Titre: Title:	Développement et caractérisation d'une solution de traçage pour systèmes hétérogènes CPU-FPGA
Auteur: Author:	Nicolas Deloumeau
Date:	2025
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Deloumeau, N. (2025). Développement et caractérisation d'une solution de traçage pour systèmes hétérogènes CPU-FPGA [Master's thesis, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/71205/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/71205/
Directeurs de recherche: Advisors:	Tarek Ould-Bachir
Programme: Program:	Génie Informatique

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Développement et caractérisation d'une solution de traçage pour systèmes hétérogènes CPU-FPGA

NICOLAS DELOUMEAU
Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Décembre 2025

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Développement et caractérisation d'une solution de traçage pour systèmes hétérogènes CPU-FPGA

présenté par **Nicolas DELOUMEAU**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Heng LI, président
Tarek OULD-BACHIR, membre et directeur de recherche
François LEDUC-PRIMEAU, membre

REMERCIEMENTS

Je souhaite d'abord exprimer ma profonde gratitude à Tarek Ould-Bachir, mon directeur de recherche, pour son accompagnement, sa disponibilité et ses conseils tout au long de ce projet. Ses orientations et sa confiance ont grandement contribué à la réalisation de ce projet.

Je remercie Mitacs et Ciena pour leur appui financier, sans lequel ce projet n'aurait pu être mené à bien. Ciena a joué un rôle clé non seulement en soutenant le projet, mais aussi en favorisant un environnement propice à l'innovation et à la collaboration entre l'industrie et la recherche universitaire.

Notamment, ma reconnaissance s'étend à l'ensemble de l'équipe de Ciena, et plus particulièrement à François Tetreault, qui a su initier ce projet et sans qui celui-ci n'aurait pu voir le jour. Je remercie également Jamie Sanderson, qui a beaucoup contribué à mobiliser le projet chez Ciena. Je remercie aussi Andrew Handke, David Evans et le reste de l'équipe pour nos rencontres régulières, leurs commentaires pertinents et leurs retours.

Je tiens également à remercier les membres du jury, les professeurs Heng Li et François Leduc-Primeau, pour le temps consacré à l'évaluation de ce mémoire et pour leurs commentaires constructifs, qui ont contribué à en améliorer la clarté et la portée.

Enfin, je souhaite remercier ma famille et mes amis pour leur soutien, ainsi que Robin pour sa présence et les nombreuses heures partagées au bureau.

RÉSUMÉ

L'évolution des architectures informatiques contemporaines s'inscrit dans un contexte où le ralentissement de la loi de Moore a renforcé l'adoption de systèmes hétérogènes combinant processeurs généralistes (CPU) et accélérateurs matériels tels que GPU, FPGA et ASIC. Si ces architectures permettent d'améliorer les performances pour des applications spécialisées, elles introduisent aussi une complexité accrue. Les outils classiques de débogage matériel ou logiciel s'avèrent insuffisants : les traceurs logiciels n'offrent qu'une vue partielle du comportement matériel, tandis que les analyseurs intégrés aux FPGA (tels que les ILA d'AMD/Xilinx) permettent seulement une observation locale, limitée en durée et en signaux.

Ce mémoire propose une infrastructure complète, unifiée et robuste de traçage CPU–FPGA, permettant une observabilité fine des systèmes hétérogènes modernes et facilitant le diagnostic, l'optimisation et la compréhension des interactions logicielles-matérielles. L'infrastructure proposée comprend un mécanisme unifié de traçage matériel–logiciel, capable d'observer simultanément l'activité du FPGA et celle du CPU, d'aligner temporellement leurs événements et de permettre une analyse globale du système.

Sur le plan matériel, la contribution majeure réside dans la conception d'un traceur FPGA générique et modulaire, construit autour : 1) Des sondes configurables détectant des signaux, bus AXI ou machines à états ; 2) Une chaîne de sondes inspirée de travaux antérieurs, agrégée en flux AXI-Stream unique ; 3) Un système de synchronisation multi-horloge ; et 4) Un mécanisme de transfert PCIe basé sur le sous-système XDMA. L'ensemble est généré automatiquement à l'aide de Chisel.

Du côté logiciel, le mémoire détaille la réalisation d'un pilote noyau Linux dédié, exposant deux périphériques : un accès mémoire mappée pour configurer les sondes et insérer des marqueurs de synchronisation, et un flux événementiel continu pour récupérer les données brutes issues du FPGA. Un module de post-traitement convertit ces flux en *Common Trace Format* (CTF), permettant une compatibilité immédiate avec Trace Compass.

Une évaluation expérimentale approfondie montre la capacité du système à supporter de très hauts débits via PCIe, à tracer en continu plusieurs domaines d'horloge, et à maintenir un taux de pertes minimal. Le mémoire analyse l'impact des profondeurs de FIFO, du débit du DMA, des rafales d'événements et des configurations multi-domaine. Les limites identifiées, telles que la mise à l'échelle et la gestion des rafales extrêmes, sont discutées, et plusieurs perspectives sont proposées, notamment l'intégration future de compresseurs de traces.

ABSTRACT

The evolution of contemporary computing architectures is occurring in a context where the slowdown of Moore's law has accelerated the adoption of heterogeneous systems combining general-purpose processors (CPUs) with hardware accelerators such as GPUs, FPGAs, and ASICs. While these architectures significantly improve performance for specialized applications, they also introduce additional complexity. Traditional hardware and software debugging tools prove insufficient: software tracers provide only a partial view of hardware behavior, while FPGA-integrated analyzers (such as AMD/Xilinx ILAs) offer only localized, short-duration observations, limited in scope and in the number of accessible signals.

This thesis proposes a complete, unified, and robust CPU–FPGA tracing infrastructure that enables fine-grained observability of modern heterogeneous systems and facilitates debugging, performance optimization, and the understanding of hardware–software interactions. The proposed infrastructure provides a unified tracing mechanism capable of simultaneously observing FPGA and CPU activity, temporally aligning their events, and enabling a global analysis of system behavior.

On the hardware side, the main contribution lies in the design of a generic and modular FPGA tracer built around: 1) Configurable probes capable of monitoring signals, AXI buses, or state machines; 2) A probe-chain inspired by prior work, aggregated into a single AXI-Stream flow; 3) Aa multi-clock synchronization mechanism; and 4) A PCIe transfer system based on the XDMA subsystem. The entire architecture is automatically generated using Chisel from a high level description.

On the software side, the thesis details the implementation of a dedicated Linux kernel driver exposing two devices: a memory-mapped interface for configuring probes and inserting synchronization markers, and a continuous event-stream interface for retrieving raw data from the FPGA. A post-processing module converts these streams into the Common Trace Format (CTF), enabling immediate compatibility with Trace Compass.

An extensive experimental evaluation demonstrates the system's ability to sustain very high throughput over PCIe, to trace multiple clock domains continuously, and to maintain a minimal event-loss rate. The thesis analyzes the impact of FIFO depths, DMA bandwidth, event bursts, and multi-domain configurations. Identified limitations (such as scalability and the handling of extreme burst scenarios) are discussed, and several future directions are proposed, including the integration of trace-compression mechanisms.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

API	Application Programming Interface
AXI	Advanced eXtensible Interface
BRAM	Block RAM
CDC	Cross-Domain Crossing
CDEV	Caracter Device
CTF	Common Trace Format
CUDA	Compute Unified Device Architecture
DMA	Direct Memory Access
DVFS	Dynamic Voltage and Frequency Scaling
FF	Flip Flop
FIFO	First-In, First-Out
FPGA	Field-Programmable Gate Array
GPU	Graphics Processing Unit
HIL	Hardware-In-the-Loop
HLS	High-Level Synthesis
HSA	Heterogeneous System Architecture
ILA	Integrated Logic Analyzer
IP	Ellipse de IP Core : Intellectual Property Core
IRQ	Interrupt Request
LFSR	Linear-Feedback Shift Register
LTTng	Linux Trace Toolkit : next generation
LUT	Lookup Table
ME/s	Million d'événements par seconde
MMIO	Memory-Mapped I/O
MSI	Message Signaled Interrupts
NTP	Network Time Protocol
OpenCL	Open Computing Language
OpenGL	Open Graphics Library
PCIe	Peripheral Component Interconnect Express
PPT-GPU	Performance Prediction Toolkit for GPUs
PTP	Precision Time Protocol
RAM	Random Access Memory
ROCm	Radeon Open Compute

ROCr	ROCm Runtime
RTL	Register Transfer Logic
SoC	System on a Chip
SSD	Solid-State Drive
TLPs	Transaction Layer Packets
TSC	Time Stamp Counter
UST	User Space Tracing
VCD	Value Change Dump
XDMA	Xilinx DMA
XML	Extensible Markup Language

CHAPITRE 1 INTRODUCTION

1.1 Contexte et problématique

Avec l'atteinte des limites de la loi de Moore, les architectures des systèmes informatiques ont dû être repensées. Le ralentissement du progrès de la miniaturisation des semi-conducteurs a fait croître la demande pour les architectures spécialisées. Plus efficaces et performantes pour des applications spécifiques, elles permettent de répondre à la demande toujours croissante en puissance de calcul. Les systèmes hétérogènes, combinant processeurs à usage général et accélérateurs dédiés (tels que GPU, FPGA ou ASIC), s'imposent comme solution de choix en alliant la flexibilité du logiciel et l'accélération matérielle.

Cependant, les systèmes hétérogènes introduisent un degré de complexité, notamment par la différence de leur architecture. Les interactions peuvent avoir des dépendances bidirectionnelles, rendant l'analyse complexe. Pour exploiter au mieux ces architectures, les concepteurs doivent disposer d'outils capables d'analyser ces systèmes. De tels outils permettent non seulement de détecter des bogues, mais aussi d'identifier des opportunités d'optimisation des performances.

Au niveau logiciel, la nécessité d'obtenir l'état d'un programme existe depuis longtemps, d'où la conception des débogueurs. Cependant, leur utilisation entraîne souvent un ralentissement important, ce qui les rend peu adaptés pour analyser des programmes à haute performance. Par la suite, des outils de profilage et de traçage ont été développés, offrant une vue plus globale des événements d'un programme. Ces outils fournissent une perspective temporelle de l'exécution. L'avantage du traçage est qu'il permet d'observer un système avec un impact réduit sur les performances. Cet aspect est essentiel, car tout ralentissement peut modifier le comportement d'un programme, par exemple en masquant certains bogues liés à la concurrence. Pour identifier avec précision les délais et obtenir le maximum de performance des systèmes modernes, il est donc important que les solutions de traçage enregistrent un flot d'événements aussi fidèle que possible à la réalité.

Dans le cas des FPGA et des ASIC, une grande partie de la validation se fait par simulation. Cette approche permet de tester en profondeur chaque module ou l'ensemble d'un design et d'observer son comportement dans le temps. Toutefois, il reste nécessaire de valider également les systèmes implémentés, car certains problèmes peuvent échapper aux simulations. En particulier pour les FPGA, des outils de débogage permettent d'obtenir des informations supplémentaires. Pour les approches dites au niveau de transfert de registres (*Register Trans-*

fer Level — RTL), les fabricants de FPGA proposent des analyseurs de signaux capables de capturer des traces temporelles et d'offrir une vue locale sur un intervalle restreint. Cette solution est efficace pour obtenir une observation précise des signaux, mais elle est limitée par la quantité de données générées. Elle est donc surtout utile lorsque le problème est bien localisé. En revanche, il peut être difficile de sélectionner les bons signaux à observer et de définir les conditions de déclenchement de l'enregistrement.

Malgré les outils logiciels et matériels existants, il reste difficile de localiser certains bogues dans les systèmes hétérogènes. Les traces logicielles seules ne révèlent qu'une partie du problème et cachent de nombreuses informations liées au matériel. Les interactions entre le matériel et le logiciel peuvent être longues et complexes, ce qui rend l'usage des analyseurs de signaux traditionnels peu adapté. Il devient alors essentiel de disposer d'une vue globale du système, qui retrace un fil d'exécution commun entre le logiciel et le matériel.

De plus, les équipes responsables du logiciel et celles des FPGA possèdent des expertises très différentes, ce qui complique encore l'identification des causes profondes d'un bogue. Obtenir une vue unifiée des événements faciliterait ainsi la collaboration entre ces différentes équipes.

1.2 Objectifs de recherche

L'objectif général de cette recherche est de concevoir, implanter et valider une infrastructure complète de traçage adaptée aux systèmes hétérogènes CPU–FPGA. Cette infrastructure vise à offrir une observabilité continue, une synchronisation temporelle précise et une intégration transparente avec les outils existants de traçage logiciel, afin de permettre l'analyse conjointe des interactions entre le processeur hôte et la logique programmable.

Pour atteindre cet objectif, plusieurs objectifs spécifiques ont été définis :

- Réaliser le traçage d'un bus AXI, de signaux et de machines à états en continu dans un FPGA avec plusieurs horloges ;
- Transférer la trace en temps réel vers un système d'exploitation par PCIe ;
- Générer une trace dans le format CTF et permettre la visualisation dans l'outil Trace Compass ;
- Synchroniser les événements FPGA avec les événements logiciels ;
- Caractériser la solution de traçage par ses performances et analyser son impact sur l'application.

1.3 Contributions de la recherche

La contribution de cette recherche réside dans la conception et la mise en œuvre d'une solution complète de traçage matériel–logiciel pour des systèmes hétérogènes CPU-FPGA. Premièrement, une solution de traçage en continu a été réalisée pour un bus AXI, des signaux internes et des machines à états, et cela avec plusieurs horloges. Deuxièmement, un transfert en temps réel des traces vers un système d'exploitation a été assuré au moyen de l'interface PCIe, garantissant un débit élevé et un transfert efficace des données. Troisièmement, les traces ont été générées au format CTF et intégrées à l'outil Trace Compass, permettant une analyse conjointe entre le matériel et le logiciel. De plus, les événements FPGA et logiciels ont été synchronisés, ce qui permet une corrélation temporelle entre les deux architectures. Enfin, la solution a été caractérisée en termes de performances et d'impact sur les applications observées, offrant ainsi une évaluation complète de l'approche proposée.

Il convient également de mentionner que les résultats de ce travail ont fait l'objet de présentations et de publications dans des forums scientifiques. Une première communication a été présentée au *Tracing Summit 2025* [1], mettant en évidence l'approche de traçage en continu pour les interfaces AXI. Par la suite, un article a été soumis au *International Symposium on Field-Programmable Gate Arrays 2026* de l'ACM [2], présentant en détail la méthode de synchronisation des événements CPU–FPGA et les résultats expérimentaux obtenus.

1.4 Organisation du mémoire

L'organisation de ce mémoire est la suivante :

- Le Chapitre 2 présente les fondements théoriques et une revue de littérature couvrant les principales approches de traçage logiciel, GPU, FPGA et hétérogène, ainsi que les outils d'analyse associés.
- Le Chapitre 3 décrit l'architecture matérielle développée, en détaillant la conception des sondes, leur chaînage, le contrôleur, ainsi que les mécanismes de transfert de données entre le FPGA et le processeur hôte, incluant la mémoire tampon, la scrutation, le DMA et les interruptions.
- Le Chapitre 4 est consacré à l'architecture logicielle, incluant le pilote Linux, le logiciel de traçage, la gestion du format CTF, la synchronisation entre événements matériels et logiciels, ainsi que l'intégration avec Trace Compass pour la visualisation et l'analyse.

- Le Chapitre 5 présente l'évaluation expérimentale de la solution proposée, avec une analyse des performances du DMA, de la chaîne de sondes et du traitement logiciel, et propose une discussion critique des résultats obtenus, en soulignant les limites du système et les perspectives d'évolution pour le traçage matériel–logiciel dans les plateformes hétérogènes.
- Le Chapitre 6 conclut le mémoire.

CHAPITRE 2 FONDEMENTS ET REVUE DE LITTÉRATURE

2.1 Contexte général

2.1.1 Observabilité et traçage dans les systèmes modernes

Les systèmes informatiques contemporains reposent sur des architectures de plus en plus hétérogènes, combinant des processeurs généralistes (CPU), des accélérateurs matériels spécialisés (GPU, FPGA, ASIC) et des couches logicielles complexes [3,4]. Cette évolution s'accompagne d'un accroissement notable de la parallélisation, du débit des interfaces et de la diversité des domaines d'horloge. Dans ce contexte, l'observabilité devient essentielle pour comprendre le comportement temporel des applications, diagnostiquer les erreurs, mesurer l'efficacité des communications et optimiser l'utilisation des ressources [5]. Cependant, cette observabilité doit être obtenue sans perturber le système instrumenté, ce qui impose des mécanismes de traçage à très faible intrusion et capables de résister à de fortes charges événementielles [6,7].

L'un des défis majeurs réside dans la collecte cohérente d'informations provenant d'unités très différentes en termes de granularité temporelle et de modèles d'exécution. Les processeurs exécutent des séquences d'instructions organisées autour d'événements logiciels relativement rares, tandis que les FPGA produisent des événements matériels potentiellement à la fréquence du cycle, avec des débits pouvant dépasser plusieurs dizaines de millions d'événements par seconde [8]. À cela s'ajoutent les effets des communications inter-composants, telles que les transferts DMA sur PCIe, qui introduisent des délais variables. Obtenir une vision unifiée nécessite donc des mécanismes robustes d'horodatage, de synchronisation et de gestion de flux, capables de capturer simultanément des événements logiciels et matériels dans un même cadre temporel.

2.1.2 Concepts fondamentaux du traçage

Le traçage repose sur la capture structurée d'événements décrivant l'activité d'un système. Un événement représente l'occurrence d'une action significative (interruption, accès mémoire, changement d'état d'un signal ou appel de fonction) horodatée grâce à une source temporelle locale [9]. Ces événements, générés à vitesse potentiellement élevée, doivent être stockés temporairement dans des tampons circulaires avant d'être transmis ou enregistrés. L'efficacité du système de traçage dépend ainsi de la profondeur des tampons, du mode de transmission et du coût d'instrumentation, qui doit être minimal pour éviter de modifier le comportement du système observé.

Un concept clé du traçage est celui du format de trace. Des standards tels que *Common Trace Format* (CTF) [10] permettent d'organiser les événements en flux temporels ordonnés et compatibles avec des outils d'analyse avancée. Une difficulté apparaît lorsque les traces proviennent de multiples domaines d'horloge : chaque source utilise sa propre référence temporelle, rendant l'alignement non trivial. Les systèmes modernes doivent donc intégrer des mécanismes de synchronisation, parfois basés sur des balises temporelles, ainsi que des algorithmes de compensation des décalages [11].

2.1.3 Motivations pour un traçage unifié CPU–FPGA

Les plateformes intégrant CPU et FPGA sont de plus en plus utilisées pour l'accélération de processus critiques [12], la simulation en temps réel [13], ou le traitement de données à très haut débit [14]. Cependant, les outils existants ne permettent généralement d'observer que l'un des deux mondes : les traceurs logiciels capturent les interactions noyau/utilisateur, tandis que les outils embarqués sur FPGA se limitent à des fenêtres temporelles restreintes ou à des signaux pré-sélectionnés. Cette séparation complique l'analyse des interactions fines entre le logiciel et la logique matérielle, notamment pour diagnostiquer des retards, des blocages ou des incohérences entre les commandes du CPU et les réactions du FPGA.

Un traçage unifié CPU–FPGA vise à fournir une chronologie complète des événements, couvrant simultanément les couches matérielles et logicielles, et permettant d'établir des relations causales globales [15]. Une telle approche facilite le débogage de systèmes hybrides, l'analyse de la performance des interfaces AXI [16]/PCIe et la compréhension du comportement inter-couches. L'existence d'une ligne de temps cohérente ouvre la voie à des outils d'analyse, de visualisation et de détection d'anomalies plus avancés, tout en réduisant le temps de développement lors du diagnostic de problèmes complexes.

2.2 Traçage logiciel

2.2.1 Principes généraux

Le traçage logiciel permet d'observer l'exécution d'un système, au niveau du noyau ou des applications, dans l'espace utilisateur. Il repose sur différents mécanismes d'instrumentation (statiques ou dynamiques), qui insèrent des points d'observation dans le flot d'exécution. Ces points génèrent des événements horodatés décrivant l'activité du système, tels que des interruptions, des appels système, des changements d'état ou des transitions logiques dans un programme [9].

Pour être utilisables à grande échelle, ces événements doivent être collectés efficacement et avec très peu d'intrusion. Ils sont généralement stockés dans des tampons circulaires et transmis périodiquement vers un moteur de collecte. La précision des horodatages, souvent fondée sur des compteurs matériels, est essentielle pour permettre une analyse temporelle fine [17]. Enfin, l'utilisation de formats de trace normalisés (comme le CTF) assure la compatibilité avec des outils avancés de visualisation, de corrélation et d'analyse post mortem.

2.2.2 LTTng : architecture et fonctionnement

LTTng (Linux Trace Toolkit : next generation) [18] est un outil open source de traçage conçu pour l'analyse des performances et le débogage de systèmes Linux. Il permet de collecter des traces du noyau et des applications en espace utilisateur, avec un impact minimal sur les performances du système instrumenté. Plusieurs études ont montré que son surcoût est nettement inférieur à celui des autres traceurs logiciels [9], ce qui en fait un outil privilégié pour le traçage continu.

L'écosystème LTTng se compose de plusieurs composantes complémentaires :

- LTTng-modules [19] est l'extension du noyau responsable de l'instrumentation de bas niveau. Elle permet de capturer des événements tels que les interruptions matérielles, les appels système, les informations sur les processus ou encore l'activité des pilotes.
- LTTng-UST (User Space Tracer) [20] est le composant destiné aux applications en espace utilisateur. Il permet aux développeurs d'ajouter des points de trace directement dans leur code à l'aide d'une API (*Application Programming Interface*) dédiée. Cette instrumentation permet l'observation du déroulement logique d'une application, la corrélation avec les événements du noyau et la détection d'éventuels goulots d'étranglement dans le flot d'exécution.
- LTTng-sessiond [21] est un processus d'arrière-plan qui gère les sessions de traçage. Il assure la coordination entre les différents traceurs, la configuration des canaux de trace, le démarrage et l'arrêt des sessions, ainsi que la collecte et l'écriture des données sur le disque. Ce composant permet d'unifier les traces noyau et logicielles.

L'outil offre plusieurs options pour s'adapter aux besoins : il est possible de sélectionner les événements à tracer, de définir des filtres ou encore de configurer les tampons mémoire utilisés pendant une séance de traçage. Les traces produites par LTTng sont enregistrées au format CTF, dans une structure conçue pour représenter efficacement des événements horodatés provenant de flux multiples. Ce format sert de base à plusieurs outils d'analyse, dont Babeltrace et Trace Compass.

2.2.3 Outils associés

Outre les mécanismes d'instrumentation et de collecte fournis par LTTng, l'écosystème de traçage s'appuie sur plusieurs outils complémentaires destinés à la manipulation, à la conversion et à l'analyse avancée des traces. Ces outils permettent de parcourir des flux d'événements volumineux, de les fusionner lorsqu'il s'agit de plusieurs sources, ou encore de produire des visualisations adaptées à l'étude de systèmes complexes.

Les deux outils les plus utilisés dans ce contexte sont *Babeltrace*, pour le traitement programmatique et la conversion des traces CTF, et *Trace Compass*, une plateforme graphique d'analyse interactive.

Babeltrace

Babeltrace [22] est un ensemble d'outils et de bibliothèques open source utilisés pour lire, convertir, analyser et visualiser des traces d'exécution, notamment celles produites au format CTF, comme c'est le cas pour LTTng et d'autres outils de traçage. Babeltrace permet également de fusionner et de parcourir des traces provenant de différentes sources.

L'écosystème Babeltrace comprend une interface en ligne de commande ainsi que des API en C et en Python, facilitant l'intégration dans des scripts et des pipelines d'analyse automatisée. Ces interfaces permettent d'extraire des événements, d'appliquer des filtres, de restructurer des traces ou de générer des formats intermédiaires pour d'autres outils. Par ailleurs, Babeltrace sert de backend à plusieurs environnements de visualisation, dont Trace Compass, ce qui renforce son rôle au sein de l'écosystème du traçage logiciel.

Trace Compass

Trace Compass [23] est un outil de visualisation et d'analyse de traces d'exécution conçu pour faciliter le diagnostic des performances et le débogage de systèmes complexes. Développé initialement par le projet Eclipse, Trace Compass permet de traiter de grands volumes de données issues de traces logicielles CTF, telles que celles générées par LTTng. Trace Compass peut également intégrer d'autres types de traces, comme `perf` [24] ou `ftrace` [25], ou encore des systèmes embarqués avec `barectf` [26].

L'outil fournit une interface graphique interactive qui permet différentes analyses, dont des vues chronologiques, des statistiques sur les événements et des représentations des ressources. Ces fonctionnalités permettent d'examiner le comportement temporel d'un système, d'identifier des goulots d'étranglement et de corréler des événements provenant de sources différentes.

Trace Compass a une architecture modulaire et extensible. Il est possible de développer des plug-ins d'analyse adaptés à des besoins spécifiques, par exemple en créant des analyses ou des vues sur mesure en XML [27]. La possibilité d'intégrer de multiples sources de traces permet également son utilisation lorsque les traces proviennent de systèmes distribués ou hétérogènes. Des fonctionnalités telles que la synchronisation entre différentes traces sont disponibles dans Trace Compass [28].

2.3 Traçage matériel et hétérogène

2.3.1 Traçage GPU

Le traçage GPU vise à observer et analyser le comportement interne des unités de calcul massivement parallèles utilisées pour l'accélération d'applications scientifiques et d'apprentissage profond. Plusieurs solutions commerciales offrent des outils puissants, mais fermés, centrés sur l'analyse des performances.

Plusieurs solutions commerciales existent pour le traçage des GPU. Sur les plateformes NVIDIA, l'environnement Nsight [29] permet de tracer les files de commandes du GPU, les appels système et l'activité des cadriciels (CUDA, Vulkan, DirectX), facilitant l'identification des points de contention, des latences de synchronisation et des phases d'inactivité du GPU. Du côté d'AMD, l'écosystème ROCm [30] propose des outils de profilage et de traçage en logiciel libre, exploitant les appels à l'API et les compteurs de performance matériels.

Cependant, la majorité des solutions non commerciales demeurent limitées, en raison de la nature propriétaire des architectures GPU. Quelques initiatives récentes ont néanmoins tenté de combler ce manque en proposant des cadres de modélisation et d'analyse traçables. Arafa *et al.* [31] ont présenté PPT-GPU (*Performance Prediction Toolkit for GPUs*), un outil de modélisation des performances des GPU basé sur des traces d'instructions et de mémoire. Leur approche permet de prédire les performances tout en conservant une bonne extensibilité et une portabilité entre générations de GPU NVIDIA. Plus récemment, Darche *et al.* [7] ont proposé un traçage des GPU basé sur la modélisation et la compréhension du comportement microarchitectural.

2.3.2 Traçage embarqué

Pour les processeurs embarqués, `barectf` est un générateur de traceurs conçu pour produire des flux de traces au format CTF. À partir d'un fichier de configuration, `barectf` génère à la fois les métadonnées CTF et du code source C, ce qui permet d'intégrer directement un

traceur dans un programme C ou C++. L'intégration se fait via le fichier d'en-tête, dont les fonctions peuvent être appelées aux emplacements souhaités dans le code pour écrire des événements dans des paquets CTF.

Dans le domaine des FPGA SoC (*System-on-Chip*), Fabien *et al.* présentent CoChrono [32], un outil d'instrumentation innovant permettant d'analyser de manière unifiée les performances des systèmes embarqués conçus à l'aide d'outils de synthèse de haut niveau (*High-Level Synthesis* — HLS). CoChrono fournit une API C++ unique, utilisable de manière transparente dans le logiciel, les accélérateurs matériels générés par HLS et la co-simulation SystemC. L'outil repose sur des compteurs pour le calcul du temps dans le FPGA.

2.3.3 Traçage FPGA

Traçage matériel continu

Blochwitz *et al.* [33] proposent une architecture de surveillance continue du RTL qui enchaîne des modules de trace de manière sérielle et transmet les événements déclenchés par des changements au système hôte via PCIe pour conversion en format VCD. Leur conception évite les captures fenêtrées propres aux ILA et prend en charge les conceptions multi-horloges, tout en réduisant l'utilisation des ressources du FPGA jusqu'à 70% par rapport à l'ILA. Leur système atteint un débit allant jusqu'à 3,10 GB/s et permet une observation continue.

Penneman *et al.* [34] présentent une solution similaire pour l'observation en temps réel des signaux internes. Leur méthode horodate les transitions logiques et les transfère en continu par Ethernet vers l'hôte. L'architecture met en évidence la faisabilité d'un traçage non intrusif basé sur une communication asynchrone, qui assure un flux régulier de données et une synchronisation stable, malgré les variations de latence réseau.

Ces approches démontrent l'intérêt d'un traçage matériel continu capable de couvrir de longues périodes d'observation, contrairement aux approches fenêtrées limitées par la profondeur de mémoire des ILAs. Cette continuité est essentielle pour l'étude des phénomènes rares ou intermittents, tels que les congestions de bus, les violations des contraintes de temps ou les interblocages transitoires. De plus, le transfert asynchrone des traces vers l'hôte permet de dissocier la fréquence d'échantillonnage interne du débit de sortie. Enfin, ces travaux précèdent les approches de traçage hétérogène, où la continuité et la synchronisation inter-domaines forment des prérequis pour corréler efficacement les événements entre le matériel et le logiciel.

Instrumentation pour conceptions HLS ou OpenCL

Une partie importante des travaux modernes consiste à améliorer l'observabilité dans les conceptions générées par HLS ou OpenCL. Bensalem *et al.* [35] introduisent un cadre d'instrumentation dans le FPGA pour les accélérateurs OpenCL, qui insère des sondes capturant au cycle près avec un surcoût en ressources extrêmement faible. Cela permet une analyse des performances et, par conséquent, d'éliminer les blocages. Un autre travail propose des sondes OpenCL à précision d'un cycle d'horloge, capables d'observer arbitrairement des variables et des comportements de pipeline [36]. Ces approches répondent au manque d'observabilité dans les flux HLS/OpenCL et fournissent une visibilité sur le comportement interne de l'accélérateur.

Au niveau HLS, Choi et Cong proposent HLScope [37], qui automatise l'insertion de moniteurs de performance au sein de Vivado HLS et utilise la simulation logicielle pour identifier rapidement les blocages et d'autres goulots d'étranglement. HLScope fournit donc une vue à un haut niveau, intégrée à l'outil HLS. Kim et Hao proposent RealProbe [38], qui permet de profiler des modules HLS à l'aide de directives directement dans le code. Ils extraient alors les signaux de contrôle utiles de manière automatisée et les connectent à leurs noyaux de propriété intellectuelle (*Intellectual Property Core*, abrégé en *IP core*, ou simplement IP par ellipse).

Monson et Hutchings [39] introduisent des ports d'observation et des tampons de traces par événement, accompagnés de méthodes pour reconstruire les relations temporelles entre événements stockés dans des mémoires tampons indépendantes. De même, Goeders et Wilton [40] proposent des techniques de traçage adaptées aux circuits HLS, incluant la compression, le changement dynamique des signaux enregistrés et leur reconstitution par la suite, ce qui permet d'augmenter la profondeur de trace.

2.3.4 Compression de trace en temps réel

Tsai *et al.* [41] développent un compresseur de traces en deux niveaux pour les FPGA, combinant une première compression basée sur des compteurs et une compression par similarité. Leur conception repose sur une structure en pipeline synchronisée avec l'horloge d'échantillonnage, garantissant un traitement en temps réel avec une consommation minimale de ressources logiques. Cette approche atteint un rapport de compression pouvant aller jusqu'à $\times 100$, tout en maintenant un débit synchronisé avec l'horloge du système observé.

Ce travail pionnier illustre l'intérêt d'une compression matérielle intégrée, en particulier pour les outils de débogage embarqués, où la taille de la mémoire FPGA constitue un facteur limitant majeur. En réduisant drastiquement la quantité de données stockées, cette approche ouvre la voie à des traceurs à longue profondeur et à un diagnostic plus exhaustif des transitions d'état, des erreurs de synchronisation et des comportements transitoires dans les systèmes sur puce.

2.4 Traçage hétérogène

Comme évoqué précédemment, le traçage hétérogène vise à offrir une visibilité unifiée sur l'exécution conjointe d'architectures combinant des CPU et des accélérateurs matériels tels que les GPU ou les FPGA. Dans ces environnements, la majorité des travaux récents s'est attachée à étendre les outils de traçage existants pour capturer simultanément le CPU et le GPU, et reconstruire une chronologie cohérente de leurs interactions.

2.4.1 Traçage CPU–GPU unifié

Une première catégorie de travaux s'est concentrée sur l'intégration de capacités de traçage système dans les environnements d'exécution GPU hétérogènes. LTTng-CLUST, présenté par Couturier et Dagenais, propose un traçage unifié CPU–GPU pour les applications OpenCL [42]. Les auteurs étendent LTTng afin d'intercepter les appels à l'API OpenCL et de les combiner avec le traçage complet du CPU, permettant ainsi une visibilité globale de l'exécution hétérogène. Le résultat est une faible surcharge de traçage et l'intégration de mécanismes existants de LTTng-UST, ce qui permet de corréler les activités du CPU, l'envoi des commandes du GPU et l'exécution des noyaux au sein d'un cadre de traçage unique.

LTTng-HSA [5], proposé par Margheritta et Dagenais, introduit une instrumentation pour les systèmes GPU basés sur la norme HSA (*Heterogeneous System Architecture*) en intégrant LTTng au sein de ROCr, l'environnement d'exécution (*runtime*) de ROCm (*Radeon Open Compute*), la pile logicielle ouverte d'AMD destinée au calcul hétérogène. Leur approche repose sur le préchargement de bibliothèques pour intercepter les appels d'API GPU sans nécessiter de recompilation. Les traces produites unifient les appels API, les activités des noyaux GPU, les files de commandes et les métriques matérielles, ainsi que les événements du pilote Linux, tout en étant compatibles avec Trace Compass.

2.4.2 Corrélation et visualisation multi-domaines

Une évolution de ces travaux, qui inclut la visualisation et la corrélation de données issues de traces hétérogènes, a été étudiée par Fiorini et Dagenais [6]. Les auteurs proposent un cadre de profilage et de traçage combinant des événements CPU, des noyaux GPU, des appels d'API et des transferts de mémoire. S'appuyant sur ROCm, la plateforme de Radeon Open Compute, ROC-tracer, ROC-profiler et Trace Compass, leur approche fournit des vues globales de l'exécution hétérogène.

D'autres travaux, comme GLTraceSim, présenté par Sembrant *et al.*, analysent les interactions mémoire dans les systèmes hétérogènes, les systèmes mémoire CPU–GPU [43]. GLTraceSim capture les appels OpenGL via l'outil `Apitrace`, les rejoue dans un rendu logiciel à l'aide du moteur Mesa/LLVMpipe et en extrait les traces d'accès mémoire du GPU, synchronisées avec les accès de la file de rendu CPU. Les traces produites peuvent ensuite être rejouées dans des simulateurs haut niveau ou à granularité fine (comme `gem5`) afin d'évaluer les interactions mémoire entre le CPU et le GPU, notamment la contention sur les caches et la DRAM.

2.4.3 Traçage dans les systèmes embarqués hétérogènes

Les systèmes embarqués hétérogènes combinent plusieurs types de processeurs et parfois de la logique programmable, ce qui rend leur traçage particulièrement complexe. Les contraintes de ressources, la diversité des architectures et l'absence d'un système d'exploitation complet imposent des solutions légères et adaptées au bas niveau.

Pagano *et al.* [44] ont proposé une approche initiale pour le traçage multi-cœurs, fondée sur une instrumentation logicielle minimale et une synchronisation temporelle simplifiée. Leur travail a mis en évidence l'importance d'une corrélation cohérente entre les événements produits par des unités de calcul distinctes.

Poursuivant cette idée, Bertauld et Dagenais [45] proposent une méthodologie générique pour la corrélation des traces dans de tels environnements. Leur approche repose sur un mécanisme de synchronisation interprocessus par mémoire partagée, suivi d'un réalignement temporel basé sur l'algorithme du *convex hull*. Cette méthode permet d'obtenir une chronologie cohérente d'événements issus de multiples cœurs hétérogènes, tout en conservant une faible empreinte mémoire.

2.5 Instrumentation des FPGA

2.5.1 Instrumentation interne

Plusieurs outils commerciaux permettent l'acquisition interne de signaux dans les FPGA. AMD propose l'Integrated Logic Analyzer (ILA) [46] et Intel propose SignalTap [47]. Ces outils fonctionnent en insérant, au moment de la synthèse du design FPGA, un bloc matériel constitué de sondes connectées aux signaux internes, d'une logique de déclenchement et d'une mémoire tampon basée sur des blocs RAM internes. Les sondes échantillonnent les signaux directement dans le FPGA, pendant que la logique de déclenchement analyse en continu ces valeurs afin de détecter une condition prédéfinie. Lorsque cette condition est vraie, le système active la capture et enregistre les échantillons dans la mémoire tampon.

Une fois la capture effectuée, un contrôleur JTAG embarqué transfère la mémoire interne vers l'outil logiciel. Ce contrôleur communique avec l'ordinateur externe sans interrompre le fonctionnement du FPGA, permettant une extraction non intrusive des données. L'outil logiciel reconstruit alors les signaux sous forme d'ondes temporelles.

Hung *et al.* proposent un travail sur l'insertion incrémentale de tampons pour le débogage sur FPGA [48]. Ils démontrent qu'il est possible d'éviter les recompilations complètes nécessaires lorsque les signaux observés sont modifiés. En gardant le routage initial du circuit et en utilisant les ressources libres restantes, leur approche permet d'insérer ou de modifier des connexions de trace de façon incrémentale. Comparée aux analyseurs logiques standards, cette méthode est jusqu'à 98 fois plus rapide, réduit la longueur de routage et n'impacte presque pas le chemin critique (<1%).

2.5.2 Instrumentation multi-FPGA

Rafii *et al.* présentent Pharos [49], un outil d'analyse des performances pour les systèmes multi-FPGA. L'approche repose sur une instrumentation matérielle modulaire intégrée directement dans le tissu logique, permettant des mesures précises de latence et de débit sans dépendance vis-à-vis des protocoles réseau utilisés. Pharos permet de mesurer la latence entre les FPGA grâce à une synchronisation temporelle dérivée du PTP (*Precision Time Protocol* : un protocole de synchronisation de réseaux à haute précision), d'évaluer les délais dans les communications et de surveiller le trafic interne aux FPGA, comme le débit et la taille des paquets. Le protocole pPTP synchronise les horloges des FPGA avec une précision d'un cycle d'horloge, sans nécessiter de matériel externe de synchronisation.

Les résultats expérimentaux obtenus sur des FPGA Xilinx UltraScale+ connectés via un

réseau de 100 Gb/s montrent que Pharos maintient une précision temporelle inférieure à 10 ns pour des intervalles de resynchronisation de 10 ms, et un écart maximal de 4,5 µs pour une seconde. Le système reste précis et non intrusif, même en condition de congestion, avec une latence d'insertion inférieure à 10 ns pour un trafic réseau proche du débit maximal. Grâce à son architecture modulaire et à son indépendance protocolaire, Pharos constitue une solution de référence pour le traçage distribué et la corrélation d'événements dans les environnements multi-FPGA, en surmontant les limites des ILAs et APM traditionnels.

2.6 Enseignements de l'état de l'art

2.6.1 Synthèse des approches existantes

L'état de l'art propose de nombreuses solutions ciblées pour le traçage et l'observabilité dans les systèmes FPGA. Cependant, ces approches restent fragmentées et ne couvrent pas l'ensemble des besoins des systèmes hétérogènes modernes. Des solutions pour les systèmes hétérogènes CPU–GPU ont été proposées, mais il manque des solutions unifiées pour le traçage CPU–FPGA.

Cette lacune est problématique dans les applications où de larges conceptions RTL interagissent avec des logiciels, car les outils classiques n'offrent ni la cohérence temporelle ni la visibilité globale nécessaires pour analyser ces interactions. L'écosystème LTTng/CTF/Trace Compass ne propose actuellement aucun support natif pour le traçage des FPGA.

2.6.2 Orientations du mémoire

Il est donc nécessaire de développer des outils permettant d'intégrer le traçage FPGA dans cet environnement, afin de bénéficier des avantages d'un écosystème mature : corrélation des événements, visualisation avancée et compatibilité avec des solutions existantes. Une telle intégration ouvre la voie à l'observation unifiée de systèmes de plus en plus complexes.

Pour combler ces manques, ce mémoire introduit une infrastructure complète de traçage unifié CPU–FPGA, couvrant à la fois :

- Une architecture matérielle de sondes configurables et synchronisées ;
- Une infrastructure logicielle basée sur un pilote Linux et un post-traitement CTF.

Les Chapitres 3 et 4 présentent successivement ces deux aspects, suivis d'une évaluation expérimentale détaillée au Chapitre 5.

CHAPITRE 3 ARCHITECTURE MATÉRIELLE

3.1 Vue d'ensemble de l'architecture matérielle

Ce chapitre présente l'architecture matérielle du traceur hétérogène, ses principaux blocs fonctionnels, ainsi que les mécanismes permettant la synchronisation et le transfert des événements entre le FPGA et le processeur hôte. Cette architecture vise à effectuer la capture continue d'événements au sein du FPGA tout en permettant leur transfert en temps réel vers le système hôte. Elle constitue le cœur du dispositif de traçage et a été conçue pour être générique, modulaire et faiblement intrusive, de manière à pouvoir être intégrée à tout design matériel sans perturber son fonctionnement normal.

L'objectif principal du système est de fournir une infrastructure d'observation unifiée pour les architectures hétérogènes CPU–FPGA. Du point de vue matériel, cela implique de concevoir une chaîne de collecte d'événements capable d'opérer sous plusieurs domaines d'horloge, de gérer des débits élevés de données, et de transférer ces informations efficacement vers le CPU à travers l'interface PCIe.

L'ensemble du dispositif de traçage repose sur quatre blocs fonctionnels principaux :

1. Les sondes (*probes*), qui instrumentent les signaux ou bus à observer et encapsulent les événements détectés avec un identifiant et un horodatage local.
2. La chaîne de sondes (*probe-chain*), qui regroupe et sérialise les flux issus de multiples sondes dans un flux AXI-Stream unique.
3. Le contrôleur de traçage, qui gère la configuration des sondes et la synchronisation entre les domaines d'horloge du FPGA.
4. Le mécanisme de transfert de données, basé sur le sous-système XDMA (Xilinx DMA), qui effectue la communication bidirectionnelle entre le FPGA et le système d'exploitation hôte via un bus PCIe.

3.2 Système de traçage

La Figure 3.1 illustre une application du traceur sur un FPGA avec plusieurs modules. Un ou plusieurs modules observés (numérotés de #1 à #n) sont connectés à un interconnecteur relié au pont PCIe–AXI. Chaque module peut être observé par des sondes qui génèrent des événements, représentés sur la figure par une ligne dorée reliée à un disque également doré.

Ces événements sont collectés par une chaîne de sondes (*probe-chain*) et transmis sous forme

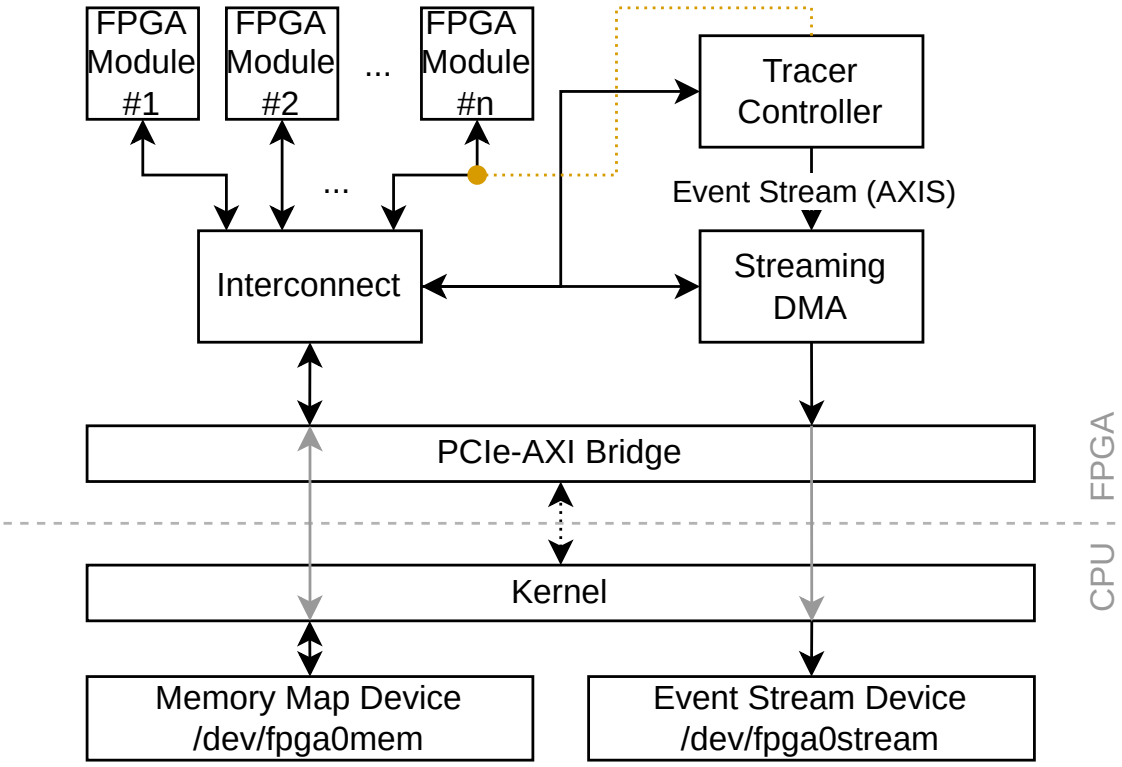

FIGURE 3.1 Vue globale d'un système avec traçage.

de flux AXI-Stream vers le module DMA de transfert. Le contrôleur de traçage effectue la configuration du système et la synchronisation entre domaines d'horloge au moyen de registres de contrôle accessibles en mémoire mappée.

Du côté hôte, le pilote Linux expose deux périphériques : un périphérique de contrôle en mémoire mappée (`/dev/fpga0mem`), qui offre à l'espace utilisateur un accès direct aux registres du contrôleur de traçage, et un périphérique de flux de données (`/dev/fpga0stream`), qui fournit le flux d'événements transféré par le DMA. Le programme en espace utilisateur interagit avec ces périphériques pour configurer les sondes, injecter des événements de synchronisation et enregistrer les traces sur disque en vue de leur post-traitement.

Le système de traçage est conçu pour fonctionner de manière totalement découplée du reste du design. Il comprend des entrées pour les signaux à inspecter, un port de sortie AXI-Stream pour le flux d'événements sortants, ainsi qu'une interface AXI-Lite esclave pour la configuration. En utilisant des protocoles standards, le système peut ainsi être intégré indépendamment du fabricant et de la plateforme, à un système qui permet l'écriture dans une interface AXI et de recevoir un flux de données AXI-Stream.

3.2.1 Sonde

La sonde est l'élément de base du système. Pour chaque événement tracé, une sonde est créée. La sonde dispose de deux modes d'enregistrement des signaux.

1. Mode de détection de changement d'état, dans lequel la sonde surveille un signal, et un événement est enregistré à chaque transition de ce signal.
2. Mode déclenché par signal externe, où l'enregistrement est initié par un signal de déclenchement externe. Ce mode est utile pour l'observation de bus, car il permet de n'enregistrer les données qu'au moment d'une poignée de main (*handshake*).

Dès qu'un événement est enregistré, celui-ci est horodaté à l'aide d'un compteur, et le numéro de la sonde est ajouté. L'événement est ainsi encapsulé dans un paquet, puis transféré vers une FIFO configurable. Cette mémoire tampon permet à la sonde d'absorber un flux continu d'événements, potentiellement à chaque cycle d'horloge, dans les limites de sa capacité maximale. Lorsque la FIFO est pleine, les événements seront perdus. Les événements sont ensuite transférés sur un bus AXI-Stream. Les paquets sont découpés en plusieurs segments de la largeur du bus. Lorsque le bus est occupé, les paquets restent en attente dans la FIFO. Un port de configuration est disponible sur chaque sonde, permettant de sélectionner les sondes souhaitées.

3.2.2 Chaîne de sondes

La chaîne de sondes fait la collecte et l'agrégation des événements issus de l'ensemble des sondes réparties dans le design. La Figure 3.2 illustre le principe de la chaîne de sondes, où une chaîne (tracée en pointillé doré) traverse un ensemble de points de sondes (disques dorés) qui traversent la fabrique FPGA.

La chaîne de sondes est inspirée des travaux de Blochwitz et Klink [33] : elle regroupe tous les flux AXI-Stream générés par les sondes dans un flux d'événements continu unique. Le flux étant un bus AXI-Stream, la chaîne est découplée du système et permettrait à toute solution acceptant ce bus de recevoir le flux de données.

La Figure 3.3 illustre les éléments de la chaîne de sondes. Pour agréger les données, la chaîne est constituée d'une série de multiplexeurs qui fusionnent les flux de données provenant de toutes les sondes. Chaque sonde émet un flux d'événements local, horodaté et encapsulé, transmis à une hiérarchie de combinateurs AXI-Stream. Ces combinateurs (ou multiplexeurs) fusionnent progressivement plusieurs flux en un seul : chaque niveau de la hiérarchie réduit le nombre d'entrées jusqu'à obtenir un flux global unique, qui sera acheminé de manière

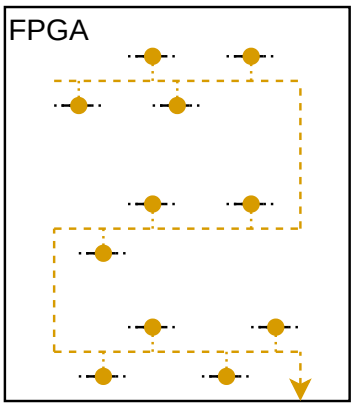

FIGURE 3.2 Chaîne de sondes

ordonnée et continue vers le DMA.

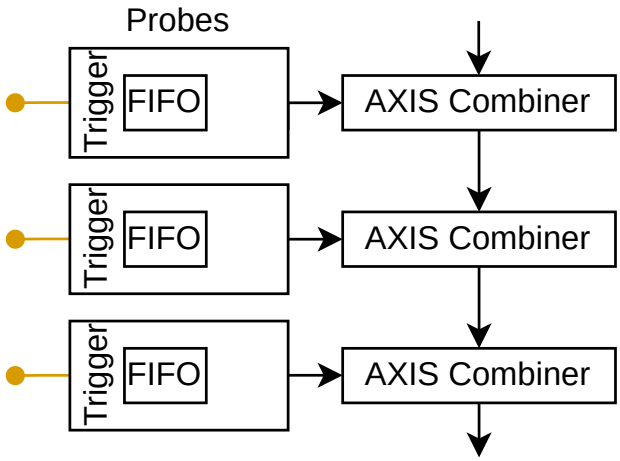

FIGURE 3.3 Chaîne de sondes détaillée

3.2.3 Contrôleurs et registres

Les contrôleurs et registres de configuration sont l'interface entre la partie matérielle du traceur et le logiciel exécuté sur l'hôte. Ils permettent de configurer dynamiquement le système de traçage et de faire la synchronisation entre les événements matériels et logiciels. Deux registres sont dédiés à la gestion des sondes : ils permettent d'activer ou de désactiver sélectivement les sondes intégrées dans le design, permettant de filtrer les signaux pertinents pour une session d'observation.

Deux autres registres sont associés à la synchronisation temporelle. Le premier permet d'in-

sérer, à la demande, un événement de synchronisation dans le flux de traçage pour corriger le décalage entre les horloges du FPGA et celles du processeur hôte. Le second sert à appliquer une compensation temporelle pour les délais associés au bus. Cette correction permet un alignement des événements lors du post-traitement et de la fusion des traces. Le mécanisme de synchronisation sera discuté plus en détail dans la Section 3.4, portant sur le traçage en présence de multiples domaines d'horloge.

3.3 Transfert de données

Le transfert de données est découplé du système de traçage. Dans le cadre du projet, la carte FPGA utilisée est la AMD Xilinx Alveo U280, qui se connecte à l'ordinateur hôte via PCIe. AMD/Xilinx propose plusieurs solutions pour communiquer avec l'hôte en PCIe. La première est XRT (Xilinx Runtime library), qui fournit un environnement matériel et logiciel pour faciliter le développement FPGA. Cette solution a été écartée, car son système d'abstraction présente des limitations pour les architectures sur mesure et n'était pas applicable à une solution bas niveau pour le traçage de sous-systèmes.

L'autre option choisie est de concevoir une solution native, avec un bitstream entier généré par Vivado. Plusieurs IPs sont disponibles pour effectuer le transfert des données :

- Le premier est le bloc PCIe intégré (*UltraScale+ Device Integrated Block for PCI Express*), qui est le plus flexible et permet un accès direct au protocole PCIe.
- Le XDMA (*DMA/Bridge Subsystem for PCI Express*), qui fournit des interfaces AXI et propose une version avec ou sans DMA intégré.
- Le QDMA (*QDMA Subsystem for PCI Express*), qui propose un DMA à haute performance avec un système de files d'attente multiples.

La solution avec PCIe intégré a été écartée, car le développement d'une solution sur mesure était hors du cadre de la recherche. Le QDMA a également été écarté en raison de sa complexité, qui n'était pas nécessaire pour démontrer le bon fonctionnement de la solution. Les sous-sections suivantes présentent deux approches explorées pour l'échange de données entre le FPGA et l'hôte, selon le mode d'exploitation choisi : le XDMA.

3.3.1 Modes de fonctionnement du XDMA

Le transfert de données entre le FPGA et l'hôte est assuré par le sous-système XDMA (*DMA/Bridge Subsystem for PCI Express*), qui établit la liaison entre le protocole PCIe et les interfaces AXI utilisées dans la conception. Ce module peut fonctionner selon deux modes

principaux : le mode DMA et le mode Bridge. Le mode DMA se décline lui-même en deux sous-modes : le mode *flux de données* (AXI-Stream) et le mode *mémoire mappée* (AXI-MM).

Dans le mode DMA, un pilote Linux fourni par AMD prend en charge la gestion complète des transferts de données. En mode AXI-Stream, le pilote crée des périphériques de caractères (`CDEV`) permettant à l'utilisateur de lire et d'écrire directement dans ces périphériques ; chaque opération correspond à un transfert sur les ports AXI-Stream d'entrée ou de sortie.

En mode AXI-MM, l'utilisateur interagit avec des plages d'adresses mappées en mémoire. Les opérations de lecture et d'écriture effectuées sur ces adresses sont traduites par le pilote en transactions AXI, puis transférées via le DMA. Le pilote orchestre ainsi les échanges entre les mémoires tampons de l'hôte et les modules internes du FPGA.

Le mode Bridge offre un fonctionnement plus bas niveau, sans dépendre d'un pilote fourni. Dans ce mode, le module expose deux interfaces : un port maître et un port esclave. Le port maître permet d'effectuer des transferts de données par accès mémoire (*Memory-mapped I/O*, MMIO), typiquement des écritures ponctuelles de 32 ou 64 bits.

Le port esclave donne quant à lui un accès direct à des zones de la mémoire hôte, configurées préalablement pour permettre des lectures et écritures à partir du FPGA. Ce mode est adapté à la conception de solutions sur mesure, où le contrôle du protocole et de la gestion mémoire est fait entièrement par la logique utilisateur.

3.3.2 Mémoire FPGA et scrutation

La première approche utilise le XDMA en mode DMA et mémoire mappée. Le mode mémoire mappée est choisi car le XDMA ne propose pas de solution hybride entre mémoire mappée et flux de données. Ainsi, comme il était nécessaire d'avoir accès à la mémoire en mode mappé, le mode flux de données a été écarté.

Pour l'utilisation de ce mode, un module de mémoire tampon, le *AXI Trace Buffer*, a été conçu. Ce module a une entrée AXI-Stream ainsi qu'un port esclave AXI-MM. Une mémoire tampon circulaire de taille prédéfinie est intégrée, transférant continuellement les données du flux vers celle-ci. Les informations de remplissage sont disponibles par des registres accessibles à l'utilisateur. Lorsqu'il y a des données disponibles, l'utilisateur effectue une lecture sur le CDEV correspondant, ce qui se traduit par une transaction de lecture dans le FPGA. Le pilote et le DMA interne gèrent ensuite le transfert de données. Une fois les données lues, l'utilisateur informe le module, et la mémoire est libérée.

3.3.3 DMA et interruptions

La deuxième approche utilise le XDMA sans le DMA intégré, également appelé *Bridge*. Un driver a été conçu pour utiliser ce mode. La mémoire MMIO est accessible à l'utilisateur en mémoire mappée, permettant la lecture et l'écriture des registres matériels.

Dans cette configuration, un module DMA en flux de données a été conçu afin d'envoyer continuellement les données issues de l'entrée AXI-Stream vers l'interface AXI maître, donnant ainsi accès à la mémoire de l'hôte. À intervalles réguliers, lorsque des données sont disponibles, le DMA déclenche une interruption pour informer le noyau de la présence de nouvelles données. Le noyau traite ensuite ces données, puis informe le DMA que les données ont été consommées, permettant la reprise du transfert.

3.4 Traçage avec multiples domaines d'horloge

Les conceptions FPGA modernes comportent souvent plusieurs domaines d'horloge, introduisant une complexité supplémentaire lors du traçage. Dans le système développé, les sondes peuvent être instanciées dans différents domaines d'horloge afin de tracer des événements issus de régions hétérogènes du FPGA. Chaque sonde fonctionne dans le domaine d'horloge du signal observé et horodate localement les événements capturés. Ces horodatages préservent l'ordre temporel à l'intérieur de chaque domaine, mais posent un défi pour l'alignement des événements entre domaines lors du post-traitement.

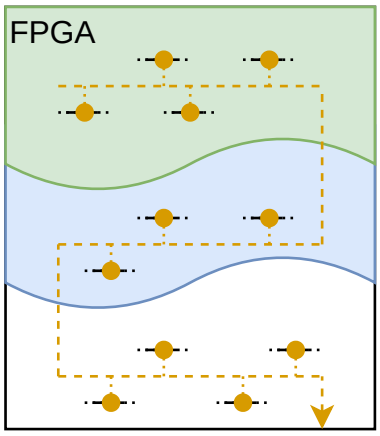

FIGURE 3.4 Chaîne de sondes

La Figure 3.4 illustre le passage de la chaîne de sondes à travers trois domaines d'horloge. La chaîne peut passer à travers un nombre configurable de domaines d'horloge. Chaque sonde est placée dans le domaine d'horloge correspondant au signal qu'elle observe, ce qui permet la

capture d'événements au rythme local du sous-système instrumenté. Les sondes d'un même domaine sont regroupées afin de minimiser le nombre de transitions d'horloge dans la chaîne de sondes pour réduire la latence.

3.4.1 Traversée des domaines d'horloge

Lorsque le flux d'événements doit passer d'un domaine d'horloge à un autre, la chaîne de sondes intègre automatiquement des FIFO de traversée de domaine d'horloge (*clock-domain crossing*, CDC), tel qu'illustré à la Figure 3.5. Les FIFO de traversée de domaine d'horloge permettent à l'information de passer entre les différences de phase et de fréquence entre les domaines.

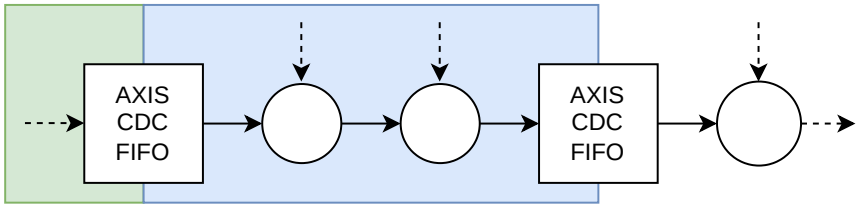

FIGURE 3.5 Changement d'horloge dans la chaîne de sondes

Les FIFO CDC sont insérées uniquement lorsqu'un changement d'horloge est détecté, pour minimiser l'utilisation des ressources FPGA.

Les horodatages générés par chaque sonde voyagent avec les événements tout au long de la chaîne. Ainsi, même après leur sérialisation dans un flux unique, l'ordre relatif des événements au sein de chaque domaine d'horloge est conservé, ce qui simplifie le réalignement temporel qui sera effectué en post-traitement.

3.4.2 Synchronisation multi-horloge

Pour faire la correspondance temporelle entre domaines d'horloge et permettre la fusion des traces matérielles et logicielles, un synchroniseur dédié est ajouté à chaque domaine. Comme le montre la Figure 3.6, tous les synchroniseurs sont associés à un registre de synchronisation accessible depuis le processeur hôte. Lorsqu'une écriture est effectuée sur ce registre, l'opération est horodatée en parallèle par l'horloge du CPU et par le compteur local de chaque domaine FPGA. Ces paires d'horodatages, appelées balises de synchronisation, servent ensuite de points d'ancrage pour estimer les décalages et dérives entre les horloges.

Durant le post-traitement, ces balises permettent de convertir les horodatages FPGA dans la même référence temporelle que celle du CPU. Cette approche rend possible une analyse

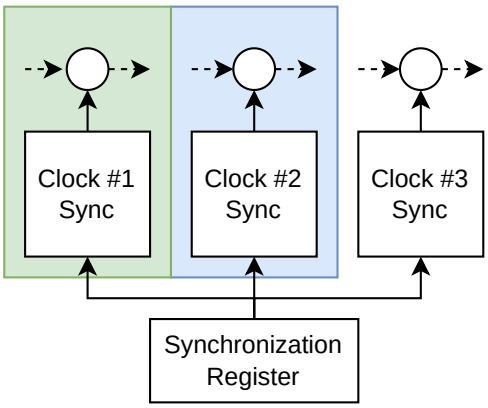

FIGURE 3.6 Synchronisation multi-horloge

unifiée dans les outils comme Trace Compass, ce qui permet une visibilité complète sur les interactions entre le processeur hôte et la logique FPGA, même en présence de multiples horloges internes.

3.5 Processus de conception

Le système de traçage est généré automatiquement à l'aide de Chisel, un langage de description matérielle intégré à Scala, permettant la création de générateurs matériels paramétrables et réutilisables [50, 51]. La Figure 3.7 illustre le processus complet de génération.

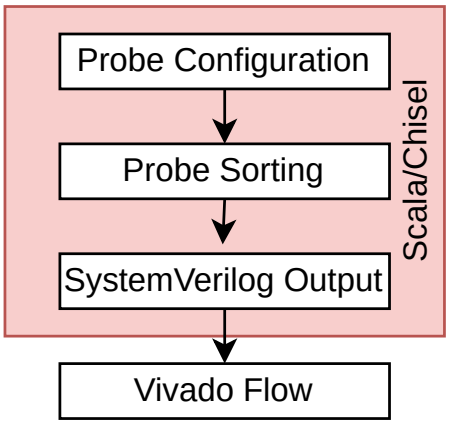

FIGURE 3.7 Flow de génération matérielle basé sur Chisel.

Le processus débute par un fichier de configuration des sondes, dans lequel sont définis les signaux à tracer, leurs domaines d'horloge respectifs et les conditions de déclenchement associées. Les sondes sont ensuite triées et regroupées par domaine d'horloge afin de réduire le nombre de traversées inter-domaines nécessaires dans la conception. Ce tri optimise la

topologie de la chaîne de sondes en minimisant l'insertion de FIFO de traversée de domaine d'horloge, ce qui diminue à la fois la latence et l'utilisation des ressources FPGA. À partir de cette description, Chisel génère automatiquement l'ensemble du système de traçage matériel, comprenant la chaîne de sondes, le contrôleur, les modules de synchronisation et les interfaces AXI requises. Le code RTL produit est émis en SystemVerilog, puis intégré dans le processus de conception standard de Vivado pour la synthèse, le placement et le routage.

Cette méthode de génération présente plusieurs avantages :

- Les sondes, domaines d'horloge, profondeurs de FIFO et identifiants peuvent être ajustés à la volée à partir du fichier de configuration ;
- Les modules générés sont indépendants de la conception observée et peuvent être intégrés à d'autres projets de traçage ou de surveillance ;
- La génération automatique garantit la cohérence des connexions AXI et des registres de configuration ;
- Le code produit est directement compatible avec Vivado et peut être intégré à d'autres outils de simulation ou de synthèse (par exemple Model Composer ou Vitis HLS).

Ce processus de conception automatisé a permis d'intégrer rapidement le traceur à différents projets FPGA et de générer plusieurs variantes matérielles destinées à l'évaluation expérimentale présentée au Chapitre 5.

CHAPITRE 4 ARCHITECTURE LOGICIELLE

4.1 Vue d'ensemble de l'architecture logicielle

Ce chapitre décrit l'architecture logicielle du traceur hétérogène, qui fait la configuration matérielle, la collecte des événements, leur conversion en trace conforme au format CTF, et leur visualisation par des outils d'analyse existants. Le dispositif logiciel fait le lien entre le FPGA, le système d'exploitation hôte, le post-traitement des données et leur analyse graphique.

L'architecture a été conçue selon trois objectifs principaux :

1. Offrir un contrôle du matériel via des interfaces standard du noyau Linux ;
2. Permettre un transfert en continu et sans perte des événements issus du FPGA ;
3. Produire des traces exploitables par les outils d'analyse CTF tels que Trace Compass.

Le système logiciel repose sur quatre composants principaux :

1. Le pilote noyau Linux, qui gère la communication PCIe avec le FPGA. Il expose deux périphériques de caractères : `/dev/fpga0mem` (pour la configuration par registre mappé) et `/dev/fpga0stream` (pour le flux d'événements transféré par le DMA ou le mode bridge).
2. Le processus utilisateur, qui configure les sondes, synchronise les horloges FPGA–CPU et capture le flux d'événements depuis `/dev/fpga0stream`. Il produit un fichier brut structuré en paquets.
3. Le module de post-traitement, qui transforme le fichier brut en trace CTF, fait le réordonnancement temporel des événements multi-domaines, et fusionne les données issues des sondes matérielles avec des traces logicielles si nécessaire.
4. Les outils d'analyse, tels que Babeltrace et Trace Compass, permettent d'afficher, d'explorer et d'interpréter les traces CTF générées.

La Figure 4.1 illustre l'architecture logicielle complète du système de traçage. Elle est structurée en cinq couches successives. La première, située sur le FPGA, regroupe les sondes matérielles, la chaîne de collecte AXI-Stream et le module DMA chargé du transfert des événements vers le processeur hôte via PCIe, tels que présentés au chapitre précédent. La seconde couche correspond au pilote noyau Linux, qui fournit deux interfaces vers l'espace utilisateur : un périphérique mémoire mappée pour la configuration des registres matériels et un périphérique pour le flux brut d'événements. La troisième couche, dans l'espace utilisa-

teur, orchestre la capture des événements, leur synchronisation temporelle avec le processeur et leur sauvegarde sous forme de fichier brut. Vient ensuite la couche de post-traitement, où les événements sont réordonnés et transformés au format CTF. Enfin, la couche d'analyse permet une vue interactive des événements à l'aide des outils `babeltrace` et Trace Compass pour une inspection unifiée des activités CPU–FPGA.

La Figure 4.1 distingue également les interactions entre ces différentes couches : les flux de données (flèches pleines) et les flux de contrôle (flèches en pointillés), qui traversent l'ensemble de la chaîne logicielle de traçage.

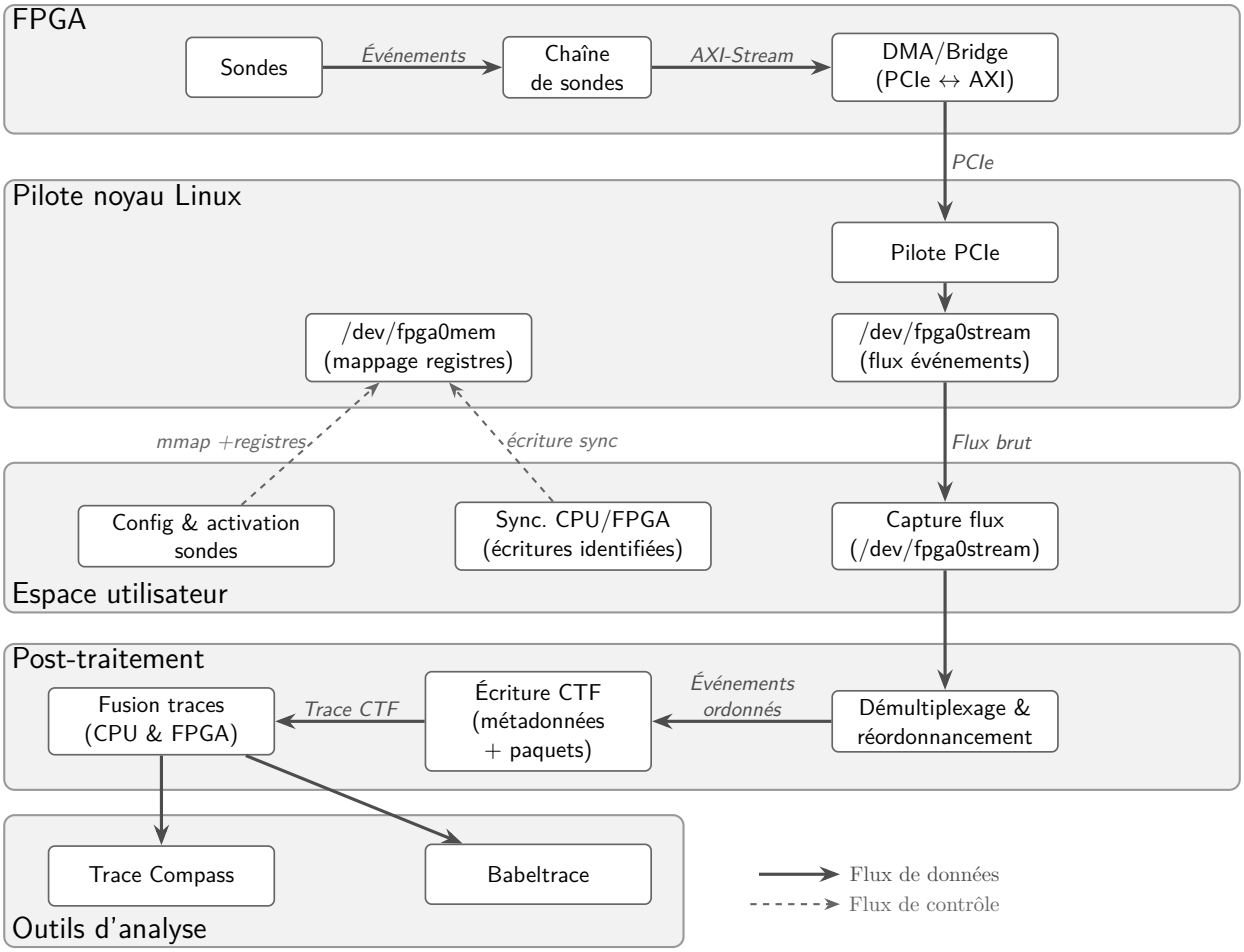

FIGURE 4.1 Architecture logicielle du système de traçage.

4.2 Pilote noyau Linux

Comme mentionné dans la section précédente, le pilote noyau Linux constitue la seconde couche de l'architecture logicielle. Il fait la communication entre le FPGA et l'espace utilisa-

teur via l'interface PCIe.

Dans l'implémentation retenue, le XDMA en mode *Bridge*, aucun pilote Linux n'est fourni, car l'implémentation est sur mesure. Il a donc été nécessaire de concevoir un pilote sur mesure. Ce pilote assure trois fonctions essentielles :

- Initialisation de l'interface PCIe et la gestion des espaces d'adressage ;
- Exposition de deux périphériques de caractères (CDEV) pour la configuration et le transfert de données ;
- Gestion des interruptions et du flux d'événements via PCIe.

Le pilote respecte les bonnes pratiques de gestion de la mémoire : toutes les ressources allouées au cours de l'initialisation ou de l'exécution sont correctement libérées, y compris en cas d'erreur ou lors du retrait du module. Les références utilisées sont la documentation du noyau Linux [52], le guide LDD3 [53] et le guide *The Linux Kernel Module Programming Guide* [54].

4.2.1 Configuration PCIe

La première section du pilote est la configuration PCIe. Pour ce faire, certaines fonctions et macros du noyau sont disponibles pour la configuration des périphériques. L'architecture des pilotes PCIe du noyau Linux est un peu différente de celle des autres pilotes. Le pilote va d'abord identifier les identifiants PCIe qu'il souhaite se voir attribuer. Dans ce cas, c'est l'identifiant qui a été assigné dans le XDMA lors de la configuration de l'IP. Ensuite, un objet est enregistré dans le noyau, ce qui va lui permettre d'appeler une fonction d'initialisation du pilote pour chaque périphérique faisant partie de la liste.

La fonction d'initialisation va d'abord identifier les zones d'adresse mémoire que le périphérique a été mappé lors de l'initialisation du système. Dans ce cas, la seule zone mémoire est mappée à un pointeur dans la mémoire virtuelle du noyau. Le périphérique est ensuite autorisé à prendre le contrôle du bus PCIe, ce qui va lui permettre d'effectuer des opérations directement sur la mémoire par la suite.

4.2.2 Interface avec l'espace utilisateur

Une fois le périphérique PCIe initialisé, le pilote expose deux périphériques de caractères qui permettent à l'espace utilisateur d'interagir directement avec le matériel.

Le premier périphérique, nommé `/dev/fpga0mem`, est destiné à la configuration des registres via une mémoire mappée. Il implémente l'appel système `mmap`, permettant aux processus

utilisateurs de mapper la mémoire PCIe directement dans leur propre espace d'adressage virtuel. Cette approche réduit la latence des accès, car elle évite les appels système (`ioctl`, `write`, etc.).

Le second périphérique, `/dev/fpga0stream`, expose un flux continu d'événements émis par le FPGA. Il s'appuie sur une zone mémoire contiguë allouée en amont, dans laquelle le module PCIe effectue les transferts DMA entrants. Cette mémoire est ensuite lue par l'utilisateur via des appels système de lecture.

Le système d'adressage virtuel permet ainsi de mapper l'espace mémoire PCIe vers l'espace mémoire du programme. Cela permet de réduire la latence, car l'utilisateur n'a pas besoin de faire appel à des appels système pour interagir avec le FPGA.

Cet espace de mémoire mappée permet aux processus de l'espace utilisateur un accès direct aux registres du contrôleur de traçage. Il permet aussi un accès à tout autre module connecté à l'interconnexion, par exemple aux applications de démonstration.

4.2.3 Gestion des interruptions et du flux de données

Le CDEV `/dev/fpga0stream` permet à l'espace utilisateur de recevoir le flux de données : il alloue un espace mémoire contigu qui va permettre au module PCIe d'effectuer des transferts de données. Les informations sur l'espace mémoire, telles que l'adresse et la taille, sont écrites dans les registres du module DMA pour lui permettre d'écrire au bon endroit de la mémoire de l'hôte pendant les transferts.

Le CDEV implémente alors les opérations de lecture. L'opération de lecture va utiliser les pointeurs de tête (*head*) et de queue (*tail*) pour copier la mémoire dans l'espace utilisateur. Quand aucune information n'est disponible, le pilote va alors mettre le processus en attente, dans une file prévue à cet effet.

La Figure 4.2 illustre les interactions entre le FPGA et le pilote. Le flux de données (*Stream*) est copié dans une petite mémoire tampon sur le FPGA. Cette mémoire est ensuite transférée vers la mémoire de l'hôte. Après avoir copié la mémoire, une interruption (IRQ) est émise. Les données sont alors accessibles via le CDEV, et le pilote interagit avec les registres pour indiquer l'espace libéré. Ce processus se répète de manière continue pendant le traçage.

Le transfert de données s'effectue de manière asynchrone à l'aide d'interruptions MSI (*Message Signaled Interrupts*), qui est une méthode plus moderne pour les interruptions PCIe, et qui permet d'éviter les états de concurrence (*race conditions*). Lorsqu'une interruption est reçue, le pilote effectue une lecture dans les registres du DMA pour obtenir le dernier pointeur de tête de l'espace mémoire alloué au DMA. Celui-ci va ensuite réveiller le dernier

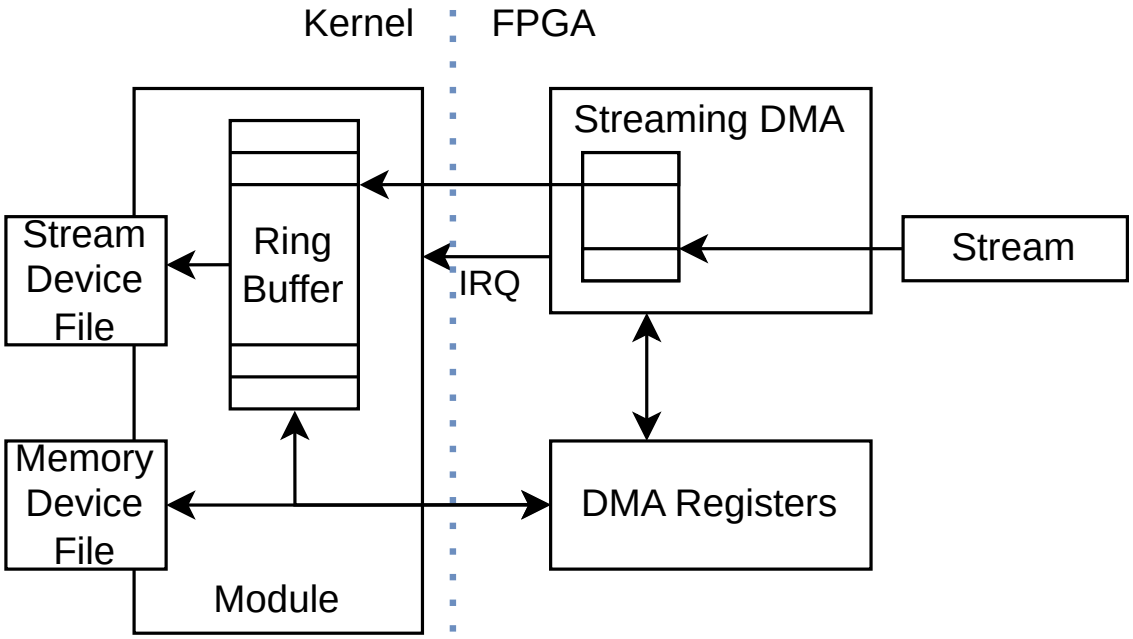

FIGURE 4.2 Architecture du DMA

processus dans la file d'attente, ce qui a pour effet de compléter la lecture et de retourner à l'espace utilisateur.

4.3 Logiciel de traçage

Dans l'implémentation basée sur le DMA, le logiciel de traçage assure deux fonctions essentielles : la sauvegarde continue du flux d'événements sur disque et la gestion de la synchronisation temporelle entre le FPGA et le processeur. Puisque le pilote noyau fait les transferts des données brutes, la logique utilisateur fait l'orchestration du flux et le traitement des métadonnées, ce qui simplifie sa conception.

Le logiciel utilise plusieurs fils d'exécution, structurés autour de trois tâches exécutées en parallèle :

- Capture du flux de données : un premier fil d'exécution lit en continu depuis le périphérique `/dev/fpga0stream` et écrit les données dans un fichier temporaire. Lorsqu'aucune donnée n'est disponible, le fil se met en attente via l'appel système `poll`, ce qui évite l'occupation CPU inutile et permet d'arrêter le processus en fin de traçage.
- Synchronisation temporelle : un second fil est chargé d'émettre régulièrement des événements de synchronisation en écrivant des identifiants uniques dans les registres matériels dédiés. Ces événements sont récupérés dans le flux d'événements et consignés dans un

fichier séparé, en vue de l'alignement temporel FPGA–CPU lors du post-traitement.

- Gestion du programme : un troisième fil permet d'activer ou de désactiver le traçage pour les sessions ou à des fins de mesure de performance.

4.4 Trace CTF

Pour assurer l'interopérabilité des données collectées et permettre une analyse avec des outils disponibles, le traçage logiciel repose sur le format *Common Trace Format* (CTF), utilisé avec LTTng pour capturer les événements matériels. Le choix du CTF se justifie par sa conception modulaire : le format sépare explicitement les *métadonnées*, qui décrivent la structure et le type des données capturées, du flux binaire brut (*packet data*). Dans notre cas, les paquets produits contiennent pour chaque événement un identifiant de 32 bits, un horodatage de 64 bits, et, le cas échéant, des données spécifiques propres à la sonde.

Cette organisation rend le CTF adapté à notre architecture matérielle. Les événements générés par les sondes FPGA sont déjà formatés de manière proche du modèle CTF : un flot séquentiel d'échantillons horodatés. Cependant, il existe une contrainte : bien que les flux de chaque sonde soient individuellement ordonnés, l'agrégation de flux provenant de sondes différentes ne garantit pas l'ordre global des événements. Or, les spécifications du format CTF imposent une stricte monotonie temporelle dans chaque fichier de trace. Pour répondre à cette exigence, les événements sont préalablement démultiplexés par type (ou par sonde) dans des flux séparés.

Une fois les événements réorganisés, un fichier distinct contenant les métadonnées est généré pour décrire la structure de chaque type d'événement, ainsi que la configuration globale de la trace. Cette étape rend le fichier pleinement compatible avec les outils d'analyse CTF comme `babeltrace` ou Trace Compass, qui peuvent alors interpréter la trace FPGA sans traitement complémentaire.

Le Listing 4.1 présente un exemple complet d'un fichier de métadonnées généré à partir d'une trace FPGA. Il contient notamment :

- Un en-tête global conforme à la version 1.8 du CTF ;
- Une définition de l'horloge matérielle (`fpga_clock`) opérant à 1 GHz ;
- La structure d'un flux de données (`stream`) accompagné de son en-tête ;
- La définition d'un événement (ici, `timesync0`) incluant son identifiant binaire, son nom symbolique, et la structure de ses champs.

Ces métadonnées, associées aux fichiers binaires ordonnés, forment ainsi une trace CTF complète, exploitable directement dans Trace Compass avec la possibilité d'étendre l'analyse

(par exemple, en superposant des métriques logicielles ou des états dérivés).

Listing 4.1 Métadonnées CTF

```
/* CTF 1.8 */

trace {
    major = 1;
    minor = 8;
    byte_order = le;
    packet.header := struct {
        integer { size = 32; align = 8; } magic;
        integer { size = 32; align = 8; } stream_id;
    };
};

clock {
    name = fpga_clock;
    freq = 1000000000;
    offset = 1759006635201228561;
};

typealias integer {
    size = 64;
    map = clock.fpga_clock.value;
    align = 8;
} := fpga_clock_int_t;

stream {
    id = 0;
    event.header := struct {
        integer { size = 32; align = 8; } id;
        fpga_clock_int_t timestamp;
    };
};

event {
    id = 0;
    name = "timesync0";
    stream_id = 0;
    fields := struct {
        integer { size = 32; align = 4; } id;
    };
};
```

4.5 Synchronisation

Une étape du traçage est l'alignement temporel des événements provenant du FPGA avec ceux enregistrés par le processeur hôte. Pour réaliser cette synchronisation, des événements spéciaux sont insérés dans le flux de données au cours de l'exécution.

4.5.1 Principe général

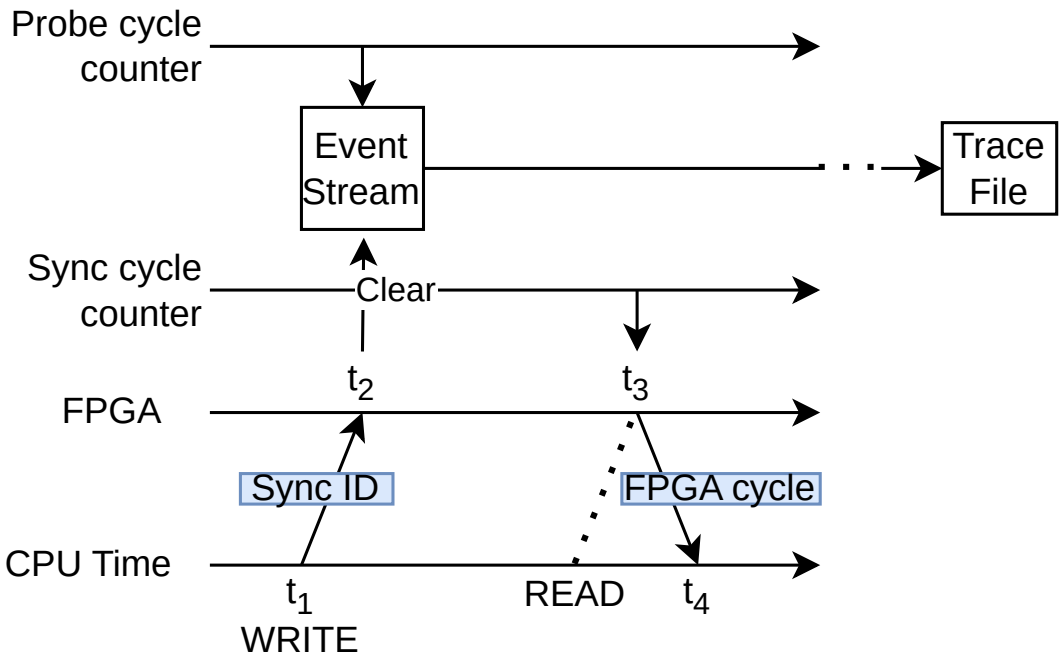

FIGURE 4.3 Synchronisation des temps CPU et FPGA

La Figure 4.3 illustre le déroulement en temps réel de ce processus. La synchronisation se fait en insérant des balises temporelles communes dans les deux domaines d'horloge. Le principe repose sur un aller-retour minimal entre l'hôte et le FPGA, comparable à une mesure de type NTP (Network Time Protocol).

1. À l'instant t_1, le logiciel utilisateur relève l'horloge du processeur et écrit un identifiant unique dans le registre de synchronisation du FPGA.
2. Ce registre est immédiatement intercepté par une sonde dans le FPGA, qui génère un événement horodaté au temps local t_2, tout en réinitialisant un compteur matériel.
3. Peu après, le logiciel interroge un registre pour lire la valeur du compteur qui contient la durée écoulée depuis la réinitialisation.
4. Enfin, le processeur mesure à nouveau son horloge à l'instant t_4 lorsqu'il reçoit cette réponse.

Ces quatre horodatages permettent d'estimer le délai aller-retour de la requête entre les domaines temporels du CPU et du FPGA. Un point de synchronisation commun peut alors être établi, servant de référence pour convertir les horodatages FPGA dans le temps processeur, en appliquant une interpolation linéaire entre les deux points les plus proches.

4.5.2 Méthodologie de synchronisation

À partir des données sauvegardées, on obtient t_1, t_2, t_3 et t_4. Le délai de transmission d'un aller-retour est donné par :

$$\delta = (t_4 - t_1) - (t_3 - t_2) \tag{4.1}$$

En supposant que le délai de transmission est symétrique, soit $t_2 - t_1 = t_4 - t_3$, le demi-aller-retour vaut $\delta/2$. On en déduit alors le temps réel auquel l'instruction a été reçue dans le domaine FPGA :

$$t_{\text{ref}} = t_1 + \frac{\delta}{2} \tag{4.2}$$

Le couple (t_{ref}, t_2) constitue un point de référence associant le temps CPU au temps FPGA. Cette opération est répétée périodiquement tout au long de la session de traçage, produisant une série de paires temporelles permettant de corriger la dérive des horloges.

En post-traitement, chaque horodatage FPGA est converti en temps CPU par interpolation linéaire entre les deux points de référence les plus proches. La formule utilisée est la suivante :

$$y = y_0 + (x - x_0)\frac{y_1 - y_0}{x_1 - x_0} \tag{4.3}$$

où (x_0, y_0) et (x_1, y_1) sont les paires formées des temps FPGA x et temps CPU de référence y les plus proches de l'événement considéré. L'application de cette interpolation permet d'obtenir un alignement temporel cohérent entre événements FPGA et CPU.

4.5.3 Multi-horloges

Pour effectuer la synchronisation entre plusieurs domaines d'horloge dans le FPGA et le processeur, le processus est dupliqué. Lorsque le FPGA reçoit la requête de synchronisation à l'instant (t_2), un événement d'horodatage est émis pour chaque domaine d'horloge et inséré dans le flux d'événements.

Lors du post-traitement, l'approche reste identique et est appliquée à chaque domaine d'horloge. Pour chacun d'eux, on obtient une liste de points associant le temps processeur au temps du domaine d'horloge correspondant. Chaque événement est ensuite synchronisé par interpolation avec son domaine d'horloge respectif.

4.6 Chaîne de traitement des données

4.6.1 Pipeline des données du FPGA jusqu'à Trace Compass

La Figure 4.4 illustre le cheminement des données du FPGA jusqu'à une vue Trace Compass. La première partie des opérations s'effectue pendant le traçage, et la seconde en post-traitement.

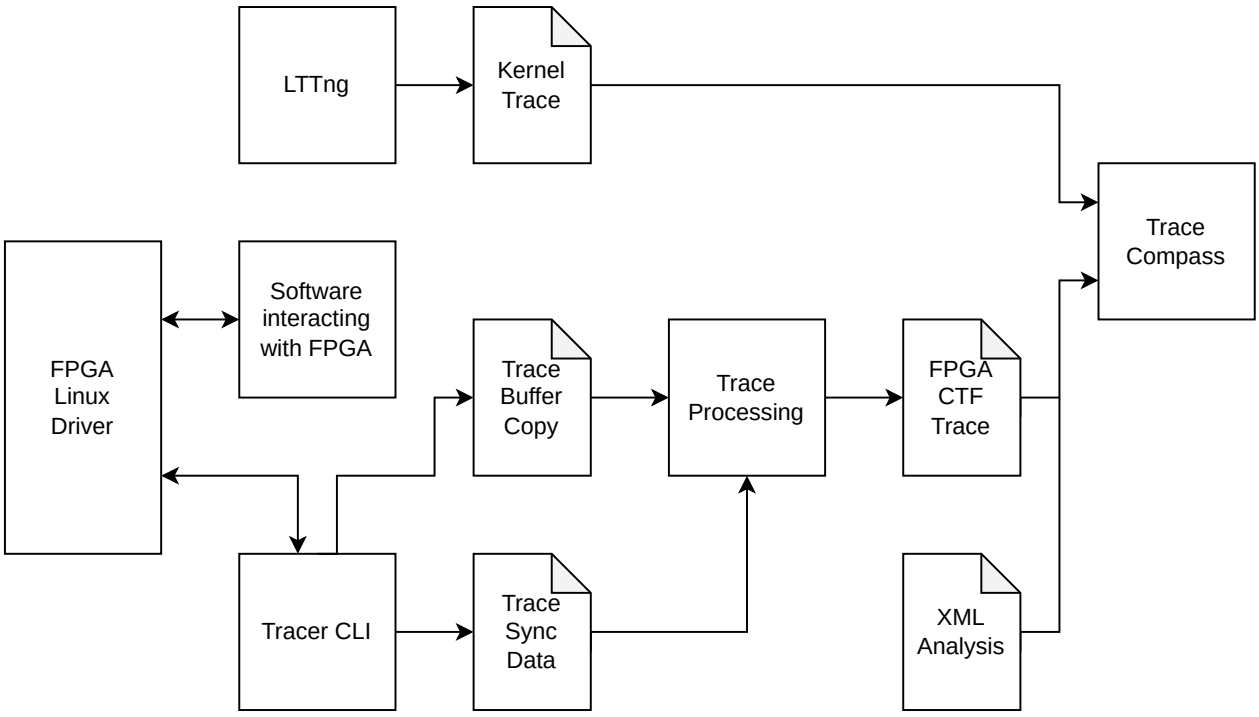

FIGURE 4.4 Pipeline des données du FPGA jusqu'à Trace Compass

Pendant le traçage, le logiciel de traçage(*Tracer CLI*) copie le flux de données dans un fichier (*Trace Buffer Copy*), tout en générant les événements de synchronisation dans un autre fichier temporaire (*Trace Sync Data*). Pendant ce temps, LTTng peut être utilisé pour tracer le logiciel et générer la trace noyau (*Kernel Trace*). De plus, le logiciel inspecté peut être en cours d'exécution.

À la fin d'une session de traçage, on obtient la trace LTTng, la trace brute du FPGA ainsi que les données de synchronisation. Après le post-traitement, on obtient une trace CTF à partir des données du FPGA (*FPGA CTF Trace*). Finalement, une analyse XML peut être utilisée pour permettre la visualisation de la trace (*XML Analysis*). En important toutes les traces générées, une vue globale est obtenue.

4.6.2 Visualisation et Trace Compass

Une fois la trace au format CTF générée, elle peut être chargée dans des outils compatibles tels que `babeltrace` ou Trace Compass. La Figure 4.5 montre l'affichage tabulaire d'une trace dans Trace Compass, où chaque événement est listé avec son horodatage, son nom, et ses données binaires décodées. Outre cette présentation en tableau, Trace Compass offre des vues temporelles interactives pour analyser la densité des événements, inspecter leur chronologie, ou naviguer dans de larges volumes de données à l'aide de filtres et de signets.

	Timestamp	Channel	CPU	Event type	Contents
	<srch>	<srch>	<srch>	<srch>	<srch>
	10:42:01.184 168 375	stream6		axi1_read_start	addr=48
	10:42:01.184 168 390	stream7		axi1_read_end	addr=48

FIGURE 4.5 Tableau des événements dans Trace Compass

Cependant, dans leur structure brute, les événements ne définissent souvent que des instants ponctuels, sans préciser explicitement la relation temporelle ou logique entre eux. C'est typiquement le cas dans le traçage d'un bus AXI, où une transaction d'écriture se compose d'un ensemble d'événements émis à différents moments (adresse, données, réponse, etc.).

Pour résoudre cela, Trace Compass propose une infrastructure de modélisation basée sur un langage XML (*Extensible Markup Language*). Celui-ci permet de définir des automates d'états, interprétant la séquence des événements pour reconstruire des transactions ou phases d'activités. Par exemple, un événement d'adresse valide (`AWVALID`) peut représenter l'entrée dans un état `écriture`, tandis que la réception d'un événement de fin de transaction (`BVALID`)

marque la sortie de cet état. Ces états sont ensuite représentés graphiquement sous forme de segments colorés sur une ligne du temps, comme le montre la Figure 4.6. Ce mécanisme permet une analyse visuelle des opérations non ponctuelles du matériel.

FIGURE 4.6 Vue FPGA dans Trace Compass

CHAPITRE 5 ÉVALUATION DES PERFORMANCES ET DISCUSSIONS

Ce chapitre présente les résultats expérimentaux obtenus avec l'infrastructure de traçage proposée. Les mesures visent à évaluer l'utilisation des ressources FPGA, la performance du transfert DMA, la capacité de la chaîne de sondes à supporter des débits élevés d'événements, ainsi que la capacité du système à corréler des événements hétérogènes entre le logiciel et le FPGA dans un contexte multi-domaines d'horloge. Toutes les expérimentations ont été réalisées sur une carte AMD Alveo U280 avec un hôte x86 (Intel i7-10700KF) sous Ubuntu 24.04. Le processeur hôte est relié au FPGA via un lien PCIe 3.0 $\times 1$.

5.1 Utilisation des ressources FPGA

L'utilisation des ressources FPGA a été évaluée en fonction du nombre de sondes instrumentées et de la profondeur des FIFO (*First-In, First-Out*). Le Tableau 5.1 illustre l'occupation en LUT et en bascules (FF) pour différentes configurations.

Tableau 5.1 Utilisation des ressources FPGA en fonction du nombre de sondes et de la profondeur des FIFO.

# Sondes	Profondeur de la FIFO par sonde					
	4		8		16	
	LUT	FF	LUT	FF	LUT	FF
20	5 774	22 444	8 061	31 020	12 452	48 128
50	12 364	49 656	17 745	69 932	28 132	110 380
100	23 317	95 008	33 827	134 784	54 222	214 132

Les mesures rapportées ont été obtenues à partir d'une chaîne de sondes de 64 bits, alimentée par une mémoire tampon dédiée au DMA configurée à 32 kbits. Chaque événement généré occupe 128 bits, répartis en trois champs : un horodatage de 64 bits, un identifiant de sonde de 32 bits, et un champ de données de 32 bits, comme illustré à la Figure 5.1.

Figure 5.1 Structure d'un événement FPGA de 128 bits composé d'un horodatage, d'un identifiant de sonde et de données associées.

Comme attendu, l'utilisation des ressources croît approximativement de façon linéaire avec le nombre de sondes, chaque sonde ajoutant sa propre logique de déclenchement et d'horodatage. En revanche, la profondeur de la FIFO influe principalement sur les ressources en bascules (FF). Contrairement aux analyseurs logiques intégrés (*Integrated Logic Analyzers* — ILA) proposés par les fournisseurs, qui s'appuient sur des BRAM pour capturer de longues fenêtres de traces, notre solution vise un flux d'événements continu avec des tampons FIFO de faible profondeur, implémentés en logique.

Ce choix permet de préserver les blocs de BRAM, tout en maintenant un débit de transfert stable. L'inconvénient est une augmentation de l'utilisation en registres (FF), qui devient plus marquée lorsque la profondeur des FIFO augmente. Toutefois, même avec 100 sondes et des FIFO de profondeur 16, l'empreinte reste limitée à environ 4,5 % des LUT et 8,6 % des FF du FPGA Alveo U280, qui en compte respectivement 1,2 M et 2,5 M [55].

5.2 Caractérisation des performances en débit

5.2.1 Débit du DMA

Pour valider le fonctionnement du module DMA et de son pilote logiciel, un banc d'essai a été mis en place. Afin d'éliminer les goulots d'étranglement liés au stockage, les données de traçage ont été redirigées vers un disque `tmpfs` (système de fichiers en RAM), pour que le SSD interne ne biaise pas les mesures de performance.

Sur le FPGA, il était essentiel de disposer d'une génération de données à la fois rapide (une donnée par cycle d'horloge) et suffisamment variée pour permettre la vérification de leur ordre d'arrivée à large bande passante. Pour répondre à ces contraintes, un générateur matériel de séquences pseudo-aléatoires a été implémenté au moyen d'un registre à décalage à rétroaction linéaire (*Linear Feedback Shift Register* — LFSR). Ce générateur produit un flux AXI-Stream continu, dont le comportement déterministe permet à la fois de solliciter pleinement la chaîne de traçage et de vérifier l'intégrité du flux reçu côté hôte.

Ce flux alimente directement le module DMA en mode *streaming*, qui se charge de le transférer vers la mémoire de l'hôte au moyen d'un accès AXI Memory-Mapped (AXI-MM). Comme illustré à la Figure 5.2, les données transitent du LFSR vers le DMA, puis traversent le pont PCIe avant d'être exposées à l'espace utilisateur via un périphérique de type caractère (CDEV). Cette architecture garantit un débit soutenu sur l'ensemble de la chaîne, permettant une évaluation précise des capacités de transfert et des performances globales.

Pour valider l'intégrité du pilote, un programme utilisateur copie en continu le flux de données depuis le CDEV vers un fichier temporaire en RAM. Après chaque session de transfert,

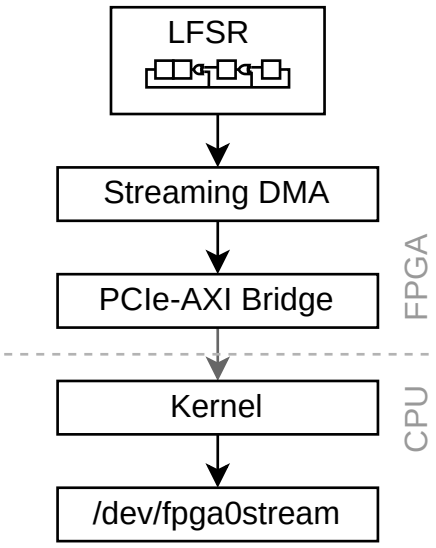

FIGURE 5.2 Générateur de séquence pseudo-aléatoire LFSR implémenté sur FPGA

un second programme logiciel recalcule les valeurs attendues du LFSR et les compare aux données générées par le FPGA.

Les essais de débit ont montré qu'avec un lien PCIe 3.0 ×1, le module DMA atteint un débit soutenu de 6,87 Gb/s, soit environ 87 % du débit utile théorique maximal. En effet, un lien PCIe 3.0 ×1 offre une bande passante brute de 8 GT/s (gigatransferts par seconde), mais utilise un encodage *128b/130b*, ce qui réduit la bande passante utile à :

$$8 \text{ GT/s} \times \frac{128}{130} \approx 7{,}877 \text{ Gb/s} \tag{5.1}$$

En pratique, ce débit subit encore une réduction supplémentaire liée aux en-têtes de trame PCIe, à la structure des paquets transactionnels (*Transaction Layer Packets* — TLPs) et à la surcharge du protocole. La valeur de 6,87 Gb/s mesurée représente donc environ :

$$\frac{6{,}87}{7{,}877} \approx 87 \ \% \tag{5.2}$$

du débit utile maximal théorique, ce qui confirme l'efficacité du pipeline de transfert. De plus, le caractère déterministe de la séquence LFSR a permis de vérifier que l'ensemble des données reçues était correct, sans perte ou corruption.

5.2.2 Performances en débit du traçage des événements

Afin d'évaluer la capacité du système à capturer des événements sans perte, dix compteurs matériels ont été instrumentés à l'aide des sondes. Leur fréquence de mise à jour a été ajustée pour contrôler le débit global d'événements injectés dans la chaîne de traçage. Chaque événement produit était codé sur 128 bits, incluant son horodatage, son identifiant et ses données associées. Chaque essai a duré 5 secondes et a été répété 5 fois afin de vérifier la reproductibilité des mesures.

D'après les résultats obtenus lors de l'évaluation du DMA, le débit maximal théorique pouvant être soutenu par la chaîne complète est de 53,67 millions d'événements par seconde (ME/s). Ce débit est calculé en divisant le débit maximal mesuré sur le lien PCIe par la taille d'un événement encodé :

$$\text{Débit maximal théorique} = \frac{\text{Débit maximal PCIe}}{\text{Taille d'un événement}} = \frac{6{,}87 \text{ Gb/s}}{128 \text{ bits}} = 53{,}67 \text{ ME/s} \qquad (5.3)$$

Ce résultat constitue donc la limite supérieure du débit soutenable par le système dans un cas idéal sans surcharge supplémentaire.

5.2.3 Évaluation de l'effet de la taille de la mémoire de tampon du DMA

Les premiers résultats des tests de performance ont montré que les pertes d'événements survenaient de manière groupée, indiquant l'existence de congestions ponctuelles dans la chaîne de transfert entre le FPGA et l'hôte. Cette observation suggère que le DMA ne parvient pas toujours à vider ses tampons internes assez rapidement lorsque la cadence d'émission augmente. Pour quantifier l'impact de la profondeur de mémoire sur ce phénomène, nous avons évalué plusieurs configurations de la mémoire tampon du DMA.

Les performances ont été mesurées avec une FIFO de profondeur 8 par sonde. Chaque essai a duré 5 secondes et a été répété 5 fois. La Figure 5.3 présente le taux de perte d'événements en fonction du débit global pour plusieurs tailles de mémoire tampon DMA : 32, 64, 128 et 256 kbits. L'axe des ordonnées est en échelle logarithmique afin de mettre en évidence les faibles taux de perte, et la ligne verticale pointillée grise représente la limite théorique déterminée à partir du débit maximal du DMA calculé en 5.3.

Les résultats montrent une amélioration progressive de la tolérance aux pertes à mesure que la taille du tampon augmente. Le taux de perte reste nul jusqu'à environ 25 ME/s avec 32 kbits, 38 ME/s avec 64 kbits, et 46 ME/s avec 128 kbits. Augmenter la taille de la mémoire a pour

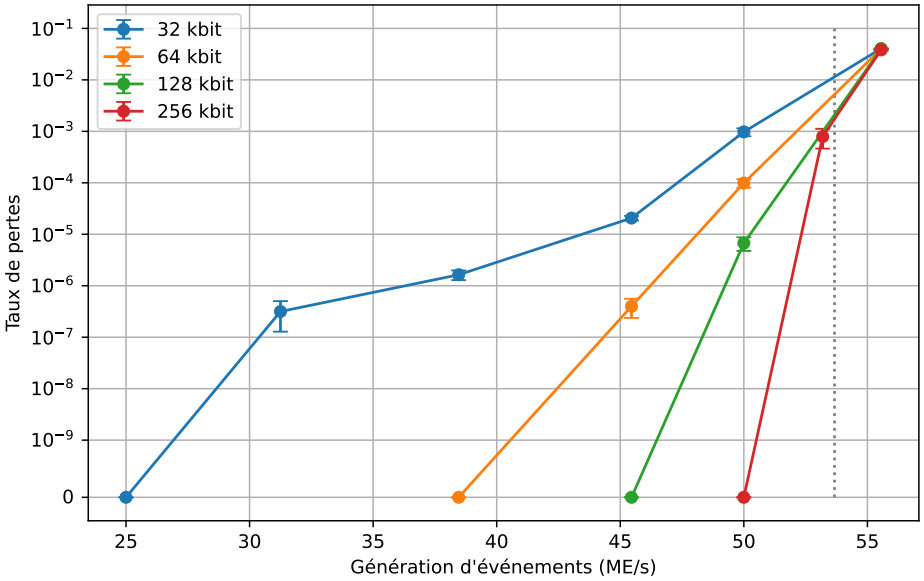

FIGURE 5.3 Taux de pertes en fonction du taux d'événements.

effet d'absorber les ralentissements dans les transferts. En l'augmentant jusqu'à 256 kbit, la taille maximale pour être contenue dans une seule UltraRAM [55], il est possible de générer des événements sans perte jusqu'à 50 ME/s. Au-delà de 50 ME/s, les tests pour différentes tailles de mémoire convergent vers environ 4 % de pertes, ce qui est attendu, puisque cela correspond au dépassement du débit soutenable du lien PCIe.

Ces résultats confirment que la taille du tampon DMA agit comme un amortisseur des fluctuations de débit : plus la mémoire disponible est grande, plus le système parvient à absorber les variations ponctuelles d'émission sans perte. Toutefois, une fois la capacité du lien saturée, l'augmentation de la mémoire ne suffit plus à compenser le dépassement du débit maximal de transfert. Il est néanmoins remarquable que des performances optimales soient atteintes avec une capacité de tampon très modeste, équivalente à un seul bloc UltraRAM.

5.2.4 Évaluation de la performance de la chaine de sondes avec rafales

La section précédente a mis en évidence l'influence de la mémoire du DMA sur la capacité du système à maintenir un flux d'événements soutenu et régulier. Nous considérons ici un scénario plus exigeant, dans lequel les sondes produisent des événements de manière non uniforme, c'est-à-dire sous forme de rafales. Ce cas reflète des situations réelles où l'activité du matériel varie dans le temps, par exemple lors d'échanges de données intenses ou d'accès concurrents au bus.

L'expérience repose sur dix compteurs matériels, instrumentés avec les sondes. Leur fréquence de mise à jour est modulée de façon à conserver un débit moyen de 50 ME/s, tout en concentrant les émissions dans des fenêtres temporelles étroites. Ce modèle permet d'évaluer la capacité de la chaîne à absorber des volumes instantanés élevés sans perte.

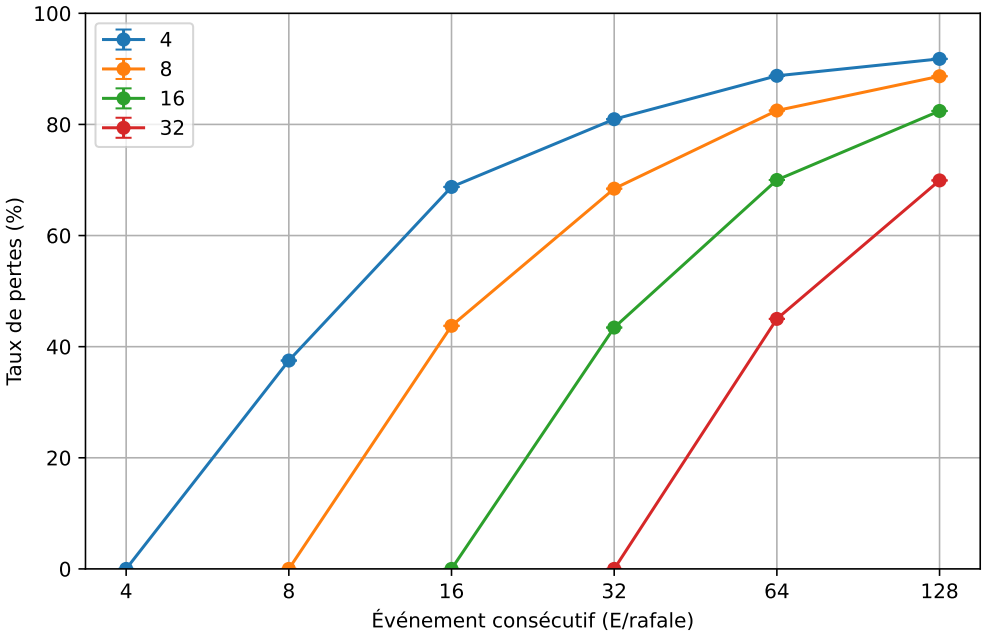

FIGURE 5.4 Taux de pertes en fonction de la taille des rafales

La Figure 5.4 illustre le taux de perte en fonction de la taille des FIFOs et des rafales. Les cas étudiés correspondent à des profondeurs de 4, 8, 16 et 32, des valeurs suffisamment faibles pour que les tampons soient entièrement implémentés en logique, sans recourir aux BRAM. Comme attendu, les sondes sont capables d'absorber autant d'événements consécutifs que leur FIFO interne le permet. Lorsque la taille de la rafale est doublée, près de la moitié des événements sont perdus, car seules certaines données ont le temps d'être absorbées par la chaîne. Cela confirme la stabilité du système dans des conditions réalistes de trafic irrégulier, lorsque le dimensionnement des FIFO locales est adéquat.

5.3 Traçage multi-domaines hétérogène

Enfin, une série d'expérimentations a été menée sur une plateforme comportant plusieurs domaines d'horloge afin de valider la capacité du système à corréler des événements issus du processeur hôte et du FPGA. Dans ce scénario, un programme utilisateur interagit avec un périphérique géré par un module Linux développé sur mesure, lequel assure les transferts de

lecture et d'écriture vers la mémoire mappée du FPGA.

La Figure 5.5 illustre le cheminement des transactions, depuis les appels effectués par le programme utilisateur (situé sur le processeur hôte, partie inférieure de la figure) jusqu'à la mémoire du FPGA accessible via le pont PCIe–AXI (partie supérieure). La zone FPGA comporte deux domaines d'horloge distincts, notés `CLK #1` et `CLK #2`, représentés respectivement en blanc et en bleu clair, permettant de visualiser la propagation des opérations entre les différents segments de la chaîne matérielle.

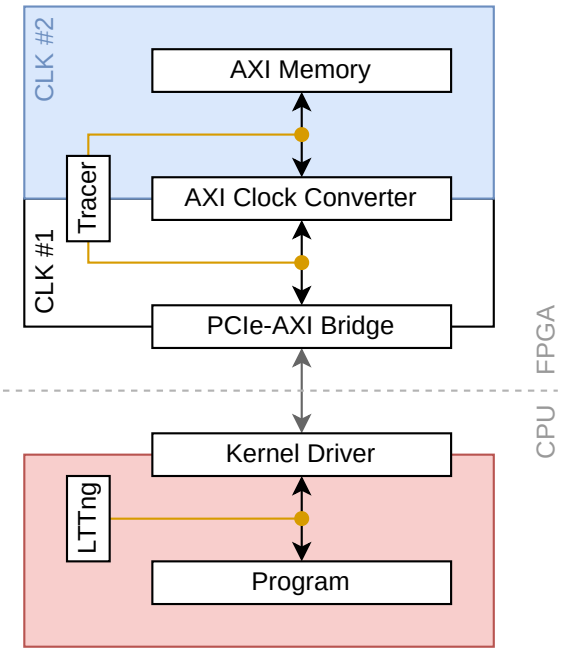

FIGURE 5.5 Traçage d'une interaction CPU–FPGA sur deux domaines d'horloge.

5.3.1 Configurations logicielles du traçage hétérogène

Deux configurations logicielles distinctes ont été évaluées afin d'observer le comportement du traçage hétérogène CPU–FPGA sous différents niveaux d'abstraction logicielle. La première, basée sur des appels systèmes standards, pour montrer la traversée complète de la pile logicielle (utilisateur, noyau, pilote, matériel) et illustre l'utilisation conjointe de LTTng côté noyau et des sondes matérielles côté FPGA. La seconde, utilisant un accès direct à la mémoire mappée, réduit la surcharge associée aux appels systèmes tout en permettant un traçage à l'aide de points personnalisés en espace utilisateur. Ces deux approches offrent des perspectives complémentaires : la première illustre la visibilité inter-domaines du système complet, tandis que la seconde démontre la faisabilité d'un traçage allégé à haute performance.

5.3.2 Traçage avec appels systèmes

Dans une première configuration expérimentale, les échanges entre le programme utilisateur et le FPGA s'effectuent au moyen d'appels systèmes standards (`read()` et `write()`), transmis au périphérique PCIe par le pilote Linux, qui les traduit en transactions mémoire.

L'architecture du processeur traduit ces interactions avec la mémoire en transactions PCIe. Le XDMA (*PCIe-AXI Bridge*) convertit ensuite les transactions PCIe en transactions AXI dans le premier domaine d'horloge AXI (`CLK #1`). Les transactions passent ensuite à travers le convertisseur d'horloge pour atteindre le second domaine (`CLK #2`) et effectuer les opérations mémoire. Après la complétion de l'opération, la transaction suit le chemin inverse vers le processeur hôte.

Pendant ce temps, LTTng capture les appels système dans l'espace noyau vers une trace CTF. En parallèle, des sondes matérielles surveillent les transactions AXI dans les deux domaines FPGA. Les événements de synchronisation injectés par le CPU permettent l'alignement temporel et une corrélation causale.

Après post-traitement, les traces logicielles et matérielles peuvent être ouvertes dans Trace Compass. La Figure 5.6 illustre la propagation d'un appel système `write()`, puis `read()`.

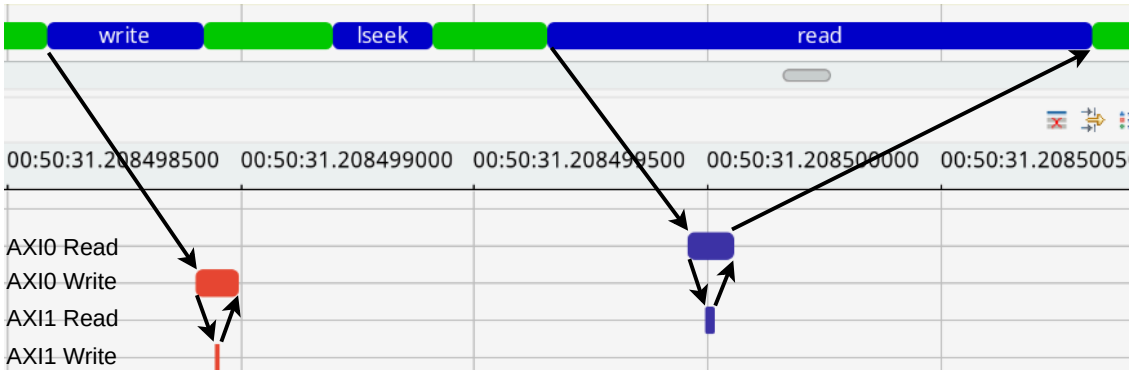

FIGURE 5.6 Vue de Trace Compass de la propagation d'appels système

On observe que la séquence des événements respecte l'ordre causal attendu :

- L'appel système d'écriture démarre en espace utilisateur, suivi de l'initiation de l'écriture AXI dans le premier domaine, puis dans le second. La complétion s'effectue en ordre inverse : d'abord dans le second domaine, puis dans le premier. Il peut être observé que l'appel système se termine avant que la transaction AXI ne soit elle-même complétée.
- L'appel système de lecture démarre en espace utilisateur, suivi de l'initiation de la lecture AXI dans le premier domaine, puis dans le second. Le retour de la lecture

s'effectue en ordre inverse : d'abord dans le second domaine, puis dans le premier, et finalement l'appel système est complété.

La Figure 5.6 met également en évidence un comportement distinct entre les appels `write()` et `read()`. L'écriture se termine rapidement côté logiciel, car le système considère l'opération comme complétée dès que la requête a été envoyée au matériel, sans attendre la confirmation complète du FPGA. À l'inverse, la lecture demeure active tant que la donnée n'a pas été reçue en retour.

5.3.3 Traçage avec mémoire mappée

Dans une deuxième configuration expérimentale, le programme utilisateur accède directement à une zone de mémoire mappée sur le périphérique PCIe. Les opérations de lecture et d'écriture sont alors traduites automatiquement par l'architecture du processeur en transactions PCIe, qui suivent ensuite le même cheminement matériel que dans le scénario précédent.

Pendant l'exécution, le programme émet des points de trace dans l'espace utilisateur, capturés par LTTng dans une trace CTF distincte. Le Listing 5.1 illustre ces points d'instrumentation. En parallèle, des sondes matérielles enregistrent les transactions AXI dans les deux domaines d'horloge du FPGA, tandis que des événements de synchronisation sont injectés par le processeur pour assurer la corrélation temporelle.

Listing 5.1 Mémoire mappée et UST

```
while (1) {
    // ...
    lttng_ust_tracepoint(hwmm, ust_write_start, addr*8,value);
    mem[OFFSET+pos] = value;
    lttng_ust_tracepoint(hwmm, ust_write_end, addr*8);

    lttng_ust_tracepoint(hwmm, ust_read_start, addr*8);
    value = mem[OFFSET+pos];
    lttng_ust_tracepoint(hwmm, ust_read_end, addr*8, value);
    // ...
}
```

Après post-traitement, les traces logicielles et matérielles sont fusionnées et visualisées dans Trace Compass. La Figure 5.7 montre la propagation des lectures et des écritures entre les deux domaines d'horloge, mettant en évidence la relation causale entre les événements d'application et les opérations matérielles correspondantes.

On observe que la séquence des événements respecte l'ordre causal attendu :

- Le point de traçage précédant l'écriture est appelé (`UST Writing`), suivi de l'initiation de l'écriture AXI dans le premier domaine (`AXI0 Writing`), puis dans le second (`AXI1`

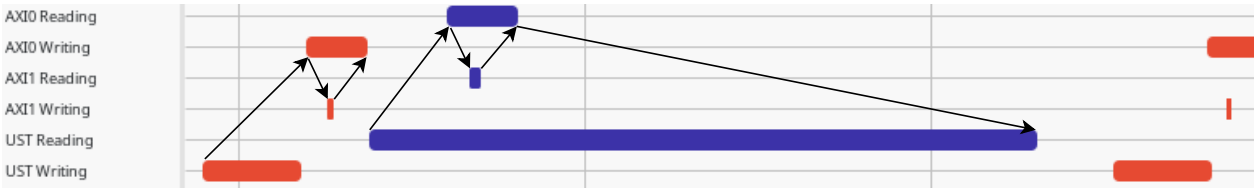

FIGURE 5.7 Vue de Trace Compass de la propagation des accès à la mémoire

`Writing`). La complétion s'effectue en ordre inverse : d'abord dans le second domaine (`AXI1 Writing`), puis dans le premier (`AXI0 Writing`). De même, le point de trace de fin d'écriture (`UST Writing`) apparaît avant que la transaction AXI ne soit elle-même complétée.

- Le point de traçage précédant la lecture démarre en espace utilisateur (`UST Reading`), suivi de l'initiation de la lecture AXI dans le premier domaine (`AXI0 Reading`), puis dans le second (`AXI1 Reading`). Le retour de la lecture s'effectue également en ordre inverse : d'abord dans le second domaine (`AXI1 Reading`), puis dans le premier (`AXI0 Reading`), et finalement dans l'espace utilisateur (`UST Reading`). Notons que la lecture demeure active du côté hôte tant que la donnée n'a pas été reçue en retour.

5.3.4 Évaluation de la synchronisation inter-domaines d'horloge

Une expérience a été mise en place pour évaluer la synchronisation entre plusieurs domaines d'horloge. Pour ce faire, des événements sont émis périodiquement par un générateur. Ces événements passent ensuite par une série de registres afin d'éviter la métastabilité. Pour chaque domaine d'horloge, la longueur de la chaîne de registres est ajustée afin que le temps de transmission soit équivalent. L'événement est ensuite enregistré par une sonde du domaine correspondant.

La désynchronisation a été évaluée pour un intervalle de synchronisation compris entre 1 ms et 1000 ms. L'expérience utilise deux horloges différentes provenant de sources distinctes, donc asynchrones : la première à 250 MHz et la seconde à 200 MHz.

Après post-traitement, la différence entre les deux domaines est analysée. Pour chaque point, la moyenne, l'écart type et le maximum sont calculés. La Figure 5.8 illustre la désynchronisation mesurée entre les deux domaines d'horloge, exprimée en nanosecondes, pour différents intervalles de synchronisation compris entre 1 ms et 1000 ms. L'axe des abscisses est en échelle logarithmique et couvre les valeurs suivantes : 1, 2, 5, 10, 25, 50, 100, 250, 500 et 1000 ms. Pour chaque point, la figure présente la désynchronisation moyenne, l'écart type et la valeur maximale obtenue.

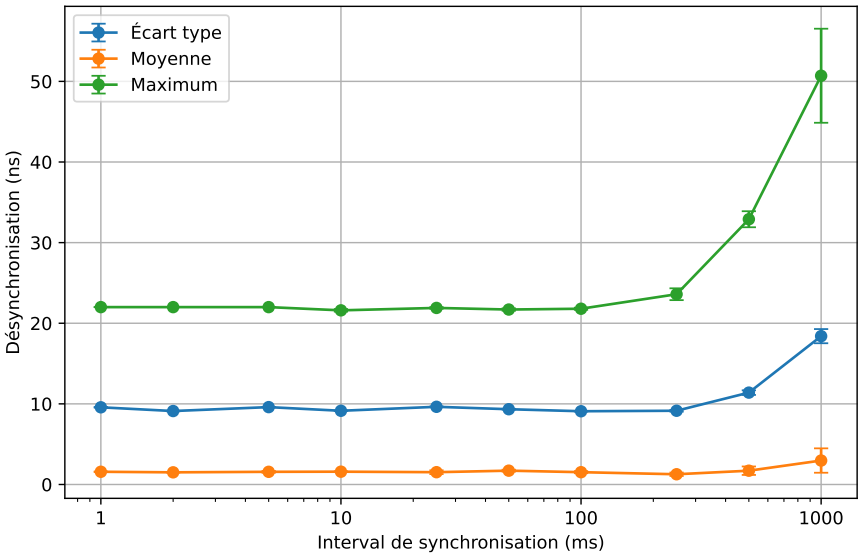

FIGURE 5.8 Désynchronisation entre les domaines d'horloges

Les résultats montrent qu'à 250 ms et en dessous, la synchronisation ne semble pas bénéficier d'une amélioration. Au-delà, les résultats varient selon les expériences. On observe une désynchronisation moyenne de 2 ns, ce qui est inférieur aux périodes d'horloge, qui sont respectivement de 4 ns et de 5 ns. L'écart-type est d'environ 9 ns, soit près de deux périodes d'horloge. Les écarts maximaux sont en moyenne de 22 ns, soit environ 4 à 6 cycles d'horloge.

5.4 Synthèse des résultats et discussion générale

Les résultats expérimentaux confirment la robustesse et la pertinence de l'infrastructure de traçage proposée, tant du point de vue des performances que de son intégration dans des environnements hétérogènes CPU–FPGA. L'ensemble des mesures portant sur l'utilisation des ressources, la capacité de transfert, le comportement sous charges intenses et la synchronisation multi-domaines permet de dégager plusieurs conclusions significatives.

5.4.1 Synthèse des capacités et limites du système de traçage

Le transfert DMA a démontré une capacité à soutenir des débits de l'ordre de 6,87 Gb/s sur une liaison PCIe 3.0 $\times 1$, soit près de 87 % du maximum théorique. La capacité à maintenir un tel débit en continu, sans perte de données, valide la robustesse de la chaîne matériel–logiciel, du module DMA jusqu'au pilote et au stockage. La solution DMA étant le lien entre la solution matérielle de traçage et son logiciel, cela permet d'effectuer des tests à haute performance sur celui-ci.

Un autre avantage majeur du traceur est sa faible occupation en ressources FPGA. Même en instrumentant 100 sondes avec des tampons locaux de profondeur 16, l'empreinte totale reste inférieure à 9 % des ressources LUT et FF, ainsi que de très peu de mémoire, d'un dispositif haut de gamme comme l'AMD Alveo U280. Ce résultat valide les choix architecturaux faits, notamment l'implémentation des FIFO en registres plutôt qu'en BRAM afin de privilégier la réduction de l'utilisation matérielle et de permettre la continuité des flux d'événements.

L'étude du traçage événementiel montre que jusqu'à 50 ME/s, le système parvient à opérer sans perte en ajustant la taille de mémoire. Ce débit s'approche des limites théoriques du DMA, qui, avec son débit maximal, serait de 53,67 ME/s. Cela confirme que le débit soutenu hors puce constitue le principal goulot d'étranglement. Ce résultat s'inscrit dans la logique des systèmes de traçage modernes, où la gestion explicite de la perte d'événements est courante, notamment avec des outils tels que LTTng.

Les expériences sur un cas d'usage hétérogène CPU–FPGA ont montré la capacité du système à corréler des événements issus de différents domaines d'horloge avec une précision sub-microseconde. L'approche combinée (traçage logiciel avec LTTng, sondes matérielles, et mécanisme de synchronisation par événements) permet de reconstruire des relations causales globales avec une continuité temporelle robuste. Cette caractéristique est essentielle pour le débogage et l'analyse de performance sur des plateformes mixtes.

5.4.2 Précision de synchronisation et conditions d'alignement temporel

Les résultats montrent que la synchronisation entre les domaines d'horloge reste stable et précise à l'échelle de sub-microsecondes. Cette précision repose en grande partie sur l'utilisation de l'horloge monotonique qui dérive du compteur temporel invariant du processeur. Ainsi, la synchronisation entre le processeur et le FPGA reste fiable, même en présence de variations de charge sur la machine hôte.

Le mécanisme de synchronisation utilisé, fondé sur l'injection d'événements périodiques, induit un surcoût minimal sur la bande passante (inférieur à 0,3 %), ce qui permet un alignement temporel continu sans perturber le flux principal des événements. Ce résultat conforte l'idée que la solution est adaptée au traçage prolongé et peut maintenir une cohérence temporelle dans un contexte hétérogène, même lorsque plusieurs domaines d'horloge sont impliqués.

5.4.3 Limites identifiées et conditions d'utilisation optimales

L'évaluation expérimentale confirme que les pertes d'événements apparaissent essentiellement lorsque le débit global approche les limites physiques du lien PCIe utilisé. Le comportement

observé (accumulation des pertes au-delà d'environ 50 millions d'événements par seconde) est cohérent avec la saturation progressive du canal de transfert. Dans ce contexte, la solution atteint un fonctionnement optimal tant que le débit généré demeure inférieur ou égal à la capacité soutenable du DMA.

De même, certaines expériences montrent que le système réagit de façon sensible à des rafales très concentrées d'événements. Bien que des tampons plus larges (jusqu'à 256 kbit) permettent d'absorber efficacement les fluctuations ponctuelles, l'extensibilité reste étroitement liée à la bande passante disponible et à la capacité du DMA à vider ses tampons. Ces limitations ne sont pas propres à notre approche : elles reflètent la nature même des systèmes de traçage en continu et correspondent au comportement observé dans les traceurs logiciels tels que LTTng, où la détection explicite des pertes fait partie intégrante du modèle.

5.4.4 Contraintes architecturales et impact sur la mise à l'échelle

Les expériences multi-domaines ont également mis en évidence certaines contraintes liées à l'architecture du pipeline de sondes. Lorsque plusieurs domaines d'horloge doivent être traversés, il est nécessaire d'organiser les sondes selon un ordre déterminé afin d'éviter la congestion au niveau des registres de synchronisation. Un chaînage allant du domaine le plus lent vers le plus rapide minimise les risques de retour en arrière ou de saturation locale. Cette organisation, bien qu'efficace, peut influencer les décisions de placement et de routage dans les conceptions FPGA complexes.

La mise à l'échelle future de la solution pourrait donc bénéficier d'architectures alternatives, par exemple un chaînage arborescent ou une fusion tardive de flux indépendants, ce qui réduirait la charge sur une chaîne linéaire unique.

5.4.5 Comparaison aux approches existantes et positionnement

Les travaux antérieurs portant sur le traçage en flux d'événements des FPGA se concentrent principalement sur la capture d'événements matériels avec une reconstruction logicielle hors ligne. Aussi, les approches existantes dédiées à l'instrumentation logicielle (par exemple, LTTng ou VTune pour FPGA Intel) ne permettent pas d'obtenir une vue unifiée couvrant à la fois les événements logiciels et les événements RTL.

La contribution principale de cette approche réside dans la capacité à aligner continuellement ces deux mondes, ce qui permet une observation causale complète entre les événements logiciels et matériels, en particulier pour le débogage haute performance. Cette capacité n'était pas présente dans les approches antérieures orientées FPGA-seul ou CPU-seul, ni dans les

solutions ILA à fenêtre limitée.

Ainsi, l'infrastructure développée se positionne comme un traceur hybride, capable de :

- Capturer des événements matériels à haut débit ;
- Capturer des événements logiciels (noyau et espace utilisateur) ;
- Aligner les deux types d'événements sur une seule échelle temporelle ; et
- Permettre une visualisation standardisée via CTF et Trace Compass.

Ce positionnement comble une lacune importante dans les outils de développement pour les plateformes hétérogènes CPU–FPGA.

5.5 Perspectives futures

Plusieurs pistes d'amélioration ont été identifiées. Sur le plan matériel, l'extension du DMA à des liens PCIe plus larges (par exemple PCIe 3.0 ×4 ou 4.0 ×4) permettrait d'augmenter sensiblement la bande passante et de réduire les pertes, même pour des fréquences de traçage plus élevées. Par ailleurs, une part importante des informations contenues dans la trace est répétée afin de respecter le format CTF. L'application d'algorithmes de compression permettrait d'accroître le débit d'information utile sans modifier l'architecture matérielle.

Une amélioration importante consisterait à mesurer les pertes d'événements en temps réel, afin de notifier immédiatement l'utilisateur lorsqu'une perte survient. Une telle fonctionnalité renforcerait la robustesse de l'outil dans un contexte de développement, où la détection explicite des événements manquants est essentielle.

L'automatisation de la solution de traçage constitue une autre voie prometteuse. Plusieurs tâches pourraient être automatisées :

- Intégration automatique du système de traçage dans une conception matérielle existante. Une telle fonctionnalité faciliterait son adoption dans divers projets et permettrait de régénérer rapidement l'instrumentation selon les besoins. Une intégration directe dans un projet Vivado serait utile.
- Ajustement automatique de la taille des FIFO en fonction de critères liés à la tolérance aux pertes. Cette adaptation dynamique permettrait de dimensionner la solution selon les contraintes de chaque application.

Un autre aspect à explorer est l'ajout de sondes post-implémentation ou incrémentales. La possibilité d'instrumenter une conception déjà implémentée ou partiellement compilée simplifierait le cycle de développement et améliorerait la flexibilité de l'outil.

Ainsi, l'évaluation menée montre que l'architecture proposée satisfait aux exigences de traçage continu, performant et faiblement intrusif dans un environnement FPGA. Elle constitue une base solide pour l'analyse détaillée des interactions CPU–FPGA et ouvre la voie à des solutions hybrides plus avancées pour la simulation, le débogage et le test en temps réel.

CHAPITRE 6 CONCLUSION

Ce projet de recherche a d'abord consisté à analyser les approches existantes en matière de traçage matériel et logiciel. Cette revue a mis en évidence l'absence de solutions unifiées pour le traçage des systèmes hétérogènes CPU–FPGA, en particulier pour la corrélation temporelle entre les événements logiciels et matériels.

Nous avons ensuite proposé une solution matérielle pour effectuer le traçage. Celle-ci est basée sur une chaîne de sondes et un DMA AXI-PCIe, s'intégrant bien à la plateforme hétérogène de processeurs x86 et Alveo. Des outils logiciels ont été développés pour l'accompagner, permettant la synchronisation et le transfert des traces, du matériel au logiciel. Des outils de post-traitement ont également été développés, permettant d'obtenir des traces synchronisées matériel–logiciel dans le format CTF.

Nous avons ensuite testé la solution dans plusieurs scénarios, en commençant par des tests de performance à partir de compteurs. Nous avons ensuite évalué la qualité de la synchronisation pour les applications multi-horloges, en comparant des événements similaires émis par une même horloge. Finalement, nous avons testé une application de démonstration hétérogène, où un programme effectue des écritures et des lectures dans une mémoire. Ce test a permis de démontrer la capacité du programme à retracer les événements du logiciel au matériel et vice-versa.

Cette preuve de concept ouvre effectivement la voie à plusieurs extensions. D'abord, la détection d'anomalies pourrait être intégrée afin de déclencher automatiquement des captures lors de comportements inhabituels, réduisant ainsi le besoin d'instrumentation manuelle. Aussi, le traçage pourrait inclure des analyses de performance, telles que la détection de valeurs aberrantes, la surveillance de l'utilisation du bus et le profilage de la latence. Enfin, des optimisations du traceur, tant au niveau architectural qu'au niveau des outils, pourraient être explorées afin de réduire la surcharge du traçage.

RÉFÉRENCES

[1] N. Deloumeau et F. Tétreault, “Development of an AXI tracing solution for FPGAs on heterogeneous systems,” Talk at Tracing Summit 2025, sept. 2025, recensé au lien url : https://cfp.tracingsummit.org/ts2025/talk/BEPWU7/, enregistrement vidéo disponible sur Youtube : https://youtu.be/FWjrkF6JkEk?si=ldPk3NwiguJFTO4G.

[2] N. Deloumeau, T. Ould-Bachir, D. Evans, A. Handke, J. Sanderson et F. Tétreault, “Synchronized CPU–FPGA tracing for heterogeneous platforms,” dans *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (SFPGA)*. Seaside, CA, USA : Association for Computing Machinery, févr. 2026, (Soumis).

[3] S. Mittal et J. S. Vetter, “A survey of CPU-GPU heterogeneous computing techniques,” *ACM Comput. Surv.*, vol. 47, n°. 4, juill. 2015.

[4] W. F. Samayoa, M. L. Crespo, A. Cicuttin et S. Carrato, “A survey on FPGA-based heterogeneous clusters architectures,” *IEEE Access*, vol. 11, p. 67 679–67 706, 2023.

[5] P. Margheritta et M. R. Dagenais, “LTTng-HSA : Bringing LTTng tracing to HSA-based GPU runtimes,” *Concurrency and Computation : Practice and Experience*, vol. 31, n°. 17, p. e5231, 2019.

[6] A. Fiorini et M. R. Dagenais, “Visualization of profiling and tracing in CPU-GPU programs,” *Concurrency and Computation : Practice and Experience*, vol. 34, n°. 23, p. e7188, 2022.

[7] S. Darche et M. R. Dagenais, “Low-overhead trace collection and profiling on GPU compute kernels,” *ACM Trans. Parallel Comput.*, vol. 11, n°. 2, juin 2024.

[8] T. Fanni, D. Madronal, C. Rubattu, C. Sau, F. Palumbo, E. Juarez, M. Pelcat, C. Sanz et L. Raffo, “Run-time performance monitoring of heterogenous Hw/Sw platforms using PAPI,” dans *International Workshop on FPGAs for Software Programmers (FSP)*, 2019, p. 1–10.

[9] M. Gebai et M. R. Dagenais, “Survey and analysis of kernel and userspace tracers on linux : Design, implementation, and overhead,” vol. 51, n°. 2, p. 26 :1–26 :33.

[10] “Common Trace Format v1.8.3.” [En ligne]. Disponible : https://diamon.org/ctf/v1.8.3/

[11] N. Schulz, F. Penczek, J. E. Wylder et E. Geva, “System for correlation of operating system and hardware trace events,” Brevet US 2017/0 091 063 A1, Mar 30 2017.

[12] F. Ratti, J. Knödtel et M. Reichenbach, “Heterogeneousrtos : A cpu-fpga real-time os for fault tolerance on cots at near-zero timing cost,” *ACM Trans. Embed. Comput. Syst.*, vol. 24, n°. 2, févr. 2025.

[13] T. Ould-Bachir, H. Saad, S. Dennetière et J. Mahseredjian, “CPU/FPGA-based real-time simulation of a two-terminal MMC-HVDC system,” *IEEE Transactions on Power Delivery*, vol. 32, n°. 2, p. 647–655, 2017.

[14] L. Li, H. Pan, X. Wan, K. Lv, Z. Wang, Q. Zhao, F. Ning, Q. Ning, S. Zhang, Z. Li, L. Luo et G. Xie, *Harmonia : A Unified Framework for Heterogeneous FPGA Acceleration in the Cloud*. New York, NY, USA : Association for Computing Machinery, 2025, p. 498–514.

[15] N. Mustafa et Y. Labiche, “The Need for Traceability in Heterogeneous Systems : A Systematic Literature Review,” dans *IEEE Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1, juill. 2017, p. 305–310.

[16] “AMBA AXI and ACE Protocol Specification Version E.” [En ligne]. Disponible : https://developer.arm.com/documentation/ihi0022/e/?lang=en

[17] M. Desnoyers et M. Dagenais, “Upcoming changes to kernel tracing : Advances from the LTTng community,” dans *Proceedings of the Linux Symposium*, vol. 101, juill. 2008.

[18] “LTTng : an open source tracing framework for Linux.” [En ligne]. Disponible : https://lttng.org//

[19] “LTTng : lttng/lttng-modules.” [En ligne]. Disponible : https://github.com/lttng/lttng-modules

[20] “LTTng : lttng/lttng-ust.” [En ligne]. Disponible : https://github.com/lttng/lttng-ust

[21] LTTng : lttng-sessiond v2.13. [En ligne]. Disponible : https://lttng.org//man/8/lttng-sessiond/v2.13/

[22] “Babeltrace : babeltrace2 v2.0.” [En ligne]. Disponible : https://babeltrace.org/docs/v2.0/man1/babeltrace2.1/

[23] Trace compass. [En ligne]. Disponible : https://eclipse.dev/tracecompass/

[24] Trace compass perf profiling user guide. [En ligne]. Disponible : https://archive.eclipse.org/tracecompass.incubator/doc/org.eclipse.tracecompass.incubator.perf.profiling.doc.user/User-Guide.html

[25] Trace compass ftrace user guide. [En ligne]. Disponible : https://archive.eclipse.org/tracecompass.incubator/doc/org.eclipse.tracecompass.incubator.ftrace.doc.user/User-Guide.html

[26] Barectf docs. [En ligne]. Disponible : https://barectf.org/docs/barectf/3.1/index.html

[27] Trace compass user guide - data driven analysis. [En ligne]. Disponible : https://archive.eclipse.org/tracecompass/doc/stable/org.eclipse.tracecompass.doc.user/Data-driven-analysis.html

[28] Trace compass user guide - trace synchronization. [En ligne]. Disponible : https://archive.eclipse.org/tracecompass/doc/stable/org.eclipse.tracecompass.doc.user/Trace-synchronization.html

[29] NVIDIA nsight systems. [En ligne]. Disponible : https://developer.nvidia.com/nsight-systems

[30] Introduction to ROCProfiler — rocprofiler 2.0.0 documentation. [En ligne]. Disponible : https://rocm.docs.amd.com/projects/rocprofiler/en/amd-master/concepts/introduction.html

[31] Y. Arafa, A.-H. Badawy, A. ElWazir, A. Barai, A. Eker, G. Chennupati, N. Santhi et S. Eidenbenz, "Hybrid, scalable, trace-driven performance modeling of GPGPUs," dans *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. New York, NY, USA : Association for Computing Machinery, 2021.

[32] F. Portas, G. Bois et Y. Savaria, "Cochrono : A unified hardware/software performance analysis tool for SoC-FPGA codesign," dans *IEEE Interregional NEWCAS Conference (NEWCAS)*, 2024, p. 328–332.

[33] C. Blochwitz, R. Klink, J. M. Joseph et T. Pionteck, "Continuous live-tracing as debugging approach on FPGAs," dans *International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, déc. 2017, p. 1–8.

[34] N. Penneman, L. Perneel, M. Timmerman et B. De Sutter, "An FPGA-Based Real-Time Event Sampler," dans *Reconfigurable Computing : Architectures, Tools and Applications*, P. Sirisuk, F. Morgan, T. El-Ghazawi et H. Amano, édit. Berlin, Heidelberg : Springer, 2010, p. 364–371.

[35] H. Bensalem, Y. Blaquière et Y. Savaria, "Toward in-system monitoring of OpenCL-based designs on FPGA," dans *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, mai 2019, p. 1–5.

[36] ——, "In-FPGA Instrumentation Framework for OpenCL-Based Designs," *IEEE Access*, vol. 8, p. 212 979–212 994, 2020.

[37] Y.-K. Choi et J. Cong, "HLScope : High-level performance debugging for FPGA designs," dans *IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, avr. 2017, p. 125–128.

[38] J. Kim et C. Hao, "Realprobe : An automated and lightweight performance profiler for In-FPGA execution of high-level synthesis designs," dans *IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2025, p. 10–18.

[39] J. S. Monson et B. Hutchings, “New approaches for in-system debug of behaviorally-synthesized FPGA circuits,” dans *International Conference on Field Programmable Logic and Applications (FPL)*, sept. 2014, p. 1–6.

[40] J. Goeders et S. J. E. Wilton, “Signal-tracing techniques for in-system FPGA debugging of high-level synthesis circuits,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, n°. 1, p. 83–96, janv. 2017.

[41] G.-R. Tsai, M.-C. Lin et C.-H. Lin, “A real-time two-level trace compressor for FPGA-based SoC on-chip debugger,” dans *International Conference on Innovative Computing, Informatio and Control (ICICIC)*, sept. 2007, p. 267–267.

[42] D. Couturier et M. R. Dagenais, “LTTng CLUST : A System-Wide Unified CPU and GPU Tracing Tool for OpenCL Applications,” *Advances in Software Engineering*, vol. 2015, p. e940628, août 2015.

[43] A. Sembrant, T. E. Carlson, E. Hagersten et D. Black-Schaffer, “A graphics tracing framework for exploring CPU+GPU memory systems,” dans *IEEE International Symposium on Workload Characterization (IISWC)*, oct. 2017, p. 54–65.

[44] G. Pagano, D. Dosimont, G. Huard, V. Marangozova-Martin et J.-M. Vincent, “Trace management and analysis for embedded systems,” dans *IEEE International Symposium on Embedded Multicore Socs*, 2013, p. 119–122.

[45] T. Bertauld et M. R. Dagenais, “Low-level trace correlation on heterogeneous embedded systems,” *EURASIP Journal on Embedded Systems*, vol. 2017, n°. 1, p. 18, janv. 2017.

[46] “Integrated logic analyzer v6.2 : LogiCORE IP product guide,” 2016. [En ligne]. Disponible : https://docs.amd.com/v/u/en-US/pg172-ila

[47] 2. design debugging with the signal tap logic analyzer. [En ligne]. Disponible : https://www.intel.com/content/www/us/en/docs/programmable/683819/25-1-1/design-debugging-with-the-logic-analyzer-69524.html

[48] E. Hung et S. J. E. Wilton, “Incremental Trace-Buffer Insertion for FPGA Debug,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, n°. 4, p. 850–863, avr. 2014.

[49] A. Rafii, W. Sun et P. Chow, “Pharos : a multi-FPGA performance monitor,” dans *International Conference on Field-Programmable Logic and Applications (FPL)*, août 2021, p. 257–262.

[50] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek et K. Asanović, “Chisel : constructing hardware in a scala embedded language,” dans *Proceedings of the Annual Design Automation Conference (DAC)*. New York, NY, USA : ACM, 2012, p. 1216–1225.

[51] “Chisel : A modern hardware design language,” nov. 2024, original-date : 2015-04-27T22 :55 :56Z. [En ligne]. Disponible : https://github.com/chipsalliance/chisel

[52] The linux kernel documentation — the linux kernel documentation. [En ligne]. Disponible : https://docs.kernel.org/index.html

[53] J. Corbet, A. Rubini et K.-H. Greg, “Linux Device Drivers, Third Edition [LWN.net],” 2005. [En ligne]. Disponible : https://lwn.net/Kernel/LDD3/

[54] P. J. Salzman, M. Burian, O. Pomerantz, B. Mottram et H. Jim, “The Linux Kernel Module Programming Guide,” 2025. [En ligne]. Disponible : https://sysprog21.github.io/lkmpg/

[55] “FPGA resource information — DS963.” [En ligne]. Disponible : https://docs.amd.com/r/en-US/ds963-u280/FPGA-Resource-Information