COLLEGE DOCTORAL BRETAGNE
MATHS, TELECOMS
INFORMATIQUE, SIGNAL
SYSTEMES, ELECTRONIQUE

Université de Rennes

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES

ÉCOLE DOCTORALE N° 601
Mathématiques, Télécommunications, Informatique, Signal, Systèmes, Électronique
Spécialité : *Informatique*

Par

Amélie MAROTTA

Effects of synchronous clock glitch on the security of an integrated circuit

Thèse présentée et soutenue à Rennes, le 23 juin 2025
Unité de recherche : Centre Inria de l'Université de Rennes

Rapporteurs avant soutenance :

Jessy Clédière	Directeur de Recherche, CEA Leti, Grenoble
Jean-Max Dutertre	Professeur, Ecole des Mines de Saint-Etienne, Gardanne

Composition du Jury :

Président :	Vincent Beroulle	Professeur, Grenoble INP-UGA Esisar, LCIS, Valence
Examinateurs :	Maria Méndez-Real	Chaire de Professeur Junior, Université de Bretagne-Sud, Lab-STICC, Lorient
	Jessy Clédière	Directeur de Recherche, CEA Leti, Grenoble
	Jean-Max Dutertre	Professeur, Ecole des Mines de Saint-Etienne, Gardanne
Dir. de thèse :	Olivier Sentieys	Professeur, Université de Rennes, IRISA, Inria
Co-dir. de thèse :	Ronan Lashermes	Ingénieur de Recherche, Inria, Rennes

Invités :

Rachid Dafali	Expert sécurité matérielle, DGA MI (co-encadrant)
Guillaume Bouffard	Expert sécurité des systèmes embarqués, ANSSI (co-encadrant)

Remerciements

En premier lieu, je souhaite remercier mon directeur de thèse Olivier Sentieys, mon co-directeur Ronan Lashermes et mes encadrants Rachid Dafali et Guillaume Bouffard. Merci pour vos précieux conseils, votre bienveillance et votre disponibilité au cours des trois dernières années, qui ont grandement contribué non seulement à la qualité des travaux scientifiques produits lors de ma thèse mais aussi à mon bien-être.

Merci à Jessy Clédière et Jean-Max Dutertre d'avoir accepté de relire cette thèse, et à Vincent Beroulle et Maria Méndez-Real d'avoir accepté de faire partie de mon jury de thèse. Je remercie également Damien Hardy et Rubén Salvador d'avoir été membres du comité de suivi.

Je tiens maintenant à remercier les nombreux membres de l'équipe TARAN, anciens ou actuels, pour leur accueil et leur soutien tout au long de cette thèse : Joseph, Simon, Louis S., Mehdi, Dylan, Romaric, Nesrine, Guillaume, Benoit, Patrice, Hery, Steven, Angeliki, Marcello, Fernando... Un merci tout particulier à Nadia, pour son aide toujours précieuse. Merci à Cédric, Rémi, Étienne, Silviu, François, Sonia et Léo, alias la team Biocoop (ou Biocool) pour tous les repas conviviaux que nous avons partagés. Merci à Jean-Michel et Seu. Nos thèses ont débuté au même moment, à un mois près, et je ne pouvais rêver avoir de meilleurs camarades pour me lancer dans cette aventure.

Merci à la team TRAITOR, Antoine et Ludo, pour nos échanges sur l'injection de fautes mais aussi et surtout pour votre amitié.

Merci aux filles de "001 Le Retour", Clara, Camille et Agathe. Je suis la dernière à soutenir, vous êtes donc un peu mes grandes soeurs de thèse. Merci à Déborah, sans qui je ne serai pas la personne que je suis aujourd'hui.

Pour terminer, je souhaite remercier ma famille. Merci à mes parents pour leur soutien inconditionnel, à la fois matériel mais surtout émotionnel, pendant ces longues années d'étude. Merci pour votre aide pour mes (nombreux) déménagements (jamais dans la même ville). Merci d'avoir toujours cru en moi, je ne suis peut-être pas devenue médecin mais je serai bien docteur ! Enfin, je souhaite remercier Aurélien d'avoir pris soin de moi

ces trois dernières années (tu as ma reconnaissance pour bien d'autres raisons, mais il faudrait une autre centaine de pages pour tout lister !).

Table of Contents

Chapter 1

Introduction

Electronic devices are part of our daily lives. Within the last two decades, their number has grown significantly, notably due to the expansion of Internet of Things (IoT) devices. These devices, connected (often wirelessly) to the Internet, are combined with sensors (to collect "useful" data) and branded as "smart": smart watches, smart locks, smart cars, smart homes, etc. IoT devices are exposed to numerous threats due to their broad use [5]. As such, they are the target of attacks, aiming to retrieve sensitive informations, gain control of the device, etc. Understanding where the vulnerability comes from (software implementation, interactions with the physical world, microarchitectural optimizations, etc.), how it is exploitable and how to fix it is essential to make better secured electronic devices.

While most of these attacks are perpetrated willingly, the devices are also vulnerable to a natural phenomenon: faults. A fault is a hardware failure, born from the interaction between an electronic component and particles (namely, when it happens naturally, cosmic rays or radiations). If not detected and corrected, faults can have consequences that range from enhancing a video game speedrun [14] to modifying an election result [13]. Furthermore, faults are a major concern for electronic devices used in space applications [41]. Hardening the devices against faults is crucial to prevent abnormal behaviours. But since natural faults are uncontrollable, several means to inject faults on purpose exist, and their impact is analyzed to either perform attacks or design countermeasures.

The impact of faults is not limited to electronic components, it translates into higher level effects, on the microarchitecture, on instructions. For example, a fault can affect the control flow of a target program and be used to bypass security mechanisms [22]. Similarly, a fault can be used to recover information such as secret keys [28].

Fault injection means are various. For example, pushing your target beyond its opera-

tional limits by underpowering or overclocking it will cause faults. Electromagnetic fault injections (EMFIs) are noteworthy as frequently employed due to their minimal setup requirements. However, an EM fault impacts several signals at once, so its study can be complex. In [8], Claudepierre *et al.* isolated a particular effect of EM on the phase-locked loop (PLL), resulting in a specific alteration of the clock, that we call the synchronous clock glitch (SCG). This glitch, when injected via TRAITOR (TRAnsportable glItch aTtack platfORm) [9], can be used in many-fault injection campaigns. Instead of injecting one or two faults, as most fault injection means are capable of, TRAITOR can inject dozens of faults, perturbing several instructions. This is particularly useful to bypass security measures [22, 39].

Several fault models explaining the impact of EMFI on electronic components [17, 36, 15] and at microarchitectural level (on instructions, data transfers, etc. [33, 34, 49, 50, 35]) have been published. Although explaining the consequences of the same fault impacts, the link between the characterization levels is, as far as we know, not the main interest of research works on fault injection. Concerning the SCG, the interest so far was on its exploitation potential to perform attacks. The only characterization was done at microarchitectural level, the glitch supposedly causing instruction skip or repeat. By performing fault injection using the SCG on an integrated circuit (IC), we quickly realised that the microarchitectural characterization was lacking, the fault models being more diverse than just instruction skip and repeat [9]. Furthermore, the existing physical fault models (fault models that consider the impact on electronic components) do not explain how the SCG leads to faults.

Our contributions, presented in this document, aim to offer an advanced characterization of the SCG, from its impact on electrical components to the instructions. We believe it is important work since this glitch has proven to be used in successful attacks and has the capability to bypass software-based countermeasures. Understanding how the SCG leads to faults in a circuit is crucial to design countermeasures against it. Furthermore, it adds general knowledge on EMFI.

Our document is organized as follows. In Chapter 2, we give an overall view of the security context of electronic devices, mainly their vulnerabilities and the existing exploitation methods. Fault injections (FIs) (also referred to as fault injection attacks (FIAs)) are one of many threats electronic devices are concerned with and are our main interest, due to their potential for nuisance and their general impact on various targets. Chapter 3 gives some basic knowledge on FI. To be able to understand correctly their impact on electronic devices, it is useful to have in mind the different impacted elements and targets. Although we only mentioned EMFI in this introduction, several other FI methods exist and are presented. Some of these methods are akin to EMFI in their impact characterization. In Chapter 4, we present state-of-the-art fault models that apply to EMFI and associated injection means. This way, we highlight the diversity and complex nature of EMFI impact on IC. From this diversity, we pick the SCG to study. Chapters 5 and 6 are dedicated to understanding as well as possible this glitch and the impact it has at different abstraction levels. First, we identify the vulnerable electronic components and the main fault mechanism, from which we theorize the Energy Threshold Fault Model (ETFM). This model is then expanded to higher levels to characterize the impact of the glitch; first on a simple circuit, and finally on a processor. This last study, done on a commercial ASIC, revealed that while we can gather some clues on how the microarchitecture elements are impacted by the SCG, further research needs to be done on targets we have more control and information on. Chapter 7 concludes this document and gives perspectives on future works.

Chapter 2

Context: the Various Sources of Security Vulnerability

Fault injection is part of a bigger ecosystem of vulnerability exploitation methods, which it can be paired with. To make the study of the security challenges of electronic devices easier, we define three sources of vulnerabilities, as depicted in Figure 2.1: software, microarchitectural, and physical levels. The boundaries between these sources are porous: vulnerabilities or their exploitation may come from the interactions between these levels.

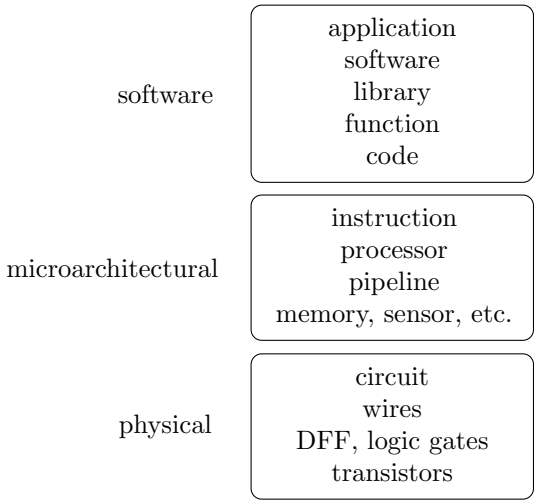


Figure 2.1 – Abstraction levels

Every aspect concerning the design of an electronic device can be a source of vulnerability exploitable by an attacker. For an electronic device to serve its purpose, it needs to run specific software. Software can be seen as a succession of code lines written by a human being with a specific goal. The software code is understandable by humans but has to be broken down into smaller, simpler instructions in another language that the hardware understands (this process is called compilation). This translation depends on the underlying processor that runs the instructions according to a specific instruction set

architecture (ISA). The most widely deployed ISAs are ARM, RISC-V, and x86. The processor is in charge of executing each instruction and controlling the other electronic components, such as memories, sensors, etc. To go even deeper, when we look at a processor at an electronic, physical level, we see an arrangement of smaller, lower-level electronic components (registers, logic gates) designed to be as efficient as possible.

Throughout this chapter, to illustrate the different sources of vulnerabilities and the different abstraction levels, we use the example of a PIN (Personal Identification Number) verifier. To unlock a smartphone or pay with a credit card, a passcode (usually composed of at least four numbers) is asked and compared with the device PIN. If the comparison is valid, the operation is successful. Otherwise, the passcode is asked again. Usually, the number of trials is limited, and each incorrect attempt decreases a counter that monitors the number of failed trials.

2.1 Software Level

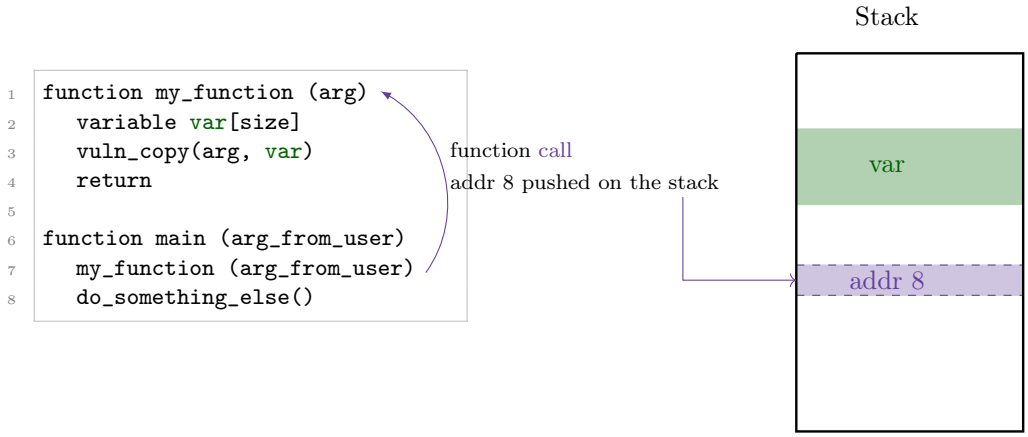
A simple PIN verification algorithm, inspired by the `VerifyPin` in [18], is presented in Listing 2.2.

```
verify_pin()
    if nb_try > 0:
        if compare(userPin, devicePin) == True:
            nb_try = 3
            return True
        else:
            nb_try--
            return False
    return False
```

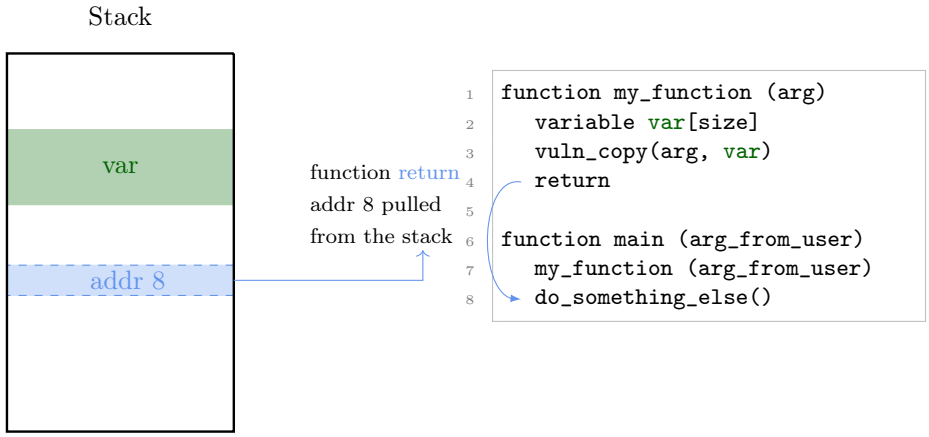
Figure 2.2 – PIN verification algorithm

Software can have numerous vulnerabilities, including the code itself, the compilation process, the environment it runs in, etc. Since humans write software, mistakes in code happen. One example is the use of a function or library that introduces vulnerabilities. For example, stack buffer overflow attacks [46] rely on the use of a function that takes data as an argument without checking its size. To explain this attack, we schematized a simple function call and return in Figure 2.3. The numbers present on the left of the code

listing serve as addresses for the sake of simplicity. In the `main` function, a function call to `my_function` occurs (Figure 2.3a). For the execution to return to the `main` function after it reaches a `return` instruction, the address of the following instruction (in our example, `8`) is stored in an accessible part of the memory called the stack. When the execution returns to the `main` function, it is removed from the stack (Figure 2.3b).



(a) Schematized function call routine



(b) Schematized function return routine

Figure 2.3 – Function call and return

The buffer overflow attack exploits this mechanism, as well as the vulnerable function defined earlier. By sending sufficient data to the vulnerable function, the attacker overwrites critical information, such as the return address of `my_function`, and replaces it with the address of the code they control, as illustrated in Figure 2.4.

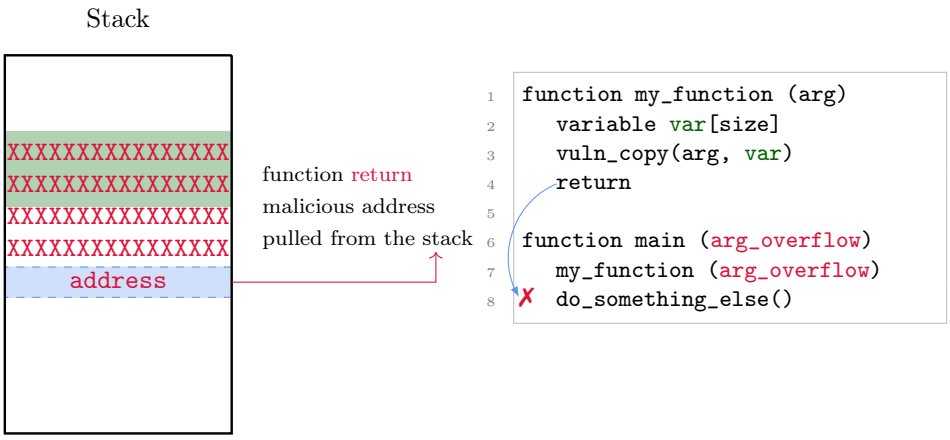


Figure 2.4 – Schematized buffer overflow

Different solutions exist to prevent this attack, such as the use of canaries [16]. Some data is placed before the return address and checked anytime a return procedure is engaged to detect a stack buffer overflow attack. Additionally, parts of the memory where the attacker could place their own code can be rendered unexploitable.

But these countermeasures are sometimes not enough. To bypass the non-executable stack, an attacker can use a Return-Oriented Programming (ROP) attack [6]. Instead of writing their own code on the stack, an attacker will craft their malicious code by using already present instructions (found in functions, libraries, etc.). The attacker still needs to perform a buffer overflow and take control of the control flow. Since the code is already in exploitable zones in memory, the countermeasure is ineffective. ROP attacks rely on the diversity of instructions accessible to the attacker. Reducing the number of instructions by changing the compilation makes it harder to perform.

Network communications are also vulnerable. Among the various attacks, we can cite spoofing (or cache poisoning), where an attacker corrupts specific data, such as IP addresses contained in the Domain Name System (DNS), to falsify their identity [25]. This can be part of man-in-the-middle attacks, where the attacker inserts themselves into a communication between two parties, acting as a malicious relay.

2.2 Microarchitectural Level

At the microarchitectural level, vulnerabilities come from the processor, its implementation, or its communications with memories, sensors, etc. For example, Spectre [27] and Meltdown [31] are attacks on the microarchitecture that target speculative execution. Some processors implement speculative execution to enhance their performance. When the execution reaches a branch, two choices are possible: the branch is taken, and the execution resumes somewhere else in the code, or the branch is not taken, and the execution resumes at the very next instruction in the code. Whether the branch is taken or not depends on a condition verification, which requires computations done beforehand. These computations might take too long, and to optimize the branch mechanism, the processor can "guess" if the condition will be verified or not. The processor can then start executing the following instructions while the computations are still running. When the computations are done, either the guess was correct, and the execution proceeds, or it was incorrect, and the execution rolls back to the branch, and the other execution path is taken. Although the instructions were "canceled," they leave traces in the processor, notably in caches, which an attacker can exploit. For example, Meltdown and Spectre can be used to access higher-privileged data.

At this microarchitectural level (also at the physical level), the application of the attacks to our example of the PIN verification code is less evident. Sometimes, attacks (like Meltdown) can be used directly. Sometimes, the exploitation of microarchitectural or physical vulnerabilities is used to perform a higher-level attack.

2.3 Physical Level

At the physical level, we consider hardware elements to be the source of vulnerabilities. Observation attacks, such as side-channel attacks (SCA), are "passive" attacks. An attacker measures the leaked physical information of a target, which includes timing, power consumption, EM leakage, temperature changes, etc. without tampering with it. Historically, the main target of SCA was cryptosystems, with the goal to retrieve part of a secret key used for decryption. These systems exist in a microarchitectural context, as they are implemented in a general-purpose processor or have their own unit. For example, this includes memory accesses, often through caches. SCA methods such as PRIME+PROBE [47] and FLUSH+RELOAD [51] monitor the cache accesses. More specifically, an attacker

can retrieve precious information, for example, on the secret key used in a cipher, by comparing the different amounts of time needed to access data in caches by the algorithm. This is an example of a porous attack between two levels: although it is the microarchitecture that is targeted, an attacker measures physical data such as time to perform their attack. Today, applications for SCA are more diverse. It can be used for malware identification [40] or to retrieve information on applications other than cryptographic, such as neural networks [11].

If observation attacks are called "passive", fault injection attacks (FIAs) (also mentioned interchangeably as FIs) are "active": by perturbing the target signals, a fault alters its behavior. Moreover, a fault affects physical and electronic elements (such as flip-flops or transistors) but has repercussions at all levels. Since FIAs are our core interest in this thesis, they will be explored in length in the following chapters.

Chapter Conclusion

Various methods exist to exploit vulnerabilities. Although it is not an absolute concept, we divided attacks into three levels: software, microarchitectural, and physical. Some attacks exploit vulnerabilities from one level, which then have repercussions on other levels. Among these attacks, fault injections are a perfect example of this phenomenon. Although considered here as physical attacks, faults are mostly exploited at the microarchitectural level [43, 26, 50]. Understanding precisely a fault's effect at several levels simultaneously consists of a complex task. To do so, we must first have a solid knowledge of the possible effects a fault has.

Chapter 3

Fault Injection

In this chapter, we give basic knowledge about fault injection. In order to inject faults, it is essential to have knowledge on the impacted elements, as well as the different levels of fault interpretation. Fault injection, as described in Chapter 2, is a physical means of attack: it affects electronic elements such as transistors or D flip-flops (DFFs). The chosen fault injection mean determines partly which elements are going to be faulted: for example, clock glitching affects clocked elements. This being said, these electronical elements often exist in bigger structures, such as processors. Thus, the fault does not only impact a transistor or a DFF, it also affects cache transfers, pipelining, instructions, etc. Interpreting the fault effects, i.e. finding fault models, can then focus on different levels. For example, why EMFI prevents a DFF from sampling, or why it causes an instruction to be skipped. Depending on the target, the interpretation is more difficult or even impossible: most fault campaigns are done on commercialized integrated circuits, and the amount of control and information on their implementation a user has can be limited.

3.1 Knowing your target

3.1.1 A bit of electronic

First, we define the most common (or at least the most frequently mentioned in the literature) electronic components (schematized in Figure 3.1) susceptible to fault injection: transistors, logic gates, and DFFs. Transistors, when combined in certain ways, form logic gates. A particular kind of gate, the not gate, is used to make DFFs.

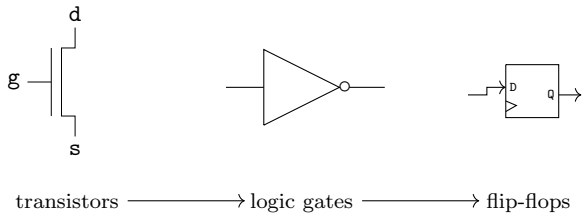


Figure 3.1 – Vulnerable electronical elements

Transistors The most basic component of a circuit is the transistor. For the scope of this thesis, transistors will be considered as switches that control the flow of current between two terminals, the drain `d` and the source `s`, depending on an input `g`. There are two kinds of transistors, nMOS and pMOS as depicted in Figure 3.2, that have complementary behaviour. When `g` equals 1 in nMOS transistors, the source and the drain are connected, thus closing the switch while they are disconnected in pMOS transistors. Inversely, when `g` equals 0, the source and the drain are connected in pMOS transistors and disconnected in nMOS transistors.

Figure 3.2 – Simplified nMOS and pMOS transistors

Logic gates As their name suggests, they perform logic operations such as `not`, `and`, `or`, etc. Let us take the example of a not gate. It is made of pMOS and nMOS transistors, linked to each other, as well as to Vdd and Vss, as depicted in Figure 3.3.

Power and ground

Vdd (or power) is a positive voltage signal. It can take different values depending on the context such as 3.3V or 1.5V, for example. Here, *Vdd* will represent a logic '1' value.
Vss (or ground, *Gnd*) is usually at 0V. Here, *Vss* will represent a logic '0' value.

When the `input` is at 0, the pMOS transistor conducts (and the nMOS transistor does not), which means that *Vdd* is connected to the `output`, which is then set to 1. When the `input` is at 1, the nMOS transistor conducts (and the pMOS transistor does not), which means that *Vss* is connected to the `output`, which is then set to 0.

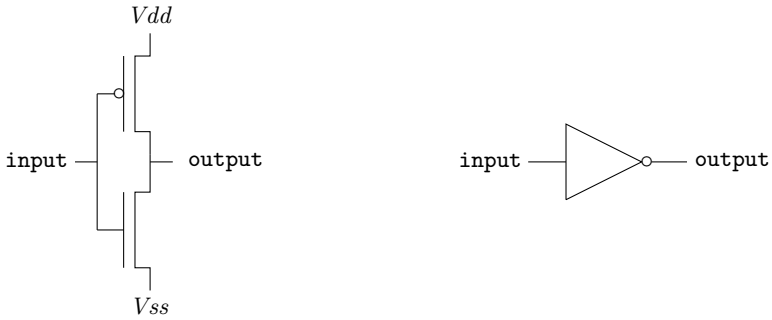


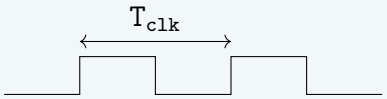
Figure 3.3 – not gate

All other logic gates are also made of a combination of transistors. It is then possible to build combinational circuits made of logic gates. Another component necessary to memorize data to build sequential circuits is the register.

Registers The main type of register considered in this thesis is the D-type Flip-Flop (DFF). First, we need to define what is a latch. Latches are level-sensitive devices: the state of the clock signal determines their behavior.

Clock

The clock is used to set the cadence of a circuit. The signal oscillates between 0 and 1 every half period $T_{clk}/2$.



Latches can be positive-level-sensitive (resp. negative), meaning they let data through (transparent mode) when the clock is at 1 (resp. at 0). When the clock signal switches to 0, the data is stored in the latch and stays latched until the next corresponding clock state changes (latch mode).

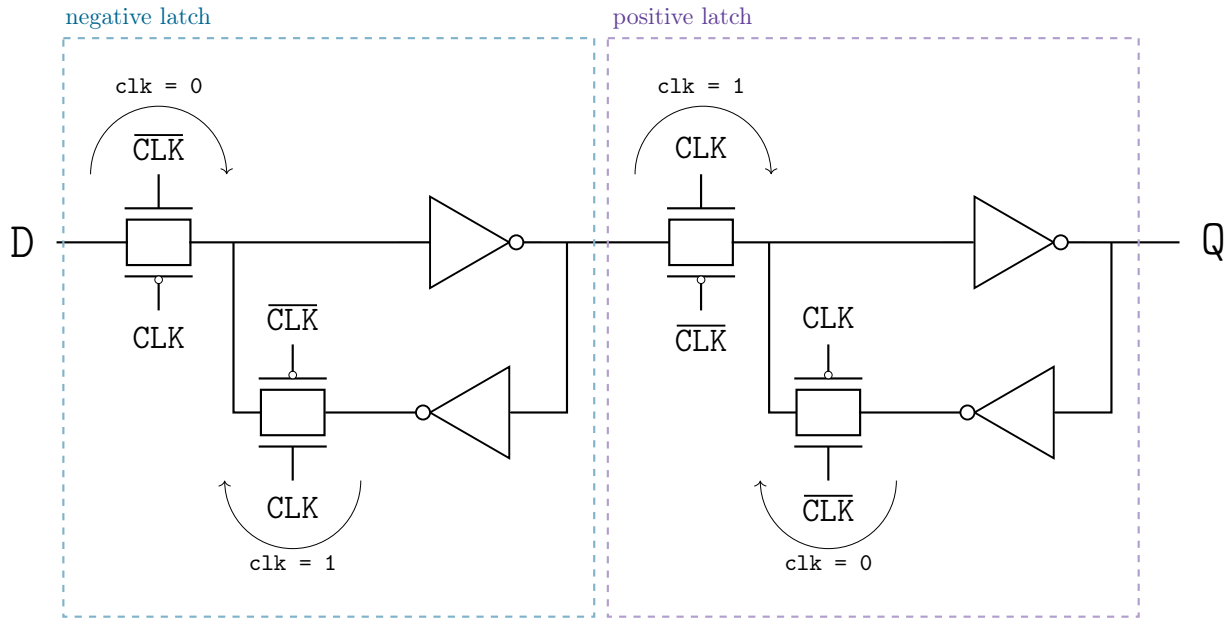


Figure 3.4 – Schematized DFF

A DFF, as depicted in Figure 3.4, is composed of two level-sensitive latches, a negative and a positive one. When the clock signal is at 0, the first latch is in transparent mode and lets the data through, while the second is in latch mode and keeps the previous input value. When the clock signal is at 1, the first latch is in latch mode while the second is in transparent mode, and the data goes through the second latch. The instant when the clock rises from 0 to 1 is called the sampling time. We call T_{D2Q} the amount of time it takes for the signal to go from D to Q after a clock rising edge.

3.1.2 Let's make a circuit

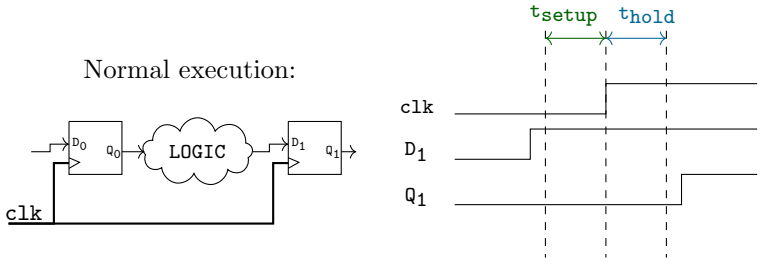


Figure 3.5 – Normal execution of a simple synchronous circuit.

Using the components we defined above, we can make a circuit. An example composed of two DFFs with some logic in between is depicted in the left part of Figure 3.5. For proper sampling (or storage) to occur from $\mathtt{D_1}$ to $\mathtt{Q_1}$ in a DFF, the data coming from $\mathtt{D_1}$ must be stable during the setup and hold time window, defined by $\mathtt{t_{setup}}$ before and $\mathtt{t_{hold}}$ after the rising edge of `clk`, respectively. This is illustrated in the right part of Figure 3.5.

Applied to the simple circuit of Figure 3.5, this window is conditioned by the propagation time of the signal from $\mathtt{Q_0}$ to $\mathtt{D_1}$, called $\mathtt{T_{prop}}$. With $\mathtt{T_{clk}}$ the clock period and $\mathtt{T_{D2Q0}}$ the propagation delay between $\mathtt{D_0}$ and $\mathtt{Q_0}$, the following equation must be verified otherwise a timing violation may occur:

$$\mathtt{T_{clk}} > \mathtt{T_{D2Q0}} + \mathtt{T_{prop}} + \mathtt{t_{setup}}$$

If the right member of the equation is less than the left member, the surplus of time is called the `slack`.

3.1.3 What makes a processor?

Now that we know some basic electronic components and how they interact with each other, we can build more sophisticated circuits such as processors (or processing units). In electronic devices (such as smartphones and computers), processors are the central piece. Processors are often composed of many elements, schematized in Figure 3.6, that interact with each other. Whether playing music, recording a video, or accessing protected data, one (or several) processor is required to perform the operations and communicate with other elements such as memories, sensors, or peripherals. How a processor behaves is defined by its microarchitecture, which depends on its instruction-set architecture (ISA).

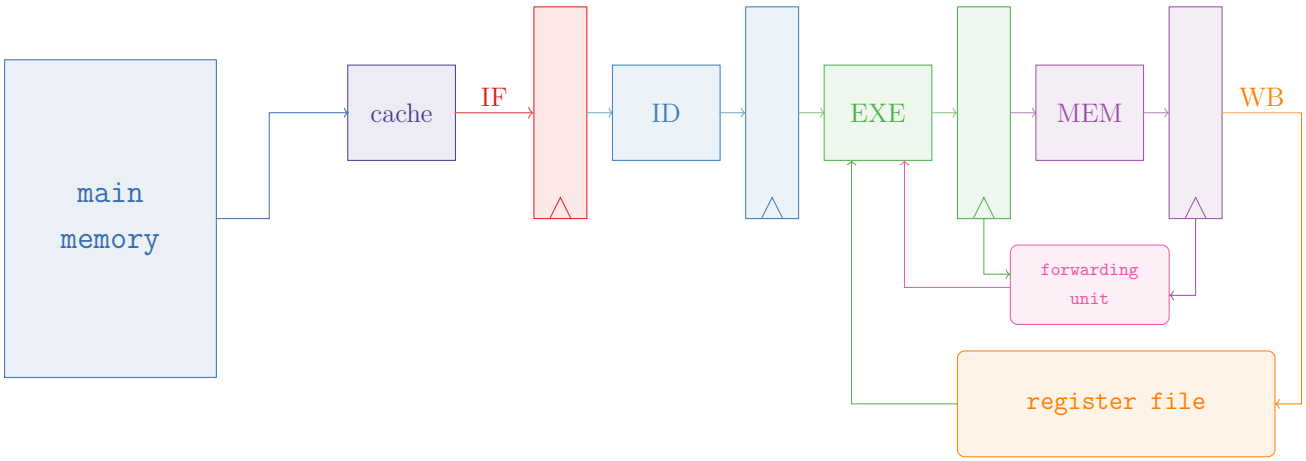


Figure 3.6 – Schematized 5-stages in-order pipeline

Pipeline

To improve performance, most processors implement a pipeline, as schematized in Figure 3.7. To allow instructions to be executed in parallel, each instruction is split into several steps or pipeline stages. Let us assume that each step takes one clock cycle to complete. Each step uses a different part of the processor, meaning that successive instructions can be executed simultaneously as long as they are not at the same step.

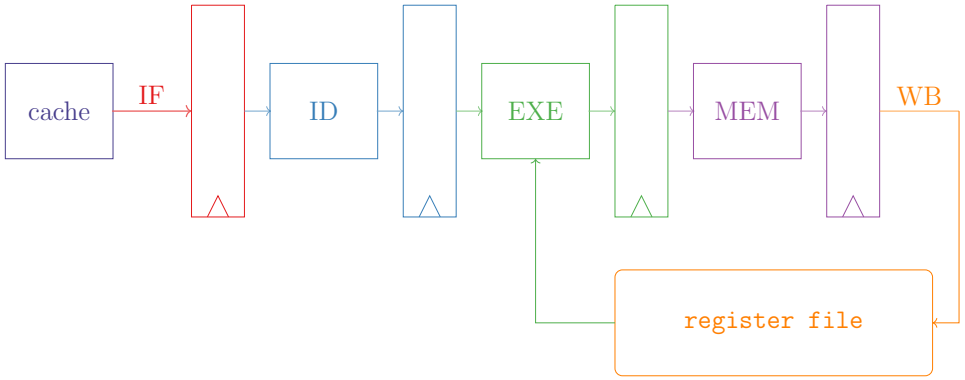


Figure 3.7 – Schematized 5-stages in-order pipeline (simplified)

The number of stages varies between processors. Figure 3.7 illustrates a 5-stage pipeline:

1. Instruction Fetch (IF): fetch the instruction from memory and update the Program Counter (PC)
2. Instruction Decode (ID): decode the instruction

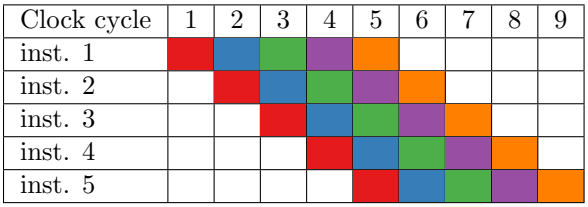


Figure 3.8 – Pipelined instruction chronogram ; each colored case corresponds to the data sampled in the corresponding colored register or the register file

3. Execute (EX): read the needed registers from the register file and execute the operations of the instruction
4. Memory access (MEM): if required, read or write data in memory
5. Write Back (WB): write the results of the previous step into the appropriate register

With a 5-stage pipeline, five instructions can be executed in parallel. At each clock cycle, an instruction achieves completion, as represented in Figure 3.8.

This simplified explanation does not consider instructions that require more than one clock cycle to complete. For example, arithmetic operations such as multiplication, or instructions that require fetching data from memory. As it is, the pipeline we introduced would need to stall for a few cycles to perform such operations. To avoid this situation, we introduce two additional optimization mechanisms linked to the pipeline: cache memory (and prefetch buffer) and forwarding.

Caches and prefetch buffer

To store data and instructions, processors need memory components that are accessible to the processor. To put it simply, if the access time of a memory is small (if retrieving data from the memory takes little time), then it is expensive to implement. Conversely, memories with longer access times are cheaper. To improve performance, processors implement both types of memories. Faster memories, called caches, are small and placed on-chip to limit costs.

As illustrated in Figure 3.9 and Figure 3.10, when the processor needs some data (including instructions), it will first look into caches. If the data is there, it is a hit, and the transfer of the data to the processor is quick (Figure 3.9). If not, it is a miss, and the data needs to be transferred from other levels (further away from the processor) of the memory hierarchy, which takes additional time (Figure 3.10).

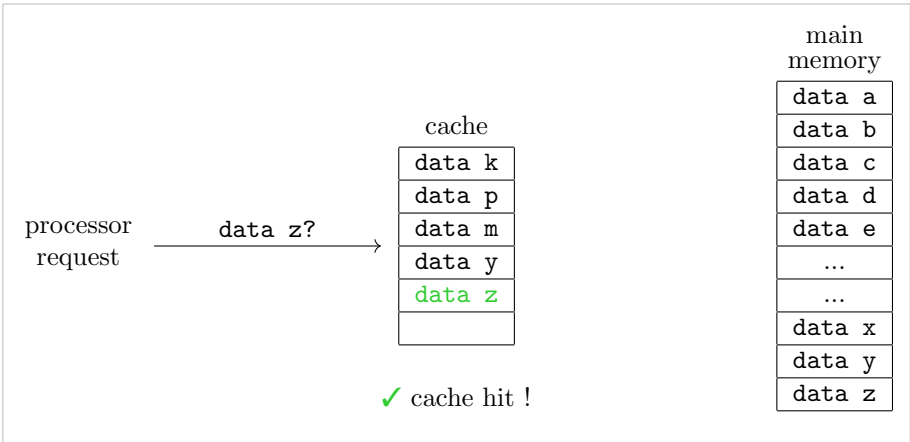


Figure 3.9 – Cache hit

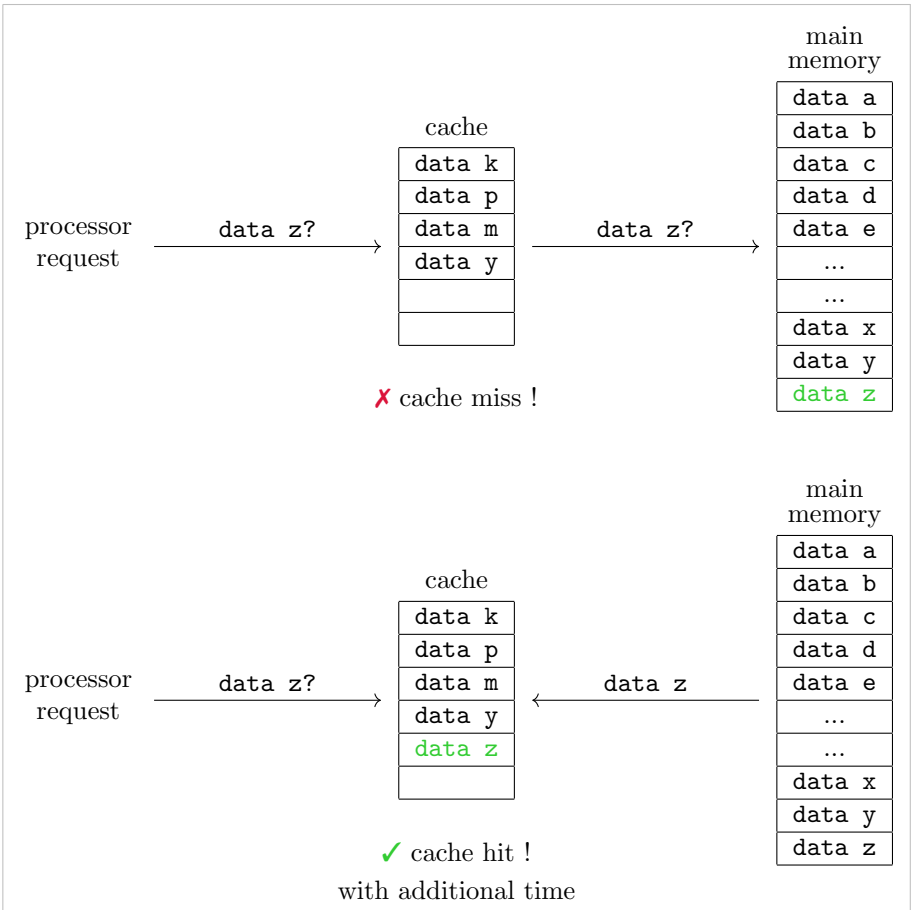


Figure 3.10 – Cache miss

In addition to caches, some processor implement a prefetch buffer: a small buffer that

can carry a limited number of instructions, accessed in priority, before any caches.

Forwarding

Some processors implement an additional optimization mechanism: the forwarding unit. To illustrate this mechanism, we take as an example a small snippet of RISC-V assembly code in Figure 3.11. *ins. 3* performs an addition between registers `t3` and `t4`

```
ins. 1   addi  t3, t3, 1 # t3 = t3 + 1
ins. 2   addi  t4, t4, 2 # t4 = t4 + 2
ins. 3   add   t5, t4, t3 # t5 = t4 + t3
```

Figure 3.11 – RISC-V assembly code example

and stores the result in register `t5`. However, `t3` and `t4` are modified in the previous instructions.

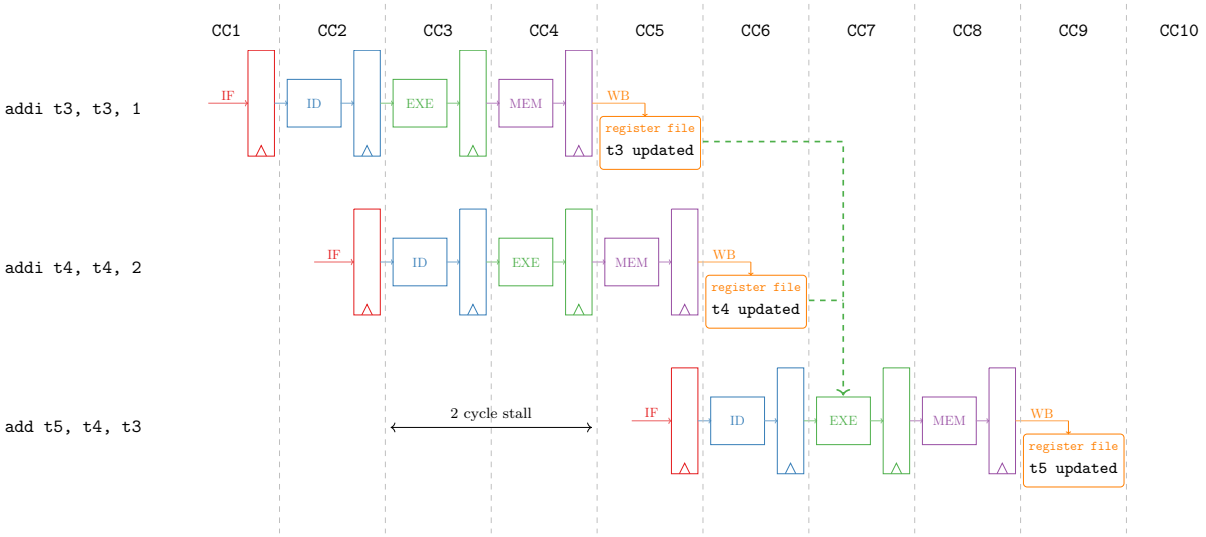


Figure 3.12 – Pipeline - without forwarding

Without forwarding, as illustrated in Figure 3.12, *ins. 1* and *ins. 2* have to complete their WB stage for *ins. 3* to access the updated value of the registers, causing a stall of two cycles. With forwarding, as illustrated in Figure 3.13, instead of only fetching the register value from the register file, it is possible to get it from the MEM or EXE stage of the previous instructions, and the instruction completes without any stall.

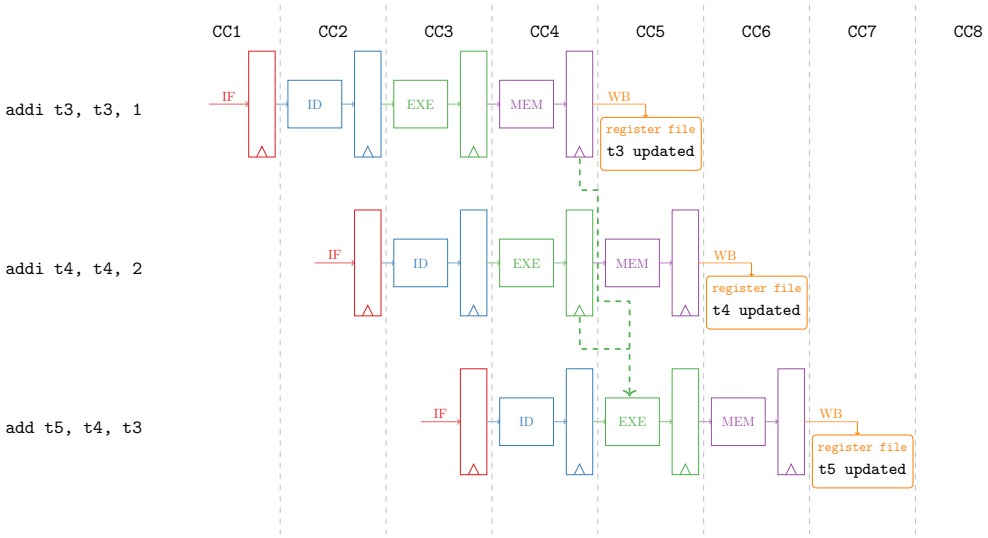


Figure 3.13 – Pipeline - with forwarding

3.2 Abstraction levels

Each section presented so far in this chapter represents a level of interpretation of the fault. The lowest level is the physical level, where fault models analyze the interaction between FI and basic electronic components. At the physical level, the goal is to understand why the photons injected from a laser pulse can switch a logic gate output or why a DFF samples an incorrect value under EMFI. This level considers the analog nature of electrical current and voltage signals.

By combining basic electronic elements into a circuit, we gain an abstraction level. This is named the register-transfer level, where a fault is modeled as a logic signal alteration. Here, the analysis focuses on how a bit flip or a 'stuck at 0' (or 1) propagates through a circuit.

Finally, at the microarchitectural level, a fault is analyzed by its impact on the microarchitecture. For instance, a bit-flip on the forwarding control signal in the pipeline can lead to an instruction skip.Microarchitectural fault models include instruction-set architecture (ISA) fault models that represent a fault as an instruction modification. In other words, at the ISA level, the consequence of a fault can be linked to one instruction being transformed into another. Some microarchitectural faults cannot be modeled at the ISA level and can impact the data cache only. Figure 3.14 summarizes these different

abstraction levels.

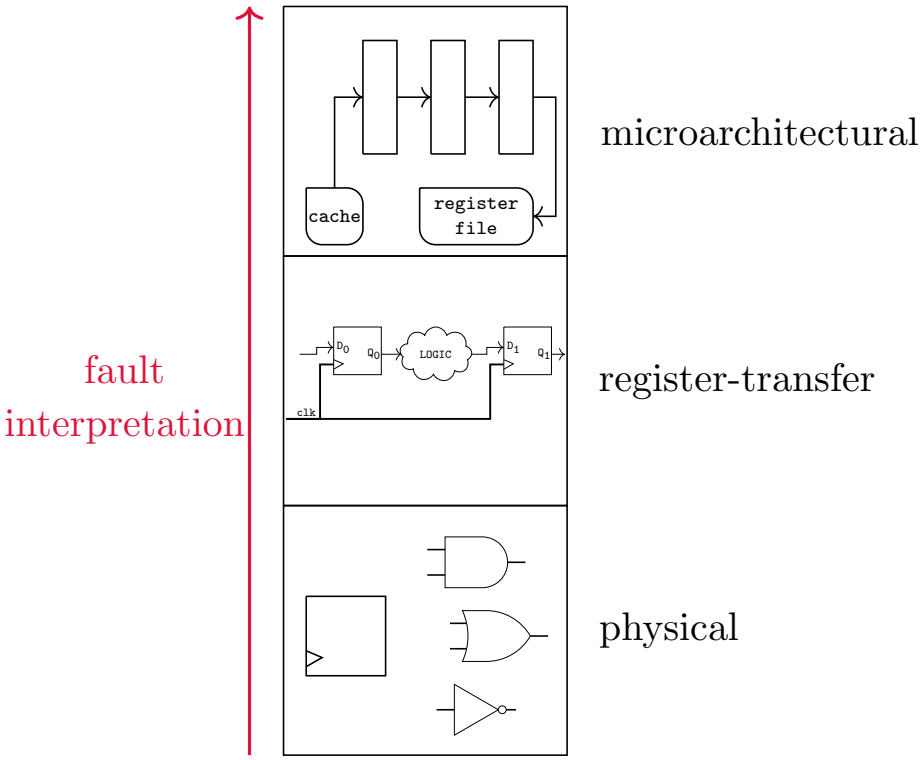


Figure 3.14 – Abstraction levels

3.3 Fault injection means

An attacker disposes of various means to inject faults. The first method of inducing faults is to push the target beyond its nominal operating conditions. For example, under-powering or overclocking the device will cause a fault in the target's critical path. These methods offer limited control over the fault.

The aim of laser fault injection is to perturb transistors by inducing localized electrical currents. At the physical level, faults induced by lasers have been studied on memory cells such as DFF [44] or SRAM cells [19], causing bit-set or bit-reset. This can translate at the microarchitectural level into instruction skip [20] or instruction and data corruption [10]. Laser fault injection has a high spatial resolution, which makes it theoretically possible to target any transistor in an electronic device with precision. However, to do so, the target has to undergo physical modifications as to ease the laser beam action (a process known

as decapping or decapsulation).

The aim of electromagnetic fault injection (EMFI) is to modify the electromagnetic field of the target device, which induces swings in signals such as the clock, *Vdd* or *Vss*. Similarly, voltage and clock glitching also induce swings in their respective networks. EM faults are injected using a probe, meaning their impact is localized. Regarding voltage and clock glitching, an attacker usually replaces the target's source with their own, and the fault impact is global to the circuit. As such, EMFI is similar to voltage and clock glitching, and the fault models associated are discussed in length in Chapter 4.

3.4 And in practice, what do we fault?

The previous sections presented out-of-context, isolated elements. When performing fault injection, we can not target an isolated register. This register is most likely part of an Integrated Circuit (IC), i.e., a device composed of electrical components designed for a specific use. This broad definition encompasses a wide range of objects, including programmable devices, memories of every kind, microcontrollers, and Systems on Chip (SoCs). For the scope of this thesis, we only focus on a subset of ICs: Application-specific Integrated Circuits (ASICs) and field programmable gate arrays (FPGAs).

3.4.1 Field Programmable Gate Array (FPGA)

Definition An FPGA is a reprogrammable IC. As illustrated in Figure 3.15, an FPGA contains the basic electronic elements defined in Section 3.1.1, grouped into Programmable Logic Blocks (PLBs). An FPGA also has a grid of wires used to connect the PLBs. At the intersection of wires, there are programmable Switching Matrices (SMs) to do so. In addition, FPGAs include proprietary elements (pre-made resources such as clock managers, etc.) whose sources are concealed.

Programming The PLBs and SMs are programmable using a Hardware Description Language (HDL) such as VHDL or Verilog. HDLs describe a circuit and its behavior over time (usually, the clock signal is used as a time reference). To transform the register-transfer level (RTL) description of the circuit into a working application (ranging from a blinking LED to a sophisticated processor softcore), we need specific software to do so, usually proprietary (such as Vivado from Xilinx) and dependent on the target FPGA.

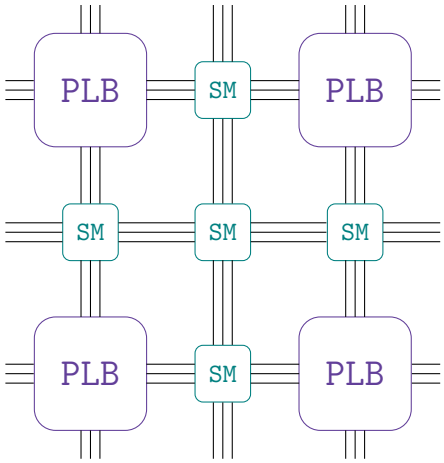


Figure 3.15 – Schematized physical layout of a FPGA

The first step of the transformation is to simulate the circuit to check the code syntax and the circuit behavior. The second step is the synthesis: the tool determines which hardware components are needed and how to link them together. It might also optimize the circuit (remove unused parts, factorize, etc.). The last step is the implementation. If placement constraints were given, the tool maps the circuit (or circuit elements) accordingly. The constraints may also specify which routes link two elements together. Otherwise, the tool maps and routes the design with time and space optimization in mind. Finally, a bitstream is created and can be uploaded into an FPGA.

Fault characterization We previously defined three levels of characterization: physical, register-transfer, and microarchitectural. For most FPGAs, we do not have access to extensive information on their layout and the proprietary elements they integrate. However, if we want to look into DFF and their behavior for example, it is possible to gather information on the fault impact by analyzing the DFF behavior.

First, since we can place and route our custom circuit at will, we can (depending on the injection method) target specific DFFs directly. Otherwise, it is possible to recover the DFF state at any time and thus locate faulted DFFs. Naturally, we can also get information at higher characterization levels. For example, at the register-transfer level (RTL), we can investigate whether the presence of logic gates between two registers influences the fault impact. At the microarchitectural level, with a processor softcore implemented in an FPGA, we can isolate the different vulnerable elements (pipeline stages, for example).

In conclusion, FPGAs appear to be ideal for theorizing fault models. Nevertheless, FPGAs are not used as much in everyday life as ASICs.

3.4.2 Application-Specific Integrated Circuits (ASICs)

Definition ASICs can be microprocessors, microcontrollers, or more complex Systems on Chip (SoCs). Our interest in this thesis mainly focuses on processors. The same electronic components as in FPGAs are present, but these ICs are mostly proprietary, so we may lack informations on the core layout and operation, which can be quite complex.

Programming We program applications to send to the core using general-purpose programming languages (such as C) compiled for the target ISA (ARM, RISC-V, etc.), which will be executed by the core (either bare-metal or through an Operating System).

Fault characterization The first challenge regarding fault injection on ASICs is to identify the vulnerable physical parts of the processor. When using spatial fault injection methods such as EM or laser, multiple fault injections are performed all over the target to associate different zones with different fault models. Regarding power or clock glitching, the fault effect is global in the processor, meaning that, in theory, any element receiving power or clock signals can be faulted.

Given the lack of information about the physical layout, it is hardly possible to gather experimental clues that confirm or deny fault models, both at physical or register-transfer levels. Concretely speaking, fault models are mainly theorized at the microarchitecture level. Contrary to FPGAs, ASICs are the most targeted ICs, as they are found in everyday electronic devices. Fault models, outside of their theoretical aspect, are used to both lead successful fault injection campaigns and develop countermeasures, meaning that understanding the fault impact at this level is crucial.

Chapter conclusion

This chapter aims to answer the following question: when performing fault injection attacks, what elements are affected? Depending on the interpretation level, the answer differs. If the goal is to design countermeasures at the physical level, then we will consider that electronic elements are impacted. On the other hand, if we want to perform a fault injection campaign to bypass security measures, we will look into the fault effect on the

microarchitecture. Additionally, reaching our goal is also conditioned by the fault injection means and the target.

In this chapter, we presented four means of fault injection: laser fault injection, EMFI, voltage glitching, and clock glitching. For the remainder of this document, we only explore the effect of the latter three because their impacts are similar. An overview of the fault models that apply to these three fault models is presented in Chapter 4.

Chapter 4

Analyzing the Fault Impact

Although electromagnetic fault injection (EMFI), voltage and clock glitching differ in their injection methods, some of their impacts are similar. For the sake of simplicity, in this chapter, we present fault models tied to EMFI. However, some models could also be applied to either voltage or clock glitching. The first section of this chapter describes the general impact of EMFI. The next three sections present fault models for each abstraction level (physical, register-transfer, and microarchitectural).

4.1 General Impact

4.1.1 Wires

In Chapter 3, we mentioned that ICs contain wires such as power, ground, and clock. It was theorized in [17] that wires that form loops, such as power and ground wires, are more likely to be impacted by an electromagnetic pulse (EMP). More precisely, wire loops have a magnetic flux passing through them. When this flux encounters an EMP, it is affected, which induces a parasitic current (swings, either positive or negative, in the signal that is proportional to the EMP amplitude) in the loop. The induction current provoked by the EMP mainly happens at the edges of the probe, with a much smaller impact at the center of the probe. Two EMP-wire couplings can happen: one with *Gnd* and one with *Vdd*. Within the same injection, both wires located underneath the probe are affected simultaneously but asymmetrically. An IC contains many more wires, and it would be hazardous to rule out couplings with other wires.

4.1.2 Capacitance

In [30], Liao *et al.* describe the Charge-based Fault Model. This model posits that an EM pulse influences the circuit's capacitance. When a circuit is overclocked or powered with sub-nominal voltage, the amount of charge present is closer to the threshold required

to flip a DFF under normal conditions. As a result, EM pulses can more easily perturb DFFs. While it presents an interesting concept, it is primarily supported by experimental data; no comprehensive explanatory model or simulation has yet been proposed. The Charge-Based Fault Model does not provide a clear explanation of why a DFF would store erroneous data. It only states that charges influence this outcome. Instead, it can be viewed as a description of how other factors might aid in facilitating EMFI.

4.1.3 PLL

To the best of our knowledge, the first mention of the influence of EMFI on the PLL was in [52]. The objective was to use the PLL as a detector for EMFI. The authors consider the booting-up phase of the PLL, where it transitions from an "*unlocked*" state to a "*locked*" one. This transition does not take the same amount of clock cycles with and without EMFI, thus demonstrating the sensitivity of the PLL to EMFI.

Claudepierre *et al.*[8] have built on this work. When doing their own EMFI experiments, they notice that the clock is significantly impacted. More precisely, the injection modifies a clock cycle: the rising edge does not reach the high state because the injection causes a drop in the signal until the next clock cycle. The glitched clock cycle delivers less energy (it has a lower voltage for a shorter duration) but remains synchronous. The authors also show that the injection may eliminate the cycle altogether, suggesting that the characteristics of glitched clock cycles may vary between injections. In this thesis, this specific clock glitch is referred to as synchronous clock glitch. Considering Yuan *et al.* work, they assume that the PLL is the component vulnerable to EMFI. They bring two additional arguments. The first one is of a geographical nature. Even if Claudepierre *et al.* do not have the exact layout of their target chip, they hint at the position of the PLL thanks to two pins carrying signals feeding it. When performing fault injections, this area is the most sensitive. Furthermore, when the PLL is deactivated, they don't observe any fault. Although Claudepierre *et al.* identify the PLL as the vulnerable part of the chip, they do not offer insight into the related mechanisms and why, when faulted, this triggers incorrect behavior.

4.2 Physical Fault Model

While EMFI has a global impact over a circuit, most physical fault models, presented in this section, aim to explain how it influences DFFs behaviour.

4.2.1 Timing Fault Model

The Timing Fault Model (TFM) was the first of its kind to be proposed. In 2008, Selman *et al.* [45] demonstrated that underpowering a circuit could lead to errors due to setup time violations. Subsequently, in 2010, Agoyan *et al.* [2] showed that shifting a clock's rising edge in time can trigger similar effects. As illustrated in Figure 4.1, D_1 is unstable during the t_{setup} time. This violation of the timing constraint can cause Q_1 to enter a metastable state, potentially leading to a fault. In other words, when the input timing constraints are not met, the value of Q_1 becomes non-deterministic. In this context, metastability refers to the phenomenon where a non-deterministic output is generated if the DFF signal constraints are not adhered to. This setup time violation can occur due to a reduction in the supply voltage of the logic, which extends its execution time (for instance, by underpowering the circuit), or by advancing a clock cycle, resulting in the logic having insufficient time to execute before the next rising edge. This model was

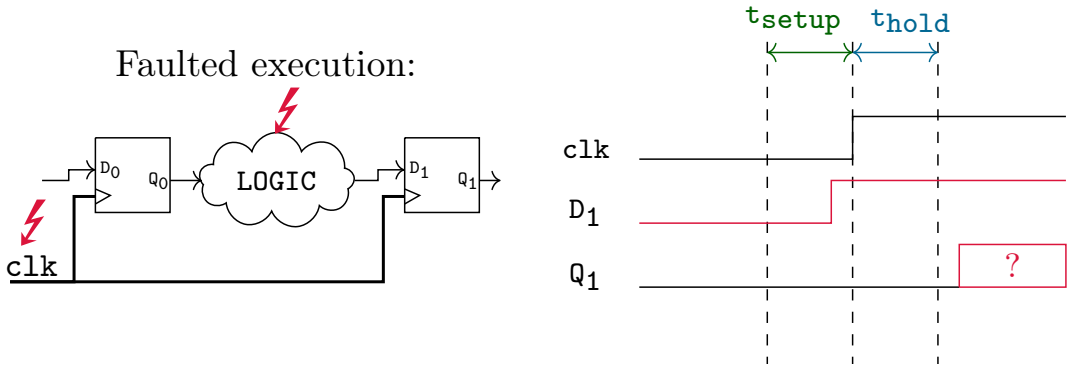


Figure 4.1 – Timing Fault Model on a simple circuit.

initially suggested [15] to explain the effects of EMFI.

The Timing Fault Model in a nutshell

→ data dependant
→ slack dependant
→ impacts power network
→ impacts critical paths first
→ fault window is timing independant

4.2.2 Sampling Fault Model

In [38], the authors question if an EMP only causes faults at the circuit level, as introduced by the Timing Fault Model. Therefore, they propose their own fault model. The Sampling Fault Model (SFM) says that faults happen at the gate level, meaning that it is the DFFs' behavior (more precisely, the sampling) that is impacted by the EMP.

The authors have developed specificities for their model to determine if a fault is due to the Timing Fault Model or the Sampling Fault Model. First, they propose to increase the target slack to prevent timing violations, thus preventing faults explained by the Timing Fault Model from occurring. Then, they look at the timing of a fault. In the case of the Timing Fault Model, an EMP injected whenever in the clock cycle should lead to a fault. In the case of the Sampling Fault Model, faults only happen around the rising clock edge because the sampling process is impacted.

To the best of our knowledge, the most recent description of the Sampling Fault Model is found in [17]. This article explains that an EM-pulse induces parasitic currents in wire loops located beneath the probe. Consequently, as depicted in Figure 4.2, fluctuations occur in the current of `Vdd` and `Gnd`, causing voltage bounces and drops, depending on the injection polarity, which impacts all other wires in the circuit (the clock tree, DFF routing, and others). In scenarios where a drop occurs, the pulse causes $\texttt{S} = \texttt{Vdd} - \texttt{Gnd}$ (and consequently all others signals) to temporarily decrease, halting circuit operation. Depending on when `S` crosses the required energy threshold for all signals to return to their nominal values, a fault may appear:

→ if `S` crosses the threshold before the nominal rising edge of `clk`, then $\texttt{D}_1$ has enough time to go back to normal and respect the timing constraints
→ if `S` crosses the threshold just before the nominal rising clock edge of `clk`, then a race between $\texttt{D}_1$ and `clk` ensues. Since the clock buffers are more powerful than

the datapath ones, `clk` is more likely to recover its initial state while D remains at 0. The authors define this as a violation of the t_{setup} time constraint, leading to a fault.

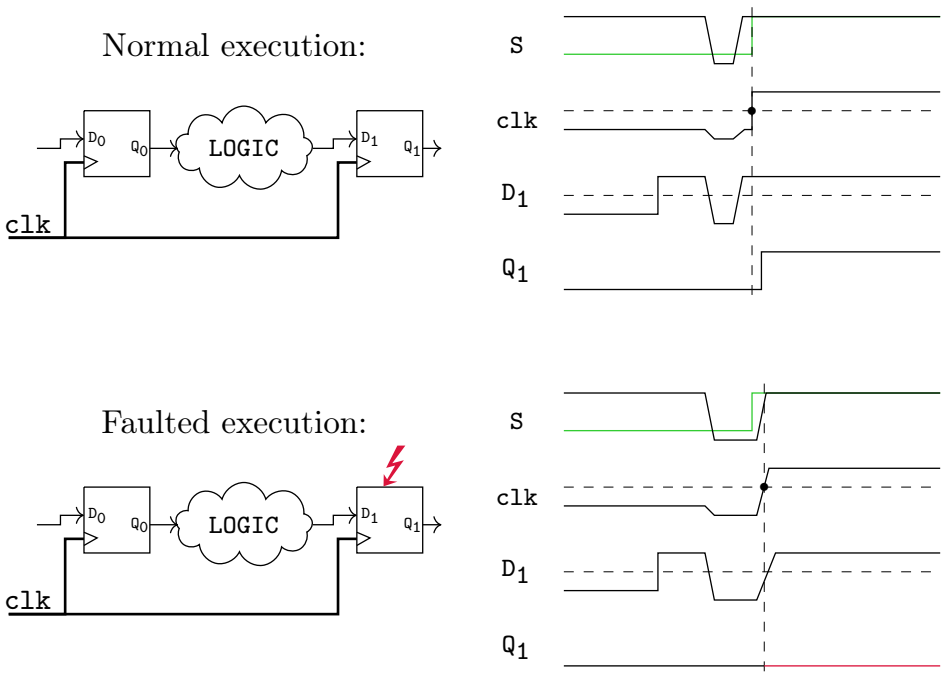


Figure 4.2 – Sampling Fault Model on a simple circuit. The green signal represents the nominal behavior of the clock.

This model has been further explored and reproduced in [53], with refinements to the coupling model and coverage of cases not previously examined: how to create a fault with a positive pulse when the DFF input is high. A distinctive characteristic of the Sampling Fault Model is the fault sensitivity window which has been experimentally confirmed as a specific timing window relative to the rising edge of the clock, during which faults can be induced. Because of the race condition between `clk` and D_1, a fault can only occur within a narrowly defined timing window around the clock's rising edge. This sensitivity window remains constant for a given circuit, irrespective of its frequency, but is influenced by the logic situated before the targeted DFFs.

The Sampling Fault Model in a nutshell

→ data dependent
→ slack independent
→ impacts all wires
→ impacts DFFs, depending on the probe position
→ fault window constant, around rising clock edge

4.2.3 Nabhan's Fault Model

In [37], the authors question the legitimacy of the Sampling Fault Model. They find inconsistencies in the Sampling Fault Model based on their experiments and subsequent observations:

— The injection window (period of time where an EMP causes faults) shrinks when the target's frequency grows, whereas the Sampling Fault Model says that this window should be constant independently of the frequency.
— The faults are data dependent, which contradicts the authors' understanding of the Sampling Fault Model.
— The fault window length is tied to the propagation path of logic, but the Sampling Fault Model says that faults happen in DFFs, so logic before it doesn't matter.
— They do not observe a link between the kind of fault (bit-set and bit-reset) and the polarity of the injection, contrary to the Sampling Fault Model.

Furthermore, the authors of [37] theorize that it is the clock wires that are impacted by EMFI, whereas the Sampling Fault Model's authors think it is the power and ground network. Nabhan *et al.* propose their fault model based on EM-generated clock glitches only. Since the authors do not name their model, we will call it Nabhan's Fault Model (NFM) in this thesis.

This model states that the EM pulse will trigger either a bounce or a drop of the clock signal, depending on the pulse polarity. It echoes the swing model in the Sampling Fault Model. They observe three cases, as illustrated in Figure 4.3.

1. The swing does not trigger any effect if
   — a drop happens when the signal is already low,
   — a bounce happens when the signal is already high.
2. The swing causes a shift of the clock's rising edge if

— a drop happens just after the rising clock edge, the edge is shifted later,
— a bounce happens just before the rising clock edge, the edge is shifted earlier.

3. The swing creates an additional clock cycle when
   — the signal is high, a drop cuts a regular clock cycle into two shorter cycles,
   — the signal is low, a bounce causes the signal to rise enough to be considered as a proper clock cycle.

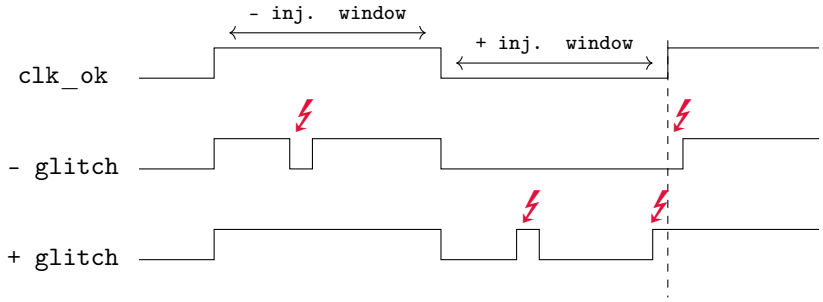


Figure 4.3 – Impact of negative and positive swing on the clock signal

The perturbations described in cases 2 and 3 may cause timing violations of t_{setup}. Nabhan shows that this depends significantly on the slack size. He considers two situations:

— the target path has a large slack, i.e., superior to half the clock period, as depicted in Figure 4.4, and
— the target path has a small slack, i.e., inferior to half the clock period, as depicted in Figure 4.5.

The Nabhan's Fault Model in a nutshell

→ data dependent
→ slack dependent
→ impacts clock network
→ impacts DFFs, with the main fault mechanism being timing violation of t_{setup}
→ fault window changes depending on the target frequency

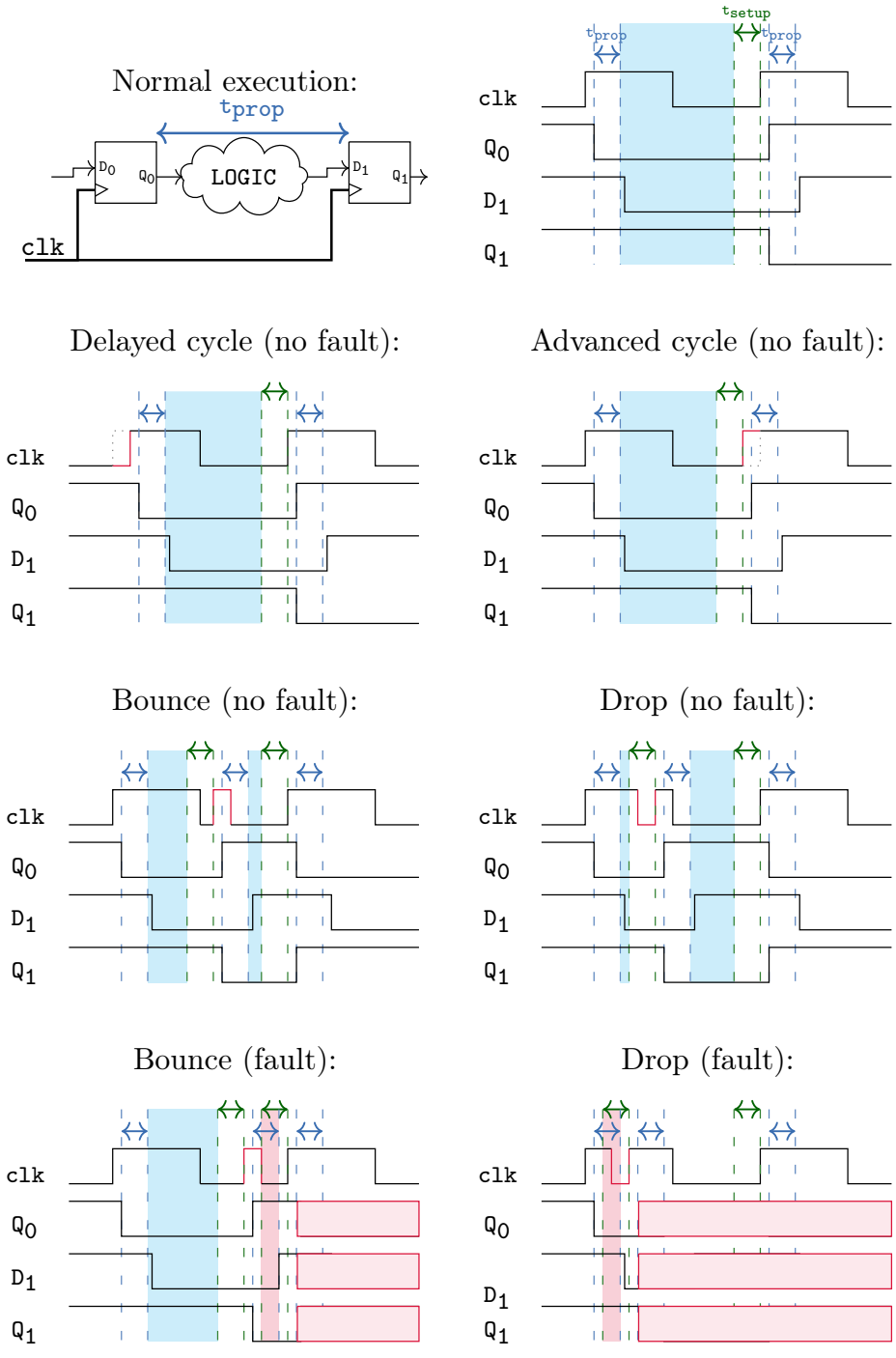


Figure 4.4 – NFM with large slack. The light blue area represents the slack. The purple squares represent the signal in a metastable state due to the timing violation.

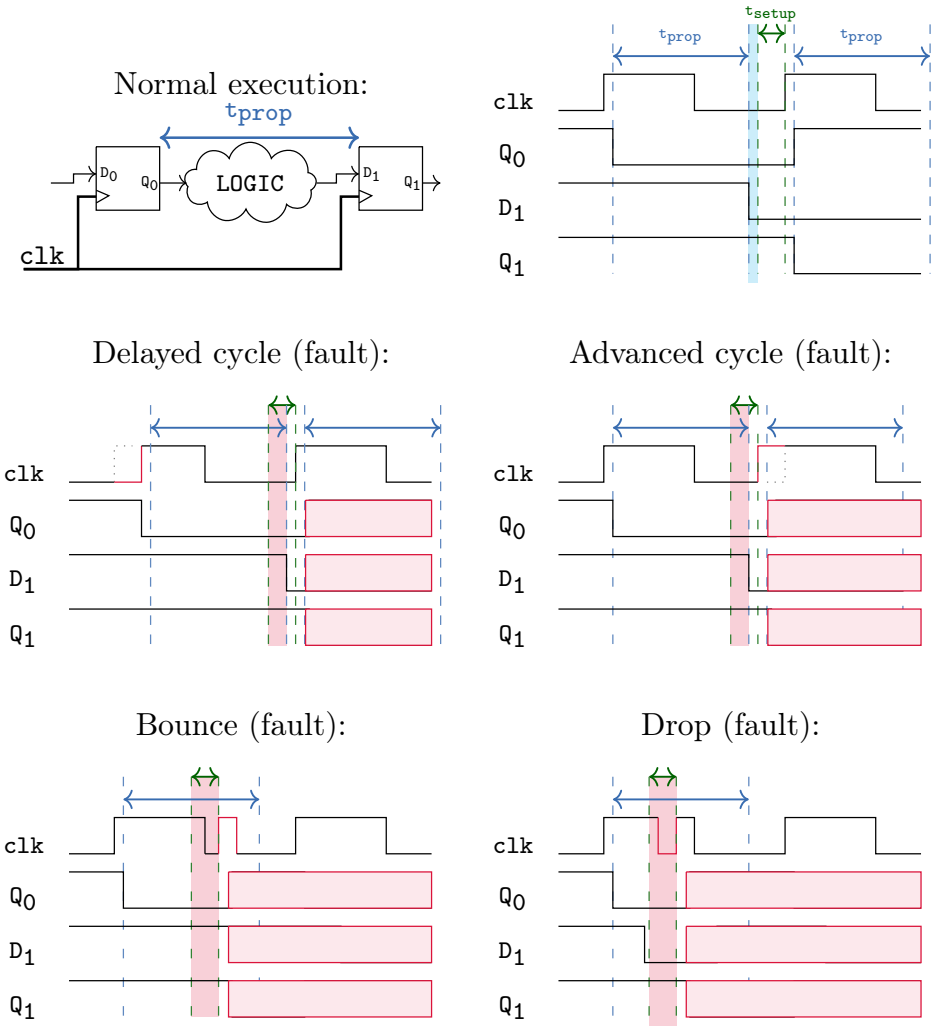


Figure 4.5 – NFM with small slack . The light blue area represents the slack. The purple squares represent the signal in a metastable state due to the timing violation.

4.2.4 Theoretical additional effects

Fault injection means such as EMFI have several complex effects on IC. When studying the previously presented fault models, we noticed that some of these effects, although theoretically possible, were not described.

Sampling Fault Model In the SFM, the fault comes from the clock winning the race against $\mathtt{D}_1$, provoked by a drop of all signals. This being the case, the DFF registers a value D that has not reached its nominal value. The authors of the SFM describe this as a timing violation. Our interpretation differs: if D has not reached the threshold towards its nominal value when the sampling happens, then this is not a timing violation (because D didn't change during $\mathtt{t}_{\mathtt{setup}}$), but the sampling of the wrong value. Since metastability is not part of this, the fault rate should be 100%. Then, D may cross the threshold during $\mathtt{t}_{\mathtt{setup}}$, causing a timing violation.

In addition, we would like to point out another cause of fault where D wins the race against the clock but not early enough, causing a timing violation of $\mathtt{t}_{\mathtt{setup}}$. This induces a metastable output. Thus, the probability of fault is not equal to 1. The clock signal goes back faster than the other signals, which might be unlikely unless D suffers a less powerful bounce than the clock.

Nabhan's Fault Model The NFM states that an EMP happening just after (0.8 to 1.2ns) a clock rising edge delays it. "Just after" indicates that the edge rises for a short time before the electromagnetic pulse (EMP). In Chapter 5, our simulation shows that for a DFF to sample, the clock energy required is actually "quite low". From these two statements, we deduce that sampling is possible before the EMP. The causes of fault might be different or even double: two timing violations back to back. We also want to introduce two phenomena not mentioned by Nabhan that could be a source of the fault. When clock cycles are close (high frequencies), an EMP could either "link" two clock cycles or deny one. This would lead to a missing clock cycle in both cases.

4.3 RTL fault models

In this section, we explore fault models at register-transfer level. Most fault models presented in this thesis rely on experimental verifications and often simulations as well. In their majority, these models focus on either the microarchitectural level or the physical

level. The propagation of the physical perturbation up to the microarchitecture, i.e., an RTL fault model, is rarely explored, at least experimentally. In other fields, such as cryptography, this level of abstraction is commonly used to model the impact of faults on cryptographic algorithms. This section details how an additional clock cycle modifies a circuit's behavior.

Alshaer *et al.* [3] offer an RTL fault model that explains the impact of an additional clock cycle at the microarchitectural level. In a pipelined processor, executing instructions

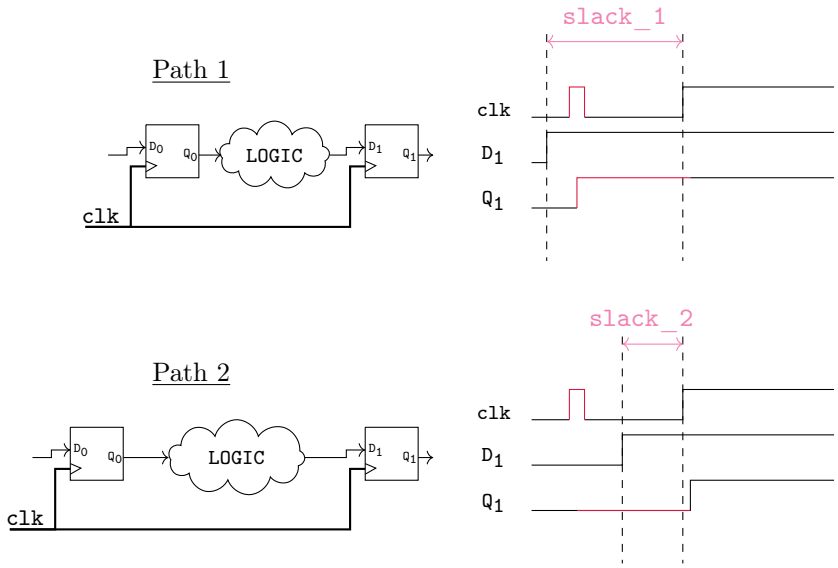


Figure 4.6 – Partial update fault model

takes several steps. The first one consists of fetching the instruction from memory. Concretely speaking, this means that each bit of the instruction's binary encoding will transit from one register to another. For Alshaer *et al.*, this transition is vulnerable to fault. At a physical level, the bits transfer corresponds to a signal (for each bit) carried out by a wire between two registers. These signals will arrive with different slack amounts, as depicted in Figure 4.6. When an additional clock cycle is injected, some destination registers will sample the newly actualized value (if the signal has a big enough slack), the previous value, or the bus's precharge value.

This model is based on a specific effect of EMFI on an IC: an additional clock cycle. How an additional clock cycle happens and why it causes faults consists of one physical

fault model, but it is not the only one proposed.

4.4 Microarchitectural fault models

At the highest level, a fault translates into a change in the program's normal execution. The first cause of perturbation comes from cache data transfers. Some processors have a prefetch buffer that fetches several instructions at once from a cache and transfers them to the pipeline, one instruction at a time. In [42], the authors demonstrate that it is possible to prevent the buffer from loading new instructions by faulting the transfer from the cache. By doing so, the buffer keeps the previous instructions, meaning they will be executed again.

In [50], the authors fault a 2-level cache hierarchy, as depicted in Figure . They show it is possible to fault multiple data transfers. Faulting a transfer between the RAM and the L2 cache shifts the position of the data in L2. Faulting a transfer between L2 and L1I (the instruction cache) results in modified instructions stored in L2. It is then possible to fault the forwarding hidden register, and these two ways are presented in [29]. First, it is possible to deactivate the forwarding; in the article, they do it to retrieve secret information. It is also possible to modify the hidden register value to modify the EXE stage arguments.

Other articles present altered program execution without giving a reason for it. In [48], the faulted instructions have modified opcodes, leading to modified operands or instruction types. In [34], the authors perform single and multiple instruction skips.

Chapter conclusion

EMFI impacts a circuit as a whole. Concerning the effects on the voltage of clock wires, EMFI, voltage and clock glitching fault models share similarities. When looking at a higher level, the fault models converge. Overall, among the various effects a fault can have on a processor, the three main explainable changes are instruction skip, instruction repeat, and instruction modification. They have the same roots, with two processor mechanisms impacted as far as we know: cache transfers and hidden registers. Most fault models either concentrate on the physical or microarchitectural level. Bridging the two levels is less common.

Among the fault effects presented in this chapter, the impact of EMFI on the PLL has not been explored outside of Claudepierre *et al.* works. Chapter 5 and Chapter 6 are dedicated to finding fault models at all levels that apply to this specific perturbation.

Chapter 5

Explaining the Synchronous Clock Glitch - Part 1: Physical Level

In Chapter 4, the Synchronous Clock Glitch (SCG) was introduced. Outside of the work done by Claudepierre *et al.* [9], where the analysis focuses only on the microarchitectural level, this perturbation has not been studied thoroughly. This chapter, as well as the next chapter, is dedicated to investigating the SCG at all levels to find a complete fault model. First, we focus on the physical and register-transfer levels.

To develop a fault model that explains why the SCG causes faults, it is essential to identify which component is most susceptible to being impacted. Given that the glitch is carried out by the clock, we hypothesize that DFFs are likely to be affected, as illustrated in Figure 5.1. The results presented in this chapter have been published at COSADE24 [32].

Furthermore, the Timing Fault Model, the Sampling Fault Model and Nabhan's Fault Model do not sufficiently explain why the SCG causes faults since:

— Only the clock is modified and other signals, particularly D_1, remain stable, there is no race condition between `clk` and D_1. Therefore, the Sampling Fault Model is excluded.

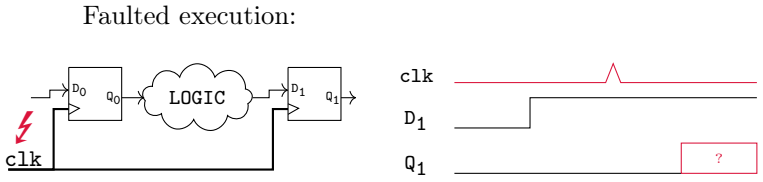


Figure 5.1 – Synchronous clock glitch impact on a simple circuit.

— There are no timing variations for either the clock or $\texttt{D}_1$, thus no setup time violations can occur, ruling out the Timing Fault Model.
— the SCG concerns only the rising edge of clock cycles, without modifying their timing. There are no additional clock cycle created either. This excludes Nabhan's Fault Model.

Given that the SCG cannot be accounted for by existing published fault models, further study is necessary to identify a fault model that accurately describes its effects.

5.1 Understanding the Synchronous Clock Glitch

In this section, we propose several hypotheses that can explain the SCG fault model.

Hypothesis (Energy Threshold) *For a DFF to correctly sample a clock's rising edge, the clock signal must meet a certain energy threshold, which represents the combination of voltage amplitude and width thresholds.*

The energy of the clock signal determines whether the DFF samples the incoming data. A failure to sample is considered a fault. Depending on the energy of the clock signal, three states of the DFF are observed, as depicted in Figure 5.2:

1. When the energy of the clock signal is too low, falling below the required energy threshold, the DFF is in a *always faulted* state.
2. Conversely, when the energy of the clock signal is sufficiently high, surpassing the threshold, the DFF is in an *always unfaulted* state.
3. When the clock signal hovers around the required threshold, the DFF enters a *sometimes unfaulted* state (i.e. when out of `X` FIs, it has sampled at least once). In this state, the output of the DFF is in a metastable state, influenced by the amount of clock energy. This phenomenon is further explored in Section 5.3.1.

This hypothesis alone is insufficient to fully explain the effect of the SCG. We propose below two additional hypotheses, following the introduction of the *fault sensitivity* concept.

Definition (Fault Sensitivity) *The minimum amplitude at which a DFF becomes sometimes unfaulted is called its fault sensitivity.*

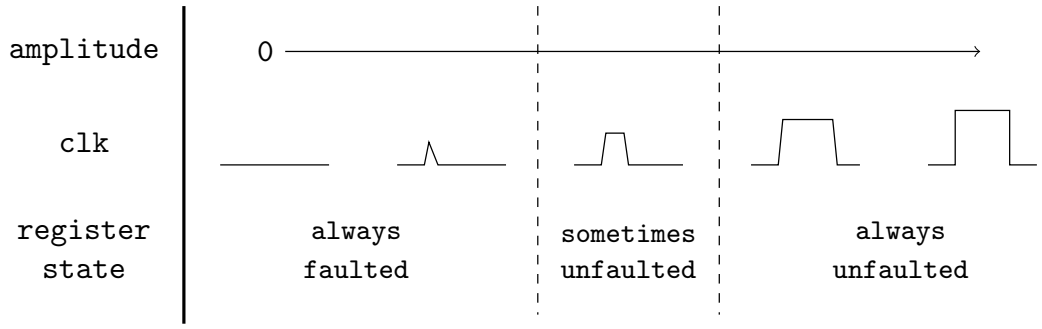


Figure 5.2 – Register behaviour depending on the rising clock edge

When faulting two DFFs, for instance, on a FPGA, their behaviors should be similar but not identical, and they may not share the same *fault sensitivity*. This difference can be attributed to variability in the manufacturing process among ICs and within individual DFFs of the same IC die. In other words, two identical DFFs, i.e., with the same characteristics and hardware layout, may not share the same *fault sensitivity*. Also, if the two DFFs are from two FPGAs of the same model, they may not share the same *fault sensitivity*.

Hypothesis (Fault Sensitivity Dependency on Intrinsic Properties) *The fault sensitivity of a DFF depends on its intrinsic properties, such as process variability and clock routing up to the DFF, among others.*

However, the intrinsic properties alone are not sufficient to explain observed variations in fault sensitivity. To add a layer of complexity, we consider the environment surrounding the glitched DFFs, specifically focusing on the wires carrying signals (e.g., `clock`, `Vdd`, `Vss` between DFFs). The energy from neighboring wires may influence the glitched clock, altering the behavior of the target DFFs. This includes both data routing between DFFs and the clock routing on the dedicated clock network.

Hypothesis (Fault Sensitivity Dependency on Extrinsic Properties) *The fault sensitivity of a DFF may also be affected by extrinsic factors, such as the activity in neighboring wires (including routing between DFFs and the routing of the clock tree).*

Experiments (either through simulation or on actual hardware) are necessary to validate these hypotheses. The experimental setup is presented in the following section.

5.2 Experimental Setup

The previous section has introduced hypotheses that may explain the effects of the SCG. In this section, we describe experiments aimed at confirming or refuting these hypotheses. The experiments are categorized into two types: physical FI and simulated experiments.

5.2.1 Physical Experiments

The SCG may be obtained with EMFI. However, this method leads to too chaotic results, so we will use a different analysis method.

TRAITOR

To reproduce EMFI clock cycle perturbations, Claudepierre *et al.* introduced an FI tool named TRAITOR [9]. TRAITOR can control the amplitude parameter, which defines the energy level of the synchronous clock glitch. This allows for more precise control over the glitch. For the remainder of this article, this perturbation is referred to as CSCG. Since there is currently no method to demonstrate equivalence, we refer to EM-induced clock glitches as synchronous clock glitch and TRAITOR-induced clock glitches as controlled synchronous clock glitch.

In their study, Claudepierre *et al.* targeted a microcontroller to analyze the TRAITOR fault model [9]. The primary induced microarchitectural fault model is an instruction skip. As a result, with its very high success rate and ability to perform a large number of faults, TRAITOR is a suitable FI tool for `NOP`-oriented programming, as explained in [39]. By carefully replacing selected instructions with a `NOP` assembly instruction, an attacker can modify a running program, akin to a Returned-Oriented Programming (ROP) attack. As demonstrated by Gicquel *et al.* in [23], without appropriate hardware countermeasures, such an attack is almost guaranteed to succeed

How does TRAITOR work?

To generate a CSCG, we can control the occurrence of the corrupted clock edge in each clock cycle and adjust a single parameter known as the **amplitude**, which shapes the corrupted edge. Figure 5.4 illustrates the generation of the corrupted edge using two phase-shifted clocks, with the phase under the TRAITOR user's control. Ideally, this

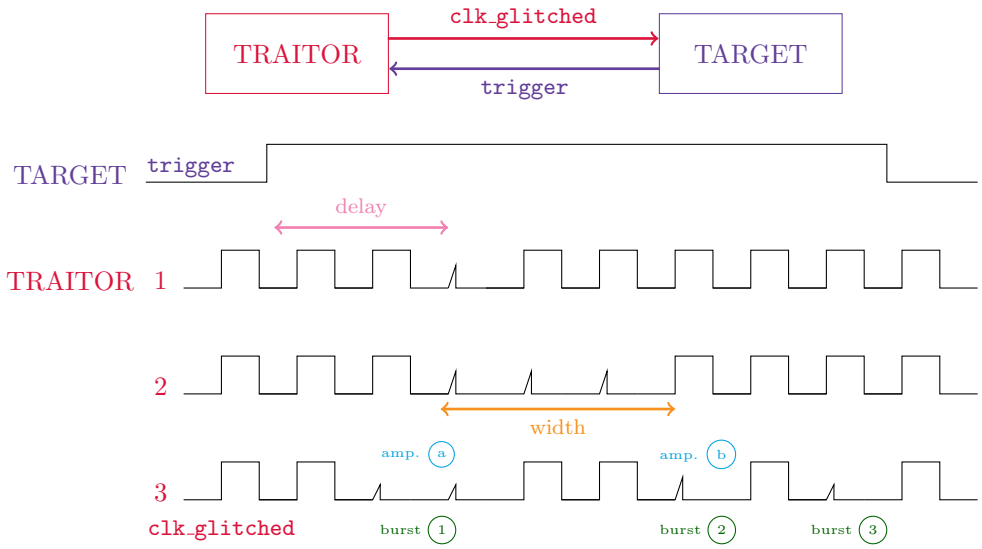


Figure 5.3 – TRAITOR's usage

method would result in a square pulse. However, the theoretical pulse width, equivalent to the phase shift, is too small relative to the circuit's inductance, preventing the signal from reaching its high value within the available time. Consequently, the **amplitude** of the corrupted edge is determined by the phase shift; a larger phase shift allows the corrupted signal to reach a higher level. Thus, the amplitude parameter influences both the height and the duration (also referred to as width) of the corrupted clock edge.

In our implementation, the phase shift is adjustable in increments of 32 ps. Throughout this paper, the term "amplitude" applied to TRAITOR will refer to the number of these 32 ps steps in the phase shift.

TRAITOR has two additional parameters. The width parameter indicates how many cycles are replaced with the CSCG. The delay parameter controls which clock cycle is affected (or the first affected in case of a width equal or superior to two). Our implementation of TRAITOR can set two different amplitudes. In this chapter, we set the delay and the width to 1, with the amplitude ranging, intending to inject a single CSCG. In other applications, TRAITOR can inject bursts (one or several successive glitches) with varying parameters.

TRAITOR produces two clocks: a regular clock, referred to as `clk_ok`, and a clock that incorporates the CSCG, referred to as `clk_glitched`. Both clocks are synchronous, operating at 16MHz, and are supplied to the device under test (DUT).

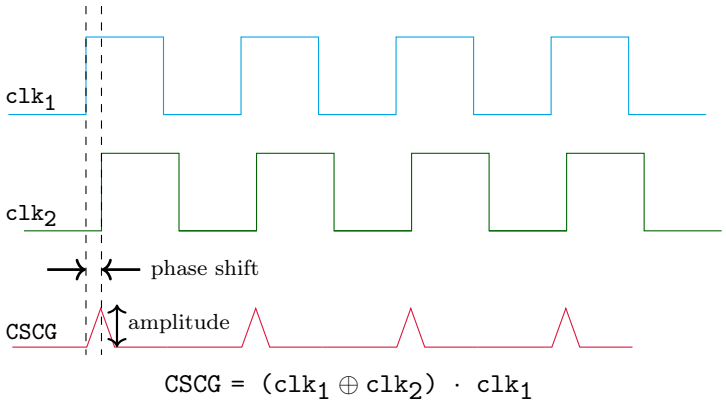


Figure 5.4 – The CSCG is generated using two out-of-phase clocks, `clk1` and `clk2`. The TRAITOR user can choose to replace the regular clock signal with CSCG.

Composition of the DUT

Physical FIs are performed using TRAITOR implemented on an Artix-7 FPGA to inject CSCG into our DUT, which can be considered as a hardware microbenchmark. To facilitate comprehension, we will begin by elucidating the use of TRAITOR for FI, embedded into our DUT. In preparation for subsequent discussions, it is imperative to make a clear distinction between logical DFF and physical DFF:

— The logical DFF represents an abstract conceptualization of a DFF in our DUT, with multiple possible mappings onto physical DFFs.

— The Physical DFFs are tangible components found on the ICs, such as FPGAs, serving as the foundational element for logical DFF. A logical DFF is mapped onto a given physical DFF.

When logical or physical is not mentioned, then the representation of the DFFs can be either.

The DUT, depicted in Figure 5.5, comprises several logical DFFs, categorized into two types:

1. Target logical DFFs that receive `clk_glitched`.
2. Control logical DFFs that receive `clk_ok`.

These DFFs are organized into groups of 6, with a group consisting of either target logical DFFs (referred to as a target chain) or control logical DFFs (referred to as a control chain). There is one control chain and 32 target chains. Each chain, whether control or target,

is fed the same input: a sequence alternating between 0 and 1. This sequence ensures that the content of every DFF, whether logical or physical, changes with each clock cycle. The outputs of the target chains are compared with the output of the control chain. Any discrepancy in at least one target output is indicative of a fault. By examining the timing between the FI and the appearance of the faulty output at the end of a chain, the specific logical DFF affected in the chain can be identified.

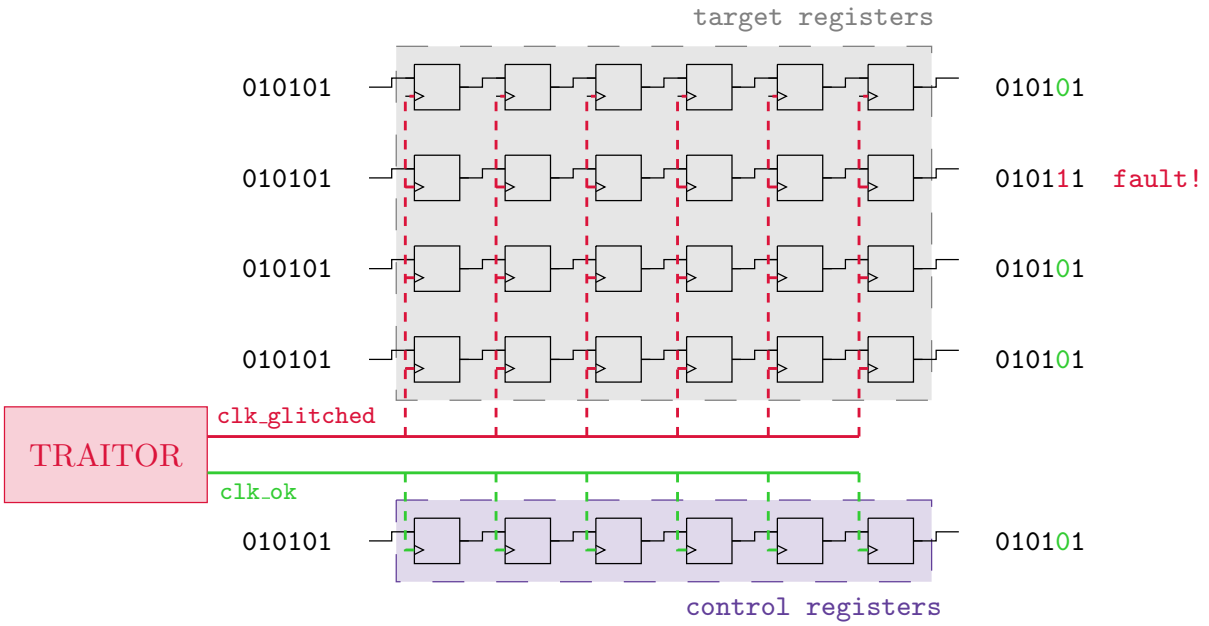


Figure 5.5 – DUT and TRAITOR on an Artix-7 FPGA.

The logical DFFs are mapped onto the physical DFFs of an Artix-7 FPGA, which are located in slices (8 physical DFFs per slice). Although slices contain other components, for clarity, these are not considered in our discussion. Two distinct mappings, as shown in Figure 5.6, are used to investigate how these mappings influence our results.

Fault Injection Protocol

The physical experiments are detailed in Section 5.3. To conduct these experiments, the following protocol is adhered to:

1. Both TRAITOR and the DUT are implemented on the same Artix-7 FPGA. This setup ensures the shortest and simplest clock paths, avoiding additional hardware components such as IOs or external wiring. To ensure consistency across all exper-

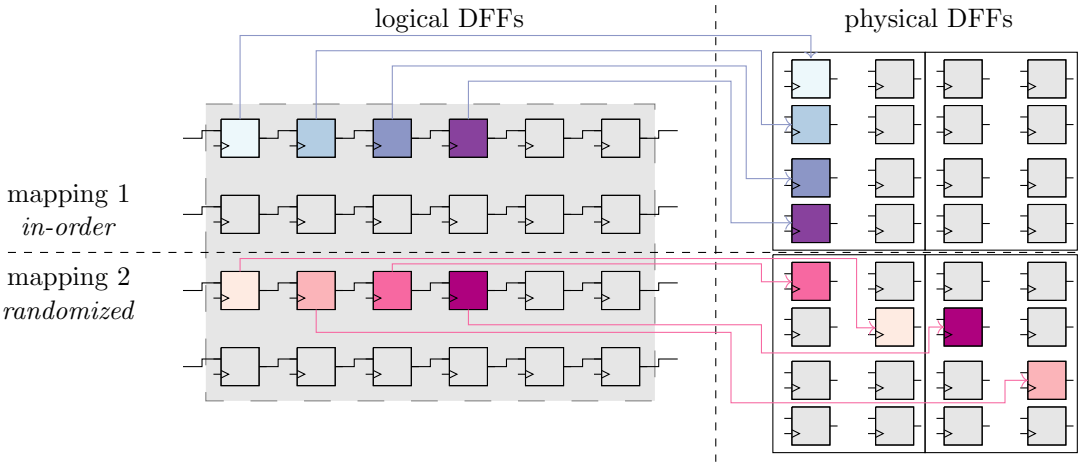


Figure 5.6 – The two logical-to-hardware mappings: mapping 1 is in-order and mapping 2 is randomized.

iments, we meticulously determine the placement of TRAITOR and our DUT on the FPGA, aiming for precision. This guarantees that TRAITOR is consistently mapped to the same location for every experiment.

2. The FI process remains constant. Upon receiving a trigger from the target, a CSCG with a specified amplitude is injected. This process is repeated 100 times for each amplitude, ranging from 0 to 29. Subsequently, we analyze which DFF, if any, has been impacted by the FI.
3. Conclusions are drawn based on the observed outcomes and the analysis of results.

5.2.2 Transistor-Level Simulations

The simulations were carried out using Eldo [21], an ASIC-oriented SPICE simulator. Given the proprietary nature of the Artix-7 FPGA design, replicating the exact 28 nm physical DFFs targeted in the physical experiments is not feasible. Instead, the simulations employ DFFs from a similar CMOS technology available in our laboratory, i.e., a 28 nm FDSOI (Fully Depleted Silicon On Insulator) Process Design Kit. These physical DFFs feature three connections: `D`, `Q`, and a clock input. However, unlike the physical DFFs used in the Artix-7 experiments, they lack a reset pin. Although the simulated DFFs and the Artix-7 physical DFFs do not have the same implementation, they do not significantly differ since they are designed for a similar technology node and tend to behave the same way.

The simulated circuit consists of two DFFs. They first undergo a normal clock cycle, followed by a glitched one. Although a fault is injected into both DFFs, only the first one is considered for analysis; the second DFF is included to more closely mirror our physical experiments by simulating a load. The clock operates at 100 MHz with a voltage amplitude ranging from 0 V to 1 V.

The simulation focuses on a state change in the first DFF, transitioning from 0 to 1. It is important to note that the metastability phenomenon observed in physical experiments is not replicable in simulation. The primary goal of the simulation is to estimate the impact of the voltage and width of the controlled synchronous clock glitch. To achieve this, we independently vary both parameters, incrementally increasing them from low values until the DFF under test samples the input.

5.3 Hypotheses validation

In this section, multiple experiments and simulations are conducted to validate the hypotheses presented in Section 5.1.

5.3.1 Hypothesis 1: Energy Threshold

We examine the behavior of physical DFFs faulted with TRAITOR, observing variations depending on the amplitude parameter. The results of the FI campaign validate Hypothesis 1. The target physical DFFs exhibit the following behaviour, shown on Figure 5.7, for 3 distinct DFFs:

1. For amplitudes ranging from 0 to 21, inclusive, all DFFs are in a *always faulted* state.
2. For amplitudes between 22 and 24, inclusive, some DFFs are in a *always faulted* state, while others are in a *sometimes unfaulted* state.
3. Starting from amplitude 25, all DFFs are in a *always unfaulted* state.

A single amplitude does not identify the energy threshold. Instead, it is characterized by a range of 2 to 3 amplitudes in this experiment, with the fault sensitivity as the lower bound. During this transition phase from faulted to unfaulted, a physical DFF progressively experiences fewer faults until it becomes entirely unfaulted. The transition phases of the 192 physical DFFs overlap but are not identical, as illustrated in Figure 5.7.

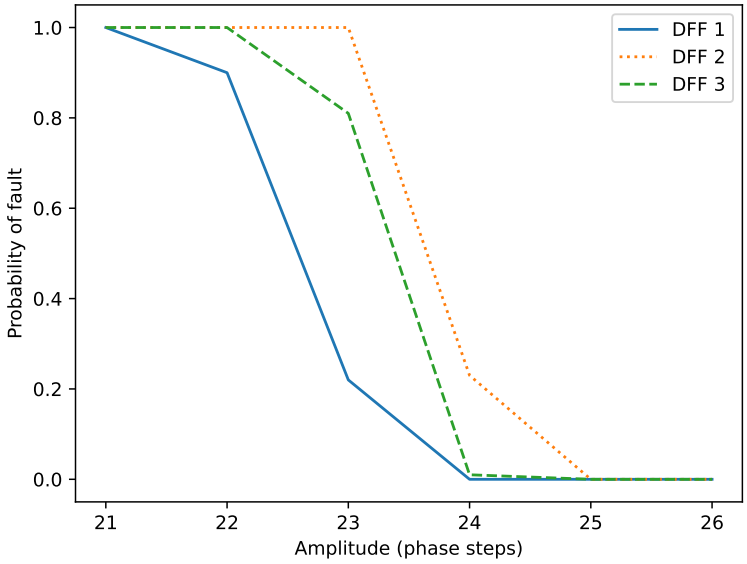


Figure 5.7 – Transitions phases of three target physical DFFs chosen since they exhibit different characteristics.

Energy Propagation and Metastability

Figure 5.7 illustrates the variation in fault occurrence probability relative to the glitch amplitude for three distinct DFFs, selected for their different characteristics. This figure shows various behaviors.

First, the three DFFs exhibit different fault sensitivities (22 for DFF 1, 23 for DFF 3, 24 for DFF 2). DFF 2 remains in a *always faulted* state at a higher amplitude, suggesting more energy loss during clock signal propagation. The causes of this energy loss are examined with Hypotheses 2 and 3. Then, at an amplitude equal to their fault sensitivity, each DFF shows a fraction of samplings being unfaulted and the rest faulted, indicative of metastable behavior. This ratio is consistent and reproducible for each physical DFF.

The standard error of the mean (SEM) is easily calculated: in the worst-case scenario where the fault probability is $p = 0.5$, the standard deviation σ is $\sigma = \sqrt{p \cdot (1-p)} = 0.5$. Therefore, the SEM is $SEM = \sigma/100 = 0.005$, as we have 100 experiments for each DFF. We can deduce that our fault probability falls within 3 error deviations ($= 0.015$) for approximately 99 % confidence. For instance, our metastability evaluation suggests that at amplitude 23, DFF 1 registers a fault in $22 \pm 1.5\%$ of injection attempts with 99 %

confidence.

What we observe in Figure 5.7 is a typical S-curve characteristic of metastable behavior due to insufficient energy at the DFF's clock pin [7]. However, only one amplitude per DFF triggers the metastable output.

As a conclusion, each DFF undergoes a transition phase, displaying a limited metastable behavior. The transition phases of different DFFs may overlap but are not identical, attributed to the energy loss during clock signal propagation. This results in a collective transition phase for all DFFs from amplitudes 22 to 24 inclusive.

Simulating the Influence of Glitch Width and Voltage Amplitude Independently

While the previous experiment emphasizes the existence of an energy threshold, the specific design of TRAITOR does not allow for independent testing of the influence of the glitch width and voltage amplitude on this threshold. To overcome this limitation, we simulate a small circuit (as described in Section 5.2.2) where we send a glitched clock pulse while varying the glitch width and voltage amplitude independently to observe if the sampling occurs.

The simulation is performed with a glitch width ranging from 0 ns to 5 ns by steps of 0.01 ns and voltage amplitude ranging from 0 V to 1 V by steps of 0.01 V). Figure 5.8 illustrates the sampling behavior with respect to glitch width and voltage amplitude parameters. The DFF successfully samples above the curve. The plot is constrained to the range of 0 ns to 1 ns, reflecting the fact that the amplitude reaches a lower plateau at 0.46 V.

Remarkably, sampling occurs for very small widths, as long as the voltage amplitude is sufficiently high; the minimum width for this occurrence is 0.03 ns at a voltage of 0.84 V. However, the opposite is not true: the glitch voltage amplitude must be at least 0.46 V for the DFF to sample, regardless of the width. In other words, it is "sufficient" for the DFF to sample that the controlled synchronous clock glitch has a high voltage amplitude for a short width, but not a long width with a low amplitude. Hence, the voltage amplitude threshold appears to be more restrictive than the width threshold.

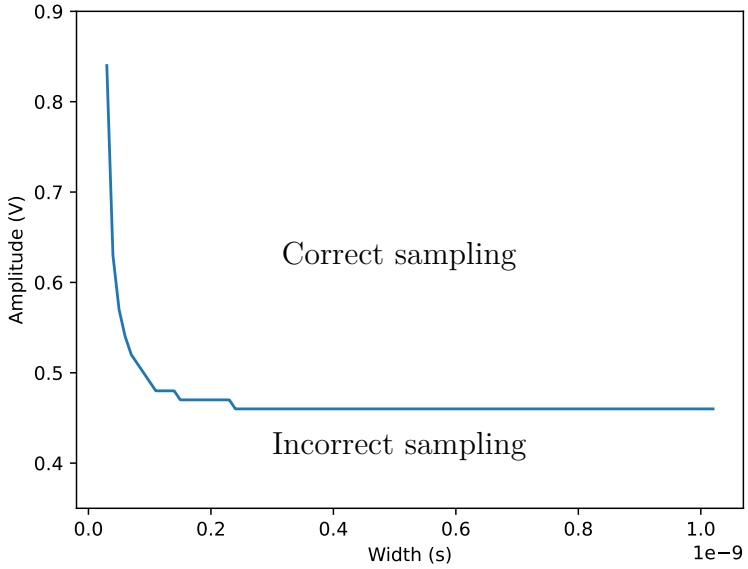


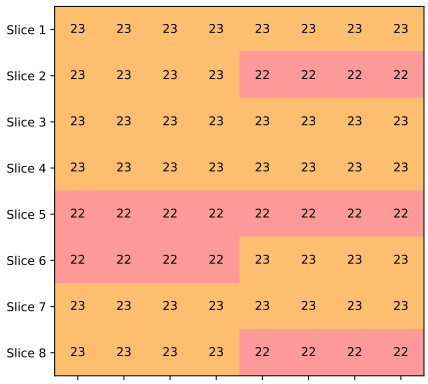
Figure 5.8 – Simulated sampling results: for a glitch with a given voltage amplitude and width above this curve, the sampling is correct. Otherwise, the sampling is incorrect.

5.3.2 Hypothesis 2: Fault Sensitivity Dependency on Intrinsic Properties

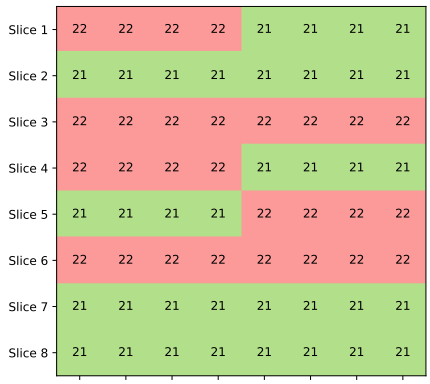
In this part of the study, we aim to understand why the transition phase, particularly the fault sensitivity, varies among physical DFFs. Our primary focus is on the potential dependency of fault sensitivity on the intrinsic properties of physical DFFs.

As discussed in Section 5.3.1, each physical DFF exhibits a specific and reproducible fault sensitivity. One primary factor influencing this sensitivity is the layout of clock routing: not all physical DFF on an FPGA share identical clock signal paths. Variations in these paths, potentially due to length differences or coupling with neighboring wires, can introduce disparities in inductance. If the layout of the clock routing was the sole intrinsic factor affecting fault sensitivity, then replicating the same design (with identical mapping) on another FPGA of the same model would result in the same sensitivity for identical logical DFFs.

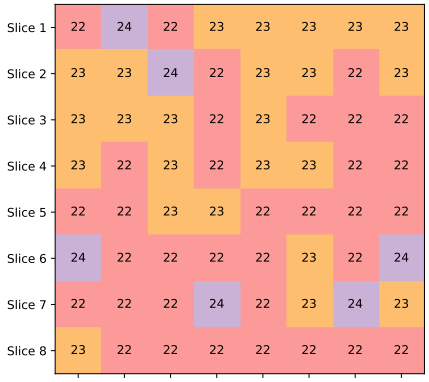
In the ensuing experiment, the same DUT is mapped onto two Artix-7 FPGAs in the exact same manner. Practically, this involves using the same bitstream FPGA image on



(a) Color coded fault sensitivities of the first 64 registers on mapping 1 *in-order* on FPGA 1.



(b) Color coded fault sensitivities of the first 64 registers on mapping 1 *in-order* on FPGA 2.



(c) Color coded fault sensitivities of the first 64 registers on mapping 2 *randomized* on FPGA 1.

Figure 5.9 – Comparing fault sensitivities between physical DFFs for various settings.

both FPGAs. The resulting fault sensitivities are depicted in Figures 5.9a and 5.9b. One can see that while the fault sensitivities of the two FPGAs do show some similarities, notable differences exist. Given that both FPGAs are programmed with the same image and, therefore, have identical clock routings, the discrepancies observed in Figure 5.9 can be attributed to process variations. The individual FPGA dies are not exactly identical, leading to variations in the inductances of clock paths, which in turn result in differing fault sensitivities for placement-equivalent physical DFFs.

Limits to the Intrinsic-Only Fault Model

If we assume that only the intrinsic properties of a physical DFF affect its fault sensitivity, then the mapping of logical DFFs to physical DFFs on a specific FPGA should not influence the fault sensitivities of these physical DFFs. This is because the clock routing is independent of the routing of other signals. Since the glitch is propagated solely by the clock signal, the fault sensitivity, assuming it depends solely on intrinsic properties, would be specific to each physical DFF.

To test this assumption, we map the same physical DFFs onto the same FPGA in two different configurations (as depicted in the two mappings of Figures 5.9a and 5.9c) and then compare their fault sensitivities. This results in varying fault sensitivities. The `clk_glitched` signal remains consistent across both mappings since it follows dedicated clock paths, suggesting that the CSCG should be identical in both experiments and consequently result in the same fault sensitivity for each physical DFF independently from the mapping. The two mappings differ in how data signals are routed between physical DFFs, which clearly impacts the fault sensitivity. This observation leads us to hypothesize that extrinsic properties, such as data signals in this case, may influence CSCG.

5.3.3 Hypothesis 3: Fault Sensitivity Dependency on Extrinsic Properties

Given that intrinsic properties alone do not account for all variations in fault sensitivity, we now turn our attention to extrinsic properties. Specifically, we examine two types of extrinsic influences:

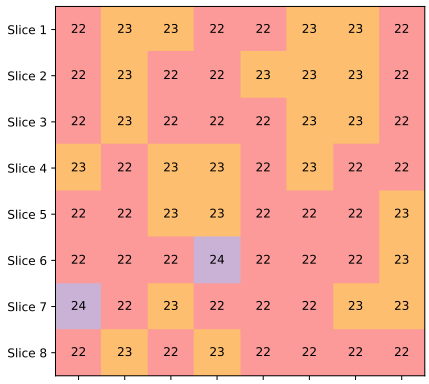
— Activity on data wires, i.e., the paths linking physical DFFs to each other.

— Activity on clock wires, responsible for carrying clock signals to the physical DFFs.

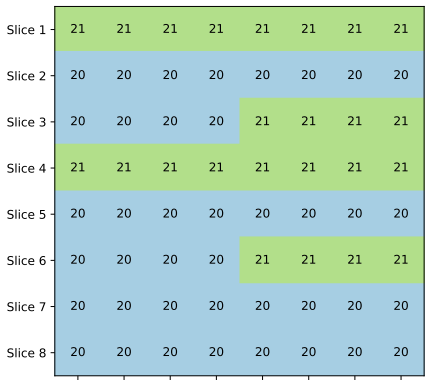
For each wire type, we conduct a separate experiment to isolate and observe its specific influence.

Impact of Data Wires

The influence of data wires on the `clk_glitched` energy was previously suggested in Section 5.3.2. We delve deeper into this aspect with the following experiment. Figures 5.9a and 5.10a illustrate the fault sensitivities of two different routings. Similar to the approach in Section 5.3.1, the target logical DFFs are mapped *in-order*. However, in this case, we



(a) Color-coded fault sensitivities of the first 64 registers on mapping 1 *in-order* with different data routing on FPGA 1, to be compared to Figure 5.9a.



(b) Color-coded fault sensitivities of the first 64 registers on mapping 1 *in-order* with a forced adjacent path for the clock on FPGA 1, to be compared to Figure 5.9a.

Figure 5.10 – Comparing fault sensitivities between physical DFFs for different routing

alter the routing between two physical DFFs resulting in a change in the data wire connections between them. Consequently, we end up with two DUT having the same logical DFFs to physical DFFs mapping, and thus identical clock routing, but differing in data wire routing between physical DFFs.

As a conclusion, the routing of data signals between physical DFFs significantly impacts the energy of the clock signal reaching these physical DFFs, thereby affecting their fault sensitivities.

Impact of Clock Wires

We further hypothesize that the `clk_glitched` signal is influenced by the proximity to the `clk_ok` signal. Previously, the mapping of the DUT and TRAITOR was carefully arranged to avoid any crossing or parallel arrangement of the two clock networks. To assess the impact of clock network interference, we now map some control DFFs on a slice adjacent to the target DFFs. This setup places both `clk_glitched` and `clk_ok` on parallel physical paths, given that the dedicated clock routes are next to each other and originate from nearby sources [1].

Figures 5.9a and 5.10b show that the fault sensitivities not only differ but are also notably lower. In all previous experiments on this FPGA, such as on Figure 5.9a, fault

sensitivities ranged between 22 and 24. However, in this setup, they range between 20 and 21. It appears that positioning the `clk_ok` signal adjacent to `clk_glitched` effectively 'adds energy' to the latter, thereby reducing its fault sensitivity.

Interpretation

The observed energy transfers, both data-to-clock and clock-to-clock, are likely the result of cross-talk between these signals. The fault sensitivity of a physical DFF highly depends on the energy delivered by the clock's rising edge, so even a small amount of energy added or subtracted through cross-talk can have a noticeable impact [24]. These findings suggest that the fault sensitivity in a DFF is an extremely precise indicator of the activity in the surrounding circuitry.

5.4 Expanding the ETFM at the Register-Transfer Level

Using the experiments presented in this chapter and the ETFM, we can theorize a fault model at the register-transfer level. The ETFM is a physical fault model explaining the effect of the CSCG on DFFs. At the register-transfer level, the goal is to understand the fault propagation in a circuit.

Let us consider two chains that are part of the same circuit, one faulted and one unfaulted, similar to the ones present in our DUT, that we schematized in Figure 5.11 and Figure 5.12. Each chain is composed of three DFFs in our example (although the number does not matter). The initial state is the same for both chains, represented at the top of the figure.

In Figure 5.11, at each clock cycle, their content changes from `0` to `1` or conversely. The unfaulted chain output after `n+9` clock cycles is `010101010`. At clock cycle `n+1`, the three registers from the faulted chain do not sample incoming data. The fault effect is twofold. First, the fault impacts the chain and its operation. The faulted value persists until the end of the chain: it takes three additional clock cycles for the chain to return to an unfaulted state. As a consequence, the output contains three faulted values: `010110110`. The second consequence concerns the the thoeritical circuit as a whole. In our example, the two chains are supposed to work identically and output the same values. Due to the

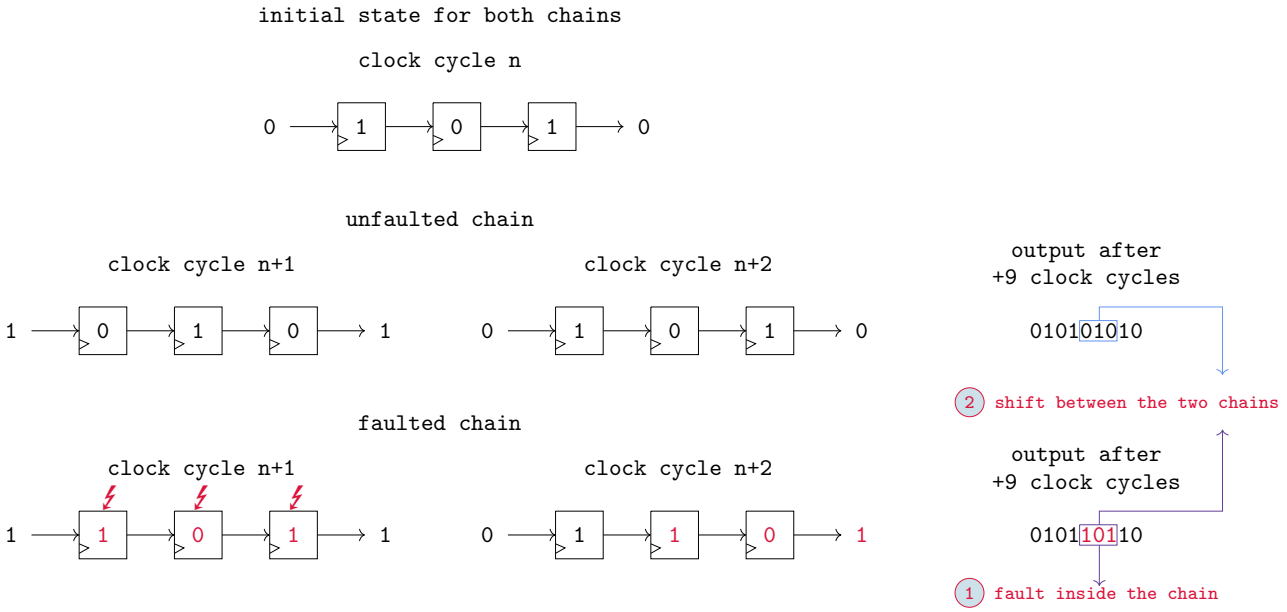


Figure 5.11 – Schematized representation of a fault propagation at register-transfer level with 10101 register transition

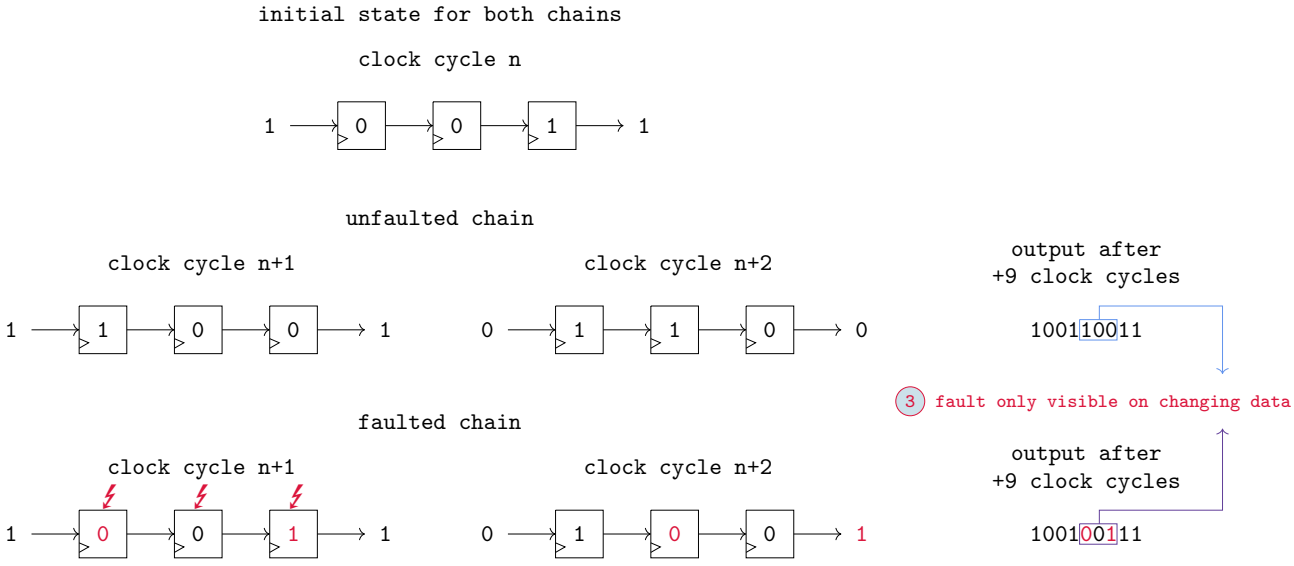


Figure 5.12 – Schematized representation of a fault propagation at register-transfer level with 10011 register transition

fault, a shift is created for three cycles between them, with potential repercussions to the circuit functioning.

In Figure 5.12, the DFFs content change every other clock cycle. The unfaulted chain output after `n+9` clock cycles is `100110011`. Although the three registers are faulted and their sampling is prevented, the fault is only visible when the incoming data differs from the previously stored data.

The two-chain example is fairly simple and illustrates the three consequences of a fault at the register-transfer level. Here, a fault disappears at the end of the faulted chain. In more complex circuits, such as processors, the faulted value may be used in computations, stored in memory for future use, etc. As such, the fault effect is more difficult to assess.

Chapter Conclusion

In this chapter, we propose a fault model explaining the CSCG impact at the physical level. The glitch impacts DFFs and prevents them from sampling incoming data. Why a DFF is affected or not depends on several parameters: the amount of energy the CSCG brings, process variability of the faulted DFF, or the activity of neighboring wires, for example. The ETFM has been presented at COSADE24 [32]. From these observations and the subsequent physical fault model, we theorize the glitch's impact at the register-transfer level.

To complete the CSCG characterization, we must now turn our attention to the microarchitectural level. In [9], Claudepierre *et al.* investigate the CSCG by faulting a commercial processor. We believe that the fault models they observe are not all there is to see. To confirm our says as well as gather other clues on the CSCG impact, we decided to fault the same target.

Chapter 6

Explaining the Synchronous Clock Glitch - Part 2: Microarchitectural Level

This chapter is a direct continuation of the previous chapter. Using the results from Chapter 4 as well as state-of-the-art observations on the SCG, a preliminary microarchitectural fault model is given. This model being theoretical, it is essential to compare it to the results of experiments done on an ASIC-implemented processor. The gathered clues from the experiments give information on the observed ISA-related fault models, the parameters that condition the fault effect, and the vulnerable part of the microarchitecture.

6.1 Theorized Microarchitectural Effects of the CSCG

In [9], Claudepierre *et al.* define the CSCG microarchitectural fault model as "skip-by-repeat" (the target instruction is repeated but does not have an additional effect on the circuit other than skipping the following instruction) or sometimes "true skip," depending on the faulted instructions. In the authors' previous paper [8] that looks into the effect of EMFI on the PLL, the fault model is defined as a "virtual NOP" caused by the modification of the instruction faulted into another instruction which hasn't any visible effect on the circuit. From these observations, we deduce that the pipeline or the cache instruction transfer is impacted. By considering the ETFM and its extension to RTL, we make the following assumptions.

We suppose that the faulted clock is distributed to the entire circuit. The different fault effects on the instructions are schematized in Figure 6.1. From amplitude 0 to a given amplitude X, no effect will be seen: a clock cycle is missing, but it is missing for the entire circuit. To reuse terminology from Section 5.1, all the registers are in an always faulted state. From an amplitude Y higher than X, all the registers will be in an always

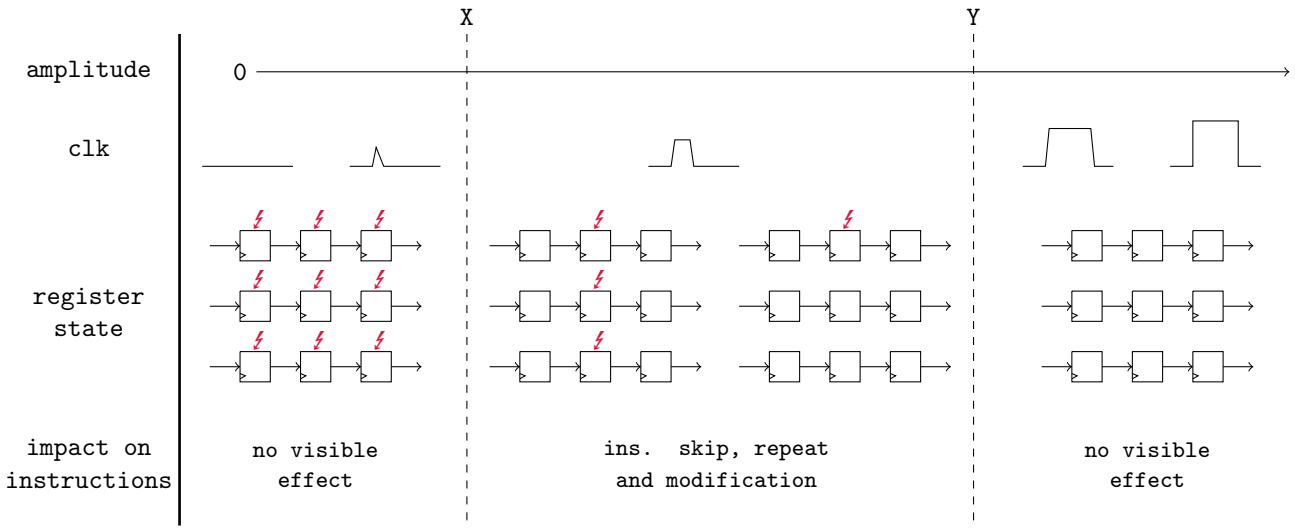


Figure 6.1 – Fault impact on instructions depending on the amplitude

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

Figure 6.2 – Binary encoding of `adds.w rd, rn, imm` with `rn` the source register and `rd` the destination register

unfaulted state. And from amplitude X to Y, the registers' behaviour will differ depending on extrinsic and intrinsic properties. Some registers are going to be in a sometimes unfaulted state, while others, for the same amplitude, are going to be either always faulted or always unfaulted. Supposedly, the coexistence of the three states brings out the faulted behaviour observed by Claudepierre *et al.* in [9]. Then, the skip and repeat and true skip fault models might be too restrictive. It would require a specific number of DFFs to be faulted simultaneously. While this is possible at a given amplitude, other fault models (such as instruction modification) are likely to be present.

As mentioned in Section 5.4, the fault effect on a DFF is only visible if the incoming data to be sampled differs from the previously sampled one. At the microarchitecture level, instructions are represented by their binary encoding (opcodes). When an instruction is fetched from a cache and goes into the pipeline, each encoding bit goes through DFFs at each data transfer. Figure 6.3 represents a theoretical instruction transfer, with the destination DFFs vulnerable to faults. At clock cycle 0, the vulnerable DFFs hold the binary encoding for `adds.w r7, r7, 1`. At clock cycle 1, when a fault is injected, the vulnerable DFFs should sample the binary encoding for `adds.w r8, r8, 1`. Since the

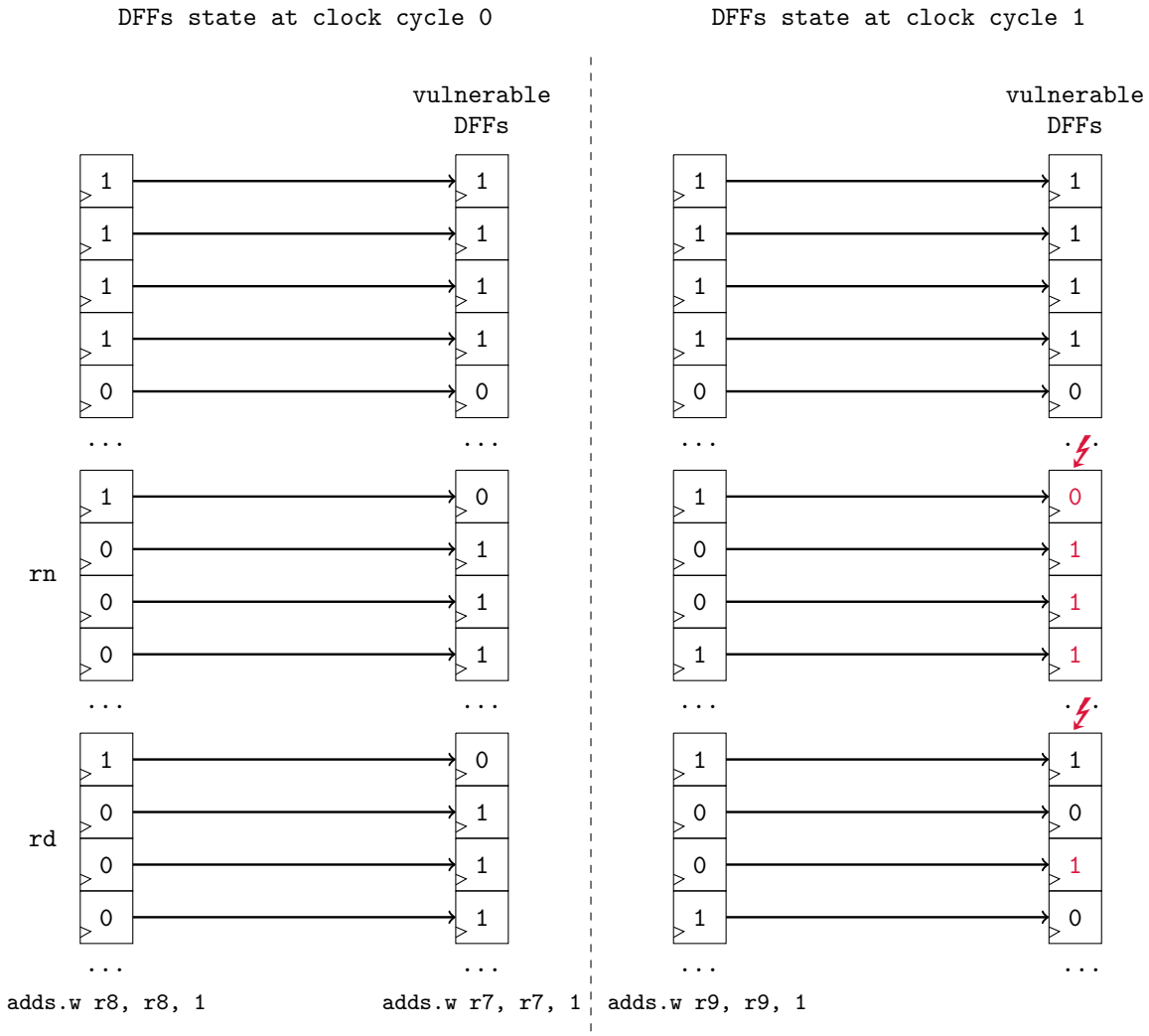


Figure 6.3 – Fault impact on instruction binary encoding DFFs

effect is only visible when the incoming bits are different from the ones previously sampled, the encoding for the immediate and the `adds.w` identification are not visibly impacted. When the difference exists, as for the encodings of source and destination registers, the fault impact can affect either the entirety (the previous encoding is repeated) or parts (the previous encoding is modified) of the bits.

Preliminary microarchitectural fault model

- → the cache instruction transfers or the pipeline are affected (among possibly other parts of the microarchitecture)
- → different amplitudes cause different ISA-level effects (instruction repeat, skip or modification)
- → the fault impact is only visible when the incoming data to be sampled by a DFF is different from the previously sampled data

The theoretical fault effects we mention here need to be verified through experiments.

6.2 Experimental Setup

As in Chapter 5, TRAITOR is used to inject the CSCG. Since we are interested in microarchitectural sources of fault and compare it with previous research, we picked the same target as in [9, 8], described below.

6.2.1 Target Description

The target processor is a Cortex-M3 embedded on a STM32F100RB. This microcontroller has a removable crystal oscillator, allowing to easily plug TRAITOR's clock instead. The Cortex-M3 [12] supports the ARMv7-M architecture profile [4]. Notably, both 16- and 32-bit long instructions are available. The processor, as schematized in Figure 6.4, implements a 3-stage pipeline, the stages being instruction fetch, instruction decode, and execute/write back. The Cortex-M3 does not have cache memory strictly speaking, but a prefetch buffer. According to the documentation, the buffer can contain up to three 32-bit or six 16-bit instructions. The instructions are prefetched from the main memory via a 32-bit bus (consequently, the bus can carry either one 32-bit or two 16-bit instructions). In [4], the number of cycles an instruction takes to execute when pipelined is given. For example, `adds` instructions take one cycle. Although the documentation does not give

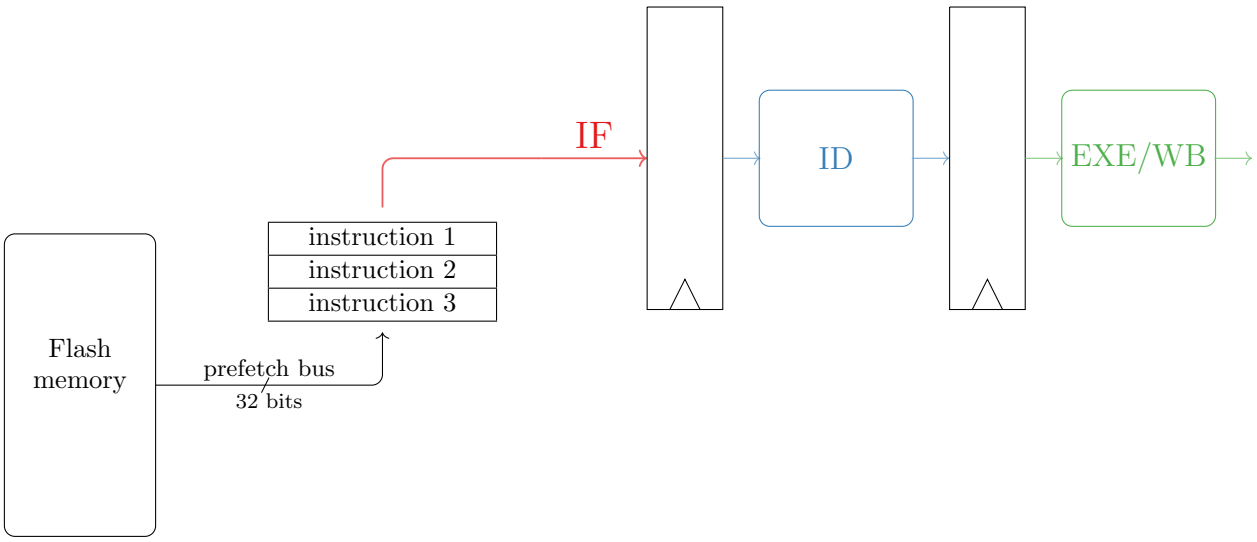


Figure 6.4 – Schematic view of the Cortex-M3

precise information about this, if we consider only 32-bit instructions executed in one cycle, the prefetch should happen every cycle.

Some details are still vague. First, it is important to note that the documentation concerns Cortex-M3 in general and that some parameters can be customized depending on their application, such as the size of the prefetch buffer. Inside the prefetched buffer, the instructions' position and whether there are transfers in between lines are unknown. How the instructions are fetched into the pipeline is also not detailed in the documentation. Finally, no information is given on the prefetching behaviour in cases of branches.

6.2.2 Fault Injection Setup

Our control over the target is limited because we do not have extensive information on its implementation. In the previous chapter, we could target isolated DFFs. With the Cortex-M3, TRAITOR's glitched clock is distributed all over to clocked elements of which we ignore the placement and interconnections. Although our lack of knowledge may hinder our comprehension, the application the processor runs can still provide clues about the fault's impact on the microarchitecture. The experiments consist in faulting variations of Listing 6.5 and Listing 6.6. The target instructions can be either aligned (the instruction address is a multiple of its size, 32 bits in our case) or unaligned.

For each listing and their variations, the same fault injection parameters are used: the amplitude ranges from 370 to 430, and the delay from 30 to 50, with only one glitched

```
1  *trigger rise*
2  movw r2, 0x6c59
3  movw r3, 0x44a3
4  movw r4, 0xd0ea
5  movw r5, 0x2624
6  movw r6, 0x2e7c
7  movw r7, 0x1248
8  movw r8, 0x3330
9  movw r9, 0xed12
10 mov.w sl, 0
11 mov.w fp, 0
12 mov.w ip, 0
13 mov.w r0, 0
14 nop.w
15 nop.w
16 ...
17  nop.w
18  nop.w
19  nop.w
20  nop.w
21  adds.w r2, r2, 1
22  adds.w r3, r3, 1
23  adds.w r4, r4, 1
24  adds.w r5, r5, 1
25  adds.w r6, r6, 1
26  adds.w r7, r7, 1
27  adds.w r8, r8, 1
28  adds.w r9, r9, 1
29  nop.w
30  nop.w
31  nop.w
32 ...
33  nop.w
34  nop.w
35  nop.w
36  *trigger fall*
```

initialize our target registers with random values

initialize other registers at 0

X nops

Figure 6.5 – Listing with variable registers and invariable immediate

```
1  *trigger rise*
2  movw  r2, 0x6c59
3  movw  r3, 0x44a3
4  movw  r4, 0xd0ea
5  movw  r5, 0x2624
6  movw  r6, 0x2e7c
7  movw  r7, 0x1248
8  movw  r8, 0x3330
9  movw  r9, 0xed12
10 mov.w sl, 0
11 mov.w fp, 0
12 mov.w ip, 0
13 mov.w r0, 0
14 nop.w
15 nop.w
16 ...
17  nop.w
18  nop.w
19  nop.w
20  nop.w
21  adds.w r7, r7, 4
22  adds.w r7, r7, 11
23  adds.w r7, r7, 5
24  adds.w r7, r7, 13
25  adds.w r7, r7, 1
26  adds.w r7, r7, 17
27  adds.w r7, r7, 19
28  adds.w r7, r7, 23
29  nop.w
30  nop.w
31  nop.w
32 ...
33  nop.w
34  nop.w
35  nop.w
36  *trigger fall*
```

initialize our target registers with random values

initialize other registers at 0

X nops

Figure 6.6 – Listing with invariable registers and variable immediate

clock cycle. For each pair (amplitude, delay), the target is reset before a fault injection attempt is done. The execution stops at the end of the second `nop.w` set of instructions thanks to a previously set breakpoint, allowing to retrieve via JTAG the value of several registers: `r0-r12`, `sp`, `lr`, `pc`, `xPSR`, `msp`, `psp`, `primask`, `basepri`, `faultmask`, `control`. Sometimes, the execution does not go smoothly and goes into an interrupt, and escalates to HardFault. Before rebooting the processor, we retrieve information about the interrupt: `xPSR` (the type of interrupt), `pc`, `lr`, several register values. This, among other things, allows us to get a sense of the temporality of the fault on our code.

After the injections are done, we look for the most present fault model(s) (at least 40 occurrences out of the 50 tries) for each couple (amplitude, delay). We then represent the fault models distribution on a bar graph. Each model is numbered, and represented by a color and a pattern.

Two STM32F100RB are used for the experiments. We quickly notice that the model distributions are the same, with slight variations in amplitudes. For the remaining of this Chapter, we present the results obtained for only one target.

When performing fault injections, we notice that for a given delay, an instruction always seems to be "skipped", i.e. the associated register is not updated. To make the analysis and subsequent explanations clearer, we use the following terminology: the "skipped" instruction is referred to as `ins. n`; the following instruction is then `ins. n+1`, etc. In addition, each instruction is also associated with a number: in all listings, instruction line 21 is `ins. 21`, etc.

6.3 Dominant Fault Models

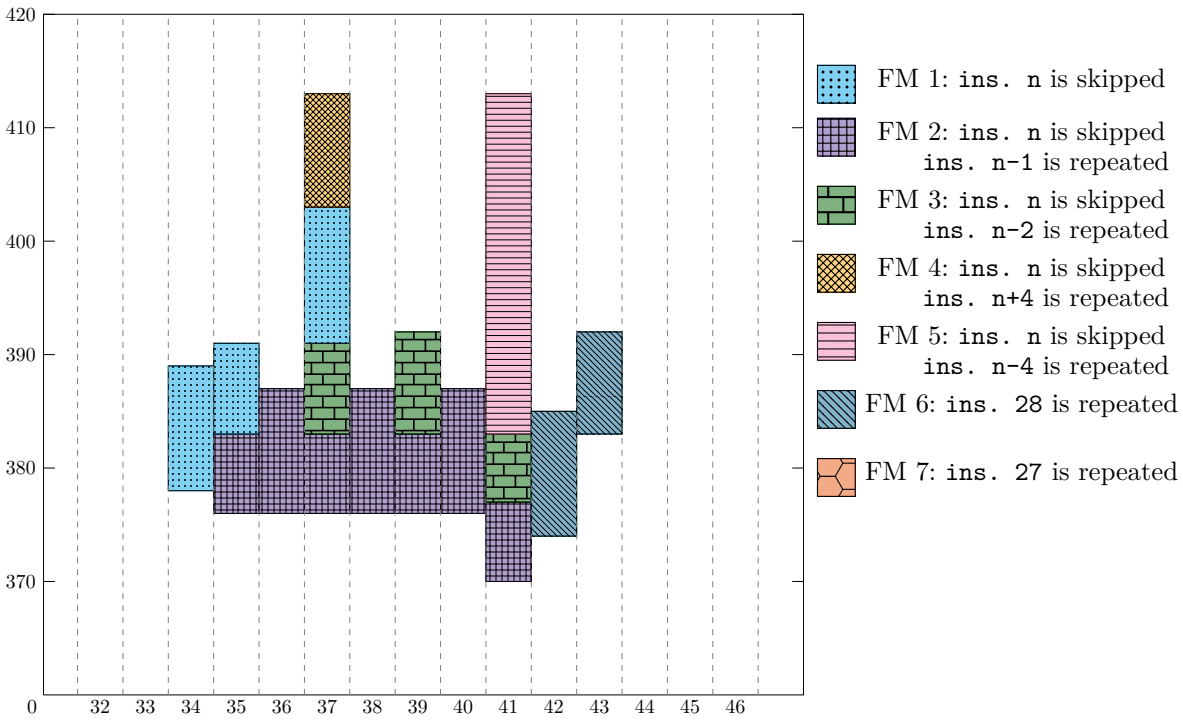
In our journey to characterize the impact of the CSCG on the microarchitecture, the first step is to conduct experiments to gather clues about the ISA-level fault models. By taking into consideration our preliminary microarchitectural model, the fault effect should be limited to changing data from one instruction to another. To simplify the task, we limit the amount of changing bits in the instruction opcode. In Experiment 6.7, only the *registers* change, while in Experiment 6.8, only the *immediates* change.

→ Initial register value:

r2	r3	r4	r5	r6	r7	r8	r9
0x6c59	0x44a3	0xd0ea	0x2624	0x2e7c	0x1248	0x3330	0xed12

→ Listing 6.5, no variation
→ Number of `nop.w` instructions: 8
→ Eight aligned in-order 32-bit `adds.w` instructions
$r_2 \rightarrow r_3 \rightarrow r_4 \rightarrow r_5 \rightarrow r_6 \rightarrow r_7 \rightarrow r_8 \rightarrow r_9$

(a) Parameters



(b) Fault models

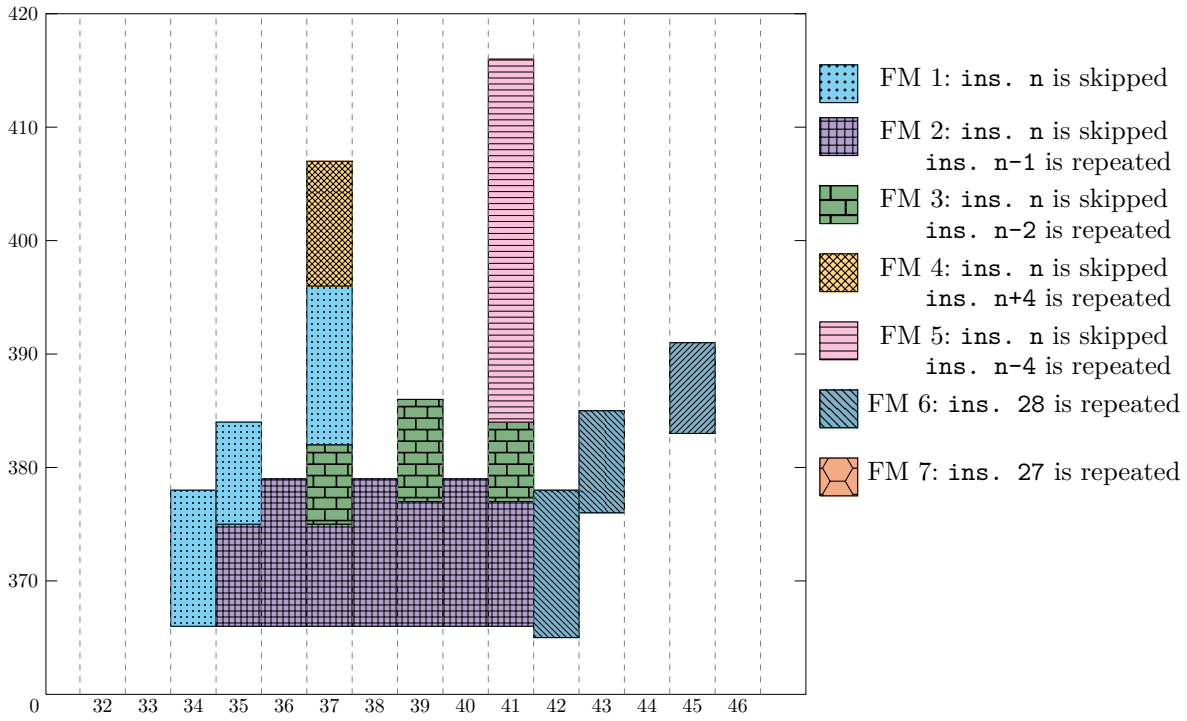
Experiment 6.7 – Finding fault models (register)

→ Initial register value:

r2	r3	r4	r5	r6	r7	r8	r9
0x6c59	0x44a3	0xd0ea	0x2624	0x2e7c	0x1248	0x3330	0xed12

→ Listing 6.6, no variation
→ Number of `nop.w` instructions: 8
→ Eight aligned in-order 32-bit `adds.w` instructions

(a) Parameters



(b) Fault models

Experiment 6.8 – Finding fault models (immediate)

To give a sense of the temporality of the fault, at delay 34, `adds.w r2, r2, 1` is skipped for Experiment 6.7 and `adds.w r7, r7, 4` is skipped for Experiment 6.8. Using our terminology, `ins. n` for this delay is `ins. 21` (for both experiments). The fault models span over ten to twelve clock cycles. First, it is important to notice that, using our terminology, we observe the same exact fault models for both experiments, except the fault model present at delay 45 for Experiment 6.8. The main difference comes from the amplitudes: the fault models in Experiment 6.8 are present for lower amplitudes than for Experiment 6.7.

Although present in both experiments, the "true skip" and "skip-by-repeat" models are insufficient to explain the variety of effects we noticed. More precisely, the "skip-by-repeat" model usually concerns adjacent instructions: a first instruction is repeated, which has for consequence to skip the next one. In both experiments, for four of the observed fault models, the repeated instruction can be two to four instructions apart, before or even after the skipped instruction. The fault model 6 observed for delay 42 and 43 is similar to the fault models 2 and 3. We theorize that `ins. 28` is repeated while one of the two following `nop.w` instructions is skipped, depending on the delay. We notice, in adequation with our hypothesis, that only either the source and destination register for Experiment 6.7 or the immediate value for Experiment 6.8 are impacted, which are the only variable opcodes for the instructions.

From these observations, we propose an alternative explanation of the fault effect. The fault prevents an instruction transfer, which has the consequence of either repeating the previous instruction (for lower amplitudes) or executing a modified instruction (for higher amplitudes) and skipping the faulted one. This being said, the modification is not what we expected. For example, let us look at the fault models present at delay 37 in Experiment 6.7. The skipped instruction is `adds.w r5, r5, 1`. Although we have no certainty over this, we guess the transition between `adds.w r4, r4, 1` and `adds.w r5, r5, 1` is faulted. In binary, `r4` is `0100` and `r5` is `0101`, meaning there is only one bit of difference. This entails that there are three possible modified instructions: `adds.w r4, r4, 1`, `adds.w r4, r5, 1` and `adds.w r5, r4, 1`. While FM 2 matches, the other three fault models do not. Additionally, both the source and the destination registers are modified, in a equal manner. Finally, we observe that the number of fault models differs depending on the delay. On every other delay, the fault models are unique, while they can

be plural otherwise.

Summary - finding fault models

Experiments revealed that the CSCG impact on instructions can not only be characterized as "skip by repeat" or "true skip". Supposedly, the CSCG prevents an instruction transfer somewhere in the microarchitecture, with the consequence of either repeating the previous instruction or executing a modified instruction, depending on the amplitude, as well as skipping the faulted instruction. We also observe that the number of fault models differs depending on the delay.

6.4 Fault Influence

Experiment 6.9 consists in shifting the target `adds.w` instructions by one clock cycle by inserting an additional `nop.w` instruction beforehand. Additionally, in Experiment 6.10, the target instructions are out-of-order. In this section, we explore the influences on the fault models. If the fault models are only tied to the instructions, then they should also be shifted by one clock cycle in the first experiment, and their order of appearance should be different in the second one.

In Experiment 6.9, since the target code has been shifted by one clock cycle, the instruction `adds.w r2, r2, 1` is skipped at delay 35. We observe that the fault models are not shifted by a clock cycle. Their number per delay stays the same as in Experiments 6.7 and 6.8. Additionally, the same global effects are seen: for example, at delay 41, fault model 5 is observed again, with `ins. n` being `ins. 27` instead of `ins. 28`.

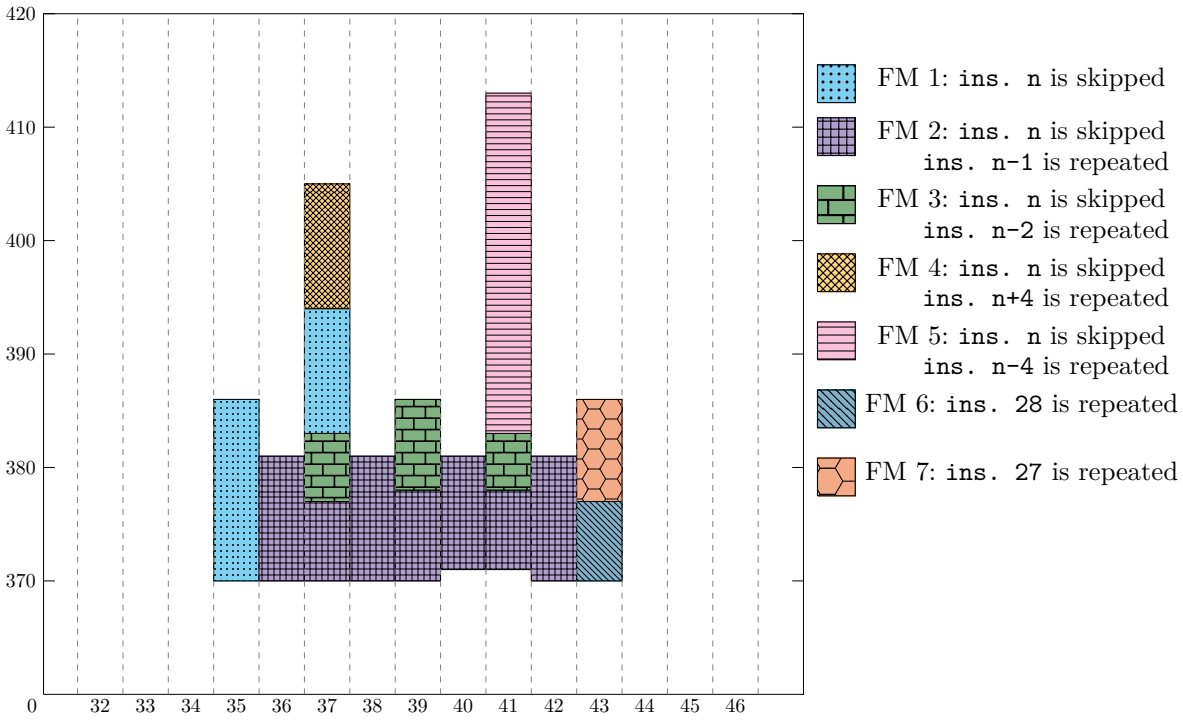
In Experiment 6.10, the same terminology applies, following the order of the instruction: at delay 34, `ins. n` is `ins. 28`, `ins. n+1` is `ins. 22`, etc. Again, the same global effects are seen only with different registers.

→ Initial register value:

r2	r3	r4	r5	r6	r7	r8	r9
0x6c59	0x44a3	0xd0ea	0x2624	0x2e7c	0x1248	0x3330	0xed12

→ Listing 6.5, variation in the number of `nop.w` instructions
→ Number of `nop.w` instructions: 9
→ Eight aligned in-order 32-bit `adds.w` instructions

(a) Parameters



(b) Fault models

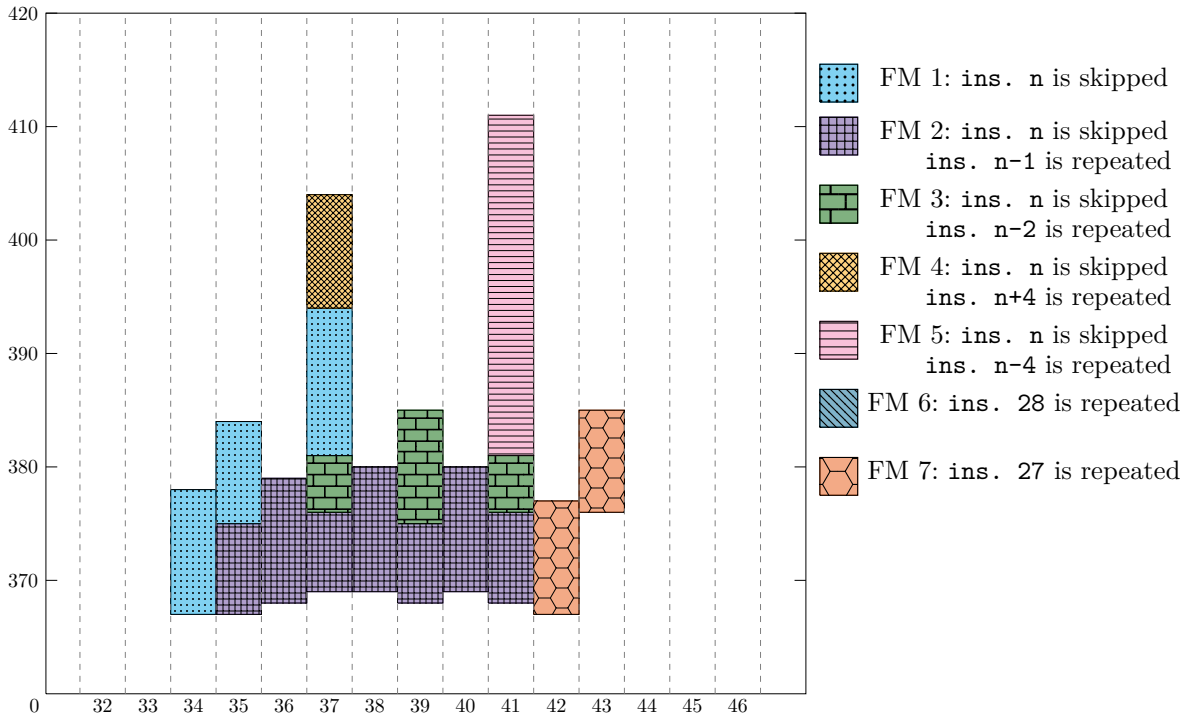
Experiment 6.9 – Shifting by one cycle

→ Initial register value:

r2	r3	r4	r5	r6	r7	r8	r9
0x6c59	0x44a3	0xd0ea	0x2624	0x2e7c	0x1248	0x3330	0xed12

→ Listing 6.5, variation in instruction order
→ Number of `nop.w` instructions: 8
→ Eight aligned in-order 32-bit `adds.w` instructions:
`ins. 28` → `ins. 22` → `ins. 21` → `ins. 25`
→ `ins. 26` → `ins. 24` → `ins. 23` → `ins. 27`

(a) Parameters



(b) Fault models

Experiment 6.10 – Out-of-order instructions

Experiments 6.9 and 6.10 contradict the assumptions made earlier: the fault models are not tied to the instructions but more to the delay and amplitude. These observations supplement our microarchitectural fault model draft. In addition to the missing transfer (either from caches or the pipeline), some other vulnerable microarchitectural mechanism that appears every two cycles is faulted. In this context, finding out which other parts of the microarchitecture are impacted is difficult. It would require extensive knowledge of the processor and its implementation, which we cannot access due to the proprietary nature of our target.

Summary - fault influence

The fault models are not only related to the instructions, but also strongly to the delays and amplitudes. We suppose that the CSCG may impact other parts of the microarchitecture in addition to instruction transfers.

6.5 Vulnerable Processor Part

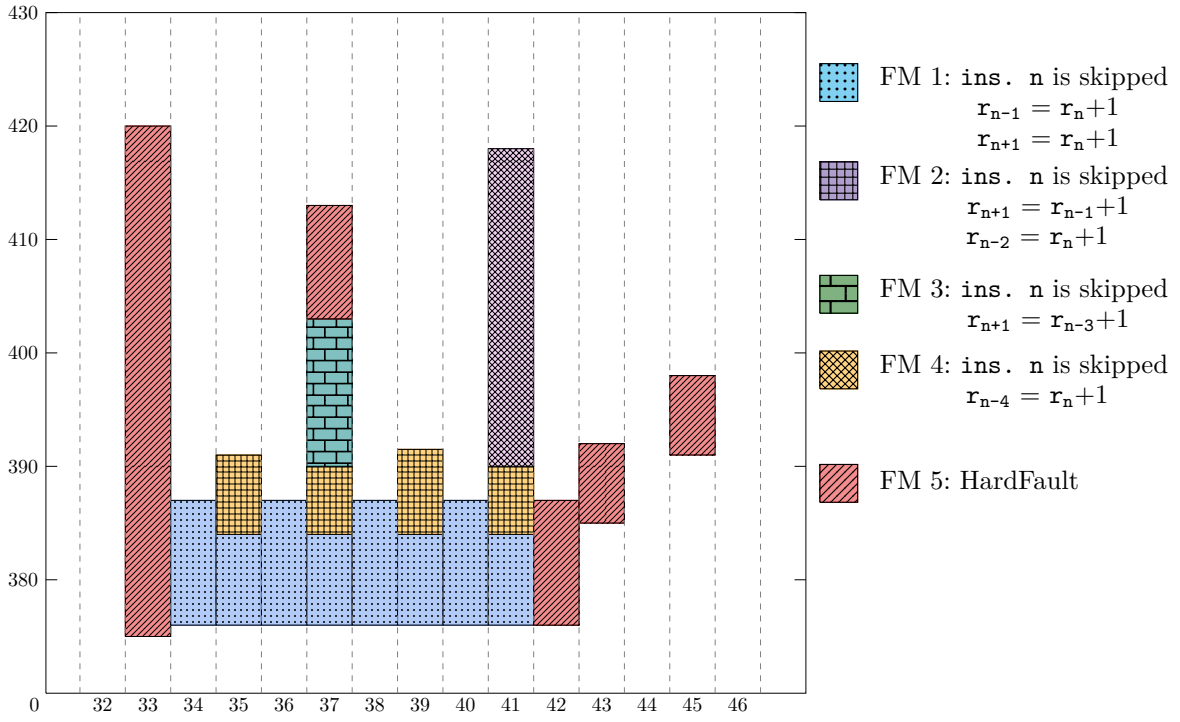
Since the fault models are partly tied to the instructions, either the prefetch mechanism or the pipeline stages are most likely vulnerable. As mentioned in the previous section, another processor part is likely affected by the CSCG, but given the lack of knowledge and control we have over the target, we can not pursue investigations.
We fault unaligned instructions in Experiment 6.11 to gather clues. More precisely, we aim to discriminate between the pipeline and the prefetch transfers. If the fault affects the pipeline stage, then the fault models should be the same since the IF stage fetches complete 32-bit instructions. On the other hand, if the prefetch is vulnerable (either the transfer between the main memory and the buffer or inner movements in the buffer), then the fault impact will be different with unaligned instructions since the fault will impact two 16-bit halves of instructions instead of a complete 32-bit one.

→ Initial register value:

r2	r3	r4	r5	r6	r7	r8	r9
0x6c59	0x44a3	0xd0ea	0x2624	0x2e7c	0x1248	0x3330	0xed12

→ Listing 6.5, variation in instruction alignment
→ Number of `nop.w` instructions: 8
→ Eight unaligned out-of-order 32-bit `adds.w` instructions

(a) Parameters



(b) Fault models

Experiment 6.11 – Unaligned instructions

The terminology for this experiment differs from the one used previously. Instead of defining the fault models solely through the instructions, we also characterize the modification to the register. The "skipped" instruction is still referred to as `ins. n`; the following instruction is then `ins. n+1`, etc. The associated register to the "skipped" instruction is referred to as r_n, the following r_{n+1}, etc.

In our experiments, the CSCG is always one cycle long; however, we observe that for some fault models, three registers are impacted. We retrieve the registers' values at each

instruction	FM	r2	r3	r4	r5	r6	r7	r8	r9
nop.w	FM1	0x6c59	0x44a3	0xd0ea	0x2624	0x2e7c	0x1248	0x3330	0xed12
	FM2	0x6c59	0x44a3	0xd0ea	0x2624	0x2e7c	0x1248	0x3330	0xed12
	FM3	0x6c59	0x44a3	0xd0ea	0x2624	0x2e7c	0x1248	0x3330	0xed12
r2	FM1	0x6c5a	0x44a3	0xd0ea	0x2624	0x2e7c	0x1248	0x3330	0xed12
	FM2	0x6c5a	0x44a3	0xd0ea	0x2624	0x2e7c	0x1248	0x3330	0xed12
	FM3	0x6c5a	0x44a3	0xd0ea	0x2624	0x2e7c	0x1248	0x3330	0xed12
r3	FM1	0x6c5a	0x44a4	0xd0ea	0x2624	0x2e7c	0x1248	0x3330	0xed12
	FM2	0x6c5a	0x44a4	0xd0ea	0x2624	0x2e7c	0x1248	0x3330	0xed12
	FM3	0x6c5a	0x44a4	0xd0ea	0x2624	0x2e7c	0x1248	0x3330	0xed12
r4	FM1	0x6c5a	0x44a4	0xd0eb	0x2624	0x2e7c	0x1248	0x3330	0xed12
	FM2	0x6c5a	0x44a4	0xd0eb	0x2624	0x2e7c	0x1248	0x3330	0xed12
	FM3	0x6c5a	0x44a4	0xd0eb	0x2624	0x2e7c	0x1248	0x3330	0xed12
r5	FM1	0x6c5a	0x44a4	0x2625	0x2624	0x2e7c	0x1248	0x3330	0xed12
	FM2	0x6c5a	0x2625	0xd0eb	0x2624	0x2e7c	0x1248	0x3330	0xed12
	FM3	0x6c5a	0x44a4	0xd0eb	0x2624	0x2e7c	0x1248	0x3330	0xed12
r6	FM1	0x6c5a	0x44a4	0x2625	0x2624	0x2625	0x1248	0x3330	0xed12
	FM2	0x6c5a	0x2625	0xd0eb	0x2624	0xd0ec	0x1248	0x3330	0xed12
	FM3	0x6c5a	0x44a4	0xd0eb	0x2624	0x6c5b	0x1248	0x3330	0xed12
r7	FM1	0x6c5a	0x44a4	0x2625	0x2624	0x2625	0x1249	0x3330	0xed12
	FM2	0x6c5a	0x2625	0xd0eb	0x2624	0xd0ec	0x1249	0x3330	0xed12
	FM3	0x6c5a	0x44a4	0xd0eb	0x2624	0x6c5b	0x1249	0x3330	0xed12

Figure 6.12 – Cycle by cycle analysis, with delay of fault equal to 37

cycle to gather more information and report them in Figure 6.12. The cells colored in blue correspond to the actualization of the register, in purple to a missing actualization, and in red to an unplanned modification of a register.

We observe that the impact is limited to two cycles. At the cycle corresponding to the completion of instruction `adds.w r5, r5, 1`, r_5 is not updated: for FM1 and FM2, the instruction was not performed and replaced by a near identical instruction, except the destination register is different; for FM3, the instruction was skipped without visible side effect. Next cycle, instead of witnessing the execution of `adds.w r6, r6, 1`, we see that, again, a near identical instruction was executed but with the wrong source register.

The binary encoding for the `adds.w` instruction we target is depicted in Figure 6.13. For the sake of clarity, the instruction is divided into two 16-bit segments: the left segment, in blue, contains information for `rn` (the source register) and the right segment, in green, contains information for `rd` (the destination register) and the added immediate. When this instruction is stored unaligned in the prefetch buffer, each half will be on a

`adds.w rd, rn, 1`

31	30	29	28	27	...	19	18	17	16	...	11	10	9	8	...	3	2	1	0
1	1	1	1	0	...	rn				...	rd				...	0	0	0	1

Figure 6.13 – Binary representation of `adds.w rd, rn, 1`

different buffer line, as schematized in Figure 6.14.

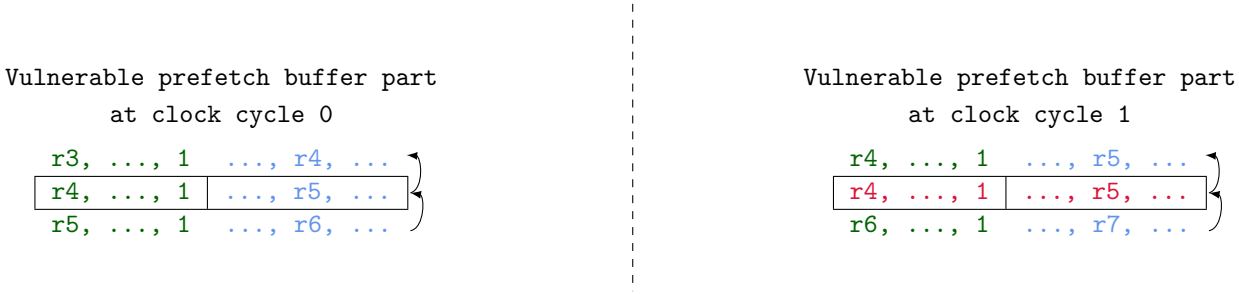


Figure 6.14 – Schematize fault effect on the prefetch buffer

The fault mechanism is the same as previously described, only affecting two instructions instead of one. When the fault occurs, this vulnerable buffer line is not updated. When the instructions are fetched into the pipeline, two successive instructions are modified, composed of information from two different instructions. The modified output is represented in Figure 6.15. It matches FM1, observed in Experiment 6.11.

```
Unfaulted instruction output:
      ...
      adds.w r3, r3, 1
      adds.w r4, r4, 1
      adds.w r5, r5, 1
      adds.w r6, r6, 1
      adds.w r7, r7, 1
      ...
```

```
Faulted instruction output:
      ...
      adds.w r3, r3, 1
      adds.w r4, r4, 1
      adds.w r4, r5, 1
      adds.w r6, r5, 1
      adds.w r7, r7, 1
      ...
```

Figure 6.15 – Unfaulted and faulted instruction outputs

Contrary to previous experiments, the CSCG causes interrupt that escalate into a Hardfault (FM 5). More precisely, it is a IBUSERR type of interrupt, which is, from the informations given in [4], a bus error during the instruction fetch that triggers only when the processor attempts to execute the instruction. We can make an educated guess to explain this effect (although we have no way to prove it). The fault might come from the same phenomenon previously described: two half instructions are repeated during prefetch, which causes two modified instructions to be executed. However, in this case, the two half instructions are not two `adds.w` instructions but one `adds.w` and one `nop.w` instructions. As a result, the processor might be unable to execute the two modified instructions, since they are not legitimate instructions.

Summary - vulnerable processor part

Although we can not identify entirely which processor parts are vulnerable to fault, we have convincing clues from previous sections that either the prefetch transfers or the pipeline are affected. By faulting unaligned instructions and analyzing the fault effects, we deduce that a single prefetch transfer is affected, although we do not know precisely which one.

Chapter Conclusion

The clues gathered in this chapter do not allow us to theorize a complete microarchitectural extension to the ETFM. The hypotheses presented in our preliminary model are globally verified, but when we examine the details, we notice several unexplained phenomena. Although we identify the prefetch mechanism as vulnerable to faults, we observe other fault effects that are probably tied to another identified part of the microarchitecture. Furthermore, some of the fault effects on instructions are not precisely what we theorized it would be. We still demonstrate that state-of-the-art fault models are incomplete and add significant knowledge to the characterization of the CSCG.

Chapter 7

Conclusion and perspectives

This thesis focuses on characterizing a particular impact of electromagnetic fault injection (EMFI) on the phase-locked loop (PLL): the synchronous clock glitch (SCG). Understanding this glitch is crucial to designing efficient countermeasures, especially as it can be used successfully in many-fault injection attacks. In addition, EMFI provokes numerous and complex effects on an integrated circuit (IC), and studying the SCG enhances the general knowledge about the fault mechanisms. The glitch characterization is done at several levels.

In Chapter 2, we presented an overview of the security challenges electronic devices are concerned with. Three levels of vulnerabilities and their associated exploitation methods have been addressed. Among the different attack means, we took an interest in fault injection, which is presented at length in Chapter 3. A fault can affect many elements in an integrated circuit from the transistor to the caches. Extensive knowledge of the possible targets is necessary to accurately characterize a fault effect. Chapter 4 considers three fault injection means (electromagnetic fault injection, voltage, and clock glitch). These methods share some similarities in their impact. The associated fault models are divided into three levels, depending on the fault interpretation level. One impact particularly interests us: the controlled synchronous clock glitch (CSCG). The main objective of this thesis is to characterize this glitch with precision.

First, since none of the state-of-the-art physical fault models applied to the synchronous clock glitch (SCG), an in-depth analysis of the fault impact on electronic components was conducted in Chapter 5. Using physical experimentations performed on an FPGA as well as simulations, we theorized several hypotheses. Since the glitch is carried out by the clock, DFFs are identified as vulnerable. The main fault mechanism concerns the DFF sampling. The glitch prevents some of the clock energy from reaching the vulnerable elements; thus, they might or might not sample incoming data. This

phenomenon is influenced by intrinsic parameters such as process variability. Two DFFs might not be entirely identical, thus requiring different clock energy to perform sampling. Additionally, due to a phenomenon called cross-talk, neighboring wires (clock or data) modify the amount of energy carried out in clock wires, leading to the vulnerable DFFs. We summarize these hypotheses in our first contribution, the Energy Threshold Fault Model (ETFM), published at COSADE24 [32].

From this model, we made assumptions about the RTL interpretation of the CSCG. Although no experimental evidence supports it, we believe that the glitch has several consequences on a simple circuit. First, a fault will only be noticed if a DFF samples different data than the previously stored data. Secondly, when a faulted DFF is part of a chain, the missing sampling will be propagated throughout the chain and in potential calculations. Finally, the missing sampling creates a shift with parallel circuit activities.

Finally, we aimed to gather clues on the SCG impact at the microarchitectural level. Using our assumptions at RTL, we theorized in Chapter 6 a coarse fault model to be verified. To do so, we performed experiments on a Cortex-M3 processor, over which we have limited knowledge and control. These experiments revealed that some fault effects remain unexplained while our coarse model is globally correct. The prefetch mechanism is faulted alongside another (or several) unidentified microarchitectural mechanisms. We also assumed that only changing data during a transfer from one instruction to another is faulted. For example, if two `adds.w` instructions only differ in their source register, so the fault does not impact the immediate. While this effect is globally observed but not as we theorized in our coarse model. Although our extension to the Energy Threshold Fault Model (ETFM) is incomplete, we still add significant knowledge to the microarchitectural effect of the SCG.

7.1 Perspectives

We built our microarchitectural extension of the ETFM by faulting a commercial processor on which we have little knowledge and control. To enhance the CSCG characterization, we believe a change of the target is necessary (while still being a processor). First, there is no guarantee that the CSCG has the same impact on a different processor. Published works so far only featured a Cortex-M3. Perhaps, more importantly, we need

a transparent, adjustable processor. This way, we can target specific parts to be faulted: instruction or data transfers, pipeline stage, optimization mechanisms, etc. Additionally, it is easier to retrieve information, possibly at any given time. The best way to achieve this is to implement a softcore on an FPGA or to rely on an ASIC on which we have complete knowledge.

Our study of the SCG uses faults injected with TRAITOR. The tool injects clock glitches, referred to as CSCG. The equivalence between the two kinds of glitches is not demonstrated.

To refine our model, it would be necessary to recreate the SCG using EMFI and study its effect. This being said, when injected using EMFI, the SCG exists among many other effects, and some are characterized in state-of-the-art works, some possibly not. To this day, a complete characterization, considering all effects, has not yet been theorized. Isolating, or at least recognizing, the SCG from the other effects is necessary to study. To do so, an in-depth study of the physical fault models is necessary, as done in Chapter 4 for state-of-the-art models and Chapter 5 for the ETFM. When looking into the details, the different effects happen at different timings, for different polarity injections, impact different signals, etc. By using these discriminatory elements, it may be theoretically possible to propose an advanced EMFI characterization encompassing every previously observed fault effect, as well as potential other effects.

BIBLIOGRAPHY

[1] *7 Series FPGAs Clocking Resources*, https://docs.xilinx.com/v/u/en-US/ug472_7Series_Clock

[2] Michel Agoyan, Jean-Max Dutertre, David Naccache, Bruno Robisson, and Assia Tria, « When Clocks Fail: On Critical Paths and Clock Faults », *in*: *Smart Card Research and Advanced Application, 9th IFIP WG 8.8/11.2 International Conference, CARDIS 2010, Passau, Germany. Proceedings*, ed. by Dieter Gollmann, Jean-Louis Lanet, and Julien Iguchi- Cartigny, vol. 6035, Lecture Notes in Computer Science, Springer, Apr. 2010, pp. 182–193, DOI: 10.1007/978-3-642-12510-2_13.

[3] Ihab Alshaer, Brice Colombier, Christophe Deleuze, Vincent Beroulle, and Paolo Maistri, « Microarchitectural Insights into Unexplained Behaviors under Clock Glitch Fault Injection », *in*: *22nd Smart Card Research and Advanced Application Conference (CARDIS 2023)*, ed. by Springer, Lecture Notes in Computer Science (LNCS), Amsterdam, Netherlands: Springer, Nov. 2023, pp. 1–20, URL: https://hal.science/hal-04273995.

[4] *ARMv7-M Architecture Reference Manuel*.

[5] Kailtlin Boeckl, Michael Fagan, William Fisher, Naomi Lefkovitz, Katerina Megas, Ellen Nadeau, Ben Piccarreta, Danna Gabel O'Rourke, and Karen Scarfone, « Considerations for Managing Internet of Things (IoT) Cybersecurity and Privacy Risks », *in*: (), DOI: 10.6028/NIST.IR.8228.

[6] Erik Buchanan, Ryan Roemer, Hovav Shacham, and Stefan Savage, « When good instructions go bad: generalizing return-oriented programming to RISC », *in*: New York, NY, USA: Association for Computing Machinery, 2008, ISBN: 9781595938107, DOI: 10.1145/1455770.1455776, URL: https://doi.org/10.1145/1455770.1455776.

[7] Doris Chen, Deshanand Singh, Jeffrey Chromczak, David Lewis, Ryan Fung, David Neto, and Vaughn Betz, « A Comprehensive Approach to Modeling, Characterizing and Optimizing for Metastability in FPGAs », *in*: *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, Feb. 2010, pp. 167–176, DOI: 10.1145/1723112.1723142.

[8] Ludovic Claudepierre and Philippe Besnier, « Microcontroller Sensitivity to Fault-Injection Induced by Near-Field Electromagnetic Interference », *in*: *Joint International Symposium on Electromagnetic Compatibility , Sapporo and Asia-Pacific International Symposium on Electromagnetic Compatibility (EMC Sapporo/APEMC)*, 2019, pp. 673–676, DOI: 10.23919/EMCTokyo.2019.8893701.

[9] Ludovic Claudepierre, Pierre-Yves Péneau, Damien Hardy, and Erven Rohou, « TRAITOR: A Low-Cost Evaluation Platform for Multifault Injection », *in*: *ASSS'21: Proceedings of the 2021 International Symposium on Advanced Security on Software and Systems, Virtual Event, Hong Kong*, ed. by Weizhi Meng and Li Li, ACM, June 2021, pp. 51–56, DOI: 10.1145/3457340.3458303.

[10] Brice Colombier, Alexandre Menu, Jean-Max Dutertre, Pierre-Alain Moëllic, Jean-Baptiste Rigaud, and Jean-Luc Danger, « Laser-induced Single-bit Faults in Flash Memory: Instructions Corruption on a 32-bit Microcontroller », *in*: *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2019, pp. 1–10, DOI: 10.1109/HST.2019.8741030.

[11] Benoit Coqueret, Mathieu Carbone, Olivier Sentieys, and Gabriel Zaid, *A Hard-Label Cryptanalytic Extraction of Non-Fully Connected Deep Neural Networks using Side-Channel Attacks*, Cryptology ePrint Archive, Paper 2024/1870, 2024, URL: https://eprint.iacr.org/2024/1870.

[12] *Cortex-M3 Technical Reference Manual*.

[13] *Cosmic particles can change elections and cause planes to fall through the sky, scientists warn*, https://www.independent.co.uk/news/science/subatomic-particles-cosmic-rays-computers-change-elections-planes-autopilot-a7584616.html.

[14] *Cosmic Ray Flips Bit, Assists Mario 64 Speedrunner*, https://hackaday.com/2021/02/17/cosmic-ray-flips-bit-assists-mario-64-speedrunner/.

[15] Amine Dehbaoui, Jean-Max Dutertre, Bruno Robisson, and Assia Tria, « Electromagnetic Transient Faults Injection on a Hardware and a Software Implementations of AES », *in*: *Workshop on Fault Diagnosis and Tolerance in Cryptography, Leuven, Belgium*, ed. by Guido Bertoni and Benedikt Gierlichs, IEEE Computer Society, Sept. 2012, pp. 7–15, DOI: 10.1109/FDTC.2012.15.

[16] Mark Dowd, John McDonald, and Justin Schuh, *The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities*, Addison-Wesley Professional, 2006, ISBN: 0321444426.

[17] Mathieu Dumont, Mathieu Lisart, and Philippe Maurine, « Modeling and Simulating Electromagnetic Fault Injection », *in*: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40.*4* (2021), pp. 680–693, DOI: 10.1109/TCAD.2020.3003287.

[18] Louis Dureuil, Guillaume Petiot, Marie-Laure Potet, Thanh-Ha Le, Aude Crohen, and Philippe de Choudens, « FISSC: a fault injection and simulation secure collection », *in*: *Computer Safety, Reliability and Security*, vol. 9922, Lecture Notes in Computer Science, Trondheim, Norway: Springer, Sept. 2016, pp. 3–11, DOI: 10.1007/978-3-319-45477-1_1, URL: https://hal.science/hal-03784107.

[19] Jean-Max Dutertre, Vincent Beroulle, Philippe Candelier, Stephan de Castro, Louis-Barthelemy Faber, Marie-Lise Flottes, Philippe Gendrier, David Hely, Régis 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*: *14th Workshop on Fault Diagnosis and Tolerance in Cryptography*, Amsterdam, Netherlands: IEEE, Sept. 2018, pp. 1–6, DOI: 10.1109/FDTC.2018.00009, URL: https://hal-emse.ccsd.cnrs.fr/emse-01856008.

[20] Jean-Max Dutertre, Timothé Riom, Olivier Potin, and Jean-Baptiste Rigaud, « Experimental Analysis of the Laser-Induced Instruction Skip Fault Model », *in*: *The 24th Nordic Conference on Secure IT Systems, Nordsec 2019*, Aalborg, Denmark, Nov. 2019, pp. 221–237, DOI: 10.1007/978-3-030-35055-0_14, URL: https://hal.science/hal-02379754.

[21] *Eldo Platform*, https://eda.sw.siemens.com/en-US/ic/eldo/.

[22] Antoine Gicquel, « Étude de vulnérabilité d'un programme au format binaire en présence de fautes précises et nombreuses », Theses, Université de Rennes, Dec. 2024, URL: https://hal.science/tel-04842415.

[23] Antoine Gicquel, Damien Hardy, Karine Heydemann, and Erven Rohou, « SAMVA: Static Analysis for Multi-fault Attack Paths Determination », *in*: *Constructive Side-Channel Analysis and Secure Design - 14th International Workshop, COSADE 2023, Munich, Germany, Proceedings*, ed. by Elif Bilge Kavun and Michael Pehl,

vol. 13979, Lecture Notes in Computer Science, Springer, Apr. 2023, pp. 3–22, DOI: 10.1007/978-3-031-29497-6_1.

[24] Ilias Giechaskiel, Kasper B. Rasmussen, and Ken Eguro, « Leaky Wires: Information Leakage and Covert Communication Between FPGA Long Wires », *in*: *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ASIACCS '18, Incheon, Republic of Korea: Association for Computing Machinery, 2018, pp. 15–27, ISBN: 9781450355766, DOI: 10.1145/3196494.3196518, URL: https://doi.org/10.1145/3196494.3196518%7D.

[25] Dan Kaminsky, *It's The End Of The Cache As We Know It*, 2008.

[26] Vanthanh Khuat, Jean-Luc Danger, and Jean-Max Dutertre, « Laser Fault Injection in a 32-bit Microcontroller: from the Flash Interface to the Execution Pipeline », *in*: *18th Workshop on Fault Detection and Tolerance in Cryptography , FDTC 2021, Milan, Italy*, IEEE, Sept. 2021, pp. 74–85, DOI: 10.1109/FDTC53659.2021.00020.

[27] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom, « Spectre Attacks: Exploiting Speculative Execution », *in*: *40th IEEE Symposium on Security and Privacy*, 2019.

[28] Ronan Lashermes, Guillaume Reymond, Jean-Max Dutertre, Jacques Fournier, Bruno Robisson, and Assia Tria, « A DFA on AES Based on the Entropy of Error Distributions », *in*: *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography*, 2012, pp. 34–43, DOI: 10.1109/FDTC.2012.18.

[29] Johan Laurent, Vincent Beroulle, Christophe Deleuze, and Florian Pebay-Peyroula, « Fault Injection on Hidden Registers in a RISC-V Rocket Processor and Software Countermeasures », *in*: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019, pp. 252–255.

[30] Haohao Liao and Catherine H. Gebotys, « Methodology for EM Fault Injection: Charge-based Fault Model », *in*: *Design, Automation & Test in Europe Conference & Exhibition, DATE 2019, Florence, Italy*, ed. by Jürgen Teich and Franco Fummi, IEEE, Mar. 2019, pp. 256–259, DOI: 10.23919/DATE.2019.8715150.

[31] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom,

and Mike Hamburg, « Meltdown: Reading Kernel Memory from User Space », *in*: *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[32] Amélie Marotta, Ronan Lashermes, Guillaume Bouffard, Olivier Sentieys, and Rachid Dafali, « Characterizing and Modeling Synchronous Clock-Glitch Fault Injection », *in*: *Lecture Notes in Computer Science*, vol. 14595, Lecture Notes in Computer Science, Gardanne, France: Springer Nature Switzerland, Apr. 2024, pp. 3–21, DOI: 10.1007/978-3-031-57543-3_1, URL: https://inria.hal.science/hal-04549548.

[33] Alexandre Menu, Shivam Bhasin, Jean-Max Dutertre, Jean-Baptiste Rigaud, and Jean-Luc Danger, « Precise Spatio-Temporal Electromagnetic Fault Injections on Data Transfers », *in*: *2019 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 2019, pp. 1–8, DOI: 10.1109/FDTC.2019.00009.

[34] Alexandre Menu, Jean-Max Dutertre, Olivier Potin, Jean-Baptiste Rigaud, and Jean-Luc Danger, « Experimental Analysis of the Electromagnetic Instruction Skip Fault Model », *in*: *2020 15th Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*, 2020, pp. 1–7, DOI: 10.1109/DTIS48698.2020.9081261.

[35] Nicolas Moro, Amine Dehbaoui, Karine Heydemann, Bruno Robisson, and Emmanuelle Encrenaz, « Electromagnetic Fault Injection: Towards a Fault Model on a 32-bit Microcontroller », *in*: *Workshop on Fault Diagnosis and Tolerance in Cryptography, Los Alamitos, CA, USA*, ed. by Wieland Fischer and Jörn-Marc Schmidt, IEEE Computer Society, Aug. 2013, pp. 77–88, DOI: 10.1109/FDTC.2013.9.

[36] Roukoz Nabhan, Jean-Max Dutertre, Jean-Baptiste Rigaud, Jean-Luc Danger, and Laurent Sauvage, « A Tale of Two Models: Discussing the Timing and Sampling EM Fault Injection Models », *in*: *2023 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*, 2023, pp. 1–12, DOI: 10.1109/FDTC60478.2023.00010.

[37] Roukoz Nabhan, Jean-Max Dutertre, Jean-Baptiste Rigaud, Jean-Luc Danger, and Laurent Sauvage, « Highlighting Two EM Fault Models While Analyzing a Digital Sensor Limitations », *in*: *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023, pp. 1–2, DOI: 10.23919/DATE56975.2023.10137124.

[38] Sebastien Ordas, Ludovic Guillaume-Sage, and Philippe Maurine, « Electromagnetic fault injection: the curse of flip-flops », *in*: *Journal of Cryptographic Engineering* 7.*3* (2017), pp. 183–197, DOI: 10.1007/s13389-016-0128-3.

[39] Pierre-Yves Péneau, Ludovic Claudepierre, Damien Hardy, and Erven Rohou, « NOP-Oriented Programming: Should we Care? », *in*: *IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2020, Genoa, Italy*, IEEE, Sept. 2020, pp. 694–703, DOI: 10.1109/EuroSPW51379.2020.00100.

[40] Duy-Phuc Pham, Damien Marion, Mathieu Mastio, and Annelie Heuser, « Obfuscation Revealed: Leveraging Electromagnetic Signals for Obfuscated Malware Classification », *in*: *ACSAC 2021 - Annual Computer Security Applications Conference*, Virtual Event, France: ACM, Dec. 2021, pp. 1–14, DOI: 10.1145/3485832.3485894, URL: https://hal.science/hal-03374399.

[41] Lorraine E. Prokop, « Historical Aerospace Software Errors Categorized to Influence Fault Tolerance », *in*: *2024 IEEE Aerospace Conference*, 2024, pp. 1–12, DOI: 10.1109/AERO58975.2024.10521061.

[42] Lionel Rivière, Zakaria Najm, Pablo Rauzy, Jean-Luc Danger, Julien Bringer, and Laurent Sauvage, « High precision fault injections on the instruction cache of ARMv7-M architectures », *in*: *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2015, pp. 62–67, DOI: 10.1109/HST.2015.7140238.

[43] Cyril Roscian, Jean-Max Dutertre, and Assia Tria, « Frontside laser fault injection on cryptosystems - Application to the AES' last round - », *in*: *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2013, pp. 119–124, DOI: 10.1109/HST.2013.6581576.

[44] Cyril Roscian, Alexandre Sarafianos, Jean-Max Dutertre, and Assia Tria, « Fault Model Analysis of Laser-Induced Faults in SRAM Memory Cells », *in*: *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2013 Workshop on*, Santa-Barbara, United States, Aug. 2013, DOI: 10.1109/FDTC.2013.17, URL: https://hal-emse.ccsd.cnrs.fr/emse-01109133.

[45] Nidhal Selmane, Sylvain Guilley, and Jean-Luc Danger, « Practical Setup Time Violation Attacks on AES », *in*: *Seventh European Dependable Computing Conference, EDCC-7 2008, Kaunas, Lithuania*, IEEE Computer Society, May 2008, pp. 91–96, DOI: 10.1109/EDCC-7.2008.11.

[46] *Smashing The Stack For Fun And Profit*, https://phrack.org/issues/49/14#article.

[47] Eran Tromer, Dag Arne Osvik, and Adi Shamir, « Efficient Cache Attacks on AES, and Countermeasures », *in*: *Journal of Cryptology*, 2010, DOI: 10.1007/s00145-009-9049-y.

[48] Thomas Trouchkine, Guillaume Bouffard, and Jessy Clédi` ere, « EM Fault Model Characterization on SoCs: From Different Architectures to the Same Fault Model », *in*: *18th Workshop on Fault Detection and Tolerance in Cryptography , FDTC 2021, Milan, Italy*, IEEE, Sept. 2021, pp. 31–38, DOI: 10.1109/FDTC53659.2021.00014.

[49] Thomas Trouchkine, Guillaume Bouffard, and Jessy Clédi` ere, « Fault Injection Characterization on Modern CPUs », *in*: *Information Security Theory and Practice - 13th IFIP WG 11.2 International Conference, WISTP 2019, Paris, France, Proceedings*, ed. by Maryline Laurent and Thanassis Giannetsos, vol. 12024, Lecture Notes in Computer Science, Springer, Dec. 2019, pp. 123–138, DOI: 10.1007/978-3-030-41702-4_8.

[50] Thomas Trouchkine, Sébanjila Kevin Bukasa, Mathieu Escouteloup, Ronan Lashermes, and Guillaume Bouffard, « Electromagnetic fault injection against a complex CPU, toward new micro-architectural fault models », *in*: *Journal of Cryptographic Engineering* 11.*4* (2021), pp. 353–367, DOI: 10.1007/S13389-021-00259-6.

[51] Yuval Yarom and Katrina Falkner, « FLUSH+RELOAD: a high resolution, low noise, L3 cache side-channel attack », *in*: *Proceedings of the 23rd USENIX Conference on Security Symposium*, 2014, ISBN: 9781931971157.

[52] Shih-Yi Yuan, Yu-Lun Wu, Richard Perdriau, Shry-Sann Liao, and Hao-Ping Ho, « Electromagnetic interference analysis using an embedded phase-lock loop », *in*: *Asia-Pacific Symposium on Electromagnetic Compatibility*, 2012, pp. 189–192, DOI: 10.1109/APEMC.2012.6237910.

[53] Maoshen Zhang, He Li, and Qiang Liu, « Deep Exploration on Fault Model of Electromagnetic Pulse Attack », *in*: *IEEE Transactions on Nanotechnology* 21 (2022), pp. 598–605, DOI: 10.1109/TNANO.2022.3214341.

COLLEGE DOCTORAL BRETAGNE
MATHS, TELECOMS
INFORMATIQUE, SIGNAL
SYSTEMES, ELECTRONIQUE

Université de Rennes

Titre : Effets des perturbations synchrones de l'horloge sur la sécurité des circuits intégrés

Mot clés : Injection de fautes électromagnétiques, modèle de faute, clock glitch, sécurité matérielle

Résumé : Lors de la conception d'un objet électronique, la sécurité est à prendre en considération. En effet, les sources de vulnérabilité peuvent être multiples, ainsi que les moyens de les exploiter. En particulier, nous nous intéressons à l'injection de fautes. Ces attaques consistent à perturber certains signaux d'un circuit (comme l'alimentation) afin de modifier son comportement. Que ce soit pour développer des contremesures ou des attaques efficaces, il est nécessaire de comprendre l'impact global des fautes sur un circuit intégré. L'injection de fautes électromagnétiques impacte plusieurs signaux à la fois, et donc son étude peut se révéler complexe. Cette thèse vise à étudier un effet en particulier des fautes électromagnétiques, les perturbations synchrones de l'horloge. Ce type de perturbation a été utilisé avec succès pour contourner des mesures de sécurité. Pourtant, une analyse de bout en bout n'a jamais été explorée. Dans un premier temps, nous explorons leur effet sur les bascules et leur échantillonnage, ce qui nous permet de déduire un nouveau modèle de faute. Dans un second temps, notre intérêt se porte sur l'effet des perturbations sur la microarchitecture. Nos buts sont multiples : faire le lien entre les paramètres d'injection et les différents effets observés, identifier les parties vulnérables du processeur, faire le lien avec le modèle de faute bas niveau. Ces deux contributions permettent d'améliorer la compréhension des effets de l'injection de fautes, notamment électromagnétiques, à divers niveaux d'abstraction.

Title: Effects of synchronous clock glitch on the security of integrated circuits

Keywords: Electromagnetic fault injection, fault model, clock glitch, hardware security

Abstract: When designing an electronic device, security is a key aspect to consider. There are numerous vulnerability sources and exploitation methods. In particular, we are interested in fault injection. These attacks consist of perturbing some of the circuit signals (such as the power supply) to modify their behaviour. Understanding the impact of faults on an integrated circuit is necessary to design effective countermeasures or attacks. Electromagnetic fault injection impacts several signals at once, so its study can be complex. This thesis aims to study one particular effect of electromagnetic faults: the synchronous clock glitch. This glitch has been used successfully to bypass security measures. However, an in-depth analysis has never been explored. First, we explore its effect on registers and their sampling, which allows us to identify a new fault model. We then focus on the effect of the glitch on the microarchitecture. Our goals are multiple: to establish the link between the injection parameters and the various effects observed, to identify the vulnerable parts of the processor, and to establish the link with the low-level fault model. These two contributions will improve our understanding of the effects of fault injection, particularly electromagnetic faults, at various levels of abstraction.