effective faults for 1, 2 and 3 simultaneous faults injected into a gate-level netlist synthesized with Cadence Genus and different effort levels.*

*Table 3.1: Area consumption of the complete circuit and extracted sub-circuit for the netlists in Case Study 1.*

| *Synthesis Effort* | *Circuit Size (NAND GE)* | *Target Circuit Size (NAND GE)* |
|---|---|---|
| high/ medium | 756 | 125 |
| low | 1140 | 142 |

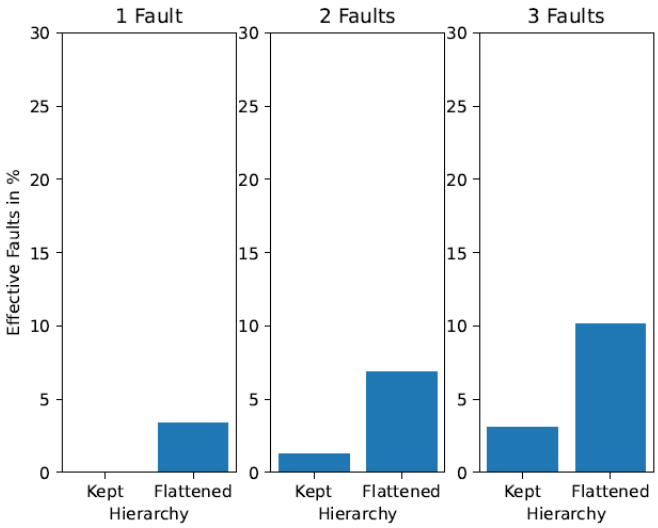
In summary, these experiments point out that, although the RTL description of a circuit seems resistant to fault attacks, there is no guarantee that the resulting gate-level netlist also provides such fault resistance. This highlights that the RTL code alone is insufficient to make guarantees about the security against fault injection attacks. The results indicate that there is a relation between synthesis effort and effectiveness of fault injection countermeasures. More specifically, with higher synthesis effort and more aggressive optimizations, the effectiveness of fault injection countermeasures decreases.

As the used synthesis tool is not open-source, it is hard to find the exact source for this behavior. However, it is obvious that, with a more aggressive synthesis setting, more effort is spent to detect redundancy. Consequently, the redundancy will be resolved in favor of lower resource consumption. As redundancy is the basis for the described fault injection countermeasure its removal introduces flaws into the fault detection capabilities and ultimately weakens the countermeasure.

### 3.3.2 Case Study 2: Comparison between Hierarchical and Flattened Netlists with different Synthesis Tooling

As mentioned in Section 2.2.1, synthesis tools perform optimizations on a design to reduce area and improve timing. By flattening a design, one can enable the synthesis tool to perform optimization across hierarchies (as hierarchies are resolved during flattening). For this case study, we use Synopsys Design Compiler and resolve all hierarchies within the design before running the synthesis and applying any optimizations.

Figure 3.5 shows the results for this experiment and indicates that resolving the hierarchies within this design leads to inferior error detection performance. Considering Table 3.2, this inferior error detection performance comes in favor of lower area consumption. These results highlight the findings of Case Study 1: it is insufficient to consider the RTL code alone as applied constraints and optimizations during synthesis have a significant influence on the effectiveness of an implemented countermeasure. A possible explanation for this behavior is that flattening hierarchies allows the synthesis tool to detect redundancy across module boundaries (as these boundaries are resolved) and enable further optimizations to remove redundancy in favor of resource consumption.




(a) Barplot with relative effective faults.

| | # Sim. Faults | Abs. # Eff. Faults | Rel. # Eff. Faults |
|---|---|---|---|
| Hierarchical | 1 | 0 | 0.0% |
| Flattened | 1 | 6 | 3.41% |
| Hierarchical | 2 | 510 | 1.27% |
| Flattened | 2 | 2,102 | 6.86% |
| Hierarchical | 3 | 120,948 | 3.03% |
| Flattened | 3 | 266,688 | 10.12% |

(b) Table with relative and absolute effective faults.

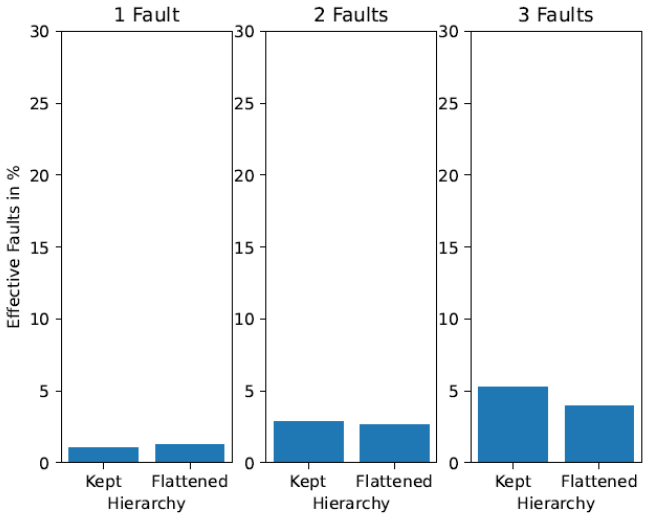
*Figure 3.5: Number of effective faults for 1, 2 and 3 simultaneous faults injected into hierarchical and flattened gate-level netlists synthesized with Synopsys Design Compiler.*

*Table 3.2: Area consumption of the complete circuit and extracted sub-circuit for the netlists generated with Synopsys Design Compiler.*

| ***Design*** | ***Circuit Size (NAND GE)*** | ***Target Circuit Size (NAND GE)*** |
|---|---|---|
| Hierarchical | 1100 | 135 |
| Flattened | 816 | 140 |

As flattening of hierarchies is an optimization available in most synthesis tools we use this setting to conduct additional FI experiments with netlists generated from Yosys and Cadence Genus. This way, we are able to consolidate previous findings and exclude (if existing) tool specific effects.

Figure 3.6 shows the results of the FI experiment when using Yosys for synthesis. In this case the number of relative effective faults is higher for two- and three-bit faults when the hierarchy is preserved. However, the absolute number of effective faults is higher in the flattened netlists. This is due to the fact that, in this case, the extracted sub-circuit is smaller (Table 3.3) with hierarchies. When using Cadence Genus, a flattened hierarchy does neither lead to inferior error detection performance nor to reduced area.




(a) Barplot with relative effective faults.

| | # Sim. Faults | Abs. # Eff. Faults | Rel. # Eff. Faults |
|---|---|---|---|
| Hierarchical | 1 | 2 | 1.03% |
| Flattened | 1 | 3 | 1.3% |
| Hierarchical | 2 | 1,068 | 2.84% |
| Flattened | 2 | 1,394 | 2.64% |
| Hierarchical | 3 | 192,818 | 5.24% |
| Flattened | 3 | 238,308 | 3.98% |

(b) Table with relative and absolute effective faults.

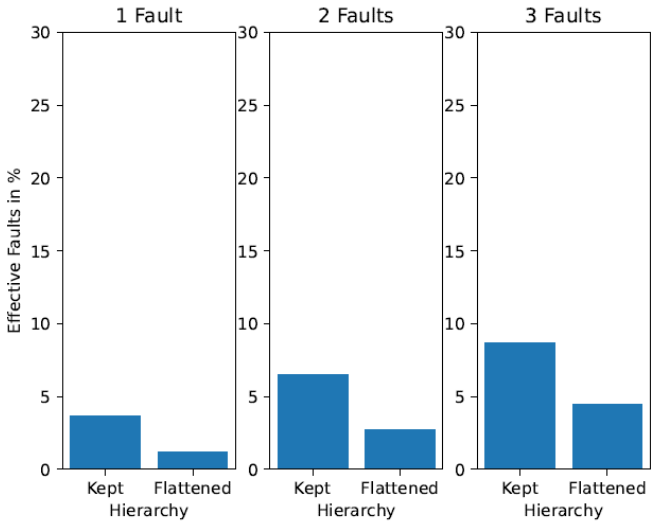
*Figure 3.6: Number of effective faults for 1, 2 and 3 simultaneous faults injected into hierarchical and flattened gate-level netlists synthesized with Yosys.*

*Table 3.3: Area consumption of the complete circuit and extracted sub-circuit for the netlists generated with Yosys.*

| ***Design*** | ***Circuit Size (NAND GE)*** | ***Target Circuit Size (NAND GE)*** |
|---|---|---|
| Hierarchical | 1170 | 136 |
| Flattened | 1160 | 148 |

Considering all results derived across all tools, indicates that the effect of optimizations applied to a circuit are tool specific. This implies that not only the RTL code and the applied synthesis constraints must be taken into account but also the synthesis tool itself.

Figure 3.7 and Table 3.4 show the results for Cadence Genus.




(a) Barplot with relative effective faults.

| | # Sim. Faults | Abs. # Eff. Faults | Rel. # Eff. Faults |
|---|---|---|---|
| Hierarchical | 1 | 6 | 3.7% |
| Flattened | 1 | 2 | 1.2% |
| Hierarchical | 2 | 1,692 | 6.53% |
| Flattened | 2 | 750 | 2.76% |
| Hierarchical | 3 | 177,822 | 8.67% |
| Flattened | 3 | 99,828 | 4.34% |

(b) Table with relative and absolute effective faults.

*Figure 3.7: Absolute and relative number of effective faults for 1, 2 and 3 simultaneous faults injected into a gate-level netlist synthesized with Cadence Genus (effort level "high").*

*Table 3.4: Area consumption of the complete circuit and extracted sub-circuit for the netlists generated with Cadence Genus (effort level "high").*

| ***Design*** | ***Circuit Size (NAND GE)*** | ***Target Circuit Size (NAND GE)*** |
|---|---|---|
| Hierarchical | 756 | 125 |
| Flattened | 778 | 125 |

### 3.3.3 Case Study 3: Netlist Analysis and Restriction of Standard Cell Types

Analyzing the gate-level netlists of Case Study 1 and 2, synthesized with Synopsys Design Compiler, reveals that synthesis tools transform circuits in a way such that OAI and AOI standard cells (see Section 2.2.1) can be used. Figure 3.8 shows an example of such an error correction circuit within these gate-level netlists for synthesis with Synopsys Design Compiler.

The usage of these more compact standard cells has the advantage of reduced area and improved timing. However, this circuit transformation introduces additional effective fault combinations into the design and therefore weakens the error detection. For example, faults after the inverter at the output of rnd_ctr[2] or rnd_ctr[3] remain undetected as there is no path to the error detection circuit from these inverters. Furthermore, the combination of OR and NAND gates (rnd_ctr[0] and rnd_ctr[1]) is unable to detect 1 → 0 faults. The combination of NOR and AND gates (rnd_ctr[2] and rnd_ctr[3]), in turn, cannot detect 0 → 1

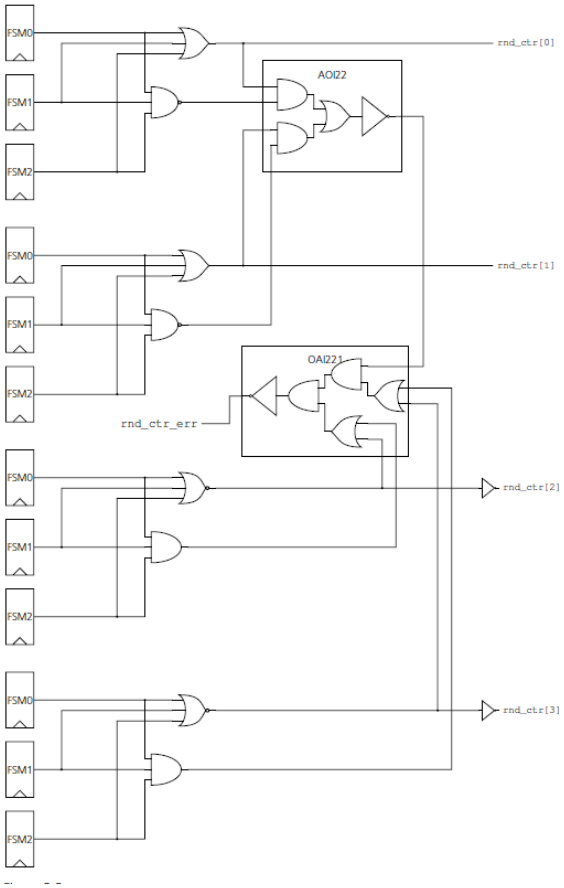



*Figure 3.8: Netlist synthesized with Synopsys Design Compiler showing a sub-circuit comprising AOI and OAI standard cells.*

faults. See Figure A. 1 and Figure A. 2 in the appendix for examples. In order to prevent this circuit transformation, we prohibit the synthesis tool from using these kind of standard cells. For this case study we focus on Synopsys Design Compiler to explain this effect more detailed. However, we also consider Cadence Genus later on in this case study. By restricting the allowed standard cell types, the output netlists comprise an error correction circuit similar to Figure 3.9, when using Synopsys Design Compiler.

The error correction performance of the gate-level netlists synthesized with Synopsys Design Compiler for both restricted and non-restricted cell types is summarized in Figure 3.10. Table 3.5 compares the area for both cases.

These results indicate that restricting the standard cell types leads to an increase in area consumption. Although the harmful circuit transformation described above is avoided and the error detection performance in terms of relative effective faults increases slightly, the circuit is still unable to detect $1 \rightarrow 0$ faults. However, due to the increased area and the consequently increased gate count, the number of absolute effective faults is higher when restricting the usage of these cells (similar to the effect for flattened netlists from Case Study 2). From analyzing Figure 3.9 in more detail, one can see that replacing the NAND gates, which feed the four inputs of the rnd_ctr_err NAND gate, with XOR gates could fix this flaw.

We conducted the same experiment with Cadence Genus as synthesis tool. The results for this experiment are illustrated by Figure 3.11 for error detection results and Table 3.6 for area results.

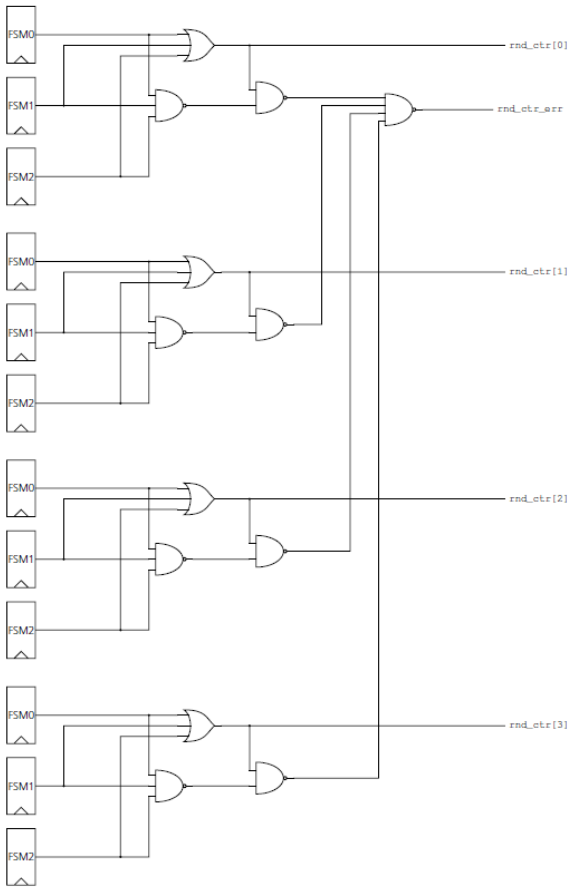
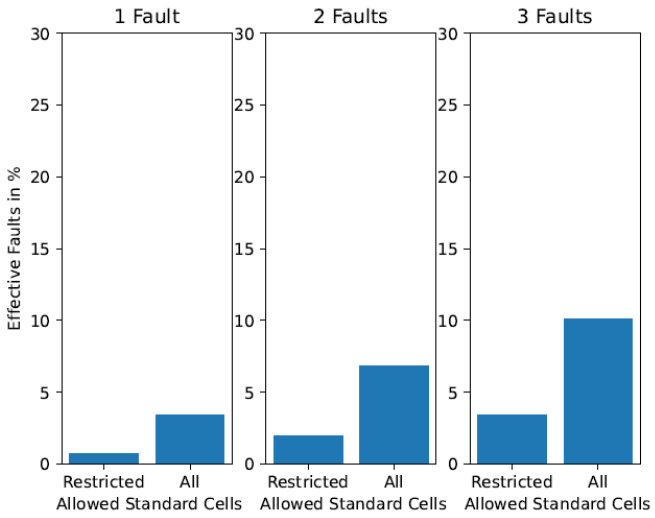



*Figure 3.9: Netlist synthesized with Synopsys Design Compiler, restricting the usage of AOI and OAI standard cells.*

A possible explanation for the usage of AOI and OAI standard cells is the fact that these standard cells enable the synthesis tool to implement the same function with fewer area overhead due to the compactness (fewer transistors) of these cells. Evidence for this is given by Table 3.5 and Table 3.6. Therefore, synthesis tools seem to map functions to these cells whenever possible. However, as pointed out in this case study, using these cells in critical spots (see Figure 3.8) might introduce flaws into the error detection capabilities of a countermeasure.




(a) Barplot with relative effective faults.

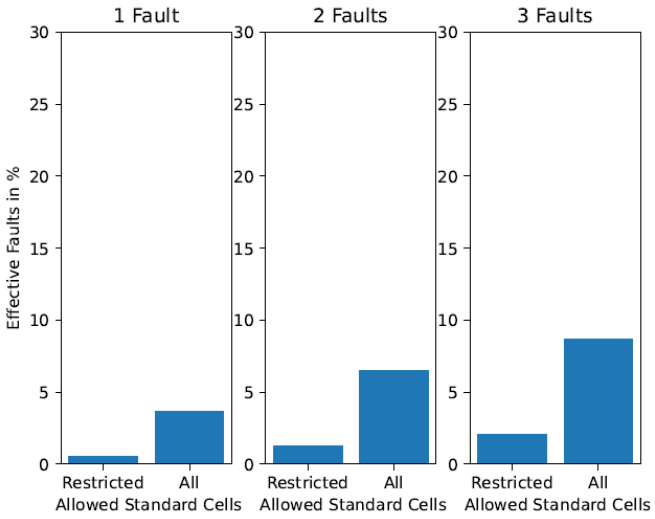
| | # Sim. Faults | Abs. # Eff. Faults | Rel. # Eff. Faults |
|---|---|---|---|
| All Cells | 1 | 4 | 2.08% |
| Restricted Cells | 1 | 2 | 0.53% |
| All Cells | 2 | 1,576 | 4.33% |
| Restricted Cells | 2 | 1,916 | 1.37% |
| All Cells | 3 | 222,660 | 6.52% |
| Restricted Cells | 3 | 609,420 | 2.36% |

(b) Table with relative and absolute effective faults.

*Figure 3.10: Number of effective faults for 1, 2 and 3 simultaneous faults injected into a gate-level netlist synthesized with Synopsys Design Compiler with and without restricted standard cell types.*

*Table 3.5: Area consumption of the complete circuit and extracted sub-circuit for the netlists generated with Synopsys Design Compiler.*

| ***Design*** | ***Circuit Size (NAND GE)*** | ***Target Circuit Size (NAND GE)*** |
|---|---|---|
| No Restriction on Cells | 1100 | 130 |
| Restricted Cells | 1240 | 148 |




(a) Barplot with relative effective faults.

| | # Sim. Faults | Abs. # Eff. Faults | Rel. # Eff. Faults |
|---|---|---|---|
| All Cells | 1 | 6 | 3.7% |
| Restricted Cells | 1 | 2 | 0.53% |
| All Cells | 2 | 1,692 | 6.53% |
| Restricted Cells | 2 | 1,738 | 1.24% |
| All Cells | 3 | 177,822 | 8.77% |
| Restricted Cells | 3 | 534,777 | 2.07% |

(b) Table with relative and absolute effective faults.

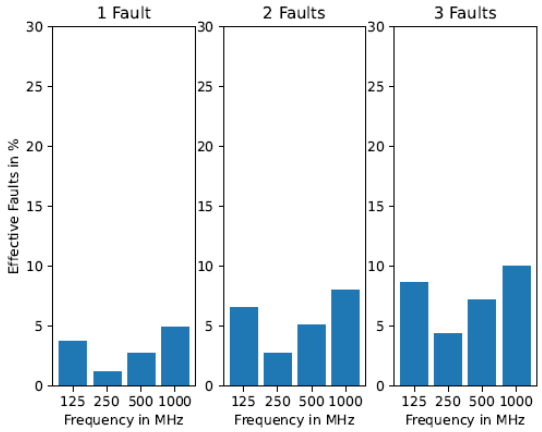
*Figure 3.11: Absolute and relative number of effective faults for 1, 2 and 3 simultaneous faults injected into a gate-level netlist synthesized with Cadence Genus (effort level "high") with and without restricted standard cell types.*

*Table 3.6: Area consumption of the complete circuit and extracted sub-circuit for the netlists generated with Cadence Genus (effort level "high").*

| ***Design*** | ***Circuit Size (NAND GE)*** | ***Target Circuit Size (NAND GE)*** |
|---|---|---|
| No Restriction on Cells | 758 | 125 |
| Restricted Cells | 924 | 167 |

### 3.3.4 Case Study 4: Effect of Timing Constraints

The previous case studies do not investigate the influence of timing constraints, such as the target frequency, on the effectiveness of the presented countermeasure. As timing constraints get tighter, synthesis tools might have to apply additional optimizations to the circuit to meet these timing constraints. Therefore, we investigate their effect in the following. To do so, we apply different synthesis constraints with different target clock frequencies (125 MHz, 250 MHz, 500 MHz and 1 GHz) to the design and synthesize it with Cadence Genus at effort level "high". As explained in Section 3.3.1, this effort level enables additional timing-driven optimizations. Therefore, it is necessary to select effort level "high" to capture all timing-driven effects. Figure 3.12 summarizes the results of the fault analysis and Table 3.7 shows the respective area consumption.




(a) Barplot with relative effective faults.

| | # Sim. Faults | Abs. # Eff. Faults | Rel. # Eff. Faults |
|---|---|---|---|
| 125 MHz | 1 | 6 | 3.7% |
| 250 MHz | 1 | 2 | 1.2% |
| 500 MHz | 1 | 4 | 2.7% |
| 1 GHz | 1 | 8 | 4.91% |
| 125 MHz | 2 | 1,692 | 6.53% |
| 250 MHz | 2 | 756 | 2.74% |
| 500 MHz | 2 | 1,106 | 5.12% |
| 1 GHz | 2 | 2,094 | 7.99% |
| 125 MHz | 3 | 177,822 | 8.67% |
| 250 MHz | 3 | 98,865 | 4.39% |
| 500 MHz | 3 | 111,801 | 7.16% |
| 1 GHz | 3 | 209,385 | 10.04% |

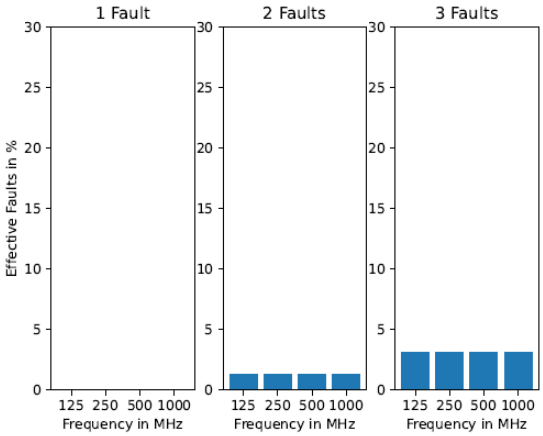
(b) Table with relative and absolute effective faults.

*Figure 3.12: Number of effective faults for 1, 2 and 3 simultaneous faults injected into a gate-level netlist synthesized with Cadence Genus at effort level "high" with different timing constraints.*

*Table 3.7: Area consumption of the complete circuit and extracted sub-circuit for the netlists generated with Cadence Genus for different target frequencies at effort level "high".*

| ***Target Frequency*** | ***Circuit Size (NAND GE)*** | ***Target Circuit Size (NAND GE)*** |
|---|---|---|
| 125 MHz | 756 | 125 |
| 250 MHz | 768 | 126 |
| 500 MHz | 786 | 124 |
| 1 GHz | 1040 | 130 |

From Table 3.7 it is evident that increasing the clock frequency results in higher area requirements. This can be explained by the fact that synthesis tools use larger cells with higher driver strength to achieve faster designs [75]. These cells have the same functionality but switch faster as a higher current can be drawn from the supply. The drawback is that they require more area. Moreover, synthesis tools might duplicate certain circuit structures to decrease the fan-out of combinational cells in the critical path. Apart from the outlier at 125 MHz, there is a tendency towards inferior error detection performance when increasing the target frequency. While cells with higher driver strength have no influence on the error detection performance, the duplication of structures might indeed lead to inferior error detection. For example, if parts of the round counter circuit are duplicated, the synthesis tool is free to structure the logic in a way that the round counter logic used by the error detection and the round counter used in the FSMs are partly or completely independent. This is possible, as - without fault injection - the value used by the error detection circuit and




(a) Barplot with relative effective faults.

| | # Sim. Faults | Abs. # Eff. Faults | Rel. # Eff. Faults |
|---|---|---|---|
| 125 MHz | 1 | 0 | 0.0% |
| 250 MHz | 1 | 0 | 0.0% |
| 500 MHz | 1 | 0 | 0.0% |
| 1 GHz | 1 | 0 | 0.0% |
| 125 MHz | 2 | 510 | 1.27% |
| 250 MHz | 2 | 510 | 1.27% |
| 500 MHz | 2 | 510 | 1.27% |
| 1 GHz | 2 | 510 | 1.27% |
| 125 MHz | 3 | 120,948 | 3.03% |
| 250 MHz | 3 | 120,948 | 3.03% |
| 500 MHz | 3 | 120,948 | 3.03% |
| 1 GHz | 3 | 120,948 | 3.03% |

(b) Table with relative and absolute effective faults.

*Figure 3.13: Number of effective faults for 1, 2 and 3 simultaneous faults injected into a gate-level netlist synthesized with Synopsys Design Compiler for different target frequencies.*

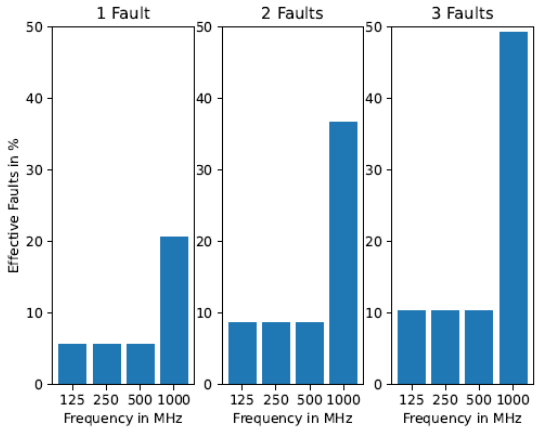
the FSM logic are always identical. In this case, faults inserted into the FSM that affect the AES operation, are not detected. It should be noted that, while the driver strength of a cell does not matter to SYNFI, it might be important for real-world fault attacks. Cells that can drive more subsequent cells feature more transistors. Therefore, an adversary that tries to inject faults by means of laser fault injection could more easily invert the output of a cell with a higher driver strength.

We repeat this experiment for Synopsys Design Compiler. The results of these investigations are summarized in Figure 3.13. In contrast to the experiments with Cadence Genus there is no inferior error detection performance when increasing the frequency. As the number of effective faults is identical for all target frequencies, Synopsys Design Compiler might compile the exact same netlist independent of the target frequency. For 125, 250 and 500 MHz, the area consumption (see Table 3.8) of the target circuit provides additional evidence for this hypothesis. An explanation for this behavior is that the timing constraints alone do not trigger additional optimizations which change the composition of the circuit.

Synopsys Design Compiler also offers synthesis for high-performance designs. This version is called Design Compiler Ultra and requires a separate license. However, details on the inner workings, such as the used algorithms or synthesis optimizations are not given by Synopsys. As high-performance designs are typically associated with high clock frequencies and the experiment with Synopsys Design Compiler leads to unexpected results, this variant of the synthesis tool is also investigated for this case study. The results of these experiments are summarized in Figure 3.14. The respective area utilization is given in Table 3.9.

*Table 3.8: Area consumption of the complete circuit and extracted sub-circuit for the netlists generated with Synopsys Design Compiler.*

| *Target Frequency* | *Circuit Size (NAND GE)* | *Target Circuit Size (NAND GE)* |
|---|---|---|
| 125 MHz | 1100 | 135 |
| 250 MHz | 1100 | 135 |
| 500 MHz | 1310 | 135 |
| 1 GHz | 1440 | 135 |




(a) Barplot with relative effective faults.

| | # Sim. Faults | Abs. # Eff. Faults | Rel. # Eff. Faults |
|---|---|---|---|
| 125 MHz | 1 | 8 | 5.56% |
| 250 MHz | 1 | 8 | 5.56% |
| 500 MHz | 1 | 8 | 5.56% |
| 1 GHz | 1 | 36 | 20.67% |
| 125 MHz | 2 | 1,762 | 8.61% |
| 250 MHz | 2 | 1,762 | 8.61% |
| 500 MHz | 2 | 1,762 | 8.61% |
| 1 GHz | 2 | 11,088 | 36.66% |
| 125 MHz | 3 | 148,710 | 10.36% |
| 250 MHz | 3 | 148,710 | 10.36% |
| 500 MHz | 3 | 148,710 | 10.36% |
| 1 GHz | 3 | 1,269,981 | 49.24% |

(b) Table with relative and absolute effective faults.

*Figure 3.14: Number of effective faults for 1, 2 and 3 simultaneous faults injected into a gate-level netlist synthesized with Synopsys Design Compiler Ultra for different target frequencies.*

*Table 3.9: Area consumption of the complete circuit and extracted sub-circuit for the netlists generated with Synopsys Design Compiler Ultra.*

| *Target Frequency* | *Circuit Size (NAND GE)* | *Target Circuit Size (NAND GE)* |
|---|---|---|
| 125 MHz | 786 | 112 |
| 250 MHz | 786 | 112 |
| 500 MHz | 799 | 112 |
| 1 GHz | 1180 | 119 |

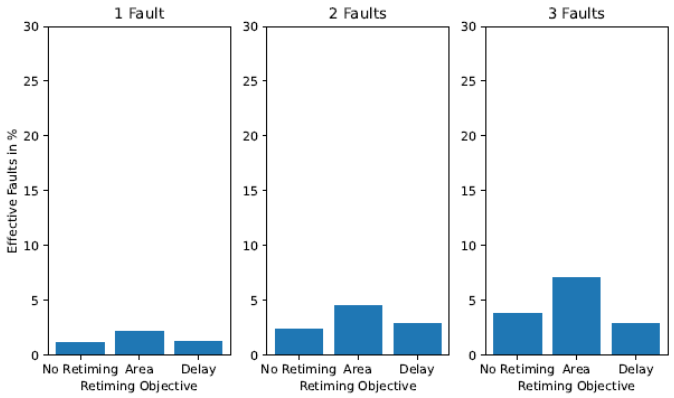
For 125, 250 and 500 MHz, the netlists seem to be identical to the previous experiment with Synopsys Design Compiler. However, for 1 GHz there is an increase in area, as well as an increase in effective faults. An explanation for this could be that for 500 MHz and below, timing constraints are easily met without further optimization, but at 1 GHz the timing constraints are too tight for the design. Consequently, the synthesis tool activates additional optimizations which weaken the countermeasure. Note that for the target clock frequency of 1 GHz the timing analysis fails as the setup time conditions fails for all our conducted experiments. Nevertheless, the synthesized netlist is functionally correct and could be clocked with a slightly lower frequency (it should be noted that delays at netlist level are only estimates, the actual timing depends on the routing during the backend design flow). We report the results for 1 GHz, as the synthesis tool activates every possible timing-driven optimization while trying to achieve this target frequency. As a result, this setting captures all effects due to tight timing constraints which might occur during synthesis.

In summary, increasing the target clock frequency might lead to inferior error detection performance as additional optimizations might be triggered to achieve tighter timing constraints. As in the preceding case studies, the influence on the countermeasure is tool- and setting-specific.

### 3.3.5 Case Study 5: Retiming Investigation

Synthesis tools such as Cadence Genus offer retiming functionality, to further optimize a design. Retiming was introduced in Section 2.2.1. Although retiming has no influence on the latency of a design, it may have significant influence on the error detection capabilities as internal registers are moved to the benefit of either area or timing. To investigate the effect of retiming on fault injection countermeasures, we synthesize the aes_cipher_control design with Cadence Genus with effort level “high” and enabled retiming. To conduct these experiments we used a slightly different version of the design with an additional register at the output of the error detection. This is necessary to ensure that the error detection circuit can be located within the netlist (see limitations of SYNFI in Section 3.2). Therefore, the results for the reference designs without retiming are not identical to the results of the previous case studies.

We conduct experiments and use retiming for different objectives, namely to optimize the design with regard to delay and to optimize the area consumption of the design. The results for this experiment (for a target frequency of 125 MHz) are shown in Figure 3.15. The area consumption is summarized in Table 3.10.




(a) Barplot with relative effective faults.

| | # Sim. Faults | Abs. # Eff. Faults | Rel. # Eff. Faults |
|---|---|---|---|
| No retiming | 1 | 2 | 1.15% |
| retiming for area | 1 | 4 | 2.13% |
| retiming for delay | 1 | 3 | 1.26% |
| No retiming | 2 | 724 | 2.41% |
| retiming for area | 2 | 1,592 | 4.56% |
| retiming for delay | 2 | 1,586 | 2.84% |
| No retiming | 3 | 95,730 | 3.76% |
| retiming for area | 3 | 228,900 | 7.12% |
| retiming for delay | 3 | 307,491 | 5.61% |

(b) Table with relative and absolute effective faults.

*Figure 3.15: Number of effective faults for 1, 2 and 3 simultaneous faults injected into a gate-level netlist synthesized with Cadence Genus (effort level "high") with different retiming objectives.*

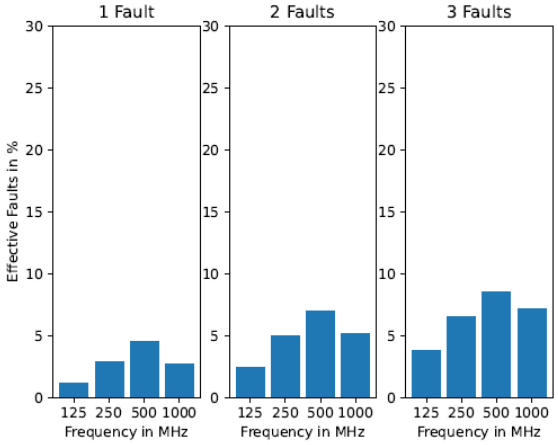
*Table 3.10: Area consumption of the complete circuit and extracted sub-circuit for the netlists generated with Cadence Genus (effort level "high").*

| ***Design*** | ***Circuit Size (NAND GE)*** | ***Target Circuit Size (NAND GE)*** |
|---|---|---|
| No Retiming | 760 | 133 |
| Retiming for Area | 773 | 139 |
| Retiming for Delay | 804 | 149 |

When observing the area results in Table 3.10 it is surprising that retiming for area leads to increased area when comparing it with the reference netlist. As in Case Studies 4 and 5 this might be due to selection of larger cells with higher driver strengths, as well as due to duplication of structures to reduce fan-outs. Another possible explanation for this is that the synthesis tool tries to balance the delays of the critical paths before optimizing for area. Further investigation is necessary to explain this behavior. When using retiming to optimize the circuit for delay we see an increase in area, as expected. As stated above, this might be due to the selection of larger cells with higher drive strengths and the duplication of structures to reduce fan-outs.

From Figure 3.15 it is evident that retiming decreases the effectiveness of the FI countermeasure. Retiming moves registers within the design and therefore changes its internal circuit structure. It is obvious that modifications applied to a countermeasure, where - among others - the redundant instantiation of registers is of extreme importance, impacts its effectiveness.

As retiming behavior might differ depending on timing constraints, we also consider clock frequencies of 125 MHz, 250 MHz, 500 MHz and 1 GHz for this analysis. The results of these experiments are summarized in Figure 3.16 for the reference design without retiming, in Figure 3.17 for retiming for area and in Figure 3.18 for retiming for delay. The respective area consumption is given in Table 3.11, Table 3.12 and Table 3.13.




(a) Barplot with relative effective faults.

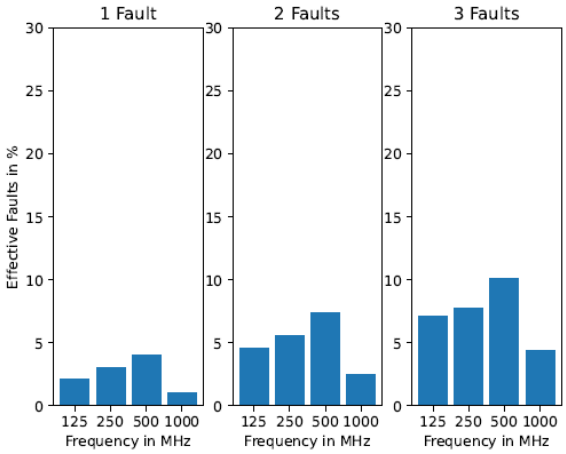
| | # Sim. Faults | Abs. # Eff. Faults | Rel. # Eff. Faults |
|---|---|---|---|
| 125 MHz - No retiming | 1 | 2 | 1.15% |
| 250 MHz - No retiming | 1 | 5 | 2.91% |
| 500 MHz - No retiming | 1 | 7 | 4.55% |
| 1 GHz - No retiming | 1 | 5 | 2.75% |
| 125 MHz - No retiming | 2 | 724 | 3.76% |
| 250 MHz - No retiming | 2 | 1,456 | 4.98% |
| 500 MHz - No retiming | 2 | 1,638 | 7.01% |
| 1 GHz - No retiming | 2 | 1,682 | 5.13% |
| 125 MHz - No retiming | 3 | 95,730 | 3.76% |
| 250 MHz - No retiming | 3 | 161,094 | 6.55% |
| 500 MHz - No retiming | 3 | 149,274 | 8.54% |
| 1 GHz - No retiming | 3 | 208,320 | 7.16% |

(b) Table with relative and absolute effective faults.

*Figure 3.16: Number of effective faults for 1, 2 and 3 simultaneous faults injected into a gate-level netlist synthesized with Cadence Genus (effort level "high" and retiming disabled).*

*Table 3.11: Area consumption of the complete circuit and extracted target circuit for the netlists generated with Cadence Genus (effort level "high" and retiming disabled).*

| ***Design*** | ***Circuit Size (NAND GE)*** | ***Target Circuit Size (NAND GE)*** |
|---|---|---|
| 125 MHz - No Retiming | 760 | 133 |
| 250 MHz - No Retiming | 785 | 132 |
| 500 MHz - No Retiming | 789 | 120 |
| 1 GHz - No Retiming | 1030 | 131 |




(a) Barplot with relative effective faults.

| | # Sim. Faults | Abs. # Eff. Faults | Rel. # Eff. Faults |
|---|---|---|---|
| 125 MHz - retiming for area | 1 | 4 | 2.13% |
| 250 MHz - retiming for area | 1 | 6 | 3.06% |
| 500 MHz - retiming for area | 1 | 7 | 4.02% |
| 1 GHz - retiming for area | 1 | 2 | 1.01% |
| 125 MHz - retiming for area | 2 | 1,592 | 4.56% |
| 250 MHz - retiming for area | 2 | 2,128 | 5.60% |
| 500 MHz - retiming for area | 2 | 2,206 | 7.37% |
| 1 GHz - retiming for area | 2 | 982 | 2.53% |
| 125 MHz - retiming for area | 3 | 228,900 | 7.12% |
| 250 MHz - retiming for area | 3 | 281,997 | 7.73% |
| 500 MHz - retiming for area | 3 | 258,009 | 10.13% |
| 1 GHz - retiming for area | 3 | 164,358 | 4.38% |

(b) Table with relative and absolute effective faults.

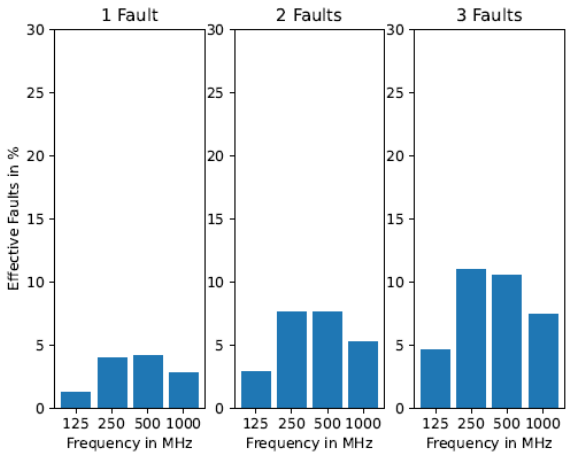
*Figure 3.17: Number of effective faults for 1, 2 and 3 simultaneous faults injected into a gate-level netlist synthesized with Cadence Genus (effort level "high" and retiming for area).*

*Table 3.12: Area consumption of the complete circuit and extracted target circuit for the netlists generated with Cadence Genus and retiming for area (effort level "high").*

| ***Design*** | ***Circuit Size (NAND GE)*** | ***Target Circuit Size (NAND GE)*** |
|---|---|---|
| 125 MHz – Retiming for Area | 773 | 139 |
| 250 MHz – Retiming for Area | 779 | 144 |
| 500 MHz – Retiming for Area | 815 | 133 |
| 1 GHz – Retiming for Area | 893 | 121 |

*Table 3.13: Area consumption of the complete circuit and extracted target circuit for the netlists generated with Cadence Genus and retiming for delay (effort level "high").*

| ***Design*** | ***Circuit Size (NAND GE)*** | ***Target Circuit Size (NAND GE)*** |
|---|---|---|
| 125 MHz – Retiming for Delay | 804 | 149 |
| 250 MHz – Retiming for Delay | 805 | 155 |
| 500 MHz – Retiming for Delay | 857 | 146 |
| 1 GHz – Retiming for Delay | 980 | 156 |




(a) Barplot with relative effective faults.

| | # Sim. Faults | Abs. # Eff. Faults | Rel. # Eff. Faults |
|---|---|---|---|
| 125 MHz - retiming for delay | 1 | 3 | 1.26% |
| 250 MHz - retiming for delay | 1 | 9 | 3.95% |
| 500 MHz - retiming for delay | 1 | 8 | 4.15% |
| 1 GHz - retiming for delay | 1 | 6 | 2.76% |
| 125 MHz - retiming for delay | 2 | 1,586 | 2.84% |
| 250 MHz - retiming for delay | 2 | 3,928 | 7.62% |
| 500 MHz - retiming for delay | 2 | 2,800 | 7.6% |
| 1 GHz - retiming for delay | 2 | 2,436 | 5.21% |
| 125 MHz - retiming for delay | 3 | 307,491 | 5.61% |
| 250 MHz - retiming for delay | 3 | 635,598 | 11.0% |
| 500 MHz - retiming for delay | 3 | 365,313 | 10.5% |
| 1 GHz - retiming for delay | 3 | 367,983 | 7.3% |

(b) Table with relative and absolute effective faults.

*Figure 3.18: Number of effective faults for 1, 2 and 3 simultaneous faults injected into a gate-level netlist synthesized with Cadence Genus (effort level "high" and retiming for delay).*

The results from these experiments demonstrate that timing constraints are an important factor when applying retiming to a design. There is a coarse tendency for more effective fault combinations when the timing constraints get tighter. However, at a certain point this relationship seems to dissolve and we observe a drop in effective faults. In case of retiming for area this point occurs at 1 GHz. When applying retiming for delay this point is reached at 500 MHz. Due to the vast parameter space and the uncertainty about the inner workings of commercial tooling, we are not able to explain this behavior.

From Table 3.12 and Table 3.13 we see that a higher target frequency leads to higher area consumption of the complete circuit. As in the case study in Section 3.3.4 this might be due to the selection of larger cells with higher drive strengths due to the duplication of structures to reduce fan-outs.

In summary, retiming impacts the effectiveness of FI countermeasures. Further, timing constraints seem to play an important role when applying retiming. As these two synthesis attributes are inherently intertwined, it is complex to thoroughly explain the reasons behind our observations. A more detailed analysis would require a huge amount of reverse engineering and is out of scope for this report.

### 3.3.6 Case Study 6: Effect of RTL Description on Fault Detection

Even though we first set out to investigate the influence of synthesis tools on a RTL design, we conduct this case study to provide more insights into the inner workings of synthesis tools and their optimizations. When considering the results of the case study in Section 3.3.3, one might notice that the effective faults in the netlists shown in Figure 3.9 can be avoided by exchanging the NAND gate with a XOR gate. Therefore, this case study investigates if we can construct such an error correction circuit through changes in the RTL description.

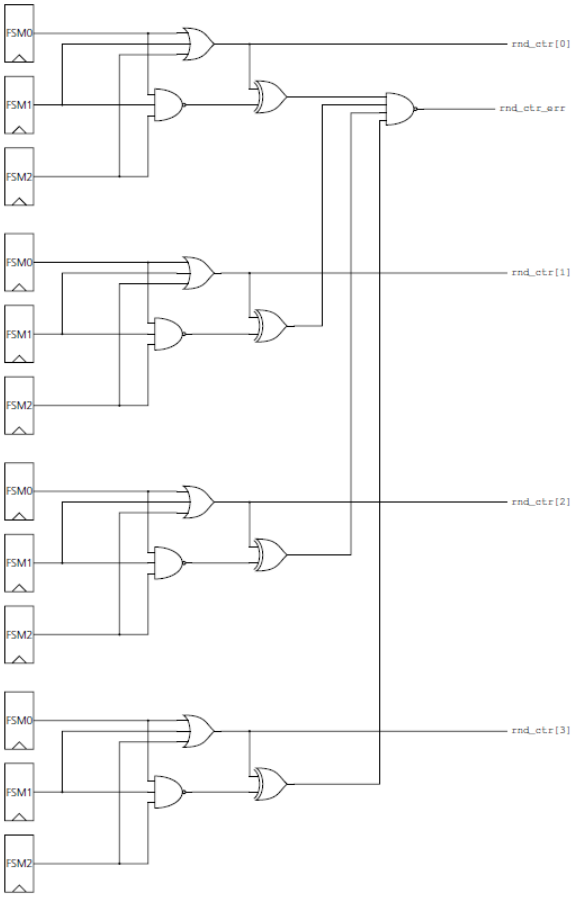



*Figure 3.19: Handcrafted error detection circuit implemented in HDL, to investigate the impact of RTL descriptions.*

When analyzing Figure 3.9 in more detail, one can see that replacing the NAND gates which feed the four inputs of rnd_ctr_err NAND gate with XOR gates fixes the flaw of not detecting 1 → 0 faults. We analyze if RTL changes avoid this flaw.

As a first step we implement an RTL description of the error detection as shown in Figure 3.19. The RTL description uses boolean equations, thus manually replacing NAND gates with XOR gates as described above. When synthesizing this design, the actual gate-level netlist is indeed composed in a similar way as the circuit in Figure 3.19.

Additionally, we investigate adding information redundancy to this circuit. To be more specific, we investigate the aes_cipher_control module at commit 097521294[4]. This version of the aes_cipher_control module implements three redundant counters similar to Figure 3.3. However, there are some important deviations from the structure depicted there.

- Each FSM contains two counters: one up-counting counter, i.e. its value corresponds to the actual AES round and one counter counting downwards, i.e. its value corresponds to the number of remaining rounds. The sum of both should always be equal to the total number of AES rounds.
- The FSMs only comprise the combinational logic for the counter and do not include a register for these counters. In each FSM, the upwards and downwards counting round counter values are updated. The combination of the round counter values from the FSMs is fed into a register outside the FSMs (see Combination block in Figure 3.20).
- For the combined round counter value, a parity bit is generated before it is fed into the register within the combination block. This parity bit is also fed into a register. The parity information of the round counter value at the output of the register is subsequently compared to the registered parity bit.

[4] https://github.com/lowRISC/opentitan/commit/097521294cd43a3e059bed8c0cd2a710b4f7f73e

- The detection block raises an alert if the sum of the two registered counters does not add up or if the parity bit is incorrect.

A simplified diagram of this circuit is depicted in Figure 3.20.

A comparison with regards to error detection performance of the different RTL countermeasure descriptions is given in Figure 3.21. Table 3.14 shows the respective area consumptions. As reference, the design without restrictions on cells from the case study in Section 3.3.3 is used. In particular, the fact that the parity bit is stored in a register seems to prohibit potentially harmful optimizations. Further, the combination of redundant instances and information redundancy yields better results. However, it is important to consider the isolated scope of this case study. For one, the protection of an AES round counter cannot be generalized for arbitrary designs. Second, this case study did only investigate simple synthesis settings. More advanced optimizations such as retiming could move registers and thus potentially optimize the parity logic. Still, we deemed it worthwhile to consider the impact of different RTL descriptions. At the least, this case study shows that there are major differences between error detection performance one would not necessarily expect, when analyzing the high-level HDL code. In accordance to all previous case studies, this emphasizes once more the complexity and inexplicability of synthesis tools.

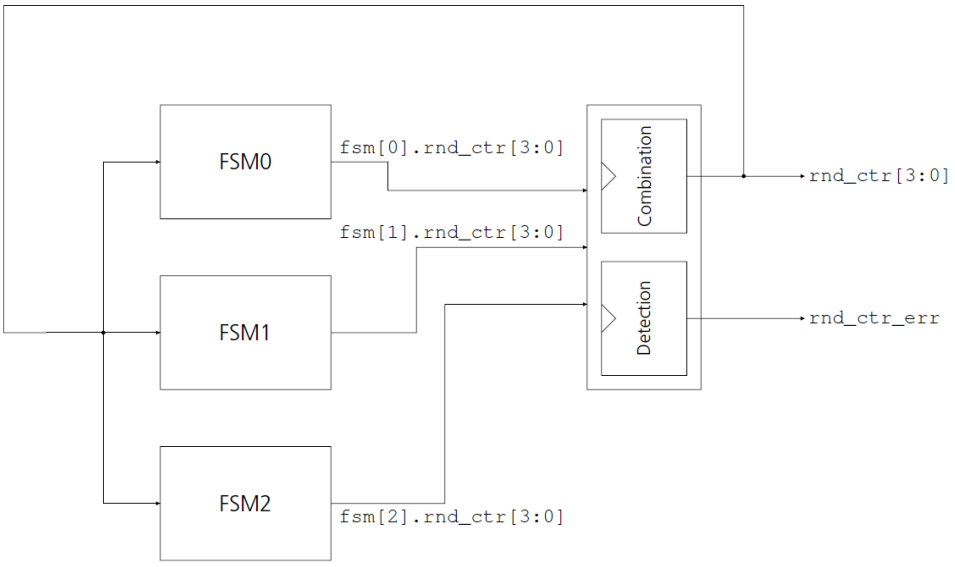
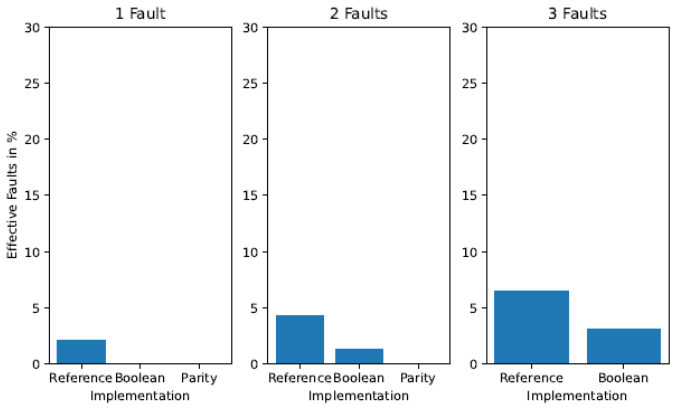



*Figure 3.20: Target circuit comprising three redundant FSMs. The "Combination" block combines the up- and down-counting counters from the FSMs. The "Detection" block implements the error detection by checking parity information and comparing the sum of up- and down-counting counters with the number of AES rounds.*

*Table 3.14: Area consumption of the complete circuit and extracted target circuit for the netlists synthesized with Synopsys Design Compiler for different RTL countermeasure descriptions.*

| ***Design*** | ***Circuit Size (NAND GE)*** | ***Target Circuit Size (NAND GE)*** |
|---|---|---|
| Reference | 1100 | 130 |
| Boolean | 1100 | 135 |
| Parity | 1540 | 154 |




(a) Barplot with relative effective faults.

| | # Sim. Faults | Abs. # Eff. Faults | Rel. # Eff. Faults |
|---|---|---|---|
| Reference | 1 | 4 | 2.08% |
| Boolean | 1 | 0 | 0.0% |
| Parity | 1 | 0 | 0.0% |
| Reference | 2 | 1,596 | 4.33% |
| Boolean | 2 | 510 | 1.37% |
| Parity | 2 | 195 | 0.07% |
| Reference | 3 | 222,660 | 6.52% |
| Boolean | 3 | 120,948 | 3.03% |
| Parity | 3 | - | - |

(b) Table with relative and absolute effective faults.

*Figure 3.21: Number of effective faults for 1, 2 and 3 simultaneous faults injected into a gate-level netlist synthesized with Synopsys Design Compiler for different RTL countermeasure descriptions.*

# 4 Masking

For this chapter, we assume that the reader has a basic knowledge of masking, as, for example, conveyed in Section 2.1.4 and the literature cited there. Section 4.1 then provides a deeper insight into the design and verification of masked circuits and related literature. In particular, the design and composition of modules at RTL level and the verification at netlist level - so after RTL synthesis - is discussed. Section 4.2 is dedicated to literature on the security evaluation of masked designs, specifically a work by Bronchain and Standaert, where a masked implementation is broken with surprisingly few traces [11]. Section 4.3.1 to 4.3.3 describe our three case studies on the security of masked designs. For our case studies, we focus on RTL synthesis, so any effects during backend design are not considered. Chapter 5 gives a short overview over the effects that might occur at this level and related literature.

## 4.1 Design and Verification of Masked Circuits

In Section 2.1.4, the *robust probing model* was introduced as a solution to formally prove whether a masked circuit withstands attacks from certain adversarial models. Further, domain-oriented masking (DOM) and threshold implementations (TI) were introduced as blueprints that allow to describe such masked circuits. Based on these seminal works, researchers tried to further optimize the construction of masked circuits and verification within the robust probing model.

**Interference Notions** The robust probing model provides the means to check the security of a synthesized netlist within certain adversarial models. This model certainly does not cover all aspects of a realistic adversary who can - by means of horizontal attacks - easily acquire thousands to millions of probes. Recall, that a probe in this context refers to a single probed value at time $t$ to which multiple wires, and - for random probing models - noise, contribute. However, researchers agree that it is the best method to date to formally verify the robustness of a masked circuit from just the netlist.

While $d^{th}$-order robust probing security is a great metric for a standalone circuit, it has no meaning when it comes to the composability of two circuits, which are both also $d^{th}$ -order robust probing secure on their own [24]. For this purpose, the notion of *probe propagation* and *simulatability* were introduced [7]. Within these terms the concept that a probed value in one circuit might expand to probed values in the preceding circuit was formalized. These sub-circuits are often referred to as *gadgets*. From the simulation, interference notions such as *non-interference* (NI), *strong non-interference* (SNI) [7], and *probe-isolating non-interference* (PINI) [14] were derived.

**Verification** The probing models and interference notions that were proposed in recent years allow to verify synthesized netlists for unwanted share recombination through glitches and other effects. However, the verification of real-world circuits and gadgets is not trivial and is still under active research. Over the last years, multiple verification tools were proposed. The approaches differ. Some were designed to verify software implementations [5], others were developed for hardware designs [3] [9] [16] [46] [56]. Even the co-verification of hardware and software was considered [36]. Further, the tools differ in their verification approach, representation of the masked implementation, and probing models. Some tools also combine automatic construction with verification [47].

**Randomness Optimizations** Masked circuits should be optimized regarding the same metrics as any circuit: area, latency, throughput, power consumption and so on. Additionally, the amount of fresh random bits that are required is an important metric, which, in turn, translates to the metrics listed above. This is because the amount of random data defines the complexity of the PRNG and the TRNG that provides its initial seed. Therefore, researchers put a lot of effort into reducing the needs for fresh entropy [26] [34] [44] [67].

For further information, the reader is referred to a recent systematization-of-knowledge paper on circuit masking by Covic et al. [25].

## 4.2 Security Evaluation of Masked Implementations

The previous section introduced state-of-the-art research on the design and verification of masked circuits. How the security of masked implementations is evaluated by the responsible agencies in practice often seems disconnected from this research. In part, this is because security evaluations are done on the manufactured IC, which makes sense, as effects during the backend design might make masking more prone to attacks, as we will discuss in Chapter 5. The introduced concepts from Section 4.1, however, focus on the synthesized netlist, which is only the outcome of the frontend design of a circuit. Further, most of the research presented above was published in recent years, and thus could not yet be considered for security evaluations.

Instead, security evaluations often focus on standardized laboratory evaluations such as test vector leakage assessment (TVLA) [71]. The gap between these methods and the formal approaches in recent works caught the attention of researchers. In [11], Bronchain and Standaert investigate a masked AES software implementation that was previously investigated by ANSSI. In a preliminary leakage assessment, following a standardized laboratory procedure with 100,000 traces, no first-order leakage was detected. However, the authors were able to recover the secret key with less than 2,000 traces. They did not target first-order leakage, but instead recovered each share separately. Usually, the underlying assumption in masking is that the noise increases exponentially with the number of shares [65]. The authors exploited that the noise of the microcontroller itself is rather low, i.e., information on each share could be recovered with a high reliability. Further, the authors demonstrated, that a probing model, once configured with the actual noise levels, is able to predict such weaknesses and model the cost of an attack realistically. This shows a potential advantage of formal verification in comparison to heuristic laboratory evaluation. The low noise level on microcontrollers and the resulting weakness of software masking, even when using higher order masking with five or more shares, is subject of ongoing research. As countermeasure to this predicament, prime field masking was proposed [18].

It is important to note that, depending on the evaluating agency and laboratory, more evaluations than TVLA are conducted. Thorough investigation includes key-recovery attacks at higher orders than the design is protected. This should have also caught the weaknesses in the discussed AES implementation, specifically for the microcontroller used in [11]. While a formal analysis in a probing model is simpler and less time-consuming than running an actual attack, it arguably does not replace it. However, such pre-laboratory evaluations can limit the laboratory work to designs that are deemed secure in these models and thus streamline laboratory efforts.

## 4.3 Case Studies

The previous sections introduced the state-of-the-art on circuit masking and the gap between this area of research and actual security evaluation. The work by [11], discussed in Section 4.2, demonstrated that formal approaches have a potential advantage compared to standardized leakage assessment. The major takeaway, however, should be the difficulty of judging the effectiveness of a countermeasure during a security evaluation. That is, even without considering the black-box behavior of EDA tools and their impact on SCA countermeasures, it is hard to assess the resilience of masked designs. To this end, we provide three case studies to investigate the impact of hardware design synthesis on the masking countermeasure. After studying our experiments, a designer should have a more thorough understanding if and how the effectiveness of masking can be evaluated before and after synthesis.

### 4.3.1 Case Study 1: Re-timing Investigation of an AES S-box

The introduction of DOM, TI, and the glitch- and delay-extended probing models in Section 2.1.4 made obvious that masked circuits require carefully designed synchronization stages to stop probe propagation. Typically, standard synchronous flip-flops are used for this synchronization. During hardware design synthesis, retiming (see Section 2.2) is allowed to move registers. The rationale behind retiming a circuit to

achieve higher frequencies or lower chip area was laid out in this preliminary section. In this case study, we investigate the effect of retiming on an AES S-box with DOM.

**RTL Design** The RTL design used as basis for this case study is an AES S-box from the OpenTitan's[5] AES core, with first-order DOM. The design is based-on and functionally identical to the S-box design in the seminal paper on DOM by Gross et al. [38]. Figure 4.1 shows the design from the original paper. $A_x$ and $B_x$ are the two 8-bit shares of the input $x$. Respectively, the output consists of the two 8-bit shares $A_y$ and $B_y$. The S-box consists of five stages, separated and synchronized by registers. The design in the OpenTitan is not pipelined and optional stages[6] are not implemented. In order to break down the AES S-box operation $S(x) = x^{-1} + 0x63$, where $x \in GF(2^8)$, into multiple operations in the sub-fields $GF(2^4)$ and $GF(2^2)$, Canright's approach is used [13]. For multiplication in $GF(2^4)$ and $GF(2^2)$, DOM multipliers are used. The registers in these multipliers - needed to synchronize shares and provide fresh randomness - make up the five register stages.

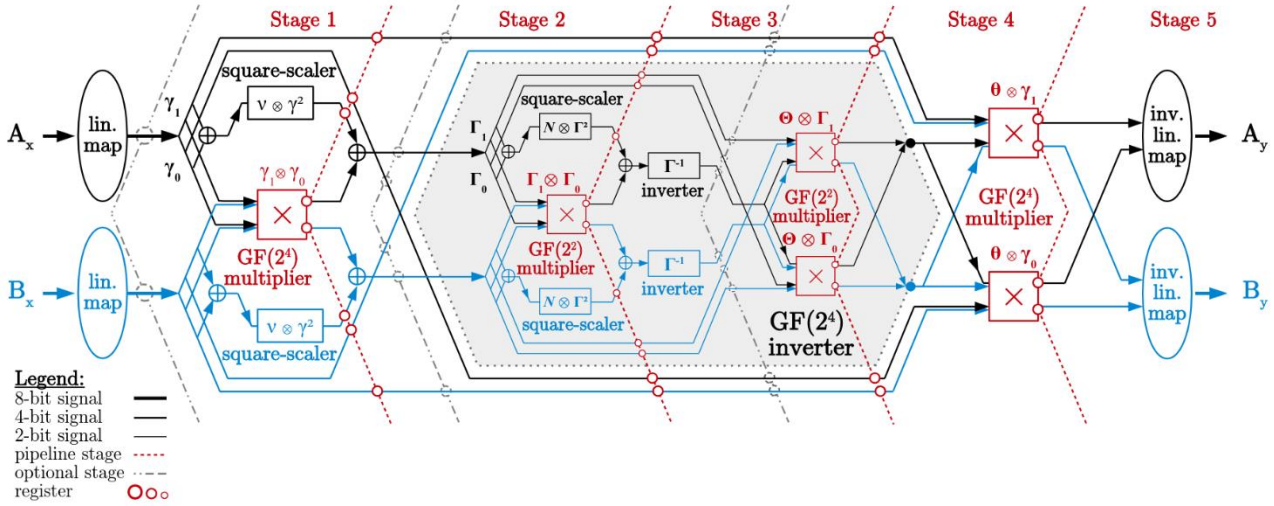



*Figure 4.1: AES DOM S-box from [38]. $A_x$ and $B_x$ are the two shares of the input x, s.t. $x = A_x + B_x$, respectively $A_y$ and $B_y$, are the two output shares.*

**Synthesis** For synthesis, we use Cadence Genus and Yosys. Genus is one of the most relevant commercial synthesis tools available (Section 2.2). It allows to explicitly retime a design with different strategies, e.g. to achieve maximum frequency, or to keep the chip area as low as possible. Per default, a design is not retimed. For Yosys, retiming is more difficult to implement. Internally Yosys uses the abc tool[7] by the Berkeley Logic Synthesis and Verification Group to optimize and retime the design. If Yosys is not explicitly configured to use a custom script, it applies a default abc script, which includes the *dretime* command. No retiming strategy to prioritize either area consumption or critical path length can be configured. This is only possible by ordering abc commands, such that the desired optimization is given priority. It is important to note that the *dretime* command is only effective if abc is allowed to optimize registers. For this, abc must be called with the *-dff* option. Further, abc only recognizes flip-flops from Yosys's internal cell library. If Yosys maps flip-flops to technology specific instances via *dfflibmap* before abc is invoked, registers will not be retimed. Finally - depending on the design and its hierarchies - it must be flattened, to enable retiming.

**Netlist Verification** For verification we use the SILVER tool, developed at Ruhr University Bochum [46]. It consists of a Verilog parser that processes gate-level netlists, the output of RTL synthesis. For synthesis, any desired combination of EDA toolchain and PDK can be used. SILVER only needs a simple wrapper per technology-dependent PDK to understand the boolean logic function of the proprietary gates. A reduced-order binary decision diagram (ROBDD) and the statistical independence of probability distributions is used to verify probing security and interference notions. SILVER supports verification regarding probing security, robust probing security, non-interference (NI), strong non-interference (SNI), and probe-isolating

[5] https://github.com/lowRISC/opentitan
[6] Can be used to reduce the length of the critical path and accordingly increase the maximum frequency.
[7] https://people.eecs.berkeley.edu/~alanmi/abc/

non-interference (PINI). The robust probing model within the tool considers glitches and transitions [56] (see Section 2.1.4 for more details on these concepts). The coupling effect is omitted, as a netlist does not contain information on the placement of cells and wires.

**SILVER - Usage and Limitations** A user needs to manually annotate a netlist to convey to SILVER the purposes of the ports of the top-level design. SILVER distinguishes ports into: clock, secret sharing, fresh randomness and control ports for both inputs and outputs. Two major limitations became obvious, when running synthesis and verification experiments.

For one, the tool struggles to handle control logic. As mentioned above, the OpenTitan DOM S-box was used as basis for experiments. This design uses flip-flops with an enable signal and control logic that enables the flip-flop only for the stage, when the register is set to a valid intermediate value. For example, registers in the fourth stage are disabled for all but the respective clock cycle. Their value changes only for this stage. Parsing this control logic with the SILVER tool resulted in an internal error. The tool uses a custom representation of the netlist which is the output of the internal parser. This representation is a simple text-based annotation of the data flow through a design. Converting complex control flows to such a simple representation is challenging. It should be mentioned that a missing intermediate representation in hardware design tooling is a general issue. Research tooling such as the SILVER tool, but also other tools with a completely different scope, would profit massively from a unified, intermediate representation. For this case study, irrelevant control logic was stripped from the design, such that the design could be parsed. Regarding verification in a probing model, this does not change the outcome. However, if the tool were to be used regularly in a commercial design process, this would introduce both a massive overhead and a critical point for errors.

Second, the SILVER tool needs a description for each standard cell in the netlist. This is due to the fact that SILVER can verify netlists synthesized for arbitrary design kits. The mapping between the standard cell's name and its functionality is provided as a simple textual description. For practical adoption, this is a nice feature, as it allows to use arbitrary PDKs in the design flow. However, more complex cells (e.g. flip-flops with two outputs, the current state and its inverse) would require modifications in the tool's source code and intermediate representation. Yosys's internal cell library is supported completely, for other design kits, the synthesis tool must be configured to not use non-supported cells. This workaround was chosen for the synthesis with the FreePDK45[8] in this case study.

**Results - Utilization** Table 4.1 shows the effect retiming has on the synthesis with Cadence Genus and Yosys. For Genus, the default retiming strategy was chosen. As stated above, the retiming configuration in Yosys is more difficult. Since no custom optimizations before retiming were carried out and abc was invoked with standard parameters, the chosen strategy can also be seen as default. The logic optimizer abc then uses retiming to balance a small gate count with a shallow circuit depth.

Originally, the S-box needs 92 flip-flops. Cadence Genus and Yosys arrive at a similar area utilization. The synthesis with Genus includes power and timing estimations. The target frequency was set to 500 MHz, which was easily met. For Yosys, power and timing estimations would have required additional plugins. The reports by Genus show that retiming increases the length of the critical path marginally. The power consumption decreases due to a reduced number of gates, in particular flip-flops.

*Table 4.1: Number of flip-flops, area, estimated critical path length and estimated power consumption for a DOM S-box synthesized with and without retiming. All results are for the FreePDK45 cell library. Yosys does not directly report critical path length and power consumption. For Genus, the target frequency was set to 500 MHz.*

| *Design* | *No. of D-Flip-Flops* | *Area (NAND GE)* | *Critical path length (ns)* | *Power (mW)* |
|---|---|---|---|---|
| Genus without retiming | 92 | 1649 | 1.44 | 1.70 |
| Genus with retiming | 75 | 1550 | 1.62 | 1.63 |

[8] https://github.com/mflowgen/freepdk-45nm

| *Design* | *No. of D-Flip-Flops* | *Area (NAND GE)* | *Critical path length (ns)* | *Power (mW)* |
|---|---|---|---|---|
| Yosys without retiming | 92 | 1659 | - | - |
| Yosys with retiming | 79 | 1566 | - | - |

**Retiming a DOM multiplier** The difference between the synthesis with and without retiming shows that both Cadence Genus and Yosys see a significant potential for optimizations in the DOM S-box. When looking at the structure in Figure 4.1 it is not directly obvious, how retiming affects the DOM S-box. Therefore, in the following, an elementary DOM multiplier as shown in Figure 4.2 is used to demonstrate optimizations and vulnerabilities caused by retiming. The domain property in DOM requires domains to be strictly separated and registers to be placed after a gadget's resharing logic. The domains are denoted by the shares *A* and *B*. Figure 4.2 shows that DOM multipliers need a resharing step after calculation of non-linear functions, to prevent first-order leakage due to the subsequent integration that combines the two intermediate results to two output shares. Further, to prevent such a recombination through temporary glitches, it needs to be ensured that the fresh mask ($Z_0$ in Figure 4.2) is valid and propagates in sync with the calculation's outputs. This is achieved by means of a flip-flop placed after resharing. Therefore, the multiplier is only secure in a robust probing model, if synthesized exactly as in Figure 4.2. From a designer's perspective, it would be desirable to merge the two flip-flops into the $Z_0$ path, such that only one flip-flop is required (see Figure 2.3). This does not change the behavior of the circuit, but it would violate robust probing security. If the registers were to be merged into the $Z_0$ path, a glitch-extended probe placed on $A_q$ would include $[A_x, A_y, A_xA_y, B_y, Z_0, ...]$ (see Section 2.1.4 for the concept of glitch-extended probes). Thus, this one probe suffices to recover the secret input $y$ from $A_y + B_y$. If the register is placed as in Figure 4.2, the probe would only include $[A_x, A_y, A_xA_y,(A_xB_y)+Z_0]$, which does not allow to recover information on $x$ or $y$.

Since both Genus and Yosys reduce the number of flip-flops, which are carefully placed in the DOM S-box (Figure 4.1), it is almost guaranteed that the retimed design is not robust probing secure.

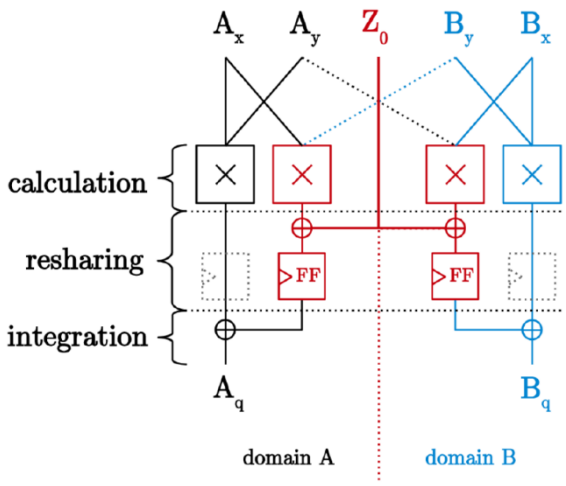



*Figure 4.2: First-order DOM-indep multiplier from [38]. It computes a masked AND, s.t. $x \times y = (A_x + B_x) \times (A_y + B_y)$, i.e. $A_x$ and $B_x$ are the two shares of $x$ and $A_y$ and $B_y$ are the two shares of $y$. The dotted registers are only required if used in a pipelined implementation.*

**Results - Verification** Running a verification with the SILVER tool confirms this suspicion. For the design synthesized without retiming, the tool reports both probing and robust probing security. It is noteworthy that the verification for simple probing security takes less than 10 seconds, whereas the verification for robust probing security needs more than 36 hours[9]. The verification of the retimed design yields that it is

[9] On a workstation with AMD Ryzen 9 3950X 16-Core Processor, multi-threading enabled, 128 GB of RAM

probing secure, but not robust probing secure. SILVER reports one probe, where robust probing security does not hold (found after less than 10 seconds) and then terminates the analysis.

**Analyzing a retimed Design** The output by SILVER provides a starting point for analyzing how the design was retimed and what possible vulnerabilities are caused by it. Within a netlist, SILVER reports the wire or port where (robust) probing security is violated. Since netlists are difficult to interpret manually, this cannot be directly mapped to a high-level representation of the circuit such as the structure in Figure 4.1 or an RTL description. Therefore, we conducted a thorough analysis. We limit this investigation to the netlist created by Genus, due to two reasons. For one, it is the smaller design. Second, Genus allows to trace retimed registers, such that every register that is introduced due to retiming is named according to a custom convention and the tool keeps track of which registers in the original design contributed to the placement of a retimed register.

Even with this setup, reverse engineering the retimed netlist to a comprehensible format is a challenging task. Therefore, Figure 4.3 shows only partially which registers were modified by retiming, in the $A_x$ path and only for Stage 1 and Stage 2. From the Genus reports it is obvious, that the fourth and final register stage stays unmodified. At this stage, two 8-bit registers are required. Since this is the width of the output shares, no instances can be saved and the tool does not seem to detect any timing advantages by moving the registers. The question-mark in Figure 4.3 illustrates that the exact placement of registers is unclear. It is important to note that the structure in Figure 4.1 does not match entirely the OpenTitan's S-box implementation. The top- and bottom-most signals can also be realized with one register, if the control logic ensures that it holds the correct input value up until stage four. Therefore, the respective registers were crossed out in black. It is also important to note that Figure 4.1 does not show fresh randomness needed for the computation. The registers in which it is stored are partially retimed, this is shown in Figure 4.3 (see prd input and register).

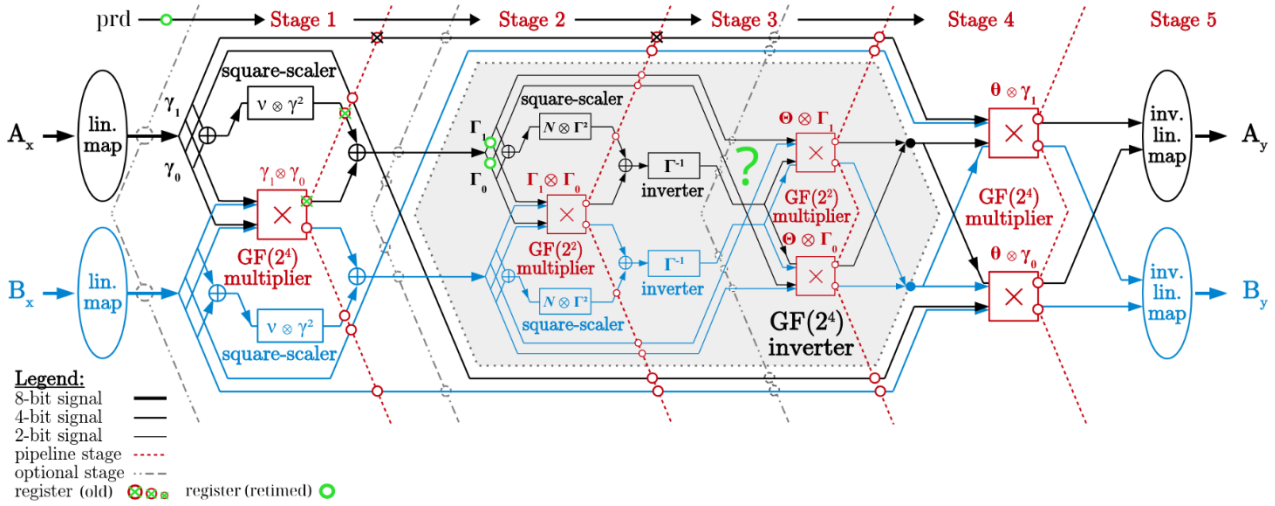



*Figure 4.3: AES DOM S-box from [38] after retiming with Cadence Genus.*

The first register stage is moved behind the XOR that combines the square-scaler's and GF($2^4$) outputs. It was found that the (extended) probe reported by SILVER as one location, where first-order robust probing security does not hold, corresponds to wires after where the register should be placed. The GF($2^4$) multiplier is a non-linear permutation for which both domains $A$ and $B$ are needed. It is implemented with a more general version of the DOM multiplier shown in Figure 4.2. For this simple DOM multiplier, an exemplary probing security violation was given above. Considering this example, it is obvious that the GF($2^4$) multiplier is not robust probing secure. Without the registers, a glitch-extended probe on one of the output shares includes all input shares. In the following, the robust probing violation is discussed in more detail and evaluated in a laboratory setup.

**Laboratory Analysis** For a detailed analysis, an excerpt of the masked S-box from Figure 4.1 is shown in Figure 4.4. It depicts the inputs and the GF($2^4$) multiplier in the S-box's first stage. For more details on the S-box implementation, the reader is referred to [38]. As stated above, it follows Canright's approach [13], by

expressing GF($2^8$) elements as multiple elements in lower sub-fields. The initial linear map creates the corresponding mapping. The following GF($2^4$) multiplier is a version of a DOM multiplier, adapted to multiply within a Galois field. In contrast to the simple design shown in Figure 4.2, it does not require the inputs to be independently shared. This refers to the composability of DOM multipliers, however it is not relevant for the subsequent analysis.

If the S-box is synthesized with retiming, the registers are not placed as in Figure 4.4, but they are merged with registers in the square-scaler output as depicted in Figure 4.3. As stated above, it is obvious that the design is no longer robust probing secure. Figure 4.4 shows this in detail. Without the registers, a glitch-extended probe on the outputs of the final XOR gates includes bits of both shares $A_x$ and $B_x$ due to the linear mapping. Related work such as [54] shows that a glitch-extended, i.e. robust, probing model is required and security in the simple probing model is not sufficient to prevent leakage. In the following, a more thorough analysis to what extent their findings apply to retimed designs is conducted.

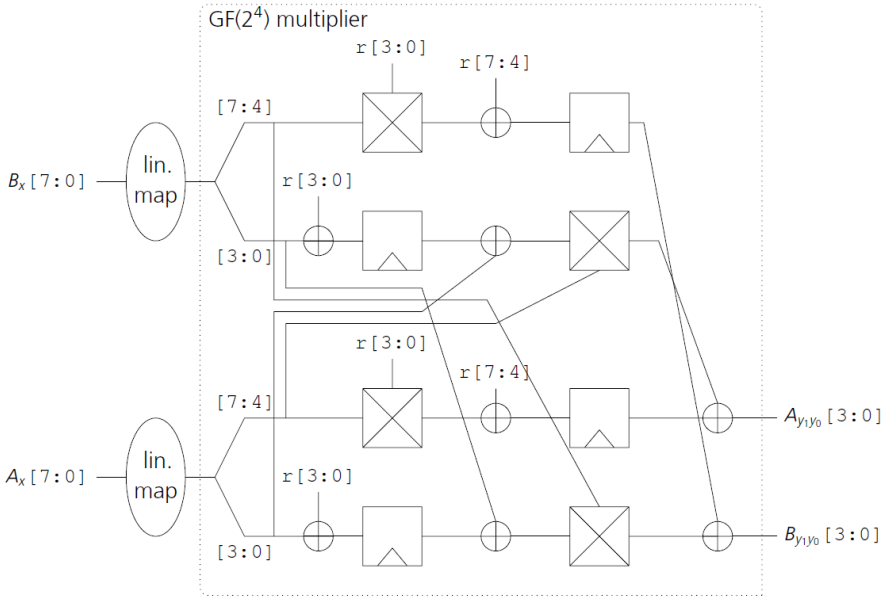



*Figure 4.4: Excerpt of the AES DOM S-box showing the input shares of x, $A_x$ and $B_x$ , the linear input map, the Galois field GF($2^4$) multiplier and its outputs $A_{y1y0}$ and $B_{y1y0}$. Fresh randomness is provided by r. The crossed rectangles correspond to an unmasked multiplication in a Galois field GF($2^4$) without registers, consisting of three multiplications and two scalings in GF($2^2$).*

**Target Platform** The laboratory investigation is conducted with the ChipWhisperer CW310 Bergen Board[10]. It features a Kintex FPGA target and is a supported platform for the OpenTitan project. It should be noted that the physical behavior of the ASIC differs from an FPGA implementation in terms of power consumption, noise, and electromagnetic emanation. However, it is common practice to evaluate designs on FPGAs, before manufacturing the integrated circuit. The production of an ASIC involves enormous effort. Measures were taken to ensure that the netlist that was synthesized and retimed by Genus maps the behavior of the FPGA implementation as closely as possible. The netlist was synthesized for a subset of the FreePDK45 design kit, to ensure that verification with SILVER is possible. For each gate, a mapping to the look-up table (LUT) architecture of the Kintex technology was created. Instead of a cell from the PDK, a single LUT that is configured to behave as this cell is instantiated. Further optimizations of this mapping are prevented with *DONT_TOUCH* and *KEEP_HIERARCHY* constraints. Our mapping ensures that FPGA specific effects that could occur by merging multiple gates into one LUT are circumvented.

[10] https://rtfm.newae.com/Targets/CW310%20Bergen%20Board/

**Laboratory Setup** Figure 4.5 shows the measurement setup. The Bergen Board features connectors to a Shunt resistor, to measure the FPGA's main powerline. We connect a ChipWhisperer CW1200 Pro capture device to this connector to record power traces. Further, we placed a Langer RF3 near-field probe[11] on the die to record electromagnetic emanation traces.

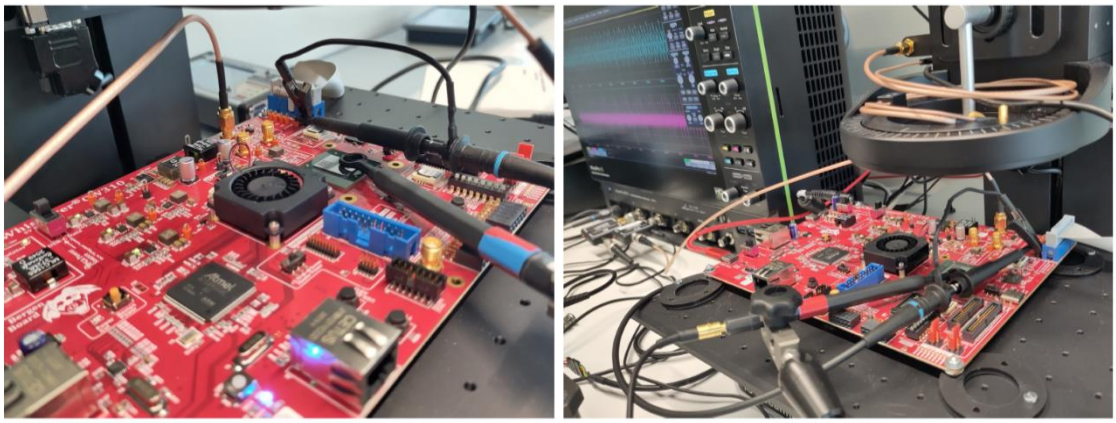


*Figure 4.5: Laboratory setup for side-channel measurements on the ChipWhisperer CW310 Bergen Board.*

**Unspecific Leakage Assessment** Laboratory assessment typically starts with an unspecific leakage assessment according to the methodology from [71]. In short, a fix-versus-random t-test can be used to check for leakage. Operations are conducted at random order, either with a fixed or random operand. In this case, either a fixed or a random key is used to encrypt a random plaintext with the AES design. Before the test, the null hypothesis is formulated: "*Traces recorded with a fixed key and traces recorded with a random key belong to the same set*". The t-values can be calculated according to [71]. Usually $|t| > 4.5$ is used as threshold, where the null hypothesis can be confidently - with a probability of 0.99999 - rejected. In this case, a fixed key can be distinguished from random keys in the power trace. One can deduce that certain effects, e.g. Hamming weight or distance leakage occur, which allow to recover secret information from power traces. For a first order masked design, $|t| < 4.5$ should hold for a basic t-test. The t-test can be conducted for higher statistical orders, or in a multivariate fashion, where sample points are combined, to evaluate higher-order masked designs.

Figure 4.6 shows a mean power trace and t-traces for the S-box with and without retiming. The power trace is shown for the S-box without retiming, but looks indistinguishable from the power trace for the S-box that was synthesized with retiming. For this experiment, the ChipWhisperer Pro capture device is used. It can be purchased for less than 4,000 USD. It allows to connect its measurement circuit to the FPGA's clock, multiply it by a factor of four, and use it for sampling. The FPGA design is clocked with 100 MHz. The ten AES rounds are roughly recognizable in the mean trace. If the S-box is synthesized without retiming, the fixed-key set is not distinguishable from the random-key set. For the retimed design, the t-trace contains multiple peaks with $|t| > 4.5$. The glitches discussed above seem to cause distinct effects in the fixed-key power traces. These effects make the fixed key distinguishable from a random key for individual sample points and first-order t-values. This is a clear indication that the first-order masking scheme does not work as expected and does not provide sufficient protection.

[11] https://www.langer-emv.com/en/product/side-channel-analysis/69/rf3-mini-set-near-field-probes-30-mhz-up-to-3-ghz/855

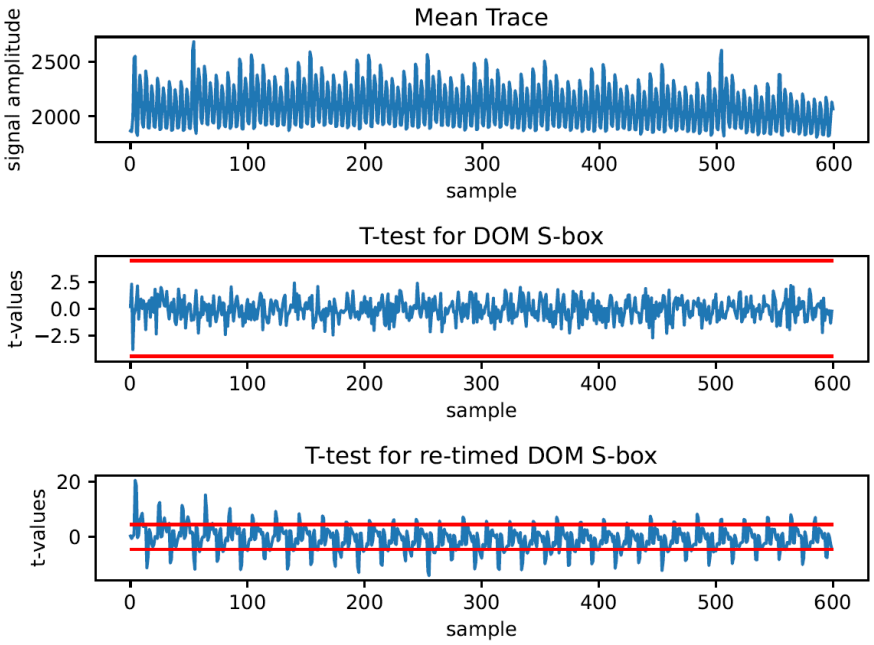



*Figure 4.6: Results of an unspecific test-vector leakage assessment of the AES DOM S-box with and without retiming for one million traces.*

**Specific Leakage Assessment** After an unspecific leakage assessment, a specific leakage assessment is conducted to establish a leakage model. Typical leakage models include Hamming weight and distance leakage. For such an investigation, the AES core is provided with random - but known - keys and plaintexts. Traces are divided into subsets according to an intermediate result and the respective leakage model, for example, the Hamming weight of a state byte at a certain round. The subsets can be composed as fix-versus-random or fix-versus-fix. Figure 4.7 shows the results of a fix-versus-random investigation. The Hamming weight of the unmasked key word going into the key schedule was selected for the fixed data set. According to the AES key schedule, the word is rotated, substituted with four parallel S-boxes, and a round constant is added. The spike in the t-trace in Figure 4.7 shows that the retimed S-box leaks information on the Hamming weight of the unmasked key. For the S-box without retiming, the same hypothesis fails to show significant leakage.

However, twenty million traces are required to detect this effect. For this experiment, the traces were recorded with a LeCroy WavePro oscilloscope with a sample rate of 1 GS/s. The FPGA's clock is set to 100 MHz. The identification of 8-bit Hamming weights within the 32-bit key word or fix-versus-fix hypotheses did not work reliably. This can be partly attributed to the parallel operation of the AES core, where key schedule and round operation are executed simultaneously. Furthermore, the actual leakage is more complex than a direct Hamming weight leakage.

The leakage of the GF($2^4$) multiplier discussed above does not directly correspond to a Hamming weight leakage of the unmasked inputs. Depending on the signal propagation, the final XOR gates in Figure 4.4 leak a pattern that depends on the unmasked inputs. The linear maps at the inputs are responsible for this, but also cause the leakage not to be directly related to the unmasked inputs. However, the linear dependency should be exploitable for an attack. To demonstrate the feasibility of such an analysis, relevant paths in the FPGA design are analyzed with Xilinx Vivado. The signals arrive at the XOR gate with differences from 0.6 to 0.1 ns. For a sampling rate of 10 GS/s and higher, such glitches can be recorded.

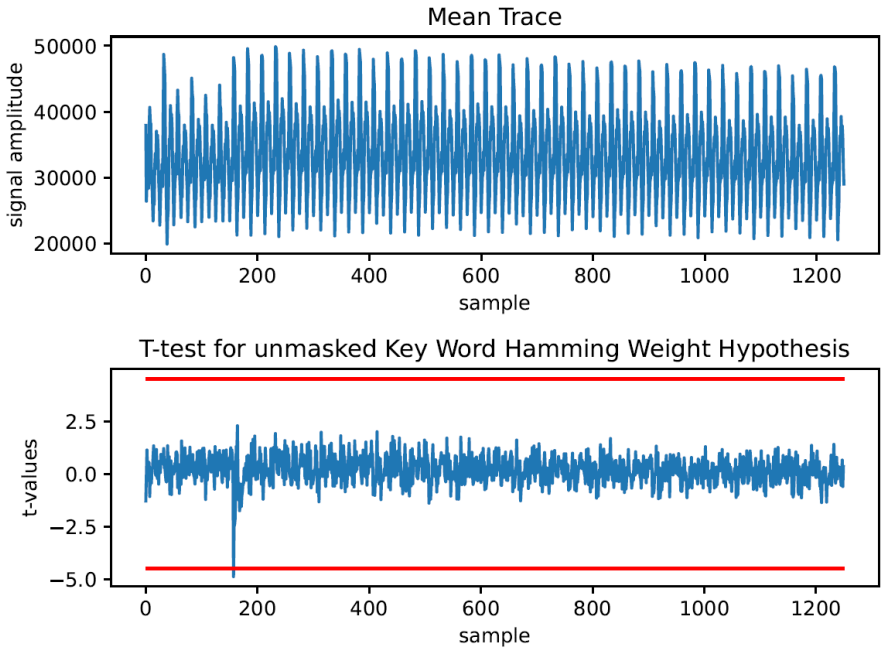



*Figure 4.7: Results of specific leakage assessment of retimed S-box for twenty million traces. The t-test distinguishes traces according to the Hamming weight of the unmasked key schedule's 32-bit input word. A fix-versus-random approach was chosen, where traces with the fixed Hamming weight of 14 were separated from all other traces.*

As part of this project, investigations toward the discussed attack approach were started. As first step, the AES's random number interface for fresh randomness was changed to use a software provided seed. This allows to investigate intermediate values and evaluate the measurability of effects in general. With the setup displayed in Figure 4.5, confirming hypotheses below 16-bit data words was not possible. Further, no glitching patterns in intermediate results could be identified. With a more advanced laboratory setup, these issues could be overcome. In particular, more localized measurements have the potential to measure the emanation of a single S-box, instead of the complete AES design. Related works such as [39][73][74] show the potential of measurements with a high spatial and temporal resolution. Attacking flawed masking schemes should work in a similar fashion.

**Conclusion** Within this case study, we established that retiming a design during synthesis could prove fatal. Using a real-world AES design, we showed that such a design is guaranteed to be modified during retiming in a way that robust probing security does no longer hold. The SILVER tool can catch such vulnerabilities, however it is not trivial to apply to real-world designs. The fact that robust probing security is needed to avoid leakage in a laboratory investigation was proven in [54]. We substantiate this with our own laboratory analysis. Further, we discuss practical attacks and undertake first steps towards such attacks.

### 4.3.2 Case Study 2: Composite Designs

The first case study revealed a fundamental problem in the verification flow of masked designs. That is, the compute intensive verification. Even verifying a single AES S-box took more than 36 hours. To this end, researchers proposed interference notions. In short, these notions allow to verify basic building blocks, so called gadgets. For more detail, the reader is referred to Section 4.1 and the papers listed there. The claim is that circuits which are composed of these gadgets, are automatically robust probing secure. This saves the effort of running a compute-expensive robust probing security verification. Within this case study, we show the benefits and pitfalls of composite designs.

**State-of-the-Art Interference Notions** The state-of-the-art interference notion is *probe-isolating non-interference* (PINI).

**Definition 4.3.1. Probe-isolating non-interference according to [14].** Let $G$ be a gadget over $d$ shares with inputs $x_{i,j}$ and outputs $y_{i,j}$, where $i \in [0, d-1]$ is the share index and $j$ the bit-index of the shared input vectors

$x$ and $y$. Further, let $P$ be a set of $t_1$ probes on wires of $G$ (called internal probes). Let $A$ be a set of $t_2$ share indices. $G$ is $t$-probe-isolating non-interfering ($t$-PINI) if and only if for all $P$ and $A$ such that $t_1 + t_2 \leq t$, there exists a set $B$ of at most $t_1$ share indices such that probes on the set of wires $P \cup y^G_{A,*}$ can be simulated with the wires $x^G_{A \cup B,*}$.

Put into simple words, PINI ensures that an adversary does not learn more than $t$ secret variables from her probes, for all possible combinations to choose the sets $P$ and $A$. By the notion of simulation, composability is ensured. If simulatability is not given, the adversary learns additional information, since the wires $x^G_{A \cup B,*}$ are no longer sufficient to simulate the probes. In this instance, depending on the gadgets that drive inputs of the gadget $G$, glitches might occur that combine secret shares. These glitches are similar to the ones caused by retiming in the previous case study (Section 4.3.1). PINI promises trivial composability at minimum overhead. The preceding notion of NI was not trivially composable. The SNI notion offered gadgets that could be placed such that arbitrary gadgets are composable, however its overhead in hardware footprint and fresh randomness is immense. PINI gadgets exist in different versions, optimized for area-efficiency, randomness consumption, throughput and latency. These gadgets are typically called hardware private circuit (HPC). They are used in tools for the automatic generation of masked hardware designs [47], and different designs [45] [53], e.g. the AES design that was the target of the 2023 CHES challenge[12].

**Composition and Verification** Figure 4.8 shows a simple example of securing an arbitrary operation by replacing basic boolean operations with the respective PINI gate. In this case, the operation corresponds to a first-order masked, i.e. two shares, calculation of $y = ((a \times b) + c) \times a$, where multiplication is a Boolean AND and addition is a boolean XOR. The trivial composition promised by PINI amounts to "connecting the output share with index $i$ always with the input share with index $i$". A violation of this is shown in Figure 4.8 in red. The twisted connection leads to a combination of the input shares $a_0$ and $a_1$ in the last AND gate. While this seems trivial to avoid, it is important that such an implementation mistake is not caught with functional tests, as the result $y$ is still correct. Dedicated verification tools such as SILVER are required to detect such a flaw. We verified that the faulty composition in Figure 4.8 is recognized by SILVER. Further, in an RTL description multiple ways to represent and index shared variables exist. Special care must be taken to compose implementations correctly. This does also apply to the composition of more complex gadgets such as Galois-field multipliers.

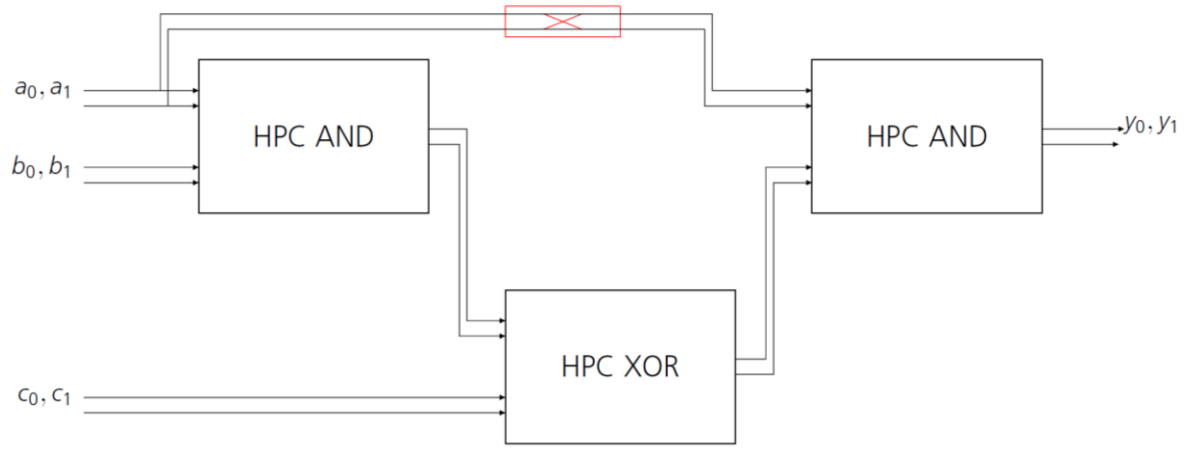



*Figure 4.8: PINI composition of masked gadgets.*

An interesting research direction is the automatic generation of masked hardware designs. In [47], the authors propose to synthesize an arbitrary design, which is annotated for its secrets, and replace each standard cell such as NAND, XOR, etc. with its HPC equivalent. Since this is done automatically, there is no

[12] https://smaesh-challenge.simple-crypto.org/

risk of incorrect composition. This approach is promising for parallel designs, but the authors of [53] report that serial designs can be massively improved by handcrafting.

Since verification of handcrafted designs is important, even when provably secure gadgets are used, the efficiency of this verification is critical. In the first case study, the SILVER tool was used, as it can verify netlists for arbitrary synthesis tools. Its ROBDD approach makes it computationally expensive. The verification of a single AES S-box is feasible, but anything larger is deemed unrealistic. The fullVerif [16] tool follows a different approach. It divides the circuit into gadgets. In a recursive fashion, each instantiated module within a gadget is also assumed to be a gadget. Once a gadget cannot be divided into sub-gadgets, it is verified for PINI. Since each gadget is verified on its own, the tool only needs to check the correct composition for the complete circuit. This allows the verification of real-world co-processors. A limitation of the tool is that it takes high-level HDL code as input and uses the open-source tool Yosys to synthesize it. Thus, a direct usage of the tool with commercial EDA tooling as allowed by SILVER is not possible. If fullVerif reports a design as secure, engineers must carefully enforce that the commercial synthesis tool does not apply optimizations such as retiming, which would make the design insecure.

We tested the fullVerif tool with the example from Figure 4.8. It correctly assesses the HPC gates to be probe-isolating non-interferent at the first order and correctly reports the composition flaw. For such a simple example it is easy to see that the netlist generated by Yosys is similar to the one generated with Cadence Genus. For more complex circuits, this is a challenging task.

**Conclusion** This case study presented interference notions and basic HPC gadgets as promising approaches to design efficient and secure hardware. A simple example emphasized that circuits must be composed correctly, which can only be detected with dedicated verification tools. The fullVerif tool uses interference notions for efficient verification, such that real-world co-processors can be investigated. In contrast to the SILVER tool, however, it cannot be used with commercial EDA tools.

### 4.3.3 Case Study 3: Higher-order Masking Effects in Synthesis

The first two case studies leave open, whether there are synthesis optimizations beyond retiming which violate the security of masked hardware designs. For this, a CMS AND gate for $2^{nd}$-order masking is analyzed.

**Second-order CMS AND** In Figure 4.9, a $2^{nd}$-order secure CMS AND gate is shown. The scheme was originally proposed for arbitrary order masking. The authors based their claim on the circular structure of the AND gate. However, the authors of [54] discovered flaws for $d > 2$. This circular structure is also investigated here. The output share $c_1$ can be written as boolean equation as follows:

$$c_1 = (a_1b_1 + R_1 + R_2) + (a_1b_2 + R_2 + R_3) + (a_1b_3 + R_3 + R_4) \tag{4.1}$$

Every term in brackets is stored in a register before the final 3-input XOR is evaluated. If the registers are ignored, the equation can be simplified to:

$$c_1 = a_1b_1 + R_1 + a_1b_2 + a_1b_3 + R_4 \tag{4.2}$$

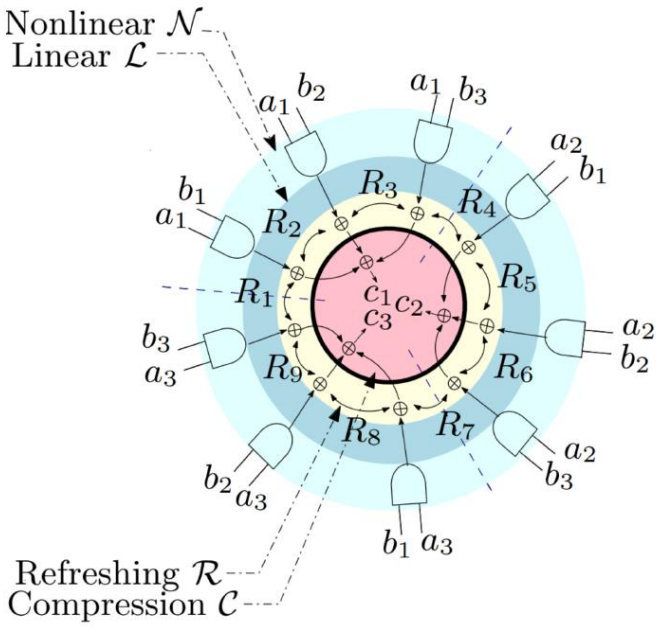



*Figure 4.9: Second-order CMS AND gate. The variables $a_i$ and $b_i$ are the input shares, where $a = a_1+a_2+a_3$, so i is the share index (analog for b). For the output $c = a \times b = c_1 + c_2 + c_3$ holds. $R_j$ denotes fresh random bits. The bold circle corresponds to a register stage.*

This is due to the fact that xor'ing a variable twice first applies and then removes the variable, such that it ultimately has no influence on the result. In contrast to Equation (4.1), this is insecure, as $a_1b_1$ and $a_1b_2$ are not sufficiently blinded. However, the two XOR gates are separated by a register. This raises the question, if optimizations on two sides of a register require retiming, or if this is not considered retiming as the location of the register does not change, only its value.

**Logic Optimization in Yosys** We use Yosys to check which optimizations are applied depending on the configuration. Since it is an open-source tool, we can access the implementation and trace its optimization features. Yosys itself applies optimizations to replace flip-flops with constant values, optimize multiplexer trees, detect redundant and unused logic, and optimize FSMs[13]. These optimizations can be triggered independently. The generic *opt* command applies them iteratively, until the result is stable. For the bulk of logic optimizations - among others retiming - the abc tool can be called from Yosys. It performs generic optimization based on an and-inverter graph (AIG) approach and directed acyclic graph (DAG) based technology mapping. The abc tool distinguishes its optimization passes into two categories, sequential and combinational. For sequential synthesis, the following transforms exist:

- **lcorr:** Detects and merges sequentially equivalent registers according to [52].
- **retime:** Different variations of retiming (most forward, most backward, minimum register, etc.).
- **scleanup:** Sequential cleanup by removing nodes and latches that do not fan-out into primary outputs.
- **ssw:** Implements signal-correspondence using K-step induction [52]. Detects and merges sequentially equivalent nodes.
- **undc, zero:** Register initialization commands.

As described in Case Study 1 (Section 4.3.1), retiming and sequential synthesis in Yosys requires in-depth knowledge of the tool. Within Yosys, abc is invoked with a default script, unless a dedicated script is specified. This script applies the *retime* command, however it can only make changes to the design, if the registers are passed to abc with the *-dff* flag and were not yet mapped to their technology-specific instances

[13] https://blog.eowyn.net/Yosys/CHAPTER_Optimize.html

with *dfflibmap*. The *lcorr* and *ssw* commands are not part of the default script. To check, if they apply optimizations across registers as described above, scripts with the respective commands were used. However, the *ssw* command is not available in Yosy's internal version of abc. The *lcorr* command did not apply the discussed optimization. Its purpose is to merge equivalent registers, not optimize the data path for one register. The *retime* command merges all three registers in the path of $c_1$, eliminating the randomness as in Equation (4.2).

**Logic Optimization in Cadence Genus** The experiment with the CMS $2^{nd}$-order AND gate was repeated with Cadence Genus. As Genus is a commercial tool and its source code is not publicly available, it is unclear which optimizations it uses internally. However, it could be verified experimentally, that the optimization as in Equation (4.2) is only applied, when retiming is allowed.

**Conclusion** Within this case study it could be verified, that only retiming applied critical optimizations, such that the CMS $2^{nd}$-order AND gate is synthesized with a flaw in the masking scheme. More research is needed to check this claim for general applicability. For commercial tools such as Cadence Genus, because we can never know its inner workings, for the open-source tools Yosys and abc, because the optimizations could not be fully investigated. However, when all commands for sequential synthesis are removed from the abc script, one can be sure that the netlist is synthesized without flaws such as probing violations, as, in this case, it can be ruled out that registers are optimized. The registers then serve as anchors, preventing harmful optimizations also to the combinational logic between register stages and inherently constraining the synthesis.

# 5 Open Questions in the Hardware Design Flow

With this report, we can confidently say that frontend synthesis can have a negative impact on hardware attack countermeasures. For fault injection, we justify this with the observation, that - depending on the synthesis settings - single bit-flips are more likely to suffice to corrupt the design. For masked designs, we showed that retiming violates security assumptions. These results do not cover all aspects of the hardware design flow, e.g. whether backend synthesis also impacts the resilience. Further, our investigations are mainly based on established, but theoretical, models such as the number of bit-flips in a netlist or (robust) probing security. We discuss these open questions in this chapter.

## 5.1 Fault Injection Modeling

We based our analysis of fault injection countermeasures mainly on the outcomes of the SYNFI tool, which reports the number of bit-flips and possible combinations that are needed to corrupt a design. For this, SYNFI uses the netlist after RTL synthesis. So far, we did not yet consider or discuss the difficulty of flipping a single bit or up to *n* bits. Contemporary work on fault models and attacks such as [6] [21] [31] [70] provide more insights. In general it can be said, that inserting two or three bit-flips is more challenging than a single bit-flip. The authors of [6] show that their laser fault injection attacks are less likely to succeed, the more bit-flips are required. They investigate information-redundant designs. However, there exists no general investigation that provides a clearer connection between the number of bit-flips required and the complexity of attacks, e.g. if it is better to have one single bit-flip vulnerability than three two bit-flip vulnerabilities. Further, the influence of information that does not exist at netlist level, e.g. the transistor layout of standard cells, the placement, the routing and clock/reset trees, was not yet evaluated in a formal setting such as SYNFI. For example, the impact of standard cells with different driver strengths was already discussed in Section 3.3.4. While SYNFI does not distinguish between cells with the same boolean behavior but different driver strengths, this might be important for real world fault attacks. Cells that can drive more subsequent cells require more transistors and thus expose more attack surface. Such an analysis, however, could only be conducted at the transistor level of a circuit.

## 5.2 Side-channel Leakage

For masking, the case studies in this report are built on verification in the robust probing model. This model's mapping to actual resilience against real-world side-channel attacks has seen more attention than fault injection models. However, it remains an open question how hard side-channel attacks on a verified implementation actually are. Further, similar to fault injection modelling, all efforts in this report were focused on the netlist. In [27], the authors demonstrate that placement and routing have a significant impact on a design's leakage. They discuss that operating conditions such as a high supply voltages, high temperatures and high frequencies amplify the influence of placement and routing. Crosstalk, i.e. electromagnetic coupling, and power supply noise are cited as reasons for this leakage. An extreme case for vulnerabilities in the backend design flow is shown in [32]. The authors showcase, that a malicious party could modify the routing, such that the length of the path violates the target's clock frequency and leakage occurs. This amounts to a hardware trojan, purely based on modified routing. As of now, there is no other way than laboratory investigation of the integrated circuit, to include these backend effects into the security evaluation. Recall, that the concept of coupling-extended probes in the robust probing model exists (Section 2.1.4). However, there is no verification tool, that can put it into practice. The information on the placement of wires, and thus which wires are adjacent to one another, exists only after floor-planning and routing and not yet after RTL synthesis in the gate-level netlist.

An important aspect beyond the backend design flow is the generation of fresh randomness for the masked design. The authors of [17] show, that it is not enough to use LFSRs to generate randomness and linear dependence between pseudo-random data can cause leakage. For this effect, dedicated verification tooling exists [34] [55].

# 6 Conclusion

As motivation for this report we raised the question: *Does RTL synthesis and the backend design flow have a negative impact on countermeasures against physical adversaries?* To this end, we investigated fault injection and side-channel attack countermeasures.

Most fault injection countermeasures are built on redundancy, such that faults are detected with a high probability. Synthesis tools are built to detect redundant components and remove them, to keep the area consumption of integrated circuits low. Therefore, it is not surprising that synthesis tools remove redundant structures required to guarantee resilience against fault injection. This deficiency was explored previously in literature [43] [58] [59] [60]. However, hardware design engineers are still largely left alone with the question of how designs must be synthesized to ensure secure implementation. We identified the verification framework from [59] as useful to analyze netlists after RTL synthesis. To allow engineers to judge the effects of synthesis appropriately, we conducted six case studies. In summary, our experiments highlight once more that considering the RTL code alone is insufficient to make a statement on the security against fault injection attacks. Furthermore, we identified the following relations between the synthesis process and the effectiveness of fault injection countermeasures: With higher synthesis effort and more aggressive optimizations, there is a tendency towards decreased effectiveness of fault injection countermeasures. Increasing the target clock frequency might lead to inferior error detection performance as additional optimizations might be triggered to achieve tighter timing constraints. Optimizations such as a resolving of hierarchies or retiming must be taken into account. We found that different RTL implementations, which serve the same purpose, yield different results regarding the effectiveness of fault injection countermeasures. While these rough correlations could be shown for all tools, we made many tool specific and incomprehensible observations.

For countermeasures against side-channel attacks, we explored the synthesis of masked designs. We showed that contemporary masked designs are typically verified in a robust probing model that includes effects such as glitches due to differences in signal propagation and transitions of memory elements. Within a case study, we demonstrated that retiming reorders registers of a masked AES S-box such that robust probing security does no longer hold. Verification tools such as [46] can catch this violation, but are not trivial to use. Further, we demonstrated that the violation of robust probing security leads to detectable leakage in an unspecific and specific leakage assessment. The roadmap towards exploiting this leakage in an actual attack was sketched. While retiming can be disabled, it is not always directly obvious if a design is retimed. Further, design engineers might be tempted to retime a design, due to the improved area cost. The second case study addressed a recent trend in masking. That is the verification of small sub-circuits, of which larger circuits such as S-boxes can be composed. We demonstrated that this facilitates the verification of real-world designs. However, we also demonstrated that the complete design must be verified again to avoid composition pitfalls. The framework from [16] can verify composite designs, but is hard to integrate with commercial EDA tools. Finally, we investigated if retiming is the only potential threat to masked designs. To this end, we investigated the consolidated masking scheme (CMS), where randomness is added to the flip-flop's input and removed from its output. For the Yosys tool, we could show that only the retiming configuration allows such optimizations. This does also seem to hold for commercial tools such as Cadence Genus. In this case, the registers in a masked design serve as anchors that - if not optimized themselves - seem to prevent further critical optimizations.

With our case studies from Chapter 3 and Chapter 4 we can answer the question whether RTL synthesis has a negative impact on countermeasures with a clear yes. A short overview of state-of-the art literature in Chapter 5 showed that the backend design flow is critical for the effectiveness of countermeasure as well and many questions in and around the hardware design flow are still left open.

The findings of this report thus emphasize, that it is not sufficient to verify countermeasures on the RTL level. We showed that the synthesized netlist must be investigated to judge the effectiveness of

countermeasures appropriately. However, the references in Chapter 5 suggest that analyzing the netlist is also not sufficient. Thus, an actual laboratory evaluation of devices seems inevitable.

Still, we want to promote the usage of tools such as SYNFI [59], SILVER [46], and fullVerif [16] in the design flow. They allow to catch possible vulnerabilities early in the design flow, before putting the design through the extensive backend flow, or producing prototypes. Further, a laboratory assessment could still miss vulnerabilities that these tools can spot. However, as described in our case studies, the tools are not trivial to use and impose constraints on the design flow. Therefore, we encourage the hardware design community to improve these tools and make their integration into the design flow as seamless as running a functional verification. The SCFI integration into Yosys is a good example for such a seamless workflow [60]. This effort needs the support of design kit and EDA tool vendors. Open and standardized interfaces and intermediate representations of the synthesized circuit could facilitate the verification flow significantly.

# Bibliography


[1] Fraunhofer AISEC and BSI. A study on hardware attacks against microcontrollers. *BSI*, 2023. URL https://www.bsi. bund.de/DE/Service-Navi/Publikationen/Studien/ Hardware-Angriffe/Hardware-Angriffe.html?nn=520750

[2] Mehdi-Laurent Akkar and Christophe Giraud. An implementation of des and aes, secure against some attacks. In *Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '01, page 309–318, Berlin, Heidelberg, 2001. Springer-Verlag. ISBN 3540425217

[3] Victor Arribas, Svetla Nikova, and Vincent Rijmen. Vermi: Verification tool for masked implementations. Cryptology ePrint Archive, Paper 2017/1227, 2017. URL https://eprint.iacr.org/2017/1227. https://eprint.iacr.org/2017/1227

[4] Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the cost of lazy engineering for masked software implementations. Cryptology ePrint Archive, Paper 2014/413, 2014. URL https://eprint.iacr.org/2014/413. https://eprint.iacr.org/2014/413

[5] Gilles Barthe, Marc Gourjon, Benjamin Grégoire, Maximilian Orlt, Clara Paglialonga, and Lars Porth. Masking in fine-grained leakage models: Construction, implementation and verification. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021:189–228, Feb. 2021. doi: 10.46586/tches.v2021.i2.189-228. URL https://tches.iacr.org/index.php/TCHES/article/view/8792

[6] Timo Bartkewitz, Sven Bettendorf, Thorben Moos, Amir Moradi, and Falk Schellenberg. Beware of insufficient redundancy: An experimental evaluation of code-based fi countermeasures. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(3):438–462, Jun. 2022. doi: 0.46586/tches.v2022.i3.438-462. URL https://tches.iacr.org/index.php/TCHES/article/view/9708

[7] Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Randomness complexity of private circuits for multiplication. In *Proceedings, Part II, of the 35th Annual International Conference on Advances in Cryptology — EUROCRYPT 2016-Volume 9666*, page 616–648, Berlin, Heidelberg, 2016. Springer-Verlag. ISBN 9783662498958

[8] Sonia Belaïd, Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Abdul Rahman Taleb. Random probing security: Verification, composition, expansion and new constructions. In *Advances in Cryptology – CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part I*, page 339–368, Berlin, Heidelberg, 2020. Springer-Verlag. ISBN 978-3-030-56783-5. doi: 10.1007/978-3-030-56784-2_12. URL https://doi.org/10.1007/978-3-030-56784-2_12

[9] Sonia Belaïd, Darius Mercadier, Matthieu Rivain, and Abdul Rahman Taleb. Ironmask: Versatile verification of masking security. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 142–160, 2022. doi: 10.1109/SP46214. 2022.9833600

[10] Alex Biryukov. The boomerang attack on 5 and 6-round reduced aes. In *Proceedings of the 4th International Conference on Advanced Encryption Standard*,

AES'04, page 11–15, Berlin, Heidelberg, 2004. Springer-Verlag. ISBN 3540265570. doi: 10.1007/11506447_2. URL https://doi.org/ 10.1007/11506447_2

[11] Olivier Bronchain and François-Xavier Standaert. Side-channel countermeasures' dissection and the limits of closed source security evaluations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(2):1–25, Mar. 2020. doi: 10.13154/tches.v2020. i2.1-25. URL https://tches.iacr.org/index.php/TCHES/ article/view/8542

[12] *Genus User Guide*. Cadence Design Systems, 2023

[13] D. Canright. A very compact s-box for aes. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, pages 441–455, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31940-5

[14] Gaëtan Cassiers and Francois-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Transactions on Information Forensics and Security*, 15:2542–2555, 2020. doi: 10.1109/TIFS.2020.2971153

[15] Gaëtan Cassiers, Sebastian Faust, Maximilian Orlt, and François-Xavier Standaert. Towards tight random probing security. In *Advances in Cryptology – CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part III*, page 185–214, Berlin, Heidelberg, 2021. Springer-Verlag. ISBN 978-3-030-84251-2. doi: 10.1007/978-3-030-84252-9_7. URL https://doi.org/10.1007/978-3-030-84252-9_7

[16] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and Francois-Xavier Standaert. Hardware private circuits: From trivial composition to full verification. *IEEE Transactions on Computers*, 70(10):1677–1690, 2021. doi: 10.1109/TC.2020.3022979

[17] Gaëtan Cassiers, Loïc Masure, Charles Momin, Thorben Moos, Amir Moradi, and Francois-Xavier Standaert. Randomness generation for secure hardware masking - unrolled trivium to the rescue. Cryptology ePrint Archive, Paper 2023/1134, 2023. URL https://eprint.iacr.org/2023/1134. https://eprint.iacr.org/2023/1134

[18] Gaëtan Cassiers, Loïc Masure, Charles Momin, Thorben Moos, and Francois-Xavier Standaert. Prime-field masking in hardware and its soundness against low-noise sca attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(2):482–518, Mar. 2023. doi: 10.46586/tches. v2023.i2.482-518. URL https://tches.iacr.org/index.php/ TCHES/article/view/10291

[19] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, page 398–412, Berlin, Heidelberg, 1999. Springer-Verlag. ISBN 3540663479

[20] Thomas De Cnudde, Begül Bilgin, Benedikt Gierlichs, Ventzislav Nikov, Svetla Nikova, and Vincent Rijmen. Does coupling affect the security of masked implementations? Cryptology ePrint Archive, Paper 2016/1080, 2016. URL https://eprint.iacr.org/2016/1080. https://eprint. iacr.org/2016/1080

[21] Brice Colombier, Paul Grandamme, Julien Vernay, Émilie Chanavat, Lilian Bossuet, Lucie de Laulanié, and Bruno Chassagne. Multi-spot laser fault injection setup: New possibilities for fault injection attacks. In *Smart Card Research and Advanced Applications: 20th International Conference, CARDIS 2021, Lübeck, Germany, November 11–12, 2021, Revised Selected Papers*, page 151–166, Berlin,

Heidelberg, 2021. Springer-Verlag. ISBN 978-3-030-97347-6. doi: 10.1007/978-3-030-97348-3_9. URL https://doi. org/10.1007/978-3-030-97348-3_9

[22] Jean-Sébastien Coron and Louis Goubin. On boolean and arithmetic masking against differential power analysis. In *Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '00, page 231–237, Berlin, Heidelberg, 2000. Springer-Verlag. ISBN 354041455X

[23] Jean-Sébastien Coron, Christophe Giraud, Emmanuel Prouff, Soline Renner, Matthieu Rivain, and Praveen Kumar Vadnala. Conversion of security proofs from one leakage model to another: A new issue. In *Proceedings of the Third International Conference on Constructive Side-Channel Analysis and Secure Design*, COSADE'12, page 69–81, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 9783642299117. doi: 10.1007/978-3-642-29912-4_6. URL https://doi.org/10.1007/978-3-642-29912-4_6

[24] Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-order side channel security and mask refreshing. *IACR Cryptol. ePrint Arch.*, page 359, 2015. URL http://eprint.iacr.org/ 2015/359

[25] Ana Covic, Fatemeh Ganji, and Domenic Forte. Circuit masking: From theory to standardization, A comprehensive survey for hardware security researchers and practitioners. *CoRR*, abs/2106.12714, 2021. URL https://arxiv.org/abs/2106.12714

[26] Joan Daemen. Changing of the guards: A simple and efficient method for achieving uniformity in threshold sharing. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, pages 137–153, Cham, 2017. Springer International Publishing. ISBN 978-3-319-66787-4

[27] Thomas De Cnudde, Maik Ender, and Amir Moradi. Hardware masking, revisited. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(2):123–148, May 2018. doi: 10.13154/tches.v2018.i2.123-148. URL https://tches.iacr.org/index.php/TCHES/article/view/877

[28] Siemen Dhooghe and Svetla Nikova. My gadget just cares for me - how nina can prove security against combined attacks. In Stanislaw Jarecki, editor, *Topics in Cryptology – CT-RSA 2020*, pages 35–55, Cham, 2020. Springer International Publishing. ISBN 978-3-030-40186-3

[29] Siemen Dhooghe and Svetla Nikova. The random fault model. Cryptology ePrint Archive, Paper 2022/1627, 2022. URL https://eprint. iacr.org/2022/1627. https://eprint.iacr.org/2022/ 1627

[30] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. *J. Cryptol.*, 32(1):151–177, jan 2019. ISSN 0933-2790. doi: 10.1007/s00145-018-9284-1. URL https://doi.org/10.1007/s00145-018-9284-1

[31] Jean-Max Dutertre, Vincent Beroulle, Philippe Candelier, Stephan De Castro, Louis-Barthelemy Faber, Marie-Lise Flottes, Philippe Gendrier, David Hély, Regis Leveugle, Paolo Maistri, Giorgio Di Natale, Athanasios Papadimitriou, and Bruno Rouzeyre. Laser fault injection at the cmos 28 nm technology node: an analysis of the fault model. In *2018 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 1–6, 2018. doi: 10.1109/FDTC. 2018.00009

[32] Maik Ender, Samaneh Ghandali, Amir Moradi, and Christof Paar. The first thorough side-channel hardware trojan. In Tsuyoshi Takagi and Thomas Peyrin,

editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 755–780, Cham, 2017. Springer International Publishing. ISBN 978-3-319-70694-8

[33] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable masking schemes in the presence of physical defaults; the robust probing model. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):89–120, Aug. 2018. doi: 10.13154/tches.v2018.i3.89-120. URL https://tches.iacr.org/index.php/TCHES/article/view/7270

[34] Jakob Feldtkeller, David Knichel, Pascal Sasdrich, Amir Moradi, and Tim Güneysu. Randomness optimization for gadget compositions in higher-order masking. Cryptology ePrint Archive, Paper 2022/882, 2022. URL https://eprint.iacr.org/2022/882. https://eprint.iacr.org/ 2022/882

[35] Benedikt Gierlichs, Jörn-Marc Schmidt, and Michael Tunstall. Infective computation and dummy rounds: Fault protection for block ciphers without check-before-output. In *Progress in Cryptology–LATINCRYPT 2012: 2nd International Conference on Cryptology and Information Security in Latin America, Santiago, Chile, October 7-10, 2012. Proceedings 2*, pages 305–321. Springer, 2012

[36] Barbara Gigerl, Vedad Hadzic, Robert Primas, Stefan Mangard, and Roderick Bloem. Coco: Co-design and co-verification of masked software implementations on cpus. Cryptology ePrint Archive, Paper 2020/1294, 2020. URL https://eprint.iacr.org/2020/1294. https://eprint.iacr.org/2020/1294

[37] Louis Goubin and Jacques Patarin. Des and differential power analysis (the Duplication method). In *Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '99, page 158–172, Berlin, Heidelberg, 1999. Springer-Verlag. ISBN 354066646X

[38] Hannes Gross, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In *Proceedings of the 2016 ACM Workshop on Theory of Implementation Security*, TIS '16, page 3, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450345750. doi: 10.1145/2996366.2996426.URL https://doi.org/10.1145/2996366.2996426

[39] Vincent Immler, Robert Specht, and Florian Unterstein. Your rails cannot hide from localized em: How dual-rail logic fails on fpgas. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems –CHES 2017*, pages 403–424, Cham, 2017. Springer International Publishing. ISBN 978-3-319-66787-4

[40] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology CRYPTO 2003*, pages 463–481, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-45146-4

[41] R. Karri, K. Wu, P. Mishra, and Yongkook Kim. Concurrent error detection schemes for fault-based side-channel cryptanalysis of symmetric block ciphers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(12):1509–1517, 2002. doi: 10.1109/TCAD.2002.804378

[42] Ramesh Karri, Kaijie Wu, Piyush Mishra, and Yongkook Kim. Concurrent error detection of fault-based side-channel cryptanalysis of 128-bit symmetric block ciphers. In *Proceedings of the 38th Annual Design Automation Conference*, DAC '01,

page 579–584, New York, NY, USA, 2001. Association for Computing Machinery. ISBN 1581132972. doi: 10.1145/378239.379027.URL https://doi.org/10.1145/378239.379027

[43] Rasheed Kibria, Farimah Farahmandi, and Mark M. Tehranipoor. ARC-FSM-G: automatic security rule checking for finite state machine at the netlist abstraction. *IACR Cryptol. ePrint Arch.*, page 1037, 2023. URL https://eprint.iacr.org/2023/1037

[44] David Knichel and Amir Moradi. Composable gadgets with reused fresh masks: First-order probing-secure hardware circuits with only 6 fresh masks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(3):114–140, Jun. 2022. doi: 10.46586/tches.v2022.i3.114-140. URL https://tches.iacr.org/index.php/TCHES/article/view/9696

[45] David Knichel and Amir Moradi. Low-latency hardware private circuits. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 1799–1812, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450394505. doi: 10.1145/3548606.3559362. URL https://doi.org/10.1145/ 3548606.3559362

[46] David Knichel, Pascal Sasdrich, and Amir Moradi. Silver – statistical independence and leakage verification. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020*, pages 787–816, Cham, 2020. Springer International Publishing. ISBN 978-3-030-64837-4

[47] David Knichel, Amir Moradi, Nicolai Müller, and Pascal Sasdrich. Automated generation of masked hardware. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(1):589–629, Nov. 2021. doi: 10.46586/tches.v2022.i1.589-629. URL https://tches.iacr.org/index.php/TCHES/article/view/9308

[48] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 388–397, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. ISBN 978-3-540-48405-9

[49] Daniel Lammers, Nicolai Müller, and Amir Moradi. Glitch-free is not enough revisiting glitch-extended probing model. Cryptology ePrint Archive, Paper 2023/035, 2023. URL https://eprint.iacr.org/2023/035. https://eprint.iacr.org/2023/035

[50] Charles E. Leiserson, Flavio M. Rose, and James B. Saxe. Optimizing synchronous circuitry by retiming (preliminary version). In Randal Bryant, editor, *Third Caltech Conference on Very Large Scale Integration*, pages 87–116, Berlin, Heidelberg, 1983. Springer Berlin Heidelberg. ISBN 978-3-642-95432-0

[51] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-channel leakage of masked cmos gates. In *Proceedings of the 2005 International Conference on Topics in Cryptology*, CT-RSA'05, page 351–365, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3540243992. doi: 10. 1007/978-3-540-30574-3_24. URL https://doi.org/10.1007/978-3-540-30574-3_24

[52] Alan Mishchenko, Michael Case, Robert Brayton, and Stephen Jang. Scalable and scalably-verifiable sequential synthesis. In *2008 IEEE/ACM International Conference on Computer-Aided Design*, pages 234–241, 2008. doi: 10.1109/ICCAD.2008.4681580

[53] Charles Momin, Gaëtan Cassiers, and François-Xavier Standaert. Handcrafting: Improving automated masking in hardware with manual optimizations. In Josep

Balasch and Colin O'Flynn, editors, *Constructive Side-Channel Analysis and Secure Design*, pages 257–275, Cham, 2022. Springer International Publishing. ISBN 978-3-030-99766-3

[54] Thorben Moos, Amir Moradi, Tobias Schneider, and Francois-Xavier Standaert. Glitch-resistant masking revisited: or why proofs in the robust probing model are needed. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(2):256–292, Feb. 2019. doi: 10.13154/tches. v2019.i2.256-292. URL https://tches.iacr.org/index.php/ TCHES/article/view/7392

[55] Nicolai Müller and Amir Moradi. Prolead: A probing-based hardware leakage detection tool. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):311–348, Aug. 2022. doi: 10.46586/tches.v2022.i4. 311-348. URL https://tches.iacr.org/index.php/TCHES/article/view/9822

[56] Nicolai Müller, David Knichel, Pascal Sasdrich, and Amir Moradi. Transitional leakage in theory and practice: Unveiling security flaws in masked circuits. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022:266–288, Feb. 2022. doi: 10.46586/tches.v2022.i2. 266-288. URL https://tches.iacr.org/index.php/TCHES/article/view/9488

[57] Adib Nahiyan, Kan Xiao, Kun Yang, Yeir Jin, Domenic Forte, and Mark Tehranipoor. Avfsm: A framework for identifying and mitigating vulnerabilities in fsms. In *Proceedings of the 53rd Annual Design Automation Conference*, DAC '16, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342360. doi: 10.1145/2897937.2897992. URL https://doi.org/10.1145/2897937.2897992

[58] Adib Nahiyan, Farimah Farahmandi, Prabhat Mishra, Domenic Forte, and Mark M. Tehranipoor. Security-aware FSM design flow for identifying and mitigating vulnerabilities to fault attacks. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 38(6):1003–1016, 2019. doi: 10.1109/TCAD.2018.2834396. URL https://doi.org/10.1109/TCAD.2018.2834396

[59] Pascal Nasahl, Miguel Osorio, Pirmin Vogel, Michael Schaffner, Timothy Trippel, Dominic Rizzo, and Stefan Mangard. SYNFI: pre-silicon fault analysis of an open-source secure element. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):56–87, 2022. doi: 10.46586/tches.v2022.i4.56-87. URL https://doi.org/10.46586/tches.v2022.i4.56-87

[60] Pascal Nasahl, Martin Unterguggenberger, Rishub Nagpal, Robert Schilling, David Schrammel, and Stefan Mangard. SCFI: state machine control-flow hardening against fault attacks. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2023, Antwerp, Belgium, April 17-19, 2023*, pages 1–6. IEEE, 2023. doi: 10.23919/DATE56975. 2023.10137038. URL https://doi.org/10.23919/DATE56975. 2023.10137038

[61] James Nechvatal, Elaine Barker, Lawrence Bassham, William Burr, Morris Dworkin, James Foti, and E Roback. Report on the development of the advanced encryption standard (aes), 2001-06-01 2001. URL https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=151226

[62] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In *Information and Communications Security*, page 529–545, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 978-3-540-49496-6. doi: 10.1007/11935308_38. URL https://doi.org/10.1007/11935308_38

[63] Elisabeth Oswald, Stefan Mangard, Norbert Pramstaller, and Vincent Rijmen. A side-channel analysis resistant description of the aes s-box. In *Proceedings of the 12th International Conference on Fast Software Encryption*, FSE'05, page 413–423, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3540265414. doi: 10.1007/11502760_28. URL https://doi.org/10.1007/11502760_28

[64] Sikhar Patranabis, Abhishek Chakraborty, and Debdeep Mukhopadhyay. Fault tolerant infective countermeasure for aes. *Journal of Hardware and Systems Security*, 1:3–17, 2017

[65] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, pages 142–159, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-38348-9

[66] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In Isabelle Attali and Thomas Jensen, editors, *Smart Card Programming and Security*, pages 200– 210, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. ISBN 978-3-540-45418-2

[67] Aein Rezaei Shahmirzadi and Amir Moradi. Re-consolidating first-order masking schemes: Nullifying fresh randomness. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(1):305–342, Dec. 2020. doi: 10.46586/tches.v2021.i1.305-342. URL https://tches.iacr.org/index.php/TCHES/article/view/8736

[68] Jan Richter-Brockmann, Aein Rezaei Shahmirzadi, Pascal Sasdrich, Amir Moradi, and Tim Güneysu. Fiver – robust verification of countermeasures against fault injections. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(4):447–473, Aug. 2021. doi: 10.46586/tches. v2021.i4.447-473. URL https://tches.iacr.org/index.php/ TCHES/article/view/9072

[69] Jan Richter-Brockmann, Jakob Feldtkeller, Pascal Sasdrich, and Tim Güneysu. Verica - verification of combined attacks: Automated formal verification of security against simultaneous information leakage and tampering. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):255–284, Aug. 2022. doi: 10.46586/tches.v2022.i4.255-284. URL https://tches.iacr.org/index.php/TCHES/article/view/9820

[70] Jan Richter-Brockmann, Pascal Sasdrich, and Tim Güneysu. Revisiting fault adversary models – hardware faults in theory and practice. *IEEE Transactions on Computers*, 72(2):572–585, 2023. doi: 10.1109/TC.2022.3164259

[71] Tobias Schneider and Amir Moradi. Leakage assessment methodology - a clear roadmap for side-channel evaluations. Cryptology ePrint Archive, Paper 2015/207, 2015. URL https://eprint.iacr.org/2015/207. https://eprint.iacr.org/2015/207

[72] Mohamed Shalan and Tim Edwards. Building openlane: A 130nm openroad-based tapeout-proven flow. In *Proceedings of the 39th International Conference on Computer-Aided Design*, ICCAD '20, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450380263. doi: 10.1145/3400302.3415735. URL https://doi.org/10.1145/ 3400302.3415735

[73] Robert Specht, Vincent Immler, Florian Unterstein, Johann Heyszl, and Georg Sigl. Dividing the threshold: Multi-probe localized em analysis on threshold

implementations. In *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 33–40, 2018. doi: 10.1109/HST.2018.8383888

[74] Florian Unterstein, Johann Heyszl, Fabrizio De Santis, Robert Specht, and Georg Sigl. High-resolution em attacks against leakage-resilient prfs explained. In Nigel P. Smart, editor, *Topics in Cryptology – CT-RSA 2018*, pages 413–434, Cham, 2018. Springer International Publishing. ISBN 978-3-319-76953-0

[75] Neil HE Weste and David Harris. *CMOS VLSI design: a circuits and systems perspective*. Pearson Education India, 2015

# Appendix

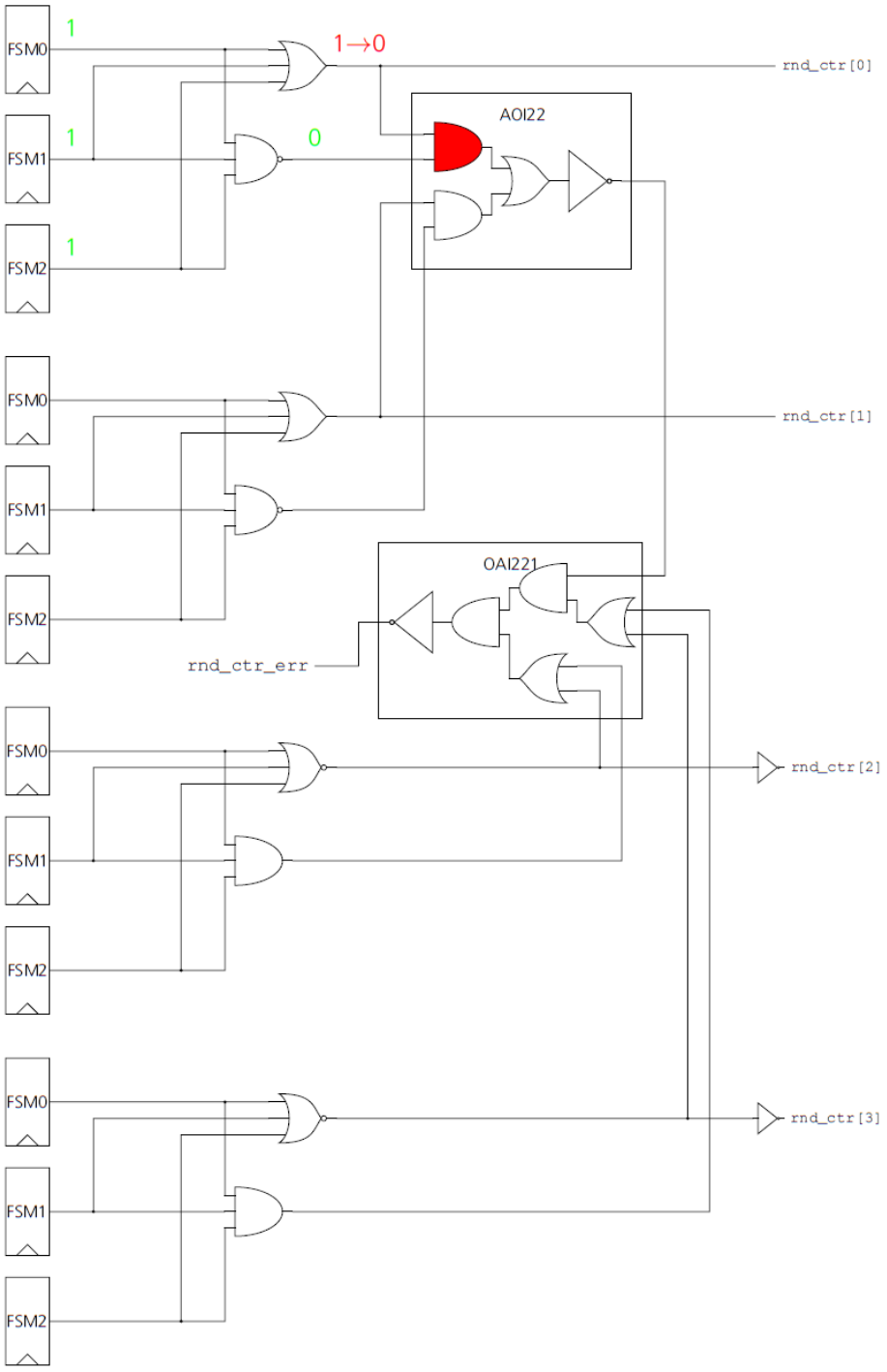



*Figure A. 1: Fault experiment with netlist synthesized with Synopsys Design Compiler showing a sub-circuit comprising AOI and OAI standard cells. For a rnd_ctr[0] value of 1, one fault (1 → 0) is injected into the OR gate of the rnd_ctr[0] path. The second input of the AND gate within the AOI22 cell marked in red is 0 and therefore masks the injected fault at the first input such that it has no effect on the rnd_cnt_err signal. Consequently, the injected fault changes the value of rnd_ctr[0] but cannot be detected.*

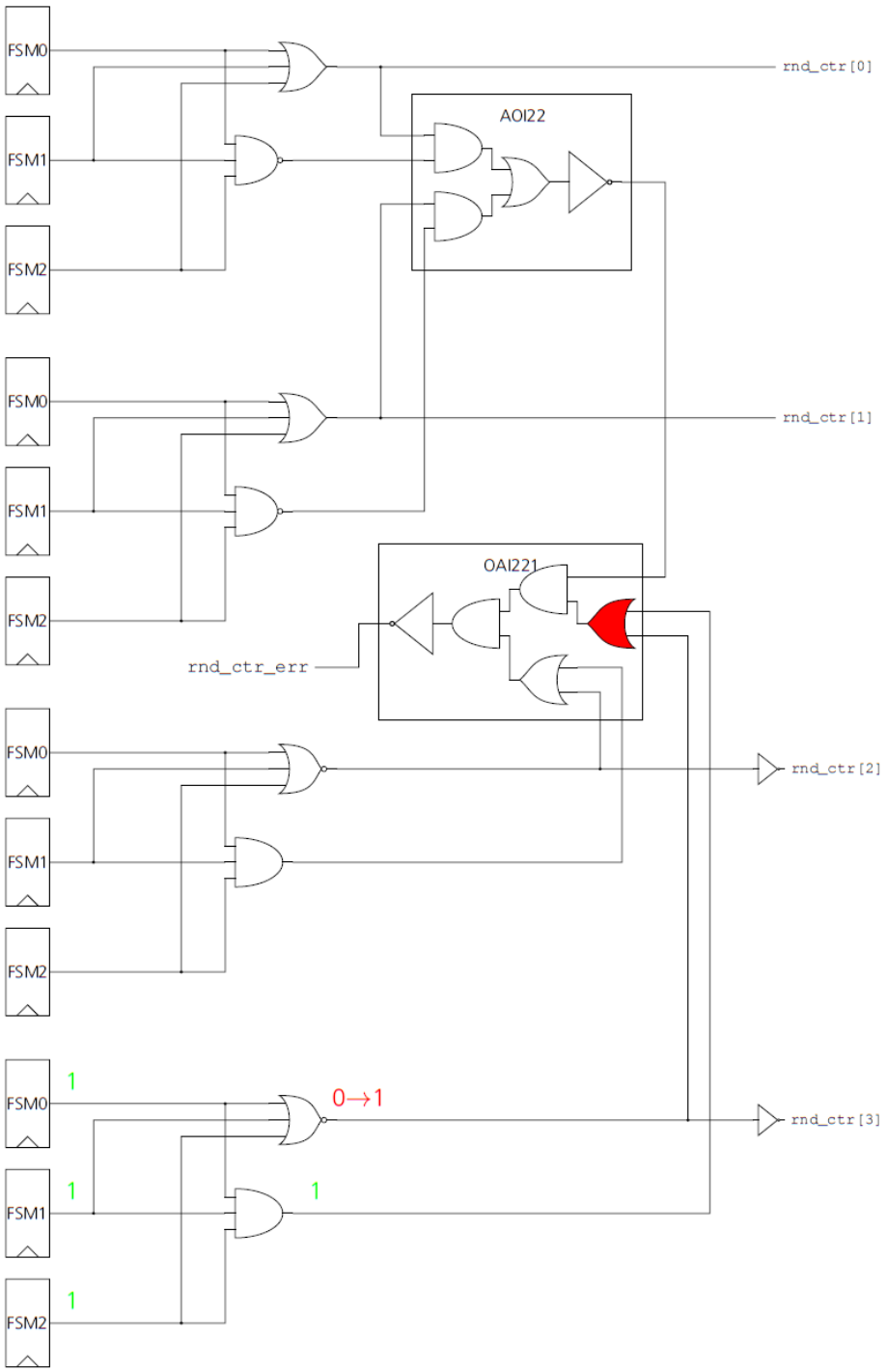



*Figure A. 2: Fault experiment with netlist synthesized with Synopsys Design Compiler showing a sub-circuit comprising AOI and OAI standard cells. For a rnd_ctr[3] value of 1, one fault (0 → 1) is injected into the NOR gate of the rnd_ctr[3] path. The first input of the NOR gate within the OAI22 cell marked in red is 1, and therefore masks the injected fault at the second input such that it has no effect on the rnd_cnt_err signal. Consequently, the injected fault changes the value of rnd_ctr[3] but cannot be detected.*

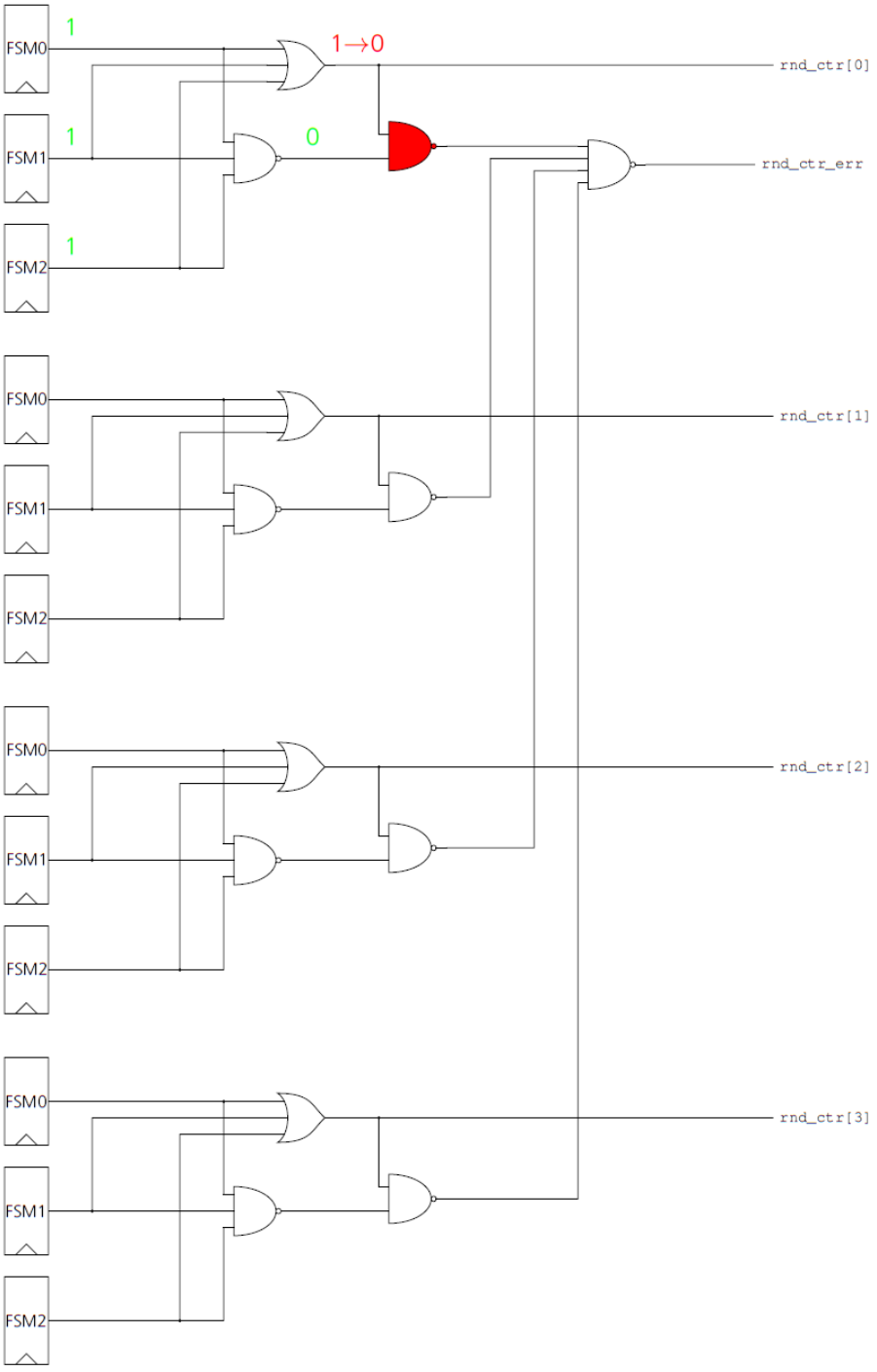



*Figure A. 3: Fault experiment with netlist synthesized with Synopsys Design Compiler showing a sub-circuit without AOI and OAI standard cells. For a rnd_ctr[0] value of 1, one fault (1 → 0) is injected into the OR gate of the rnd_ctr[0] path. The second input of the NAND gate marked in red is 1, and therefore masks the injected fault at the first input such that it has no effect on the rnd_cnt_err signal. Consequently, the injected fault changes the value of rnd_ctr[0] but cannot be detected.*