

# RAPPORT DE PROJET DE FIN D'ETUDES

## GRENoble INP Esisar 2024/2025

**Titre du projet**

Countremesures face aux attaques laser en hors tension sur cible FPGA

**Nom et adresse de l'entreprise**

LCIS (Laboratoire de Conception et d'Intégration des Systèmes)

50 Rue Barthélémy de Laffemas, 26000 Valence

**Nom et prénom de l'étudiant**

**GUERIN Enzo**

Dates du stage	03/02/2025 - 04/07/2025
Spécialité	Informatique (ISE)
Tuteur Entreprise	BEROULLE Vincent
Tuteur ESISAR	KOENIG Damien

# RAPPORT DE PROJET DE FIN D'ETUDES

## GRENOBLE INP Esisar 2024/2025

### **Mots clés :**

FPGA, attaques matérielles, attaques thermiques hors tension (POTAs), détection d'attaques, sécurité matérielle, vieillissement des circuits, oscillateur en anneau (RO), oscillateur en anneau à entrée unique (SIRO), mécanisme de test robuste (MTR), autotest, modèle d'attaquant.

### **Résumé :**

Ce rapport présente la conception et la validation d'un mécanisme de test robuste, destiné à contrer les nouvelles menaces physiques telles que les attaques thermiques hors tension sur les FPGA.

L'objectif du projet est de proposer une solution permettant non seulement de détecter ces attaques, mais aussi de limiter leurs impacts grâce à un module d'auto-test (ATM).

Le MTR repose sur deux signaux clés,  $\text{clk}_s$ , une horloge externe de référence, et  $\text{ro}_{\text{out}}$ , la sortie d'un oscillateur à anneau (RO) interne.

Ce signal  $\text{ro}_{\text{out}}$  est utilisé à la fois comme capteur passif de variation physique et comme élément de sécurité.

Une analyse théorique des modules, combinée à une modélisation mathématique et à une carte thermique expérimentale, a permis de valider la robustesse du système face aux attaques simulées.

Les résultats obtenus montrent une conformité entre la théorie et l'implémentation sur différents modèles de FPGA, confirmant la pertinence du mécanisme proposé.

# RAPPORT DE PROJET DE FIN D'ETUDES

## GRENOBLE INP Esisar 2024/2025

### **Keywords :**

FPGA, physical attacks, Power-Off Temperature Attacks (POTAs), attack detection, hardware security, circuit aging, Ring Oscillator (RO), Single Input Ring Oscillator (SIRO), Robust Test Mechanism (MTR), self-test, attacker model.

### **Abstract :**

This report presents the design and validation of a robust detection mechanism, aimed at countering new physical threats such as power-off temperature attacks (POTAs) on FPGAs.

The project's objective is to propose a solution capable not only of detecting the effects induced by these attacks but also of limiting their impact through a self-test module (ATM).

The mechanism relies on two key signals:  $\text{clk}_s$ , an external reference clock, and  $\text{ro}_{\text{out}}$ , the output of an internal ring oscillator (RO).

This module is used both as a passive sensor for physical variations and as a security element.

A theoretical analysis of the modules, combined with mathematical modeling and experimental heat mapping, validated the system's robustness against simulated attacks.

The results show consistency between theory and implementation on various FPGA models, confirming the relevance of the proposed mechanism.

# Table des matières

1	Remerciements .....	6
2	Introduction .....	7
2.1	Contexte .....	7
2.1.1	Projet POP-ANR .....	7
2.1.2	PFE .....	7
2.2	Problématique .....	8
2.3	Solution & contributions .....	8
2.4	Principe de fonctionnement du MTR .....	8
2.5	Modèle d'attaquant .....	9
3	Présentation du LCIS .....	10
4	Travaux précédents .....	11
4.1	Étude du vieillissement des ROs .....	11
4.2	Pré-étude d'un mécanisme de détection .....	12
5	Méthodologie .....	13
5.1	Déroulement du projet .....	13
5.2	Fonctionnement et limites théoriques des modules .....	13
5.3	Formalisation des modules .....	16
6	Protocole expérimental .....	19
6.1	Module DéTECTEUR et protocoles de validation .....	20
6.1.1	Fonctionnement et implémentation du détecteur .....	20
6.1.2	Protocole de validation en simulation du module DéTECTEUR .....	20
6.1.3	Protocole de validation sur FPGA du module DéTECTEUR .....	21
6.2	Module ATM et protocoles de validation .....	21
6.2.1	Fonctionnement et implémentation de l'ATM .....	21
6.2.2	Protocole de validation en simulation du module ATM .....	22
6.2.3	Protocole de validation sur FPGA du module ATM .....	22
6.3	Module RO_SIRO et protocoles de validation .....	23
6.3.1	Fonctionnement et implémentation du module RO_SIRO .....	23
6.3.2	Protocole de validation en simulation du module RO_SIRO .....	24
6.3.3	Protocole de validation sur FPGA du module RO_SIRO .....	24
6.4	Mécanisme de Test Robuste (MTR) et protocoles de validation .....	24
6.4.1	Fonctionnement et implémentation du MTR .....	24
6.4.2	Protocole de validation en simulation du MTR .....	25
6.4.3	Protocole de validation sur FPGA du module MTR .....	25
6.5	Carte thermique des états d'alarmes .....	25
7	Résultats et discussion .....	26
7.1	Module DéTECTEUR .....	26
7.1.1	Validation en simulation du module DéTECTEUR .....	26
7.1.2	Validation sur FPGA du module DéTECTEUR .....	26
7.2	Module ATM .....	27
7.2.1	Validation en simulation de l'ATM .....	27
7.2.2	Validation sur FPGA de l'ATM .....	27

7.3	Module RO_SIRO .....	27
7.3.1	Validation en simulation du module RO_SIRO .....	27
7.3.2	Validation sur FPGA du module RO_SIRO .....	28
7.4	Mécanisme de test robuste (MTR) .....	29
7.4.1	Validation en simulation du MTR .....	29
7.4.2	Validation sur FPGA du MTR .....	29
7.5	Simulation par itération des valeurs de fréquences .....	30
7.6	Campagnes d'expérimentation .....	31
8	Budget du projet et analyse RSE .....	36
9	Conclusion et perspectives .....	37
10	Annexes .....	38
10.A	Code Detector .....	38
10.B	Code ATM .....	39
10.C	Code RO_SIRO .....	41
10.D	Code MTR .....	42
10.E	Testbench Simulation Itérations .....	44
10.F	Code Python Génération Heatmap .....	46
	Bibliographie .....	49

# **1 Remerciements**

Je tiens à exprimer ma sincère gratitude à toutes les personnes dont le soutien et l'encadrement ont rendu possible la réalisation de ce PFE.

Je souhaite tout particulièrement remercier M. Vincent BEROULLE, directeur du LCIS, pour son accueil, ses conseils précieux, ainsi que pour sa disponibilité et son accompagnement tout au long de cette expérience. Son professionnalisme et sa pédagogie ont grandement contribué à mon apprentissage pratique et à mon intégration au sein du LCIS.

J'adresse également mes remerciements chaleureux à l'ensemble des membres de l'équipe du LCIS, pour leur accueil, leur collaboration et leur bienveillance durant ces semaines de formation.

Je tiens à remercier l'ESISAR ainsi que mes enseignants, et notamment M. Damien KOENIG, M. Guy DEHAY, M. Nicolas BARBOT, que ce soit pour leurs enseignements, leur suivi ou le rôle essentiel qu'ils ont joué dans mon parcours académique ou professionnel.

Je remercie Adrien GAFFET et Enzo LEMEN pour leurs précieux conseils, qui m'auront permis de mener à bien ce PFE.

Je remercie également mes parents, ainsi que ma petite amie pour m'avoir accompagné et soutenu durant ce PFE. Je n'aurais pas pu écrire ce rapport sans eux.

Enfin, je tiens à dédier ce rapport à ma grand-mère, récemment décédée après un long combat contre un cancer. Malgré la douleur et cette période profondément éprouvante, j'ai trouvé la force de mener ce projet à son terme. Je sais qu'elle aurait été fière, et c'est en pensant à elle que je l'ai achevé.

Repose en paix, mamie.

## 2 Introduction

### 2.1 Contexte

#### 2.1.1 Projet POP-ANR

Le projet ANR POP vise à explorer une nouvelle forme d'attaque physique sur des composants électroniques de sécurité, en particulier lorsqu'ils ne sont pas alimentés. L'idée centrale est de déterminer si un attaquant peut utiliser un laser pour altérer le fonctionnement de certains éléments matériels comme les PUF (Physically Unclonable Functions) ou les capteurs physiques de détection d'intrusion, même lorsque le circuit est éteint. Ces attaques pourraient permettre de cloner des dispositifs sécurisés ou de désactiver subtilement les systèmes de protection sans être détecté.

Outre les PUF et capteurs, d'autres composants comme les générateurs de nombres aléatoires ou certaines mémoires non volatiles peuvent également être analysés. Les effets du laser peuvent aller d'une modification locale des propriétés électriques des transistors jusqu'à la destruction précise de connexions dans le circuit, restant difficilement détectable par les mesures de sécurité classiques.

Le projet se divise en cinq axes principaux : la conception de circuits expérimentaux, les tests au laser et la modélisation de leurs effets, la création de protections actives (dans laquelle mon PFE s'inscrit) et passives, la mise en œuvre d'attaques avancées ainsi que la réalisation de démonstrateurs concrets. Il implique quatre laboratoires français, TIMA, LCIS, Laboratoire Hubert Curien et MSE, qui collaborent à toutes ces étapes.

À court terme, le projet ambitionne de publier des résultats scientifiques dans des revues prestigieuses. À moyen terme, il cherche à influencer les organismes spécialisés dans l'évaluation et la certification de la sécurité des systèmes électroniques. À long terme, il souhaite faire reconnaître les attaques laser hors tension comme une menace majeure, poussant à intégrer dès la conception des protections adaptées.

#### 2.1.2 PFE

Dans le cadre de ce PFE, la recherche s'est focalisée sur les attaques contre les détecteurs d'attaques sans lever d'alarme.

En effet, comme évoqué précédemment, afin de contrer la menace des attaques, les FPGAs modernes intègrent généralement des détecteurs capables de contrer certains types d'attaques physiques au moment de l'allumage du dispositif (power-on). Toutefois, ces protections restent insuffisantes face à un nouveau type d'attaques, réalisées lorsque le circuit est éteint (power-off), comme c'est le cas avec les Power-Off Temperature Attacks (POTAs).

Ces attaques récentes, comme montré dans les travaux de Maryam ESMAEILIAN [1] et d'Aghiles DOUADI [2], exploitent les variations thermiques subies par le FPGA en absence d'alimentation électrique, provoquant un vieillissement accéléré des composants. Ce phénomène peut entraîner des dégradations des performances du circuit,

notamment des variations de fréquence pouvant conduire à des erreurs critiques telles que des sauts d'instructions.

## 2.2 Problématique

Dans ce contexte, la nécessité de mettre en place de nouvelles primitives de sécurité robustes s'impose plus que jamais. En effet, si les potentiels d'attaque évoluent, les solutions de protection existantes doivent également être repensées pour garantir une sécurité accrue des FPGAs, notamment lors des phases hors alimentation (power-off).

À cet effet, nous nous sommes penchés dans le cadre de ce projet sur la conception d'un mécanisme de détection robuste, nommé MTR (Mécanisme de Test Robuste). Ce dernier est constitué d'un détecteur avec un mécanisme d'auto-test (qu'on nommera ATM) pour assurer que le détecteur n'a pas été attaqué/corrompu.

## 2.3 Solution & contributions

L'objectif consiste ainsi à créer un système capable non seulement de détecter les effets induits par des attaques de type POTAs, mais aussi d'en limiter les impacts à travers un mécanisme de test fiable et autonome.

Pour ce faire, une implémentation et validation d'un mécanisme de test robuste pour un détecteur d'attaque basé sur le temps seront réalisées.

Cette implémentation repose principalement sur deux signaux :

- `clk_s`, constituant l'horloge de référence
- `ro_out`, correspondant à la sortie du Ring Oscillator (RO)

Le MTR regroupe deux modules principaux :

1. Détecteur : primitive de sécurité préexistante, permettant de détecter la présence éventuelle de glitches sur `clk_s` selon un intervalle prédéfini.
2. Auto\_Test\_Module (ATM) : module permettant de détecter un décalage éventuel dans la fréquence de `ro_out` selon un intervalle prédéfini.

De plus, une analyse sur la sécurité du mécanisme de protection proposé (zones couvertes et limites de détection) sera menée. Enfin, une campagne expérimentale sur plusieurs FPGA permettant de valider la protection en présence d'attaques en température sera conduite en fin de projet.

## 2.4 Principe de fonctionnement du MTR

La figure ci-dessous résume les dépendances entre les modules du Détecteur et de l'ATM. Le but de la réalisation de l'ATM est de détecter les attaques éventuelles sur le Détecteur. Ce dernier est fonction de `clk_s`, et doit, pour fonctionner correctement, utiliser un `ro_out` « correct ». L'ATM, fonction de `ro_out`, vient garantir cela. Cependant, l'ATM est également dépendant de `clk_s`, dans le sens où ce dernier doit être « correct » pour que l'ATM puisse fonctionner correctement. Cela est garanti par le Détecteur. Il advient ainsi que les deux modules sont interdépendants, c'est à dire se protègent mutuellement.

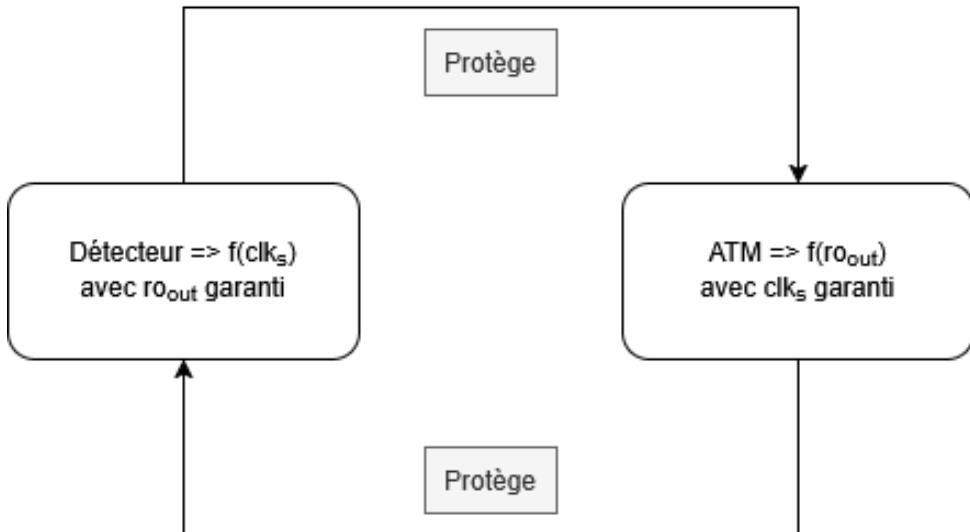


Fig. 1. – Schéma des interdépendances entre les deux modules majeurs du MTR.

## 2.5 Modèle d'attaquant

`clk_s` est contrôlable par l'attaquant (étant un élément externe au circuit, contrairement au RO qui est un élément interne du circuit). Cela n'est pas un problème étant donné que le Détecteur vient vérifier que `clk_s` n'est pas glitché/modifié hors des zones de fonctionnement normales.

Le problème intervient lorsque l'attaquant veut théoriquement mener une attaque simultanée ou par itération sur `clk_s` et `ro_out` (monter petit à petit `clk_s` et/ou `ro_out` et réajuster en fonction des résultats, ici, les alarmes levées), ce qui serait difficile à mettre en place par manque précis de contrôle sur le RO, car étant un élément interne, ainsi que par le nombre limité d'essais avant de se faire repérer (protections systèmes), limitant grandement ses tentatives d'attaque par itération.

On peut ainsi formaliser le modèle d'attaquant de la manière suivante :

1. `clk_s` est contrôlable précisément car c'est un élément externe.
2. `ro_out` est contrôlable de façon imprécise car c'est un élément interne.
3. l'attaquant a un nombre limité d'essais, car après quelques alertes (nombre défini par l'utilisateur) le système se met en sécurité et se bloque.

En observant ce schéma, et en gardant à l'esprit notre dernière remarque, une question s'en dégage : existe t-il un intervalle { `clk_s` , `ro_out` } où le système n'est pas protégé ? Une réponse à cette question sera détaillée dans l'étude analytique sur la théorie des modules.

### 3 Présentation du LCIS

Le Laboratoire de Conception et d'Intégration des Systèmes, communément appelé LCIS, est un laboratoire de recherche français spécialisé dans les domaines des systèmes embarqués et communicants. Crée en 1996, le LCIS s'inscrit dans un écosystème technologique dynamique et collabore activement avec le pôle Minalogic, dédié à l'innovation en microélectronique.

Les recherches menées au sein du laboratoire couvrent un large spectre de thématiques liées à la conception avancée de circuits intégrés analogiques, mixtes et radiofréquence, aux architectures reconfigurables telles que les FPGAs, à l'intégration 3D de composants électroniques ainsi qu'à la sécurité matérielle. Le LCIS accorde également une attention particulière aux applications critiques dans les domaines de l'aéronautique, de l'automobile, de l'énergie et du médical, où fiabilité et robustesse sont des exigences essentielles.



Fig. 2. – Logo du LCIS.



Fig. 3. – Bâtiment principal du LCIS  
(source : Google Earth)

Son rôle ne se limite pas uniquement à la recherche fondamentale car il s'agit aussi d'un acteur important dans le transfert technologique vers l'industrie. En effet, le laboratoire collabore étroitement avec des entreprises majeures telles que STMicroelectronics, Thales ou encore Airbus, dans le cadre de projets nationaux et internationaux. Par ailleurs, le LCIS participe activement à la formation d'ingénieurs, de masters et de doctorants spécialisés dans ces domaines stratégiques.

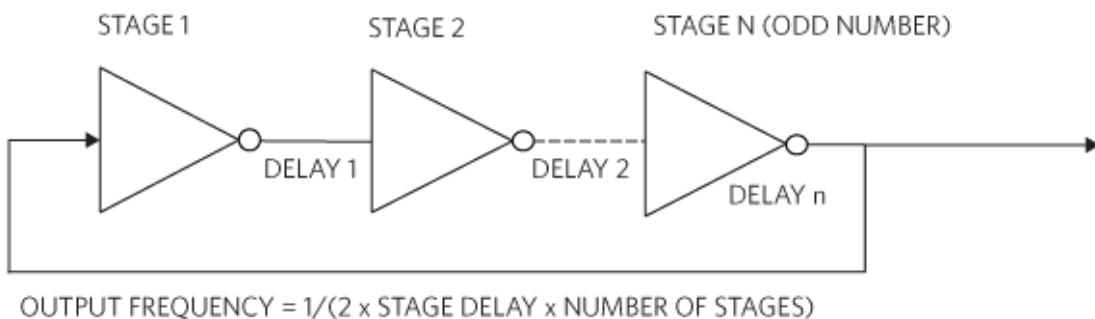
Dans le contexte actuel où la sécurisation des dispositifs électroniques contre les attaques physiques devient cruciale, le LCIS joue un rôle de premier plan dans le développement de primitives de sécurité robustes pour les circuits programmables tels que les FPGAs. C'est dans ce cadre scientifique rigoureux et innovant que s'inscrit notre travail de conception d'un mécanisme de détection robuste destiné à faire face à de nouvelles menaces comme les POTAs (Power-Off Temperature Attacks).

## 4 Travaux précédents

### 4.1 Étude du vieillissement des ROs

Les oscillateurs à anneau (Ring Oscillators ou RO) sont des circuits logiques simples mais essentiels dans de nombreux domaines électroniques. Constitués d'un nombre impair d'inverseurs CMOS connectés en boucle fermée, ils génèrent naturellement un signal oscillant dont la fréquence dépend principalement du délai de propagation individuel de chaque inverseur et du nombre total d'éléments dans la chaîne.

SIMPLIFIED PUF ELEMENT – A RING OSCILLATOR



$$\text{OUTPUT FREQUENCY} = 1/(2 \times \text{STAGE DELAY} \times \text{NUMBER OF STAGES})$$

Fig. 4. – Schéma d'un Ring Oscillator avec N inverseurs.

La période, ainsi que la fréquence d'oscillation peut être exprimée par la relation suivante :

$$\begin{cases} T_{\text{RO}} = 2 \cdot N \cdot t_d \\ F_{\text{RO}} = \frac{1}{2 \cdot N \cdot t_d} \end{cases} \quad (1)$$

où N correspond au nombre d'inverseurs

et  $t_d$  au délai moyen de propagation par inverseur

Ces circuits sont largement utilisés pour mesurer la performance technologique des transistors sur une puce, analyser l'impact du vieillissement, ou encore servir de capteurs passifs sensibles aux variations physiques telles que la température ou la tension d'alimentation.

L'une des propriétés fondamentales des ROs est leur grande sensibilité aux conditions environnementales. En particulier, une élévation de température réduit la mobilité des porteurs dans les transistors MOSFET, ce qui ralentit leur commutation et diminue la fréquence d'oscillation. Des simulations montrent que lorsque la température augmente (par exemple de 27°C à 207°C), les distributions de fréquence des ROs se resserrent autour de la moyenne, entraînant une diminution de la variabilité intrinsèque entre différents oscillateurs.

Pour simuler le chauffage d'un RO, des recherches précédentes réalisées dans le cadre d'un PFE par Maryam ESMAEILIAN [1] utilisaient une chambre climatique (étudeve)

pour induire un vieillissement sur un RO. Le problème avec cette approche réside dans la globalité du vieillissement du FPGA, au lieu du RO uniquement. Cette approche a pour inconvénient de potentiellement biaiser les résultats obtenus, car il serait impossible de distinguer les résultats induits par le vieillissement du RO, des résultats induits par le vieillissement des autres composants du FPGA.

Une étude récente réalisée par Aghiles DOUADI [2] a montré qu'il était possible de simuler une chauffe interne du FPGA en utilisant des SIRO (Single Input Ring Oscillator) en grande quantité. Les températures relevées pouvaient ainsi aller bien au-delà des 100°C, ce qui dépasse la température induite possible par la chambre climatique (~80°C). De plus, le chauffage étant induit de manière interne, les effets de vieillissement induits sur les autres composants que le RO sont minisés. Cette étude se focalisait cependant sur les effets de ces SIRO sur des RO-PUFs (Ring Oscillator - Physically Unclonnable Functions), et non pas sur d'autres systèmes basés sur des ROs.

## 4.2 Pré-étude d'un mécanisme de détection

Un travail prospectif récent mené par Yassine HMIMOU [3] (document interne) consistait à réaliser une pré-étude d'un mécanisme de détection (précurseur de l'ATM) des décalages de fonctionnement de fréquence de `ro_out`. Ce travail a ainsi permis de valider en simulation comportementale le fonctionnement du mécanisme de détection, mais pas dans le cadre d'une simulation d'implémentation, ou d'une implémentation en situation réelle.

## 5 Méthodologie

### 5.1 Déroulement du projet

Voici un Gantt résumant le déroulement du projet. Les réunions hebdomadaires n'ont pas été incluses.

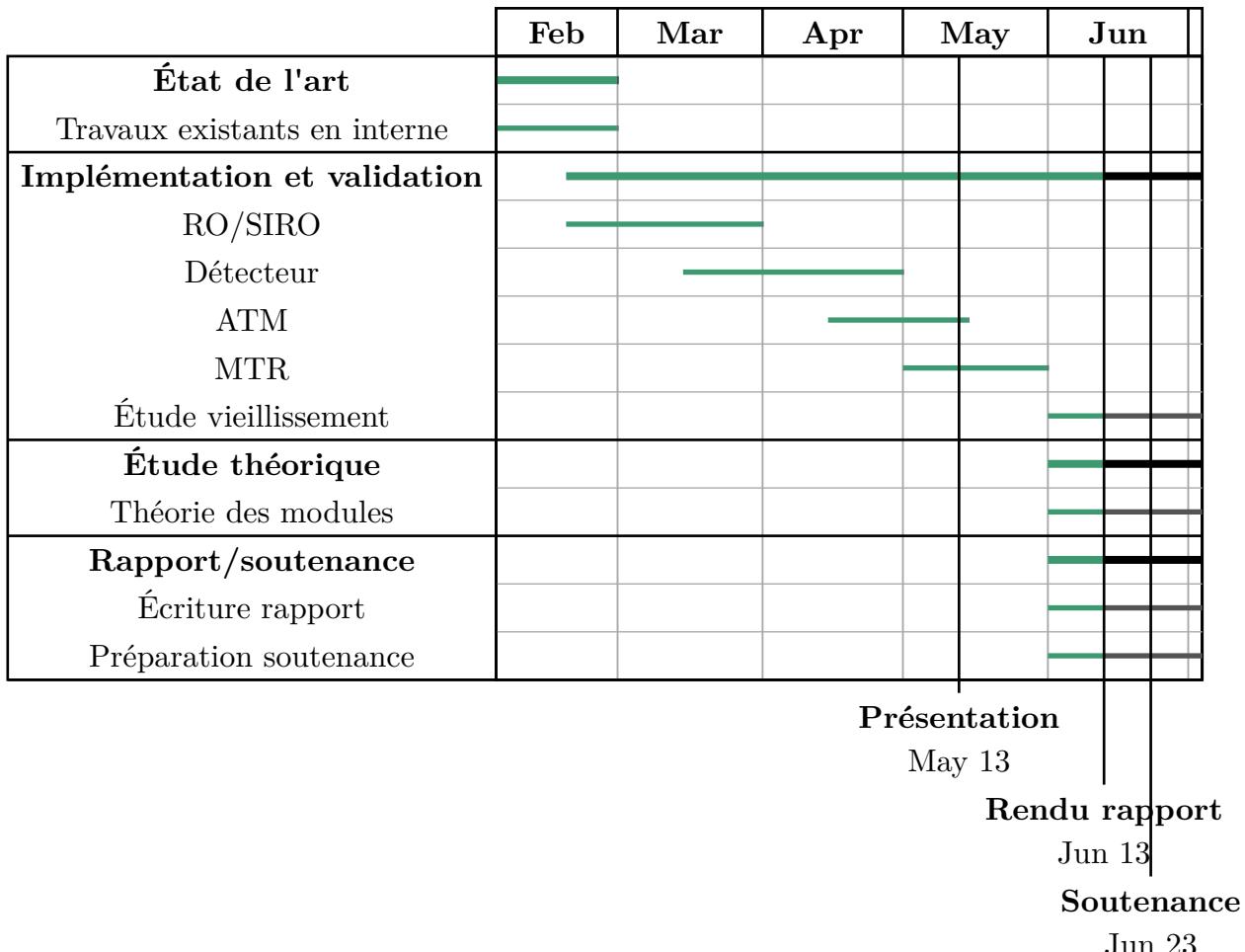


Fig. 5. – Diagramme Gantt du déroulement du projet.

### 5.2 Fonctionnement et limites théoriques des modules

Avant d'implémenter les modules, il est important de bien définir leur fonctionnement et limites théoriques. Pour ce faire, nous réaliserons une analyse des modules et de leur théorie, et réaliserons une carte thermique pour visualiser les résultats de nos simulations, et les confronter à la théorie.

Avant de rentrer dans l'étude de la théorie des modules, détaillons leurs fonctionnements respectifs.

Le module du Détecteur (**voir Annexe 10.A**) a pour but de vérifier et détecter la présence ou non de glitches sur le signal `clk_s`. Pour ce faire, il compte pendant une  $\frac{1}{2}$  période de `clk_s` le nombre de fronts montants du signal `ro_out`. Pendant chaque reste de demi-période où `clk_s` est à l'état bas, le Détecteur compare cette valeur à des intervalles prédéfinies (basés sur les valeurs nominales de fréquences de fonctionnement). Il est ainsi capable de lever une alarme si la valeur sort des intervalles.

Cette alarme restera tant que l'utilisateur n'aura pas initié un reset. Notre signal `ro_out` est ainsi considéré comme étant en « bon fonctionnement » (suffisamment proche du fonctionnement nominal).

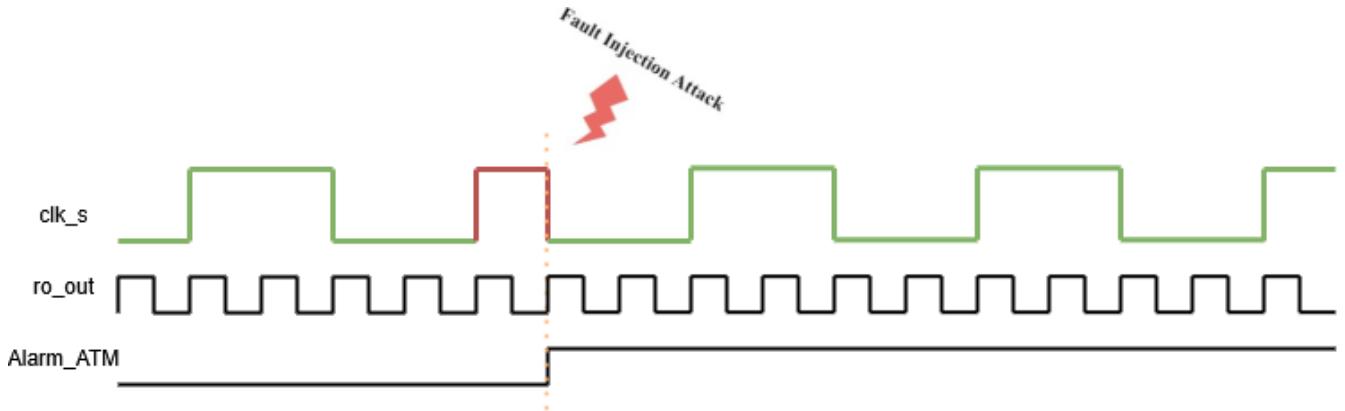


Fig. 6. – Schéma résumant le fonctionnement du DéTECTEUR.

Le module ATM (voir Annexe 10.B) a pour but de vérifier et détecter un éventuel décalage dans la fréquence de `ro_out`. Pour ce faire, il compte pendant  $N$  périodes de `clk_s` le nombre de fronts montants du signal `ro_out`. Puis, pendant une période de `clk_s`, l'ATM compare cette valeur à des intervalles prédéfinies (basés sur les valeurs nominales de fréquences de fonctionnement). Il est ainsi capable de lever une alarme si la valeur sort des intervalles. Cette alarme restera tant que l'utilisateur n'aura pas initié un reset. Notre signal `clk_s` est ainsi considéré comme étant en « bon fonctionnement » (suffisamment proche du fonctionnement nominal).

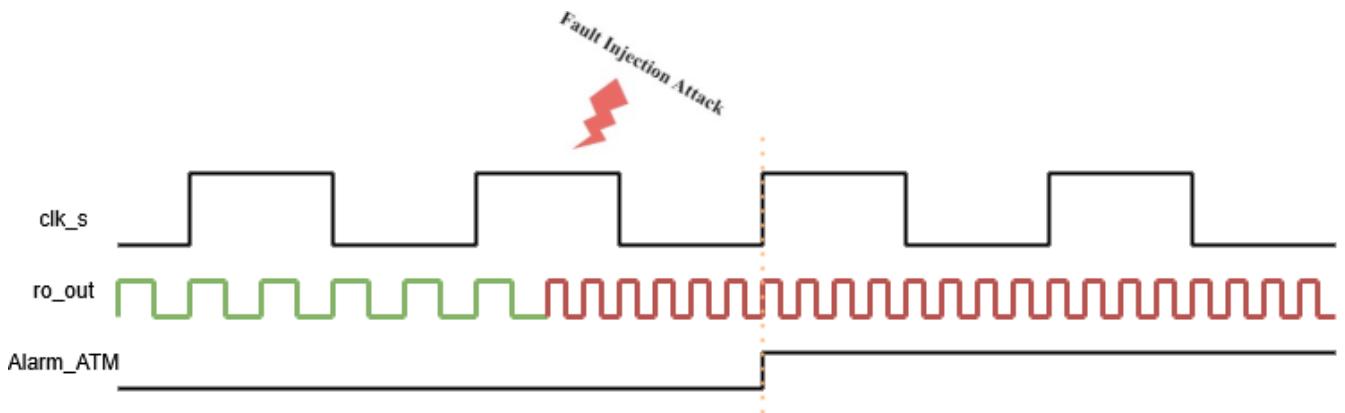


Fig. 7. – Schéma résumant le fonctionnement de l'ATM.

Pour représenter notre carte thermique, nous chercherons à balayer les valeurs de périodes pour `clk_s` et `ro_out` à partir de leurs valeurs nominales, en appliquant un pourcentage négatif ou positif. On nomme ce pourcentage pour chaque module respectif comme tel :  $P_D$  pour DéTECTEUR et  $P_A$  pour l'ATM. Pour se répétrer dans les valeurs balayées, nous utiliserons deux indices,  $i$  pour le DéTECTEUR et `clk_s`,  $j$  pour l'ATM et `ro_out`, ainsi qu'un pas,  $\Delta T$ . Les valeurs nominales sont ainsi représentées avec les indices 0, pour `clk_s` ou `ro_out`. Ainsi, les fréquences de `clk_s` et `ro_out` seront représentées respectivement comme telles :

$$\begin{cases} F_{\text{clk}_s}(i) = \frac{1}{T_{\text{clk}_s}(i)} = \frac{1}{T_{\text{clk}_s}(0) \cdot (1+i \cdot \Delta T)} \\ F_{\text{RO}}(j) = \frac{1}{T_{\text{RO}}(j)} = \frac{1}{T_{\text{RO}}(0) \cdot (1+j \cdot \Delta T)} \end{cases} \quad (2)$$

On définit ainsi le rapport de fréquence :

$$R(i, j) = \frac{F_{\text{RO}}(j)}{F_{\text{clk}_s}(i)} = \frac{T_{\text{clk}_s}(i)}{T_{\text{RO}}(j)} = \frac{T_{\text{clk}_s}(0) \cdot (1 + i \cdot \Delta T)}{T_{\text{RO}}(0) \cdot (1 + j \cdot \Delta T)} = R(0, 0) \cdot \frac{1 + i \cdot \Delta T}{1 + j \cdot \Delta T} \quad (3)$$

Le module du DéTECTeur compte les fronts montants de `ro_out` pendant  $\frac{1}{2}$  période de `clk_s`, tandis que le module de l'ATM les compte sur  $N$  périodes de `clk_s`. Les valeurs de fronts montants de `ro_out` pour DéTECTeur seront nommées  $D(i, j)$ , et  $A(i, j)$  pour l'ATM, définis comme suit :

$$D(i, j) = \frac{\frac{1}{2} \cdot \frac{1}{F_{\text{clk}_s}(i)}}{\frac{1}{F_{\text{RO}}(j)}} = \frac{T_{\text{clk}_s}(i)}{2 \cdot T_{\text{RO}}(j)} = \frac{1}{2} R(i, j) = \frac{1}{2} \cdot R(0, 0) \cdot \frac{1 + i \cdot \Delta T}{1 + j \cdot \Delta T} \quad (4)$$

Les intervalles des valeurs de fronts montants de `ro_out` pour le module DéTECTeur se calculent de la manière suivante :

$$D_{\min / \max} = D(0, 0) \cdot (1 \pm P_D) = \frac{T_{\text{clk}_s}(0)}{2 \cdot T_{\text{RO}}(0)} \cdot (1 \pm P_D) = \frac{1}{2} R(0, 0) \cdot (1 \pm P_D) \quad (5)$$

Ainsi, on définit :

$$\begin{cases} D_{\min} = \frac{1}{2} R(0, 0) \cdot (1 - P_D) \\ D_{\max} = \frac{1}{2} R(0, 0) \cdot (1 + P_D) \end{cases} \quad (6)$$

Le nombre de fronts montants de `ro_out` sur  $N$  périodes de `clk_s` (ATM) est nommé  $A(i, j)$ , et est défini comme suit :

$$A(i, j) = \frac{N \cdot \frac{1}{F_{\text{clk}_s}(i)}}{\frac{1}{F_{\text{RO}}(j)}} = \frac{N \cdot T_{\text{clk}_s}(i)}{T_{\text{RO}}(j)} = N \cdot R(i, j) = N \cdot R(0, 0) \cdot \frac{1 + i \cdot \Delta T}{1 + j \cdot \Delta T} \quad (7)$$

Les intervalles des valeurs de fronts montants de `ro_out` pour le module de l'ATM se calculent de la manière suivante :

$$A_{\min / \max} = A(0, 0) \cdot (1 \pm P_A) = \frac{N \cdot T_{\text{clk}_s}(0)}{T_{\text{RO}}(0)} \cdot (1 \pm P_A) = N \cdot R(0, 0) \cdot (1 \pm P_A) \quad (8)$$

Ainsi, on définit :

$$\begin{cases} A_{\min} = N \cdot R(0, 0) \cdot (1 - P_A) \\ A_{\max} = N \cdot R(0, 0) \cdot (1 + P_A) \end{cases} \quad (9)$$

### 5.3 Formalisation des modules

Détaillons à présent les conditions de levées d'alarmes pour le DéTECTeur et l'ATM :

$$D_{\min} \leq D(i, j) \leq D_{\max}$$

Par identification en utilisant (4) et (6) :

$$\begin{aligned} \frac{1}{2}R(0, 0) \cdot (1 - P_D) &\leq \frac{1}{2}R(0, 0) \cdot \frac{1 + i \cdot \Delta T}{1 + j \cdot \Delta T} \leq \frac{1}{2}R(0, 0) \cdot (1 + P_D) \\ 1 - P_D &\leq \frac{1 + i \cdot \Delta T}{1 + j \cdot \Delta T} \leq 1 + P_D \end{aligned} \tag{10}$$

$$\begin{aligned} (1 + j \cdot \Delta T) \cdot (1 - P_D) &\leq 1 + i \cdot \Delta T \leq (1 + j \cdot \Delta T) \cdot (1 + P_D) \\ 1 - P_D + j \cdot \Delta T - P_D \cdot j \cdot \Delta T &\leq 1 + i \cdot \Delta T \leq 1 + P_D + j \cdot \Delta T + P_D \cdot j \cdot \Delta T \end{aligned}$$

On obtient donc comme conditions de levée d'alarme pour le DéTECTeur :

$$(1 - P_D) \cdot j - \frac{P_D}{\Delta T} \leq i \leq (1 + P_D) \cdot j + \frac{P_D}{\Delta T} \tag{11}$$

$$A_{\min} \leq A(i, j) \leq A_{\max}$$

Par identification en utilisant (7) et (9) :

$$\begin{aligned} N \cdot R(0, 0) \cdot (1 - P_A) &\leq N \cdot R(0, 0) \cdot \frac{1 + i \cdot \Delta T}{1 + j \cdot \Delta T} \leq N \cdot R(0, 0) \cdot (1 + P_A) \\ 1 - P_A &\leq \frac{1 + i \cdot \Delta T}{1 + j \cdot \Delta T} \leq 1 + P_A \end{aligned} \tag{12}$$

$$\begin{aligned} (1 + j \cdot \Delta T) \cdot (1 - P_A) &\leq 1 + i \cdot \Delta T \leq (1 + j \cdot \Delta T) \cdot (1 + P_A) \\ 1 - P_A + j \cdot \Delta T - P_A \cdot j \cdot \Delta T &\leq 1 + i \cdot \Delta T \leq 1 + P_A + j \cdot \Delta T + P_A \cdot j \cdot \Delta T \end{aligned}$$

On obtient donc comme conditions de levée d'alarme pour l'ATM :

$$(1 - P_A) \cdot j - \frac{P_A}{\Delta T} \leq i \leq (1 + P_A) \cdot j + \frac{P_A}{\Delta T} \tag{13}$$

On obtient ainsi deux conditions sur des fonctions  $i(j)$ . On peut ainsi définir les équivalences suivantes :

$$\begin{cases} D_{\min} \leq D(i, j) \leq D_{\max} \Leftrightarrow i_D(j) \in \left[ (1 - P_D) \cdot j - \frac{P_D}{\Delta T}, (1 + P_D) \cdot j + \frac{P_D}{\Delta T} \right] \\ A_{\min} \leq A(i, j) \leq A_{\max} \Leftrightarrow i_A(j) \in \left[ (1 - P_A) \cdot j - \frac{P_A}{\Delta T}, (1 + P_A) \cdot j + \frac{P_A}{\Delta T} \right] \end{cases} \tag{14}$$

On rappellera ici que le module de l'ATM doit toujours être plus restrictif que celui du DéTECTeur. Ainsi, on aura toujours  $P_D > P_A$ . Ainsi, les zones où uniquement l'alarme du DéTECTeur se lèvent n'apparaîtront pas, car l'alarme de l'ATM sera toujours levée en premier.

On peut ainsi définir les différentes possibilités couvertes par nos alarmes :

- Pas d'alarmes

$$\begin{cases} D_{\min} \leq D(i, j) \leq D_{\max} \Leftrightarrow i_D(j) \in \left[ (1 - P_D) \cdot j - \frac{P_D}{\Delta T}, (1 + P_D) \cdot j + \frac{P_D}{\Delta T} \right] \\ A_{\min} \leq A(i, j) \leq A_{\max} \Leftrightarrow i_A(j) \in \left[ (1 - P_A) \cdot j - \frac{P_A}{\Delta T}, (1 + P_A) \cdot j + \frac{P_A}{\Delta T} \right] \end{cases} \quad (15)$$

- Alarme de l'ATM

$$\begin{cases} D_{\min} \leq D(i, j) \leq D_{\max} \Leftrightarrow i_D(j) \in \left[ (1 - P_D) \cdot j - \frac{P_D}{\Delta T}, (1 + P_D) \cdot j + \frac{P_D}{\Delta T} \right] \\ A_{\min} \notin A(i, j) \notin A_{\max} \Leftrightarrow i_A(j) \notin \left[ (1 - P_A) \cdot j - \frac{P_A}{\Delta T}, (1 + P_A) \cdot j + \frac{P_A}{\Delta T} \right] \end{cases} \quad (16)$$

- Alarme du DéTECTEUR
  - Impossible en connaissance du point évoqué précédemment.
- Deux alarmes.

$$\begin{cases} D_{\min} \notin D(i, j) \notin D_{\max} \Leftrightarrow i_D(j) \notin \left[ (1 - P_D) \cdot j - \frac{P_D}{\Delta T}, (1 + P_D) \cdot j + \frac{P_D}{\Delta T} \right] \\ A_{\min} \notin A(i, j) \notin A_{\max} \Leftrightarrow i_A(j) \notin \left[ (1 - P_A) \cdot j - \frac{P_A}{\Delta T}, (1 + P_A) \cdot j + \frac{P_A}{\Delta T} \right] \end{cases} \quad (17)$$

Le schéma suivant représente ainsi les divers cas d'alarmes possibles :

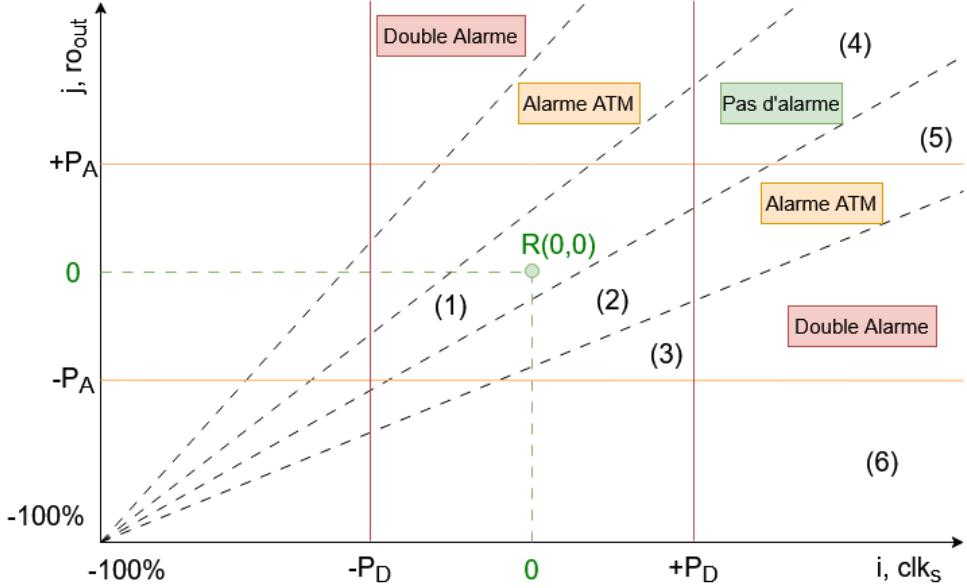


Fig. 8. – Schéma représentant les différentes combinaisons d'alarmes selon la théorie : (1) Pas d'alarme, (2) Alarme ATM, (3) Double Alarme, (4) Pas d'alarme, (5) Alarme ATM, (6) Double Alarme

Nous pouvons observer sur ce schéma basé sur les résultats de la théorie que les zones où les alarmes ne sont pas levées, et les zones où l'alarme ATM est levée s'étendent comme des faisceaux lumineux, dans une situation de quasi-proportionnalité. Ainsi, comme nous l'avions posé en hypothèse plus tôt, si l'attaquant voulait théoriquement mener une attaque simultanée ou par itération sur `clk_s` et `ro_out`, il pourrait ainsi éviter la détection en restant dans la zone centrale sans alarme, que ce soit pour le DéTECTEUR ou l'ATM. Cependant, il est important de rappeler également que cela serait difficile à

mettre en place, en raison du manque précis de contrôle sur le RO, étant un élément interne, ainsi que par le nombre limité d'essais avant de se faire repérer (protections systèmes), limitant grandement ses tentatives d'attaque par itération (voir définition du modèle d'attaquant section 3.5).

Nous pouvons ainsi répondre par la théorie à la question de l'existence ou non d'un intervalle { `clk_s` , `ro_out` } où le système ne serait pas protégé, en indiquant que cela serait possible en théorie, mais trop difficile à réaliser en pratique.

## 6 Protocole expérimental

Les tests visant à valider le fonctionnement de chaque module ont été réalisés sur les modèles de FPGA suivants :

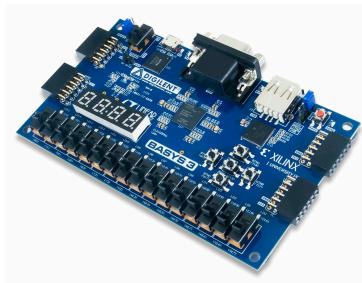


Fig. 9. – FPGA Basys 3.

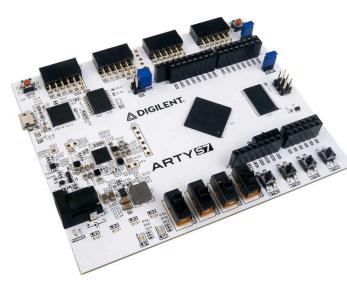


Fig. 10. – FPGA Spartan 7.

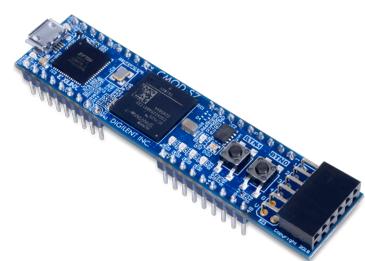


Fig. 11. – FPGA CMODS7.

Les spécifications de ces FPGA sont comparées dans ce tableau :

CARACTÉRISTIQUES	BASYS-3 (XC7A35T)	SPARTAN-7 (XC7S25)	CMOD-S7 (XC7S25)
Suffixe FPGA	1CPG236C	CSGA324	1CSGA225C
Architecture	Artix-7	Spartan-7	Spartan-7
Slices logiques	5200	3650	3650
BRAM	1.8 Mb	1.62 Mb	1.62 Mb
$F_{clk_{MAX}}$	450 MHz	450 MHz	450 MHz
Boutons	5	4	2
Interrupteurs	16	4	0
LEDs	16	4	4
PMODs	4	4	1
Total de pins	32	32	40
Connectivité USB	JTAG + UART	JTAG + UART	JTAG + UART
Intégration plaque à trous	✗	✗	✓
Intégration SIRO	✗	✓	✓
XADC	✓	✓	✓
Prix	~150 € <sup>1</sup>	~70 €	~45 €

Tableau 1. – Spécifications des FPGA utilisés dans ce projet

<sup>1</sup>Étant donné que ce modèle de FPGA était déjà disponible en grande quantité au LCIS, ce prix n'est qu'indicatif dans le cadre de ce projet.

Le Basys3 sera utilisé au départ pour les premières implémentations, en raison de son nombre abondant de LEDs et de switchs, permettant de vérifier plus facilement le bon fonctionnement des modules. Le Spartan7 sera utilisé pour passer à des implémentations avec des SIRO, permettant ainsi de simuler une chauffe sur le FPGA, tout en gardant plus ou moins les mêmes performances que le Basys3 précédemment. Enfin, le CMODS7 sera utilisé pour valider les implémentations précédentes en utilisant moins de ressources (et à un prix plus faible), tout en permettant une implémentation plus facile sur plaque à trous.

## 6.1 Module Détecteur et protocoles de validation

Nous détaillerons dans ce chapitre l'implémentation et les protocoles suivis pour la validation. Les résultats seront décrits dans le chapitre suivant celui-ci.

### 6.1.1 Fonctionnement et implémentation du détecteur

Voici le schéma d'implémentation du module du Détecteur :

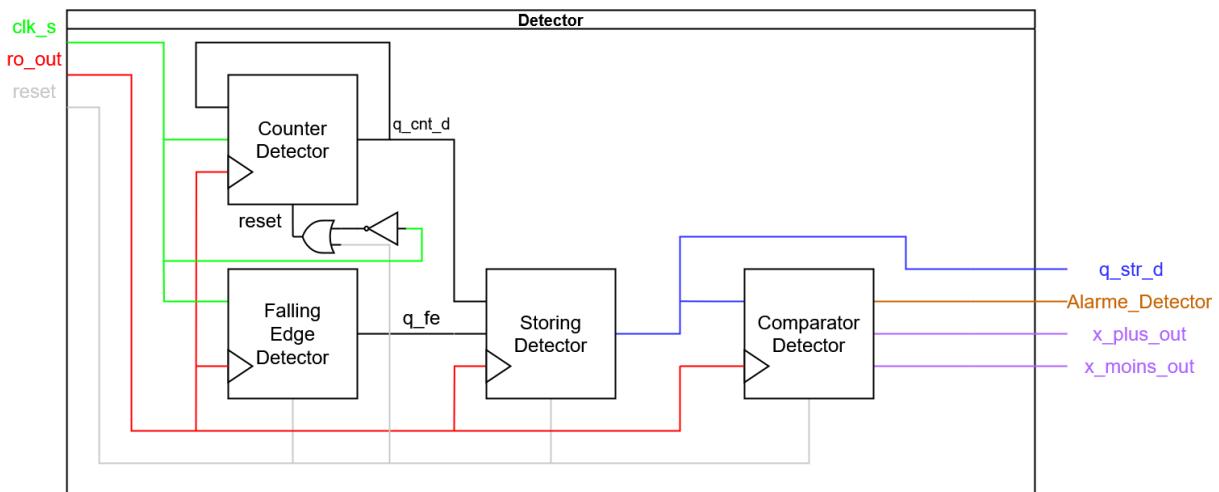


Fig. 12. – Schéma d'implémentation du Détecteur.

Cette implémentation a été réalisée en VHDL, validé par simulation, puis par implémentation. Ce module est constitué de quatres sous-modules :

Sous-modules	Fonctions
<b>Counter_Detector</b>	Compte les fronts montants de <code>ro_out</code> .
<b>Falling_Edge_Detector</b>	Déetecte les fronts descendants de <code>clk_s</code> (arrêt compteur).
<b>Storing_Detector</b>	Stocke la valeur finale de <code>Counter_Detector</code> .
<b>Comparator_Detector</b>	Compare la valeur finale de <code>Counter_Detector</code> avec les intervalles prédéfinies.

Tableau 2. – Tableau recensant les fonctions des sous-modules du module du Détecteur.

### 6.1.2 Protocole de validation en simulation du module Détecteur

Pour valider en simulation le fonctionnement du module Détecteur, le test suivant a été réalisé :

1. Garder le RO interne pendant 2 ms.
  - RO simulé après cela.
2. Poser une période initiale de `clk_s` de 10 us pendant 2 ms.
3. Changer la période de `clk_s` à 5 us
  - L'alarme du détecteur devrait se lever.
4. Remettre la période initiale sans oublier de reset.

Les intervalles ont ici été définies comme allant de 18 à 22 ( $\pm 10\%$ ).

### **6.1.3 Protocole de validation sur FPGA du module DéTECTEUR**

Pour valider sur FPGA le fonctionnement du module DéTECTEUR, le test suivant a été réalisé :

- $F_{clk\_s} = 300$  kHz et  $F_{ro\_out} = 12$  MHz.
    - Intervalles DéTECTEUR = {18, 22} avec précision de 10%
1. Installer des sondes d'oscilloscope pour pouvoir observer `clk_s` et surtout `ro_out`.
  2. Programmer le FPGA.
  3. Rester appuyé sur le bouton `enable_ro_and_siro` pour tester le RO interne.
    - Aucune alarme ne doit être levée, sous condition d'avoir bien reset pendant l'appui du bouton `enable_ro_and_siro`.
  4. Relâcher le bouton `enable_ro_and_siro` et reset.
  5. Se positionner aux fréquences nominales ( $F_{clk\_s}$  et  $F_{ro\_out}$ ) à l'aide d'un GBF.
  6. Tester le détecteur en descendant `clk_s` à moins de 270 kHz.
    - Reset à chaque modification de la fréquence pour s'assurer de la persistance de l'alarme.
  7. Revenir à la fréquence nominale de `clk_s` et reset.
  8. Tester le détecteur en montant `clk_s` à plus de 330 kHz.
    - Reset à chaque modification de la fréquence pour s'assurer de la persistance de l'alarme.

## **6.2 Module ATM et protocoles de validation**

Nous détaillerons dans ce chapitre l'implémentation et les protocoles suivis pour la validation. Les résultats seront décrits dans le chapitre suivant celui-ci.

### **6.2.1 Fonctionnement et implémentation de l'ATM**

Voici le schéma d'implémentation du module de l'ATM :

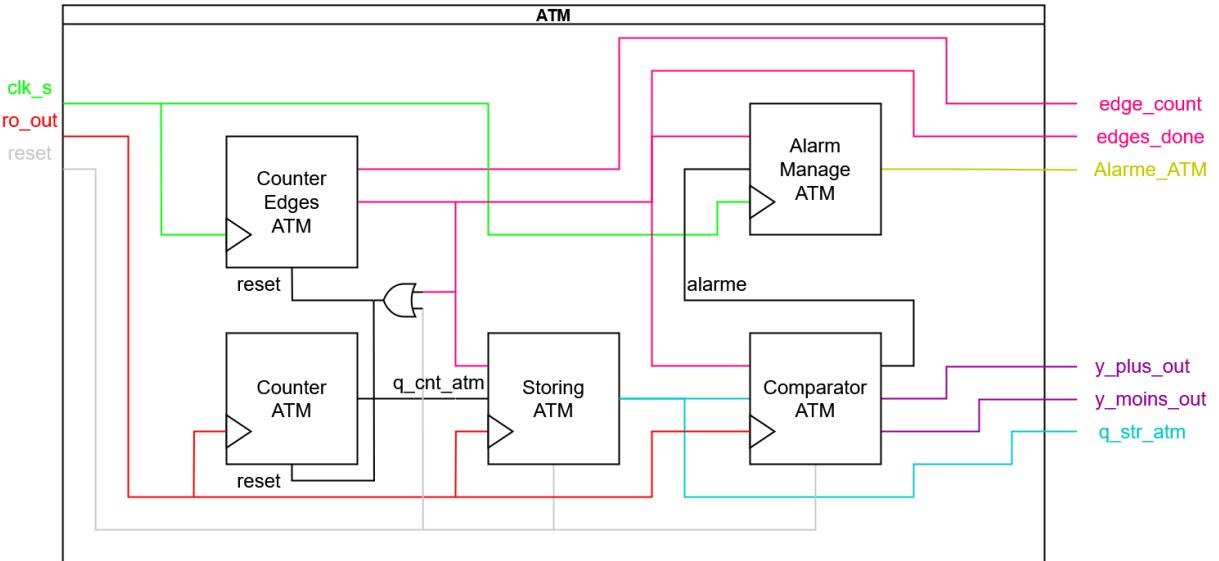


Fig. 13. – Schéma d'implémentation de l'ATM.

Cette implémentation a été réalisée en VHDL, validé par simulation, puis par implémentation. Ce module est constitué de cinq sous-modules

SOUS-MODULES	FONCTIONS
<b>Counter_Edges_ATM</b>	Compte les périodes de <code>clk_s</code> (arrêt compteur).
<b>Counter_ATM</b>	Compte les fronts montants de <code>ro_out</code> .
<b>Storing_ATM</b>	Stocke la valeur finale de <code>Counter_ATM</code> .
<b>Comparotor_ATM</b>	Compare la valeur finale de <code>Counter_ATM</code> avec les intervalles prédéfinies.
<b>Alarm_Manage_ATM</b>	Affiche l'état de l'alarme obtenu.

Tableau 3. – Tableau recensant les fonctions des sous-modules du module ATM.

### 6.2.2 Protocole de validation en simulation du module ATM

Pour valider en simulation le fonctionnement du module ATM, le test suivant a été réalisé :

1. Garder le RO interne pendant 2 ms.
  - RO simulé après cela.
2. Poser une période initiale de `ro_out` de 250 ns pendant 2 ms.
3. Changer la période de `ro_out` à 125 ns.
  - L'alarme du détecteur devrait se lever.
4. Remettre la période initiale sans oublier de reset.

Les intervalles ont ici été définies comme allant de 190 à 210 ( $\pm 5\%$ ).

### 6.2.3 Protocole de validation sur FPGA du module ATM

Pour valider sur FPGA le fonctionnement du module ATM, le test suivant a été réalisé :

- `F_clk_s` = 300 kHz et `F_ro_out` = 12 MHz.

- Intervalles ATM = {190, 210} avec précision de 5%
1. Installer des sondes d'oscilloscope pour pouvoir observer `clk_s` et surtout `ro_out`.
  2. Programmer le FPGA.
  3. Rester appuyé sur le bouton `enable_ro_and_sirop` pour tester le RO interne.
    - Aucune alarme ne doit être levée, sous condition d'avoir bien reset pendant l'appui du bouton `enable_ro_and_sirop`.
  4. Relâcher le bouton `enable_ro_and_sirop` et reset.
  5. Se positionner aux fréquences nominales (`F_clk_s` et `F_ro_out`) à l'aide d'un GBF.
  6. Tester le détecteur en descendant `ro_out` à moins de 11.4 MHz.
    - Reset à chaque modification de la fréquence pour s'assurer de la persistance de l'alarme.
  7. Revenir à la fréquence nominale de `ro_out` et reset.
  8. Tester le détecteur en montant `ro_out` à plus de 12.6 MHz.
    - Reset à chaque modification de la fréquence pour s'assurer de la persistance de l'alarme.

## 6.3 Module RO\_SIRO et protocoles de validation

Nous détaillerons dans ce chapitre l'implémentation et les protocoles suivis pour la validation. Les résultats seront décrits dans le chapitre suivant celui-ci.

### 6.3.1 Fonctionnement et implémentation du module RO\_SIRO

Voici le schéma d'implémentation du module RO\_SIRO :

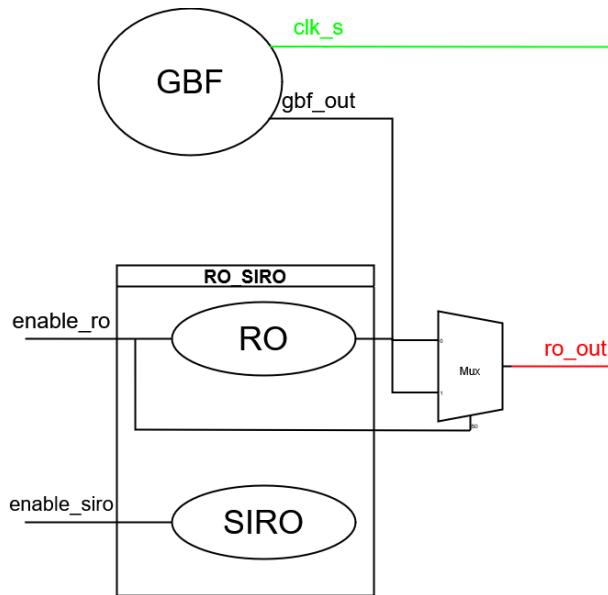


Fig. 14. – Schéma d'implémentation du module RO\_SIRO.

Le module RO\_SIRO (voir Annexe 10.C) est utilisé pour générer le signal `ro_out`. Le GBF, externe au programme du module RO\_SIRO, permet de générer le signal `clk_s`. Il peut également générer si l'utilisateur le désire une oscillation qui sera privilégiée (ou non) au signal de sortie du RO interne. L'avantage de cette approche réside dans la maniabilité du signal généré par un GBF, qui sera beaucoup utilisé dans les protocoles de validation des modules.

### 6.3.2 Protocole de validation en simulation du module RO\_SIRO

Pour valider en simulation le fonctionnement du module RO\_SIRO, le test suivant a été réalisé :

1. Garder `enable_ro_and_siro` à 0
2. Mettre `enable_ro_and_siro` à 1 après 100 ns
  - Des oscillations devraient être visibles pour le signal `ro_out`.

### 6.3.3 Protocole de validation sur FPGA du module RO\_SIRO

Pour valider sur FPGA le fonctionnement du module RO\_SIRO, le test suivant a été réalisé :

1. Installer des sondes d'oscilloscope pour pouvoir observer `clk_s` et surtout `ro_out`.
2. Mettre en place le XADC pour pouvoir observer la température tout le long de l'implémentation.
3. Programmer le FPGA.
4. Observer sur l'oscilloscope la sortie `ro_out`.
  - Aucun signal ne doit sortir à ce moment-là.
5. Rester appuyé sur le bouton `enable_ro_and_siro`.
6. Observer sur l'oscilloscope la sortie `ro_out`.
  - On doit pouvoir observer un signal de sortie, ici une oscillation.
  - A ce moment-là, la température du FPGA devrait s'approcher des 100°C.
7. Rester appuyé sur le reset.
  - La température du FPGA devrait commencer à baisser.

## 6.4 Mécanisme de Test Robuste (MTR) et protocoles de validation

Nous détaillerons dans ce chapitre l'implémentation et les protocoles suivis pour la validation. Les résultats seront décrits dans le chapitre suivant celui-ci.

### 6.4.1 Fonctionnement et implémentation du MTR

Voici le schéma d'implémentation du module MTR :

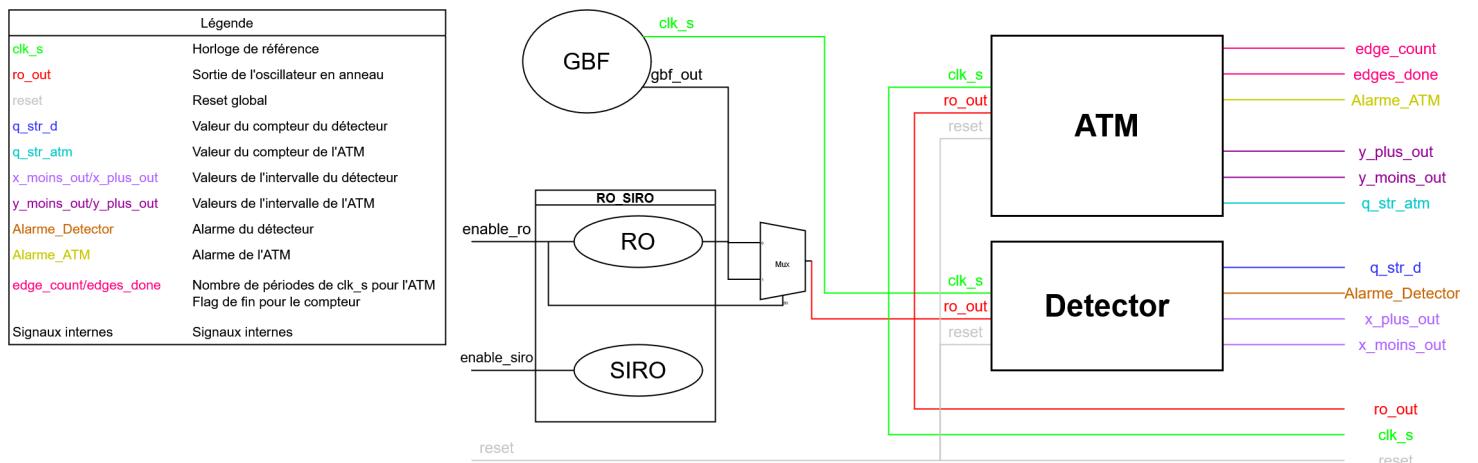


Fig. 15. – Schéma d'implémentation du MTR.

Voir les explications sur les modules RO\_SIRO, Détecteur et ATM précédentes pour le module MTR (**voir Annexe 10.D** pour le code).

#### 6.4.2 Protocole de validation en simulation du MTR

Voir les protocoles de validation en simulation des modules Détecteur et ATM précédents.

Le protocole utilisé pour tester le MTR consiste à balayer tout les cas possibles de changements de périodes pour `clk_s` et `ro_out` (nominal/nominal, anormal/nominal, anormal, anormal/anormal, nominal/anormal). On laissera 2ms entre chaque cas, et on passera d'un cas nominal à anormal de fonctionnement en divisant par deux la période de `clk_s` ou `ro_out` :

CAS	$T_{\text{clk}_s}$	$T_{\text{ro}_{\text{out}}}$
0	$10\mu\text{s}$	250ns
1	$5\mu\text{s}$	250ns
2	$5\mu\text{s}$	125ns
3	$10\mu\text{s}$	125ns

Tableau 4. – Tableau recensant les cas et valeurs des périodes pour le protocole de validation en simulation du MTR.

#### 6.4.3 Protocole de validation sur FPGA du module MTR

Voir les protocoles de validation sur FPGA des modules RO\_SIRO, Détecteur et ATM précédents.

### 6.5 Carte thermique des états d'alarmes

Après la validation du bon fonctionnement du MTR, il restera à vérifier que l'implémentation est bien conforme à la théorie détaillée précédemment. Pour ce faire, une simulation par itération des valeurs de fréquences sera réalisée (**voir Annexe 10.E**). Celle-ci implique de parcourir les valeurs nominales de fréquences de fonctionnement de `clk_s` et de `ro_out` avec un certain pourcentage prédéfini appliqué à chaque fois (ici  $\pm 90\%$  pour les minimums/maximums). Puis, une fois cette simulation réalisée, les résultats stockées dans un CSV seront traités par un programme en Python (**voir Annexe 10.F**) pour les afficher sous forme d'une carte thermique. Les résultats seront décrits dans le chapitre suivant celui-ci.

## 7 Résultats et discussion

### 7.1 Module DéTECTEUR

#### 7.1.1 Validation en simulation du module DéTECTEUR

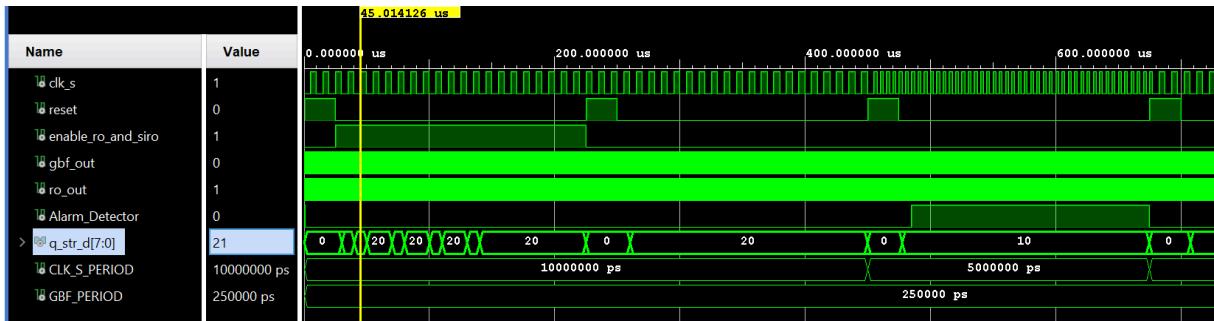


Fig. 16. – Chronogramme du test réalisé sur le module DéTECTEUR.

On peut observer sur le chronogramme que l'alarme du détecteur se lève bien au moment attendu ( $10 < 18$  pour la valeur du compteur), et se réinitialise lorsque le reset passe à 1. Le module DéTECTEUR est fonctionnel en simulation post placement et routage.

#### 7.1.2 Validation sur FPGA du module DéTECTEUR

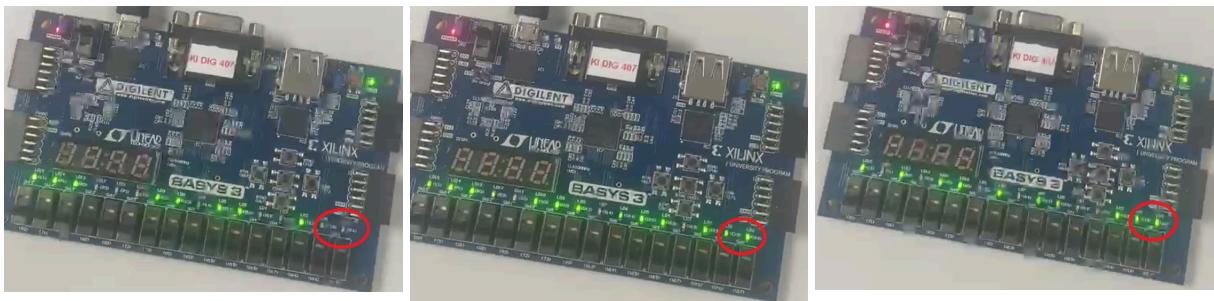


Fig. 17. – FPGA en situation : Initiale [Gauche], d'alarme (inférieur) [Centre], d'alarme (supérieur) [Droite]

On peut observer que les alarmes du détecteur se lèvent bien uniquement lorsqu'on dépasse les intervalles prédéfinis en descendant/montant la fréquence de `clk_s`. Le module DéTECTEUR fonctionne bien comme attendu.

## 7.2 Module ATM

### 7.2.1 Validation en simulation de l'ATM

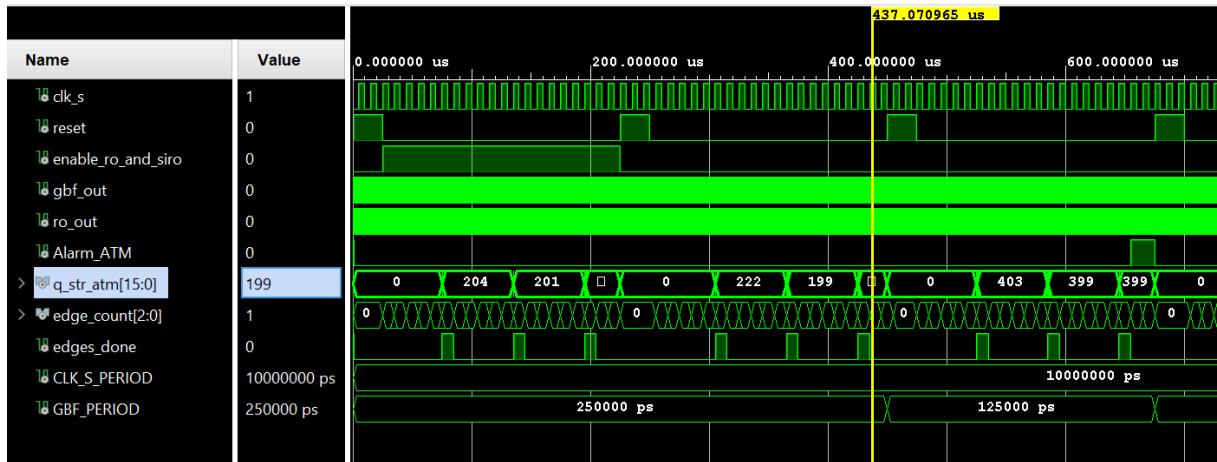


Fig. 21. – Chronogramme d'un test réalisé sur l'ATM.

On peut observer sur le chronogramme que l'alarme de l'ATM se lève bien au moment attendu ( $399 > 210$  pour la valeur du compteur), et se réinitialise lorsque le reset passe à 1. Le module ATM est fonctionnel en simulation post placement et routage.

### 7.2.2 Validation sur FPGA de l'ATM

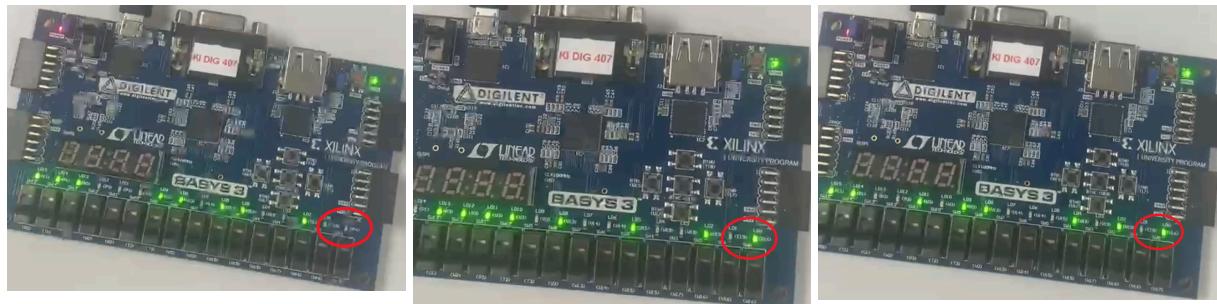


Fig. 22. – FPGA en situation : Initiale [Gauche], d'alarme (inférieur) [Centre], d'alarme (supérieur) [Droite]

On peut observer que les alarmes de l'ATM se lèvent bien uniquement lorsqu'on dépasse les intervalles pré définis en descendant/montant la fréquence de `ro_out`. Le module ATM fonctionne bien comme prévu.

## 7.3 Module RO\_SIRO

### 7.3.1 Validation en simulation du module RO\_SIRO

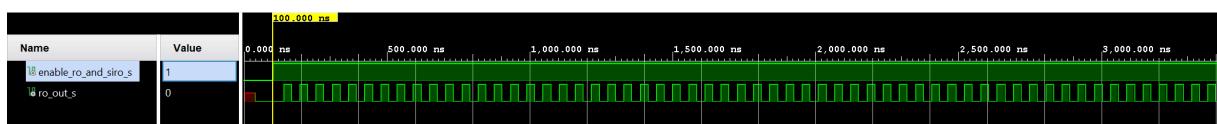


Fig. 26. – Chronogramme du test réalisé sur le module RO\_SIRO.

On peut observer sur le chronogramme qu'après l'activation d'`enable_ro_and_siro`, le signal `ro_out` se met à osciller comme prévu. Le module RO\_SIRO est fonctionnel en simulation post placement et routage.

### 7.3.2 Validation sur FPGA du module RO\_SIRO



Fig. 27. – Signaux sur l’oscilloscope après appui sur le bouton `enable_ro_and_siro`.

On peut observer sur l’oscilloscope qu’après avoir appuyé sur le bouton lié à `enable_ro_and_siro`, le signal `ro_out` se mets bien à osciller comme prévu. La partie génération du RO du module RO\_SIRO fonctionne bien comme prévu.

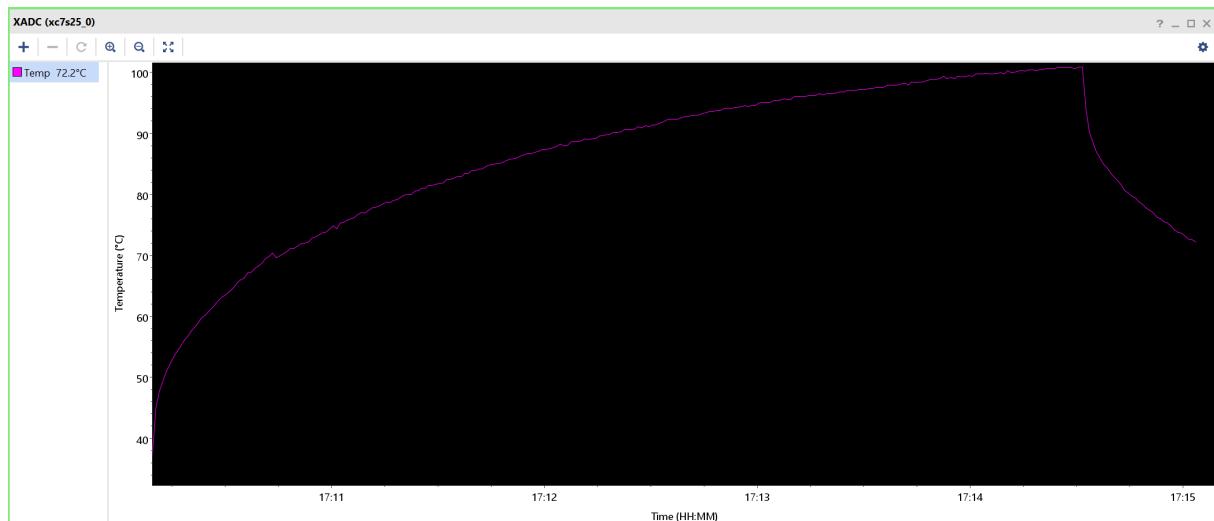


Fig. 28. – Relevé de température du FPGA réalisé par le capteur de température interne.

On peut observer que pendant les quatre premières minutes de l’implémentation, la température du FPGA ne fait qu’augmenter, jusqu’à se stabiliser aux alentours des 100°C. Puis, lorsqu’on reste appuyé sur le reset, la température baisse. La partie génération des SIRO du module RO\_SIRO fonctionne bien comme prévu.

## 7.4 Mécanisme de test robuste (MTR)

### 7.4.1 Validation en simulation du MTR

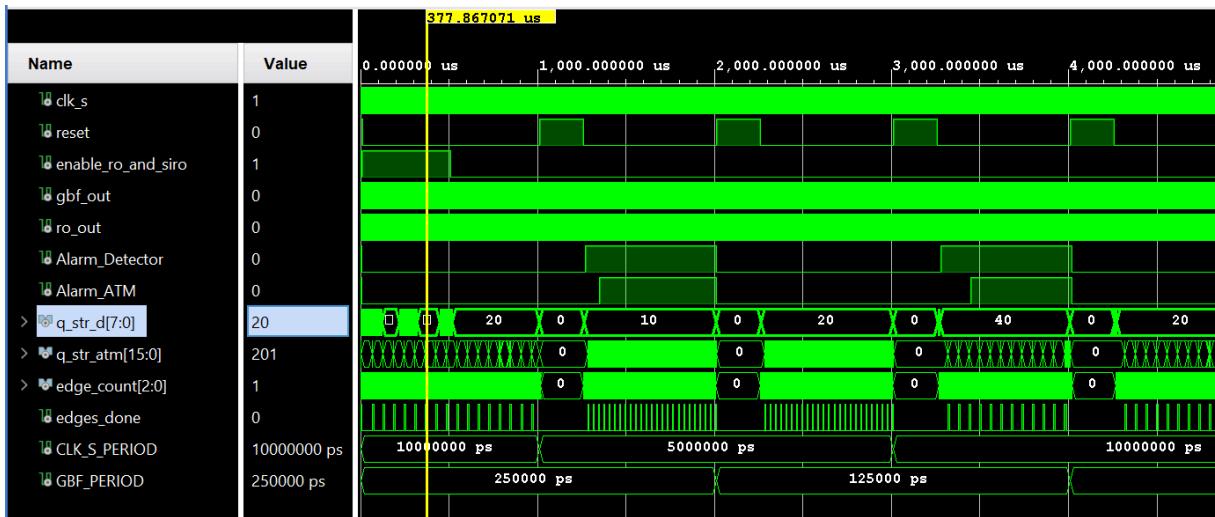


Fig. 29. – Chronogramme d'un test réalisé sur le MTR.

On peut observer sur le chronogramme que les deux alarmes se lèvent bien aux moments attendus (lorsqu'une seule des deux périodes a été modifiée), et se réinitialise lorsque le reset passe à 1. Cela s'explique par les résultats que nous avions obtenus dans la partie théorie, où nous avions conclus que si les fréquences/périodes de `clk_s` et `ro_out` étaient modifiées de manière proportionnelles, alors les alarmes ne seraient pas levées.

### 7.4.2 Validation sur FPGA du MTR

#### Détecteur :

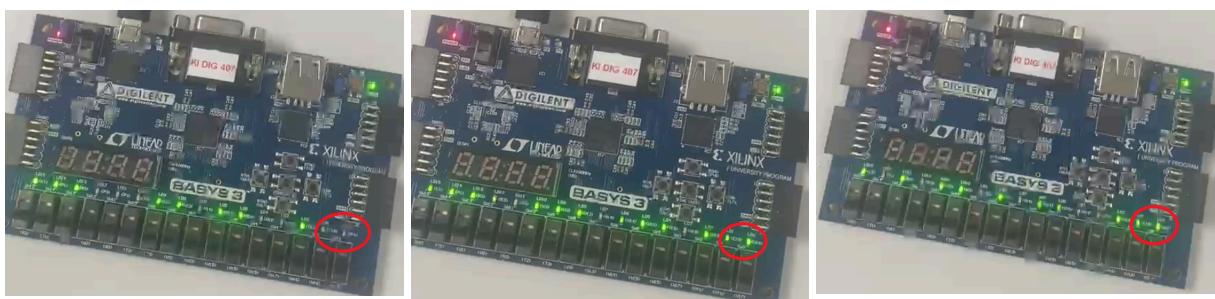


Fig. 30. – FPGA en situation : Initiale [Gauche], d'alarme (inférieur) [Centre], d'alarme (supérieur) [Droite]

#### ATM :

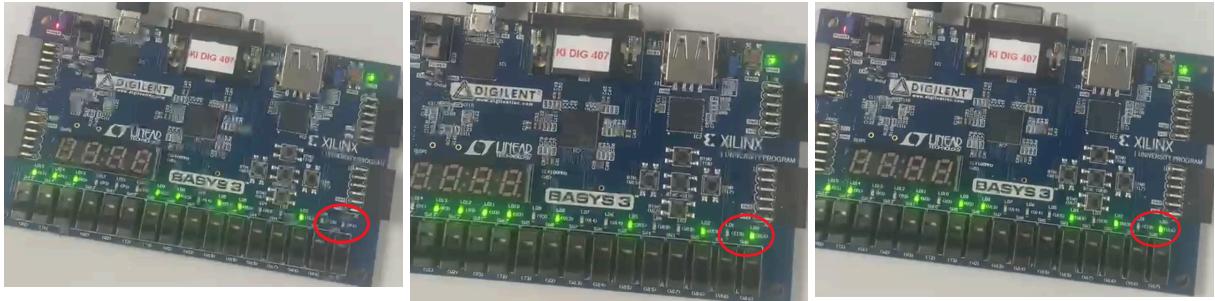


Fig. 34. – FPGA en situation : Initiale [Gauche], d’alarme (inférieur) [Centre], d’alarme (supérieur) [Droite]

On peut observer que les alarmes des modules DéTECTeur et ATM se lèvent bien uniquement lorsqu’on dépasse les intervalles prédéfinis en descendant/montant les fréquences respectives de `clk_s` ou `ro_out`. Le mécanisme de test robuste fonctionne bien comme attendu.

## 7.5 Simulation par itération des valeurs de fréquences



Fig. 38. – Chronogramme de la simulation par itération des valeurs de fréquences.

On peut observer sur le chronogramme qu’au fur et à mesure que les périodes augmentent (on part de  $-90\%$  des périodes de fonctionnement nominales pour `clk_s` et `ro_out`), les temps de levées d’alarme semblent se réduirent (puis revenir au cas de départ pour la simulation). Cela s’explique par le balayement des valeurs que nous avons effectué, et les résultats que nous avions obtenus dans la partie théorie, où nous avions conclus que si les fréquences/périodes de `clk_s` et `ro_out` étaient modifiées de manière proportionnelles, alors les alarmes ne seraient pas levées. En effet, à chaque nouvelle modification de la période de `clk_s`, on parcourt toutes les modifications possibles de la période de `ro_out`. Ainsi, on peut observer une confirmation du schéma théorique réalisé (voir partie 6.3), où l’alarme ATM est toujours levée en premier, et est plus sensible que l’alarme DéTECTeur.

Voici la carte thermique obtenue après le traitement des résultats de notre simulation :

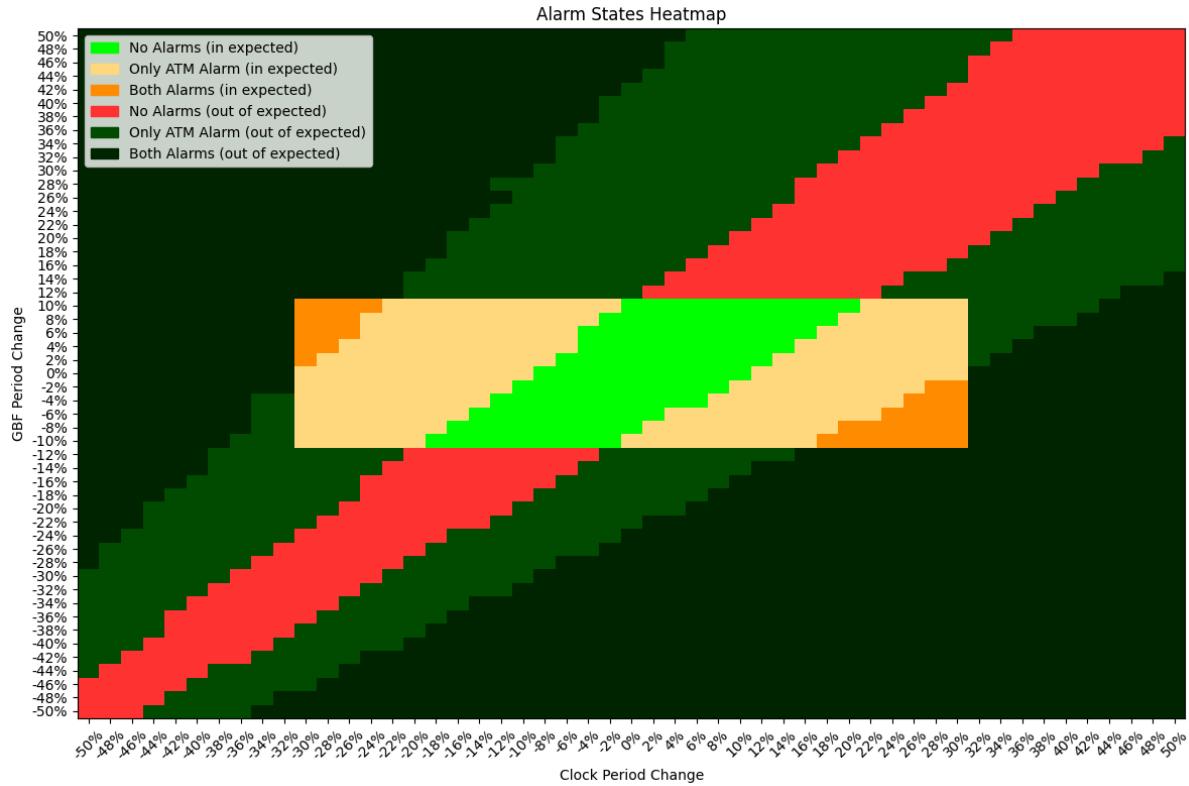


Fig. 39. – Heatmap représentant les résultats de la simulation par itération des valeurs de fréquences.

On peut observer sur cette carte thermique la confirmation de la relation de proportionnalité évoquée entre `clk_s` et `ro_out`, qui nous amène à avoir une partie centrale où aucune alarme ne sera levée, ainsi que les bandes autour de celle-ci, où plus nous nous éloignons, plus les alarmes sont levées (avec priorité à l'ATM de par sa sensibilité accrue, et sa précision plus élevée). Nous avons donc vérifié que l'implémentation donne des résultats conformes à la théorie.

## 7.6 Campagnes d'expérimentation

Dans la continuité du projet, il nous faut maintenant réaliser des campagnes d'expérimentation. Le but de ces dernières est de chercher à mesurer les effets d'un vieillissement induits par SIRO sur notre RO, ainsi que de vérifier que l'ATM n'est pas dépendant du vieillissement (il pourrait sinon être attaqué de la même manière que le DéTECTeur). Voici le protocole expérimental qui sera suivi durant ces campagnes d'expérimentation :

### Protocole :

1. Brancher le Hub USB, les 8 FPGA avec les câbles USB-A Micro-USB, et le PC avec le câble USB-A USB-B.
2. Mettre deux FPGA par plaque à trous.
3. Brancher les sondes d'oscilloscope sur l'oscilloscope et les fils sur les pins relatifs à `clk_s`, de sorte à ce qu'ils soient globaux pour tous les FPGA.

4. Brancher le VDD et le GND de chaque FPGA sur le côté des breadboards, de sorte à pouvoir placer le fil `reset` sur l'un ou l'autre, pour reset ou non chaque FPGA manuellement.
5. Se placer dans les cas de fonctionnement nominaux pour `clk_s` et `ro_out` sur le GBF.
  - Ici,  $F_{\text{clk}_s}(0) = 100 \text{ kHz}$  et  $F_{\text{RO}}(0) = 4 \text{ MHz}$ .
6. Programmer les deux FPGA de la première plaque à trous avec en paramètre 0 SIROs.
  - Ces derniers serviront de référence.
7. Programmer les deux FPGA de la deuxième plaque à trous avec en paramètre 2500 SIROs.
8. Programmer les deux FPGA de la troisième plaque à trous avec en paramètre 5000 SIROs.
9. Programmer les deux FPGA de la quatrième plaque à trous avec en paramètre 7000 SIROs.
10. Débrancher le câble USB-A USB-B.
  - Il ne sera plus nécessaire après que la programmation des FPGA ait été faite.
11. Faire vieillir le RO pendant un jour en continu.
12. Arrêter le vieillissement du RO pendant 1h pour qu'il puisse refroidir en mettant le reset à l'état haut pendant 1h.
13. Relever les fréquences de détection des alarmes pour les scénarios de test d'implémentation pour le MTR avec les mêmes intervalles 5 fois pour chaque borne de chaque détecteur (inférieure, supérieure), et noter les différences en terme de fréquence sur `ro_out`.
14. Répéter le cycle 10. à 12. jusqu'à obtenir la durée désirée de campagne d'expérimentation.

Voici une photo de l'installation de la campagne expérimentale :

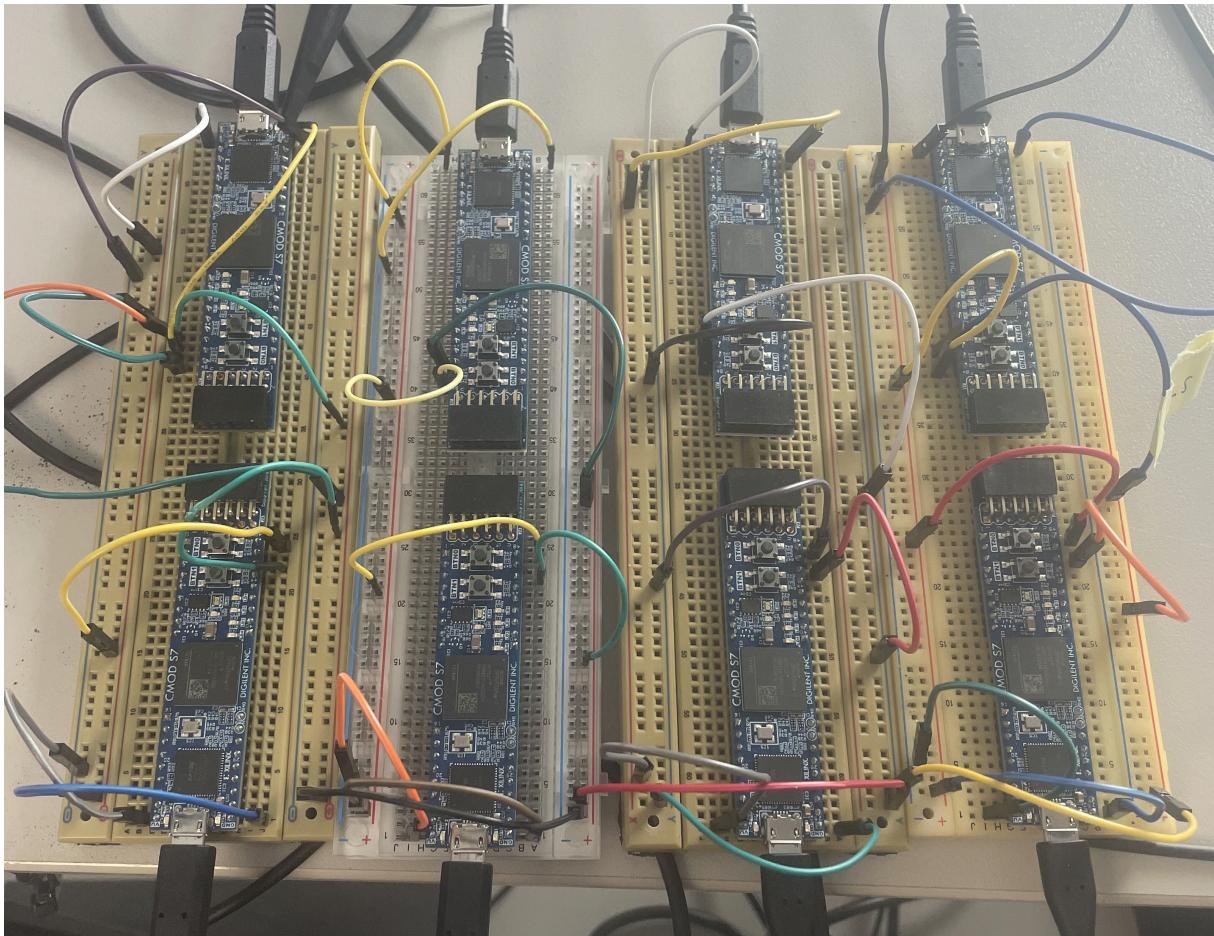


Fig. 40. – Installation expérimentale pour la première campagne d’expérimentation.

Sur chaque partie gauche des breadboards, le VDD et GND de chaque FPGA est branché, et les `reset` sont placés en fonction de la décision de reset ou non. La partie droite de chaque breadboard est utilisée pour propager le signal `clk_s`.

Lors du début du premier test de lancement de la première campagne d’expérimentation, nous avons obtenus les résultats suivants pour les caractérisations :

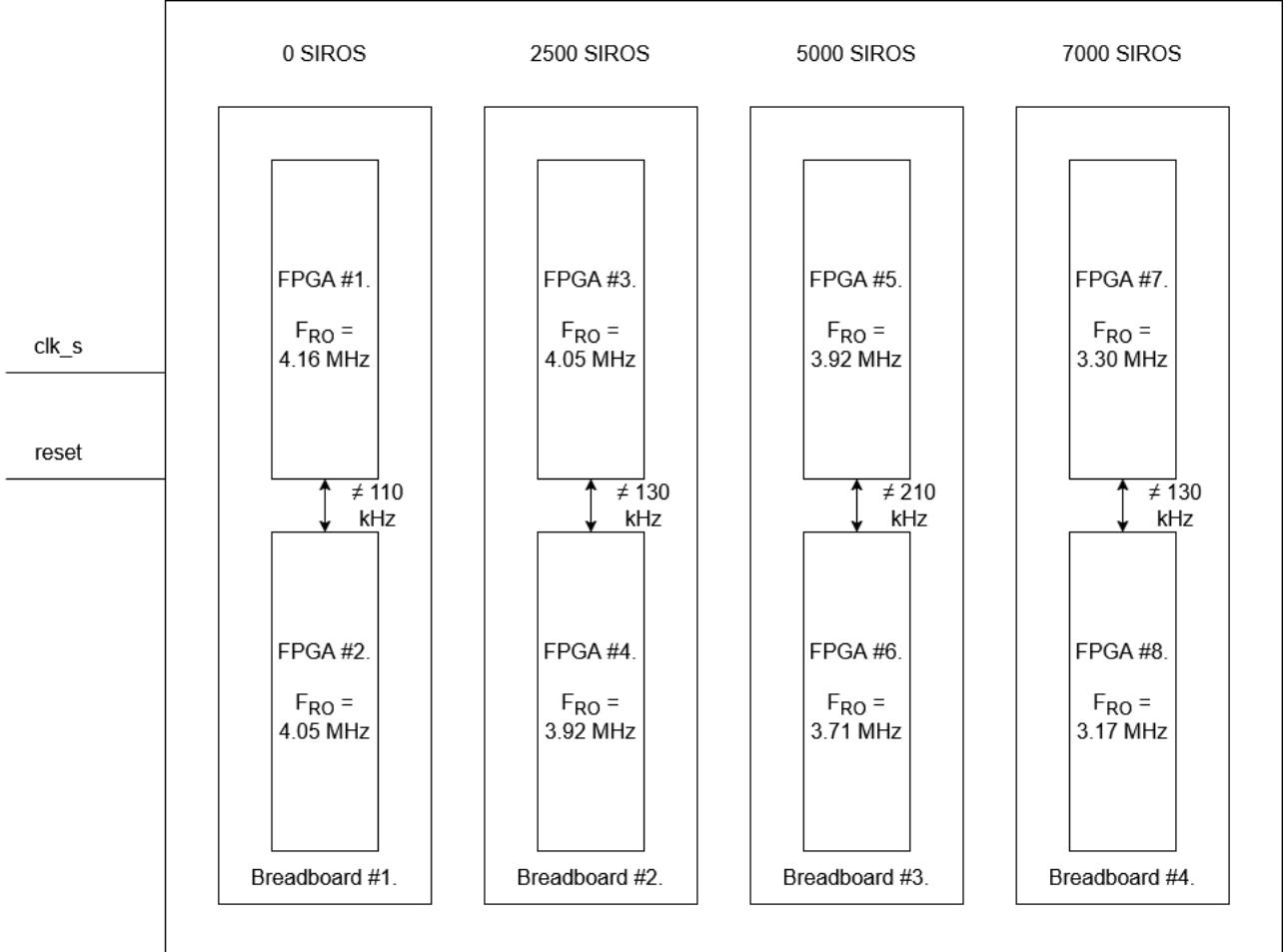


Fig. 41. – Schéma représentation les caractérisations initiales des FPGA utilisés dans la campagne d'expérimentation.

Nous pouvons en tirer deux conclusions :

1. Il y a un effet IRDrop<sup>2</sup> rentrant en jeu lorsqu'on rajoute de plus en plus d'éléments chauffants (SIRO), expliquant le décalage de la fréquence de sortie de l'oscillateur.
2. Il y a un décalage régulier de fréquence de sortie de l'oscillateur entre chaque FPGA sur la même plaque à trous (~150 kHz de différence entre le FPGA en haut et en bas).
  - Afin de vérifier cet effet, un simple test a été réalisé :

<sup>2</sup>Chute de tension se produisant surtout lorsqu'on vient faire chauffer un élément. Cela vient ainsi réduire la tension disponible pour les composants critiques, pouvant provoquer des ralentissements.

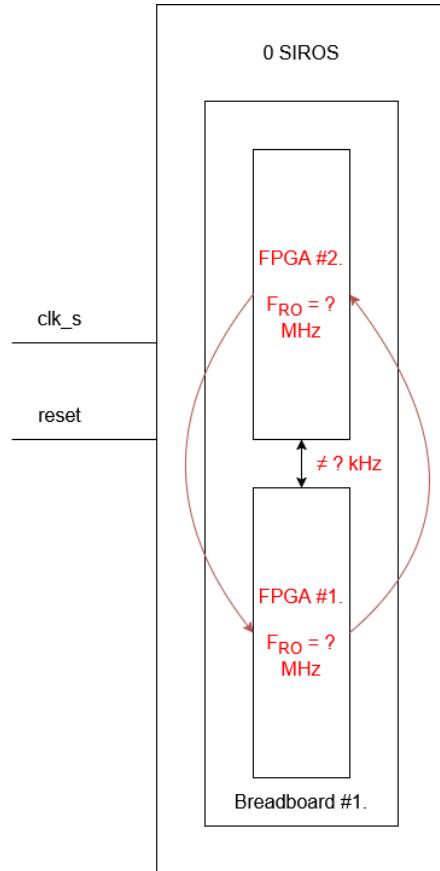


Fig. 42. – Schéma représentant le test utilisé pour déterminer la provenance de l'effet précédent observé.

Celui-ci consiste à simplement inverser les deux FPGA avec 0 SIRO (par exemple), puis à relever à nouveau leurs fréquences de sortie des oscillateurs. Si les fréquences sont les mêmes, alors cela signifie que le positionnement d'un FPGA sur la plaque à trous influence de manière non-négligeable la fréquence de sortie de l'oscillateur.

Après avoir réalisé le test, nous avons pu conclure que les fréquences s'échangeaient après avoir échangé les FPGA. Nous avons ainsi pu en conclure que le positionnement des FPGA sur la breadboard n'influence pas la fréquence de sortie des oscillateurs, mais que les FPGA, leurs états d'usures, et d'autres paramètres difficiles à contrôler, influencent la fréquence de sortie des oscillateurs. Nous avons décidé de faire avec pendant le restant de ce PFE, les différences de fréquences entre les FPGA n'allant en général pas plus loin que 0,25% de différences entre l'un et l'autre, que nous considérons acceptable.

Les résultats des campagnes d'expérimentation ne seront pas détaillés dans ce rapport, car ces campagnes débuteront après l'écriture de ce rapport.

## 8 Budget du projet et analyse RSE

Les coûts du projet se déclinent ainsi :

COÛTS	MONTANTS
<b>FPGA</b>	<ul style="list-style-type: none"> <li>• Basys3 (<math>0\text{€}^3</math>)</li> <li>• Spartan 7 (<math>\sim 70\text{€}/\text{u} \times 2 = 140\text{€}</math>)</li> <li>• CMODS7 (<math>\sim 45\text{€}/\text{u} \times 14 = 630\text{€}</math>)</li> </ul>
<b>Xilinx (Vivado)</b>	License gratuite.
<b>Équipements laboratoire</b>	<ul style="list-style-type: none"> <li>• GBF (<math>\sim 700\text{€}/\text{u} \times 1 = 700\text{€}</math>)</li> <li>• Oscilloscope (<math>\sim 725\text{€}/\text{u} \times 1 = 725\text{€}</math>)</li> </ul>
<b>Gratification</b>	4€35/h soit $\sim 609\text{€}/\text{mois}$ soit $\sim 3045\text{€}$ pour 5 mois
<b>Bureau</b>	10m <sup>2</sup> à $\sim 300\text{€}/\text{an}^4$ + PC ( $\sim 1000\text{€}$ pour 5 ans)
<b>Plateforme expérimentale</b>	PC ( $\sim 1000\text{€}$ pour 5 ans)
<b>Établissement projet M. BEROULLE</b>	Coût réel environné pour l'établissement ( $\sim 100\text{k€}/1600\text{h}$ soit $62.5\text{€}/\text{h}$ ). 40h de réunions dans le projet (2h/semaine) soit $62.5 \times 40 = 2500\text{€}$
<b>Total</b>	$  \begin{aligned}  & 140\text{€} + 630\text{€} + 700\text{€} + 725\text{€} + 3045\text{€} + 300\text{€} + 1000\text{€} \\  & + 1000\text{€} + 2500\text{€} \\  & = \mathbf{\sim 10040\text{€}}  \end{aligned}  $

Tableau 5. – Coûts finaux du projet.

En plus des impacts financiers de ce projet, ce dernier a eu des impacts sur la RSE (Responsabilité Sociétale des Entreprises), notamment de par l'ajout de l'ATM, qui aide à diagnostiquer les problèmes sur le RO, amenant à changer uniquement la puce sans changer la totalité du système. En effet, une détection précoce d'une attaque matérielle telle qu'une POTA sur un FPGA permet d'éviter en général la détérioration de ce dernier. Ces dernières ont de mauvaises conséquences sur l'environnement, rentrant ainsi dans les préoccupations du RSE.

De plus, les systèmes numériques doivent être sécurisés car ils donnent accès à des informations confidentielles ou sécurisées. Ces travaux contribuent à la sécurité des données des utilisateurs et à la confiance dans les systèmes, rentrant également dans les préoccupations du RSE.

---

<sup>3</sup>Étant donné que ce modèle de FPGA était déjà disponible en grande quantité au LCIS, le prix n'est pas compté dans le calcul.

<sup>4</sup>À rapporter au coût d'exploitation du bâtiment auquel je n'ai pas connaissance.

## 9 Conclusion et perspectives

Durant ce projet, un module d'auto-test, et plus généralement d'un mécanisme de test robuste a été développé afin de faire face aux nouveaux vecteurs d'attaques matérielles que sont les POTA, nées de la diversification et multiplication des vecteurs d'attaque sur les composants électroniques. Nous avons également réalisé une analyse théorique des modules et de leurs limites, avec un modèle d'attaquant. Les validations par simulation et par implémentation sur cible, ainsi que les vérifications des limites de l'implémentation ont donné des résultats conformes à cette théorie. Enfin, nous avons également commencé à mener des validations expérimentales de la robustesse de l'ATM face aux attaques power-off (vieillissement).

À court terme, le projet se poursuivra avec des campagnes d'expérimentation plus longues. À moyen terme, on peut considérer que le projet continuera avec des campagnes d'expérimentation avec plus de détecteurs, de fréquences, et de FPGA différent(e)s. Enfin, à long terme, le projet introduira dans les campagnes d'expérimentation d'autres types de détecteurs ou d'attaques.

## 10 Annexes

### 10.A Code Detector

VHDL

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 entity Detector is
6     Port (
7         clk_s          : in STD_LOGIC;
8         reset          : in STD_LOGIC;
9         ro_output      : in STD_LOGIC;
10        Alarm_Detector : out STD_LOGIC;
11        q_str_d       : out STD_LOGIC_VECTOR(7 downto 0)
12    );
13 end Detector;
14
15 architecture Behavioral of Detector is
16     signal q_cnt           : STD_LOGIC_VECTOR(7 downto 0);
17     signal q_fe            : STD_LOGIC;
18     signal q_str_d_internal : STD_LOGIC_VECTOR(7 downto 0);
19
20     component Counter_Detector
21         Port (
22             clk_s      : in STD_LOGIC;
23             reset      : in STD_LOGIC;
24             ro_output  : in STD_LOGIC;
25             q_cnt      : out STD_LOGIC_VECTOR(7 downto 0)
26         );
27     end component;
28
29     component Falling_Edge_Detector
30         Port (
31             clk_s      : in STD_LOGIC;
32             reset      : in STD_LOGIC;
33             ro_output  : in STD_LOGIC;
34             q_fe       : out STD_LOGIC
35         );
36     end component;
37
38     component Storing_Detector
39         Port (
40             ro_output : in STD_LOGIC;
41             reset    : in STD_LOGIC;
42             q_fe     : in STD_LOGIC;
43             q_cnt    : in STD_LOGIC_VECTOR(7 downto 0);
44             q_str_d  : out STD_LOGIC_VECTOR(7 downto 0)
45         );
46     end component;
47
48     component Comparator_Detector
49         Port (
50             clk_s      : in STD_LOGIC;
51             reset    : in STD_LOGIC;
52             q_str_d  : in STD_LOGIC_VECTOR(7 downto 0);
53             Alarm_Detector : out STD_LOGIC
54         );
55
```

```

55     end component;
56
57 begin
58     q_str_d      <= q_str_d_internal;
59
60     Counter_Detector_UUT: Counter_Detector
61         port map (
62             clk_s      => clk_s,
63             reset      => reset,
64             ro_output  => ro_output,
65             q_cnt      => q_cnt
66         );
67
68     Falling_Edge_Detector_UUT: Falling_Edge_Detector
69         port map (
70             clk_s      => clk_s,
71             reset      => reset,
72             ro_output  => ro_output,
73             q_fe       => q_fe
74         );
75
76     Storing_Detector_UUT: Storing_Detector
77         port map (
78             ro_output  => ro_output,
79             reset      => reset,
80             q_fe       => q_fe,
81             q_cnt      => q_cnt,
82             q_str_d    => q_str_d_internal
83         );
84
85     Comparator_Detector_UUT: Comparator_Detector
86         port map (
87             clk_s      => clk_s,
88             reset      => reset,
89             q_str_d    => q_str_d_internal,
90             Alarm_Detector => Alarm_Detector
91         );
92 end Behavioral;

```

## 10.B Code ATM

VHDL

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity ATM is
5     Port (
6         clk_s      : in STD_LOGIC;
7         reset      : in STD_LOGIC;
8         ro_output  : in STD_LOGIC;
9         Alarm_ATM   : out STD_LOGIC;
10        q_str_atm  : out STD_LOGIC_VECTOR(15 downto 0);
11        edge_count  : out STD_LOGIC_VECTOR(2 downto 0);
12        edges_done   : out STD_LOGIC
13    );
14 end ATM;
15
16 architecture Behavioral of ATM is
17     signal q_cnt_atm_internal  : STD_LOGIC_VECTOR(15 downto 0);

```

```

18 signal q_str_atm_internal : STD_LOGIC_VECTOR(15 downto 0);
19 signal edges_done_internal : STD_LOGIC;
20 signal alarm : STD_LOGIC;
21 signal edge_count_internal : STD_LOGIC_VECTOR(2 downto 0);
22
23 -- Component declarations
24 component Counter_ATM
25     Port (
26         ro_output : in STD_LOGIC;
27         reset : in STD_LOGIC;
28         edges_done : in STD_LOGIC;
29         q_cnt_atm : out STD_LOGIC_VECTOR(15 downto 0)
30     );
31 end component;
32
33 component Counter_Edges_ATM
34     Port (
35         clk_s : in STD_LOGIC;
36         reset : in STD_LOGIC;
37         edges_done : out STD_LOGIC;
38         edge_count : out STD_LOGIC_VECTOR(2 downto 0)
39     );
40 end component;
41
42 component Comparator_ATM
43     Port (
44         clk_s : in STD_LOGIC;
45         reset : in STD_LOGIC;
46         q_str_atm : in STD_LOGIC_VECTOR(15 downto 0);
47         edges_done : in STD_LOGIC;
48         alarm : out STD_LOGIC
49     );
50 end component;
51
52 component Alarm_Manage_ATM
53     Port (
54         clk_s : in STD_LOGIC;
55         reset : in STD_LOGIC;
56         edges_done : in STD_LOGIC;
57         alarm : in STD_LOGIC;
58         Alarm_ATM : out STD_LOGIC
59     );
60 end component;
61
62 component Storing_ATM
63     Port (
64         ro_output : in STD_LOGIC;
65         reset : in STD_LOGIC;
66         edges_done : in STD_LOGIC;
67         q_cnt_atm : in STD_LOGIC_VECTOR(15 downto 0);
68         q_str_atm : out STD_LOGIC_VECTOR(15 downto 0)
69     );
70 end component;
71
72 begin
73     q_str_atm <= q_str_atm_internal;
74     edge_count <= edge_count_internal;
75     edges_done <= edges_done_internal;
76

```

```

77 Counter_ATM_UUT: Counter_ATM
78     port map (
79         ro_output  => ro_output,
80         reset      => reset,
81         edges_done => edges_done_internal,
82         q_cnt_atm  => q_cnt_atm_internal
83     );
84
85 Counter_Edges_ATM_UUT: Counter_Edges_ATM
86     port map (
87         clk_s       => clk_s,
88         reset      => reset,
89         edges_done => edges_done_internal,
90         edge_count  => edge_count_internal
91     );
92
93 Storing_ATM_UUT: Storing_ATM
94     port map (
95         ro_output  => ro_output,
96         reset      => reset,
97         edges_done => edges_done_internal,
98         q_cnt_atm  => q_cnt_atm_internal,
99         q_str_atm  => q_str_atm_internal
100    );
101
102 Comparator_ATM_UUT: Comparator_ATM
103     port map (
104         clk_s       => clk_s,
105         reset      => reset,
106         q_str_atm  => q_str_atm_internal,
107         edges_done  => edges_done_internal,
108         alarm       => alarm
109    );
110
111 Alarm_Manage_ATM_UUT: Alarm_Manage_ATM
112     port map (
113         clk_s       => clk_s,
114         reset      => reset,
115         edges_done  => edges_done_internal,
116         alarm       => alarm,
117         Alarm_ATM   => Alarm_ATM
118    );
119 end Behavioral;

```

## 10.C Code RO\_SIRO

VHDL

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity RO_SIRO is
5     Port (
6         enable_ro_and_sirop : in STD_LOGIC;
7         ro_out              : out STD_LOGIC
8     );
9 end RO_SIRO;
10
11 architecture Behavioral of RO_SIRO is
12     signal activate_SIRO : STD_LOGIC;

```

```

13  constant SIRO_number : integer := 1;
14
15  attribute KEEP          : string;
16  attribute DONT_TOUCH    : string;
17  attribute S              : string;
18  attribute KEEP_HIERARCHY: string;
19  attribute MAX_FANOUT    : string;
20
21  attribute KEEP of activate_SIRO      : signal is "true";
22  attribute DONT_TOUCH of activate_SIRO: signal is "true";
23  attribute S of activate_SIRO        : signal is "true";
24  attribute KEEP_HIERARCHY of activate_SIRO: signal is "true";
25  attribute MAX_FANOUT of activate_SIRO: signal is "1";
26
27  component RO
28    Port (
29      | enable_ro_and_siro : in STD_LOGIC;
30      | ro_out      : out STD_LOGIC
31    );
32  end component;
33
34  component SIRO
35    Port (
36      | enable_ro_and_siro : in STD_LOGIC
37    );
38  end component;
39
40 begin
41   activate_SIRO <= enable_ro_and_siro;
42
43   UUT_RO : RO
44     port map (
45       | enable_ro_and_siro => enable_ro_and_siro,
46       | ro_out      => ro_out
47     );
48
49   GENERATE_SIRO: for i in 0 to SIRO_number-1 generate
50   begin
51     UUT_SIRO : SIRO
52       port map (
53         | enable_ro_and_siro => activate_SIRO
54       );
55   end generate GENERATE_SIRO;
56
57 end Behavioral;

```

## 10.D Code MTR

VHDL

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity MTR is
5   Port (
6     | clk_s           : in STD_LOGIC;
7     | reset          : in STD_LOGIC;
8     | enable_ro_and_siro : in STD_LOGIC;
9     | gbf_out        : in STD_LOGIC;
10    | ro_out         : out STD_LOGIC;

```

```

11      Alarm_Detector    : out STD_LOGIC;
12      Alarm_ATM        : out STD_LOGIC;
13      q_str_d           : out STD_LOGIC_VECTOR(7 downto 0);
14      q_str_atm         : out STD_LOGIC_VECTOR(15 downto 0);
15      edge_count         : out STD_LOGIC_VECTOR(2 downto 0);
16      edges_done          : out STD_LOGIC
17  );
18 end MTR;
19
20 architecture Behavioral of MTR is
21   signal ro_internal      : STD_LOGIC;
22   signal ro_output        : STD_LOGIC;
23
24   component ATM
25     Port (
26       clk_s              : in STD_LOGIC;
27       reset              : in STD_LOGIC;
28       ro_output          : in STD_LOGIC;
29       Alarm_ATM         : out STD_LOGIC;
30       q_str_atm         : out STD_LOGIC_VECTOR(15 downto 0);
31       edge_count         : out STD_LOGIC_VECTOR(2 downto 0);
32       edges_done          : out STD_LOGIC
33  );
34 end component;
35
36   component Detector
37     Port (
38       clk_s              : in STD_LOGIC;
39       reset              : in STD_LOGIC;
40       ro_output          : in STD_LOGIC;
41       Alarm_Detector    : out STD_LOGIC;
42       q_str_d            : out STD_LOGIC_VECTOR(7 downto 0)
43  );
44 end component;
45
46   component RO_SIRO
47     Port (
48       enable_ro_and_siro : in STD_LOGIC;
49       ro_out              : out STD_LOGIC
50  );
51 end component;
52
53 begin
54   ro_output <= ro_internal when enable_ro_and_siro = '1' else gbf_out;
55
56   ro_out <= ro_output;
57
58   ATM_UUT: ATM
59     port map (
60       clk_s              => clk_s,
61       reset              => reset,
62       ro_output          => ro_output,
63       Alarm_ATM         => Alarm_ATM,
64       q_str_atm         => q_str_atm,
65       edge_count         => edge_count,
66       edges_done          => edges_done
67  );
68
69   Detector_UUT: Detector

```

```

70     port map (
71         clk_s          => clk_s,
72         reset         => reset,
73         ro_output     => ro_output,
74         Alarm_Detector => Alarm_Detector,
75         q_str_d       => q_str_d
76     );
77
78     RO_SIRO_UUT: RO_SIRO
79     port map (
80         enable_ro_and_siro => enable_ro_and_siro,
81         ro_out           => ro_internal
82     );
83 end Behavioral;

```

## 10.E Testbench Simulation Itérations

VHDL

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4 use STD.TEXTIO.ALL;
5
6 entity MTR_TB is
7 end MTR_TB;
8
9 architecture Behavioral of MTR_TB is
10
11    component MTR
12        Port (
13            clk_s          : in STD_LOGIC;
14            reset         : in STD_LOGIC;
15            enable_ro_and_siro : in STD_LOGIC;
16            gbf_out       : in STD_LOGIC;
17            ro_out        : out STD_LOGIC;
18            Alarm_Detector : out STD_LOGIC;
19            Alarm_ATM    : out STD_LOGIC;
20            q_str_d       : out STD_LOGIC_VECTOR(7 downto 0);
21            q_str_atm    : out STD_LOGIC_VECTOR(15 downto 0);
22            edge_count    : out STD_LOGIC_VECTOR(2 downto 0);
23            edges_done    : out STD_LOGIC
24        );
25    end component;
26
27    signal clk_s          : STD_LOGIC := '0';
28    signal reset         : STD_LOGIC := '0';
29    signal enable_ro_and_siro : STD_LOGIC := '0';
30    signal gbf_out       : STD_LOGIC := '0';
31
32    signal ro_out        : STD_LOGIC;
33    signal Alarm_Detector : STD_LOGIC;
34    signal Alarm_ATM    : STD_LOGIC;
35    signal q_str_d       : STD_LOGIC_VECTOR(7 downto 0);
36    signal q_str_atm    : STD_LOGIC_VECTOR(15 downto 0);
37    signal edge_count    : STD_LOGIC_VECTOR(2 downto 0);
38    signal edges_done    : STD_LOGIC;
39
40    signal CLK_S_PERIOD   : TIME := 10 us;
41    signal GBF_PERIOD     : TIME := 250 ns;

```

```

42
43 constant CLK_S_BASE      : TIME := 10 us;
44 constant GBF_BASE        : TIME := 250 ns;
45 constant STEP_PERCENT    : real := 1.0;
46 constant STABLE_TIME     : TIME := 500 us;
47 constant NUM_STEPS       : integer := 50;
48
49 file results_file : TEXT;
50 constant results_filename : string := "...\\...\\...\\...\\...\\...\\
51 \Simulation_Results\\simulation_results.csv";
52 begin
53
54   UUT: MTR
55     port map (
56       clk_s          => clk_s,
57       reset          => reset,
58       enable_ro_and_siro => enable_ro_and_siro,
59       gbf_out        => gbf_out,
60       ro_out         => ro_out,
61       Alarm_Detector => Alarm_Detector,
62       Alarm_ATM     => Alarm_ATM,
63       q_str_d        => q_str_d,
64       q_str_atm      => q_str_atm,
65       edge_count     => edge_count,
66       edges_done     => edges_done
67 );
68
69   clk_s <= not clk_s after CLK_S_PERIOD / 2;
70
71   gbf_out <= not gbf_out after GBF_PERIOD / 2;
72
73   process
74     variable clk_s_period_current : TIME;
75     variable gbf_period_current   : TIME;
76     variable reset_time          : TIME;
77     variable line_out            : LINE;
78     variable clk_s_freq          : real;
79     variable gbf_freq            : real;
80   begin
81     file_open(results_file, results_filename, WRITE_MODE);
82
83     write(line_out,
84     string'("clk_s_T(s),gbf_T(s),q_str_d,Alarm_Detector,q_str_atm,Alarm_ATM"));
85     writeline(results_file, line_out);
86
87     reset <= '1';
88     enable_ro_and_siro <= '0';
89     wait for 5 us;
90
91     enable_ro_and_siro <= '1';
92     wait for 5 us;
93
94     reset <= '0';
95     wait for 5 ms;
96
97     enable_ro_and_siro <= '0';
98     wait for 1 ms;

```

```

99      for i in -NUM_STEPS to NUM_STEPS loop
100          clk_s_period_current := CLK_S_BASE * (1.0 + real(i) *
101              STEP_PERCENT / 100.0);
102          CLK_S_PERIOD <= clk_s_period_current;
103          for j in -NUM_STEPS to NUM_STEPS loop
104              gbf_period_current := GBF_BASE * (1.0 + real(j) *
105                  STEP_PERCENT / 100.0);
106              GBF_PERIOD <= gbf_period_current;
107              reset_time := CLK_S_PERIOD;
108              reset <= '1';
109              wait for reset_time;
110              reset <= '0';
111
112              wait for STABLE_TIME;
113
114              write(line_out, clk_s_period_current / 1 ns * 1.0e-9);
115              write(line_out, string(','));
116              write(line_out, gbf_period_current / 1 ns * 1.0e-9);
117              write(line_out, string(','));
118              write(line_out, integer'image(to_integer(unsigned(q_str_d))));
119              write(line_out, string(','));
120              write(line_out, std_logic'image(Alarm_Detector));
121              write(line_out, string(','));
122              write(line_out,
123                  integer'image(to_integer(unsigned(q_str_atm))));
124                  write(line_out, string(','));
125                  write(line_out, std_logic'image(Alarm_ATM));
126                  writeline(results_file, line_out);
127
128          end loop;
129      end loop;
130
131      file_close(results_file);
132
133      report "Simulation completed. Results written to " & results_filename;
134      wait;
135  end process;
136 end Behavioral;

```

## 10.F Code Python Génération Heatmap

Python

```

1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from matplotlib.patches import Patch
5 from datetime import datetime
6
7 def plot_alarm_heatmap(
8     csv_filename,
9     lowest_percentage,
10    highest_percentage,
11    step_percent,
12    detector_interval,    # e.g., 30 for ±30%
13    atm_interval           # e.g., 10 for ±10%
14):

```

```

15     COLOR_LIGHT_GREEN = (0.0, 1.0, 0.0)
16     COLOR_LIGHT_ORANGE = (1.0, 0.85, 0.5)
17     COLOR_DARK_ORANGE = (1.0, 0.55, 0.0)
18     COLOR_RED = (1.0, 0.2, 0.2)
19     COLOR_DARK_GREEN = (0.0, 0.3, 0.0)
20     COLOR_DARKER_GREEN = (0.0, 0.15, 0.0)
21
22     df = pd.read_csv(csv_filename)
23
24     pivot_data = df.pivot_table(
25         index='gbf_T(s)',          # y-axis
26         columns='clk_s_T(s)',      # x-axis
27         values='Alarm_Detector',    # placeholder
28         aggfunc='first'
29     )
30
31     color_matrix = np.zeros((len(pivot_data.index), len(pivot_data.columns),
32                               3))
33
34     x_values = list(pivot_data.columns)
35     y_values = list(pivot_data.index)
36
37     for i, gbf_T in enumerate(y_values):
38         for j, clk_T in enumerate(x_values):
39             match = df[(df['clk_s_T(s)'] == clk_T) & (df['gbf_T(s)'] == gbf_T)]
40
41             if match.empty:
42                 color_matrix[i, j] = [0, 0, 0]
43                 continue
44
45             row = match.iloc[0]
46
47             detector_alarm = int(str(row['Alarm_Detector']).strip(' '))
48             atm_alarm = int(str(row['Alarm_ATM']).strip(' '))
49
50             gbf = lowest_percentage + i * step_percent
51             clk = lowest_percentage + j * step_percent
52             if (-detector_interval <= clk <= detector_interval) and (-
53                 atm_interval <= gbf <= atm_interval):
54                 in_expected = True
55             else:
56                 in_expected = False
57
58             if in_expected:
59                 if detector_alarm == 0 and atm_alarm == 0:
60                     color_matrix[i, j] = COLOR_LIGHT_GREEN
61                 elif detector_alarm == 0 and atm_alarm == 1:
62                     color_matrix[i, j] = COLOR_LIGHT_ORANGE
63                 elif detector_alarm == 1 and atm_alarm == 1:
64                     color_matrix[i, j] = COLOR_DARK_ORANGE
65                 else:
66                     color_matrix[i, j] = [0, 0, 0]
67             else:
68                 if detector_alarm == 0 and atm_alarm == 0:
69                     color_matrix[i, j] = COLOR_RED
70                 elif detector_alarm == 0 and atm_alarm == 1:
71                     color_matrix[i, j] = COLOR_DARK_GREEN
72                 elif detector_alarm == 1 and atm_alarm == 1:
73                     color_matrix[i, j] = COLOR_DARKER_GREEN
74                 else:
75
76

```

```

71     color_matrix[i, j] = [0, 0, 0]
72
73 plt.figure(figsize=(12, 8))
74
75 x_labels = [f"{lowest_percentage + j * step_percent}%" for j in
76 range(len(x_values))]
77 y_labels = [f"{lowest_percentage + i * step_percent}%" for i in
78 range(len(y_values))]
79
80 plt.imshow(color_matrix[::-1], aspect='auto')
81 plt.xticks(range(len(x_labels)), x_labels, rotation=45)
82 plt.yticks(range(len(y_labels)), y_labels[::-1])
83
84 plt.xlabel('Clock Period Change')
85 plt.ylabel('GBF Period Change')
86 plt.title('Alarm States Heatmap')
87
88 legend_elements = [
89     Patch(color=COLOR_LIGHT_GREEN, label='No Alarms (in expected)'),
90     Patch(color=COLOR_LIGHT_ORANGE, label='Only ATM Alarm (in expected)'),
91     Patch(color=COLOR_DARK_ORANGE, label='Both Alarms (in expected)'),
92     Patch(color=COLOR_RED, label='No Alarms (out of expected)'),
93     Patch(color=COLOR_DARK_GREEN, label='Only ATM Alarm (out of
94 expected)'),
95     Patch(color=COLOR_DARKER_GREEN, label='Both Alarms (out of expected)')
96 ]
97 plt.legend(handles=legend_elements, loc='upper left')
98
99 plt.tight_layout()
100
101 timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
102 save_path = f"Heatmaps/heatmap_{timestamp}.png"
103 plt.savefig(save_path)
104 print(f"Heatmap saved to {save_path}")
105
106 plt.show()
107
108 plot_alarm_heatmap("Simulations_Results/
109 simulation_results_50_2percent_v9.csv", -50, 50, 2, False, 30, 10)
110
111 from IPython.display import Image, display
112
113 display(Image(filename="MTR_Latest_Simulation_Results.png"))

```

## Bibliographie

- [1] M. Esmaeilian, V. Beroule, et D. Hély, « On Power-Off Temperature Attacks Potential Against Security Sensors », 2025, doi: 10.3390/s25061912°.
- [2] A. Douadi, G. Di Natale, P. Maistri, E.-I. Vatajelu, et V. Beroule, « A Study of High Temperature Effects on Ring Oscillator Based Physical Unclonable Functions », in *2023 IEEE 29th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2023, p. 1-7. doi: 10.1109/IOLTS59296.2023.10224886°.
- [3] Hmimou, « Rapport de stage Master 1, Protection des systèmes embarqués, LCIS », 2024.