# AsteRISC : Un coeur RISC-V flexible

Jonathan SAUSSEREAU Christophe JEGO Camille LEROUX Jean-Baptiste BEGUERET
Univ. Bordeaux, CNRS, Bordeaux INP, IMS, UMR 5218, F-33400 Talence, France

**Résumé –** Le jeu d'instructions open source RISC-V est prometteur pour les applications liées aux systèmes embarqués à faible consommation. Cet article présente une architecture de processeur RISC-V configurable offrant un compromis entre le nombre de cycles d'horloge requis pour exécuter une instruction, la fréquence maximale de fonctionnement et l'utilisation des ressources. Cette flexibilité architecturale permet d'adapter le processeur aux contraintes applicatives, sur cibles FPGA ou ASIC.

**Abstract –** The RISC-V open source instruction set architecture is a promising solution for applications related to low power embedded systems. This paper presents a configurable RISC-V processor architecture providing a compromise between the number of clock cycles required to execute an instruction, the maximum operating frequency, the resource utilization and the power consumption. This architectural flexibility enables the core to be adapted to fit application constraints, on either FPGA or ASIC.

## 1 Introduction

L'émergence des jeux d'instructions libres, en particulier RISC-V [5], a conduit à l'apparition d'une variété d'implémentations matérielles de processeurs open source. Cela facilite l'utilisation d'architectures programmables à des fins académiques et commerciales. Cependant, il est difficile de trouver une implémentation de processeur qui soit parfaitement adaptée à l'ensemble des contraintes applicatives. Des travaux ont été réalisés pour comparer plusieurs implémentations RISC-V, tant dans la gamme des soft-cores [3] de faible performance que dans celle des performances moyennes à élevées [2]. Néanmoins, ces travaux ne peuvent pas être exhaustifs et les performances des cœurs ainsi que l'utilisation des ressources ne sont pas absolues. Elles sont influencées par la technologie ciblée et la taille des mémoires choisies [7]. Ainsi, un processeur peut être plus performant qu'un autre dans certaines conditions, mais moins performant dans d'autres conditions.

Une autre approche pour trouver l'architecture appropriée pour un ensemble de contraintes applicatives et pour une technologie donnée consiste à utiliser la flexibilité comme moyen d'explorer l'espace de conception. En effet, il est plus facile de mettre en place et de tester un seul cœur hautement configurable que plusieurs cœurs indépendants peu configurables.

Les architectures flexibles peuvent être décrites à un niveau d'abstraction élevé à l'aide d'outils de synthèse d'architecture (HLS). Néanmoins, seuls quelques processeurs [4] à usage général ont été conçus à l'aide de la HLS, et ce, malgré l'amélioration continue des outils de HLS open source. De plus, l'utilisation d'outils de synthèse de haut niveau peut être un frein pour certains concepteurs. Il en va de même pour la diminution des performances obtenues avec les alternatives disponibles. Une autre option intéressante pour des conceptions flexibles est le langage de construction de matériel Chisel. Il offre une bonne flexibilité au niveau du SoC et permet de simplifier la conception du cœur. Une approche alternative est d'implémenter la flexibilité directement dans la description RTL de l'architecture. Un bon exemple de conception de cœur flexible est BRISC-V : une boîte à outils d'exploration de l'espace de conception d'architecture. Cet outil permet de générer plusieurs processeurs RISC-V, d'un processeur à cycle unique à des processeurs pipeline, et supportant le multi-cœur. Néanmoins, des approches génériques sont nécéssaires afin de concevoir des cœurs flexibles optimisés au niveau de la taille, et adaptés à la fois aux circuits FPGA et aux ASIC.

Cet article présente AsteRISC, une description au niveau RTL, en SystemVerilog, d'un cœur RISC-V. L'originalité de ce cœur est de posséder des registres optionnels aux points clés de son chemin de données. Cela permet au concepteur d'avoir un contrôle direct sur le chemin critique. La configuration de registre est sélectionnée par des paramètres avant la synthèse logique. Par conséquent, il n'est pas nécessaire pour l'utilisateur d'éditer la description RTL pour modifier la micro-architecture. Selon la configuration retenue, seuls certains registres sont implémentés. De plus, l'unité de contrôle implémentée consiste en une machine d'états adaptée à la configuration. Il est ainsi possible de rechercher parmi les configurations celle qui répond au mieux aux contraintes de l'application au niveau du nombre de cycles par instruction, de fréquence maximale de fonctionnement, d'utilisation des ressources et de consommation d'énergie. Les résultats d'implémentation, obtenus tant sur des circuits FPGA que sur des technologies ASIC, jusqu'au layout, tendent à montrer que les cœurs flexibles permettent d'obtenir une configuration offrant le meilleur compromis entre les contraintes applicatives pour une cible technologique donnée.

## 2 Approche pour la flexibilité

Faisant partie de la famille des jeux d'instructions réduits (RISC), les architectures RISC-V sont des architectures load and store. Pour mettre en œuvre la baseline RV32I, un processeur doit ainsi effectuer au moins les tâches suivantes :

- IF (Instruction Fetch) : récupérer l'instruction en cours dans la mémoire.
- ID (Instruction Decode) : séparer et traiter les champs du code de l'instruction.
- RF (Register File read) : lire les valeurs des registres.
- EX (EXecute) : exécuter l'opération de l'instruction.
- MA (Memory Access) : lire ou écrire dans la mémoire, si nécessaire.
- WB (Write Back) : écrire le résultat de l'opération dans le registre de destination, si nécessaire.

Dans l'implémentation RISC-V AsteRISC, ces tâches sont implémentées dans des blocs combinatoires connectés entre eux par des registres optionnels. Ces registres sont identifiés par les étiquettes 1 à 6 dans le schéma de la Figure 1.

Des registres optionnels permettent de bufferiser des points clés du chemin de données du processeur, afin de limiter le chemin critique. Cette bufferisation se fait au prix de cycles d'exécution supplémentaires pour les instructions dont les données suivent ce chemin. L'implémentation ou non de chacun de ces registres est sélectionnée par des paramètres Verilog de telle sorte que l'utilisateur n'a pas besoin d'éditer la description RTL de l'architecture. Dans cet article, une combinaison de registres optionnels est appelée *configuration*.

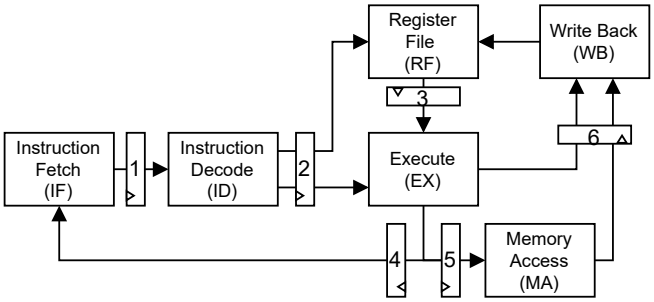



FIGURE 1 : Chemin de données du cœur AsteRISC

Une architecture pipeline utilise généralement une logique supplémentaire pour pouvoir exécuter toutes les tâches simultanément. Cela permet d'atteindre un nombre de cycles par instruction proche de 1. Comme ce cœur est destiné à être simple et optimisé en taille, ce n'est pas l'approche qui a été retenue. En effet, la micro-architecture a été conçue pour permettre de réduire la longueur du chemin critique tout en acceptant un surcout modéré en cycles par instruction (CPI).

# 3 Cœur configurable proposé

## 3.1 Configuration architecturale de base

En fonction de la configuration de registre sélectionnée, une machine à états finis (FSM) différente est générée. La configuration de base (M000) ne contient aucun registre optionnel. Comme illustré sur le graphe de la Figure 2a, la FSM correspondante ne contient que 3 états.

Lorsque aucun registre optionnel n'est implémenté, nous obtenons le CPI le plus bas possible et, bien sûr, le chemin de données le plus contraignant. Il en résulte une faible fréquence maximale. Bien qu'elle ne convienne pas aux applications axées sur les performances, cette solution est tout à fait adaptée pour un fonctionnement à basse fréquence, comme c'est le cas pour des applications privilégiant une faible consommation.

Dans l'architecture proposée, les mémoires d'instructions et de données sont supposées être implémentées en SRAM avec lecture et écriture synchrones. De ce fait, les étages IF et MA nécessitent chacun un cycle d'horloge. D'autre part, la file de registres peut aussi bien être implémentée avec des bascules ou en SRAM. Dans ce dernier cas, un cycle d'horloge supplémentaire est nécessaire pour la lecture de la file de registres (RF). Dans les deux cas, un cycle est utilisé pour le write-back (WB).

Comme les étapes IF, MA et WB nécessitent un cycle dédié, les étapes ID, RF et EX peuvent être réalisées au cours du même cycle d'exécution. Pour éviter les corruptions de données, sans ajouter de logique supplémentaire, il n'est pas possible d'effectuer à la fois une lecture (RF) et une écriture (WB) dans la file de registres pendant le même cycle. Par contre, il est possible d'écrire dans la file de registres (WB) durant le même cycle que IF. Cela permet de gagner un cycle sans coût supplémentaire.

De plus, seules les instructions load et store nécessitent un accès mémoire (MA). Cela signifie qu'il est possible de gagner un cycle pour d'autres instructions en n'accédant pas à l'état MA. Ainsi, seules les transitions $t_1$ et $t_2$ entre l'état 'WB & IF' et l'état 'ID & RF & EX' sont utilisées, comme le montre la Figure 2b.

De plus, l'étage WB n'est pas utilisé pendant les instructions store. Nous pouvons donc charger l'instruction suivante durant l'état MA, et utiliser la transition $t_4$ pour éviter l'état 'WB & IF', comme illustré par la Figure 2c.

Enfin, les instructions load nécessitent à la fois un accès à la mémoire (MA) et un write-back (WB). Par conséquent, comme le montre la Figure 2d, les instructions load sont exécutées en 3 cycles. Les autres instructions peuvent être exécutées en 2 cycles, comme nous pouvons le voir sur les Figures 2c et 2b.


(a) Diagramme global (b) Pas d'accès mémoire

(c) Écriture en mémoire (d) Lecture en mémoire

FIGURE 2 : FSM de la configuration de base (M000)

## 3.2 Gestion des accès mémoires

Pour les technologies FPGA comme pour ASIC, le chemin critique est généralement situé dans la logique d'adressage de la mémoire. L'utilisation d'une hiérarchie de mémoire basée sur le temps de réponse est une solution pour surmonter cette limitation. Cependant, elle est très coûteuse en surface. Il est également possible d'ajouter un cycle supplémentaire pour les accès à la mémoire de données (DMEM), en mettant en œuvre le registre 5 de la Figure 1. Cela s'applique également à la mémoire d'instructions (IMEM) en allouant le registre 4.

## 3.3 Gestion des accès à la file de registre

Dans certains cas, le chemin critique peut se trouver dans le bloc combinant les tâches ID, RF et EX. Par exemple, l'implémentation de l'extension C (instructions compressées) contraindrait davantage le chemin de données de l'étape ID. Afin de pallier cette contrainte, il est possible de séparer la tâche de décodage (ID) des autres tâches en allouant le registre 2. De plus, il est possible d'allouer le registre 6, lié à l'étage WB, sans ajouter un cycle supplémentaire. En effet, le write-back est ainsi réalisé durant le même cycle que le décodage. La configuration M036, donnée en Figure 4, est ainsi obtenue.

D'autre part, une implémentation synchrone de la file de registres permet de diminuer l'utilisation des ressources au prix

d'un cycle d'exécution supplémentaire. Cela peut être aussi bien à la place ou en plus d'un étage de décodage séparé. La configuration M037, telle qu'illustrée par la Figure 5, possède des états dédiés à la fois pour RF et ID.

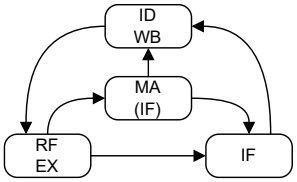



FIGURE 4 : Config. M036

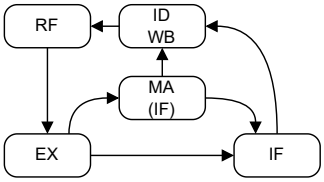



FIGURE 5 : Config. M037

Finalement, les états de bufferisation pour les mémoires d'instructions et de données, utilisés par la configuration M024, peuvent être combinés avec M036 et M037. Ceci permet d'obtenir les configurations M060 et M061, présentées respectivement par les Figures 6 et 7.

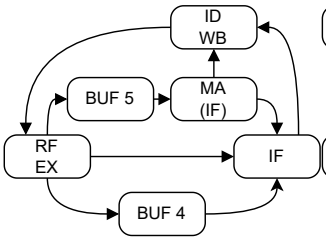



FIGURE 6 : Config. M060

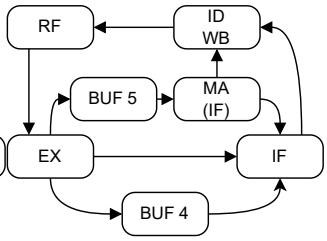



FIGURE 7 : Config. M061

# 4 Résultats d'expérimentation

## 4.1 Conditions expérimentales

Afin d'obtenir des résultats d'implémentation significatifs, le CPU a été testé au sein d'un SoC minimaliste comprenant une mémoire d'instructions, une mémoire de données et un périphérique GPIO à 8 entrées/sorties. Au niveau de l'architecture, la baseline RV32I est utilisée et les compteurs alloués à des fins de benchmarking sont désactivés. Un outil a été développé pour générer et tester toutes les configurations d'AsteRISC simplement. Les résultats sont automatiquement mis en forme dans un tableau, mettant en évidence les configurations qui obtiennent les meilleures performances.

Le PicoRV32 [6], un cœur similaire de l'état de l'art, a été choisi comme point de référence pour les comparaisons. Il a été testé dans le même SoC minimaliste. Les paramètres par défaut ont été sélectionnés, à l'exception du décalage à deux étages, des compteurs, des interruptions et de l'interface mémoire look-ahead, qui ont été désactivés pour garantir une comparaison équitable.

## 4.2 Benchmark

Les configurations présentées dans la section précédente ne sont pas les seules possibles. Le tableau 1 résume 14 des configurations possibles d'AsteRISC et les paramètres associés. Ce tableau affiche également les performances obtenues sur le benchmark Dhrystone. Ces résultats sont exprimés en DMIPS/MHz, où DMIPS est le nombre de millions d'instructions par seconde obtenu lors de l'exécution du benchmark.

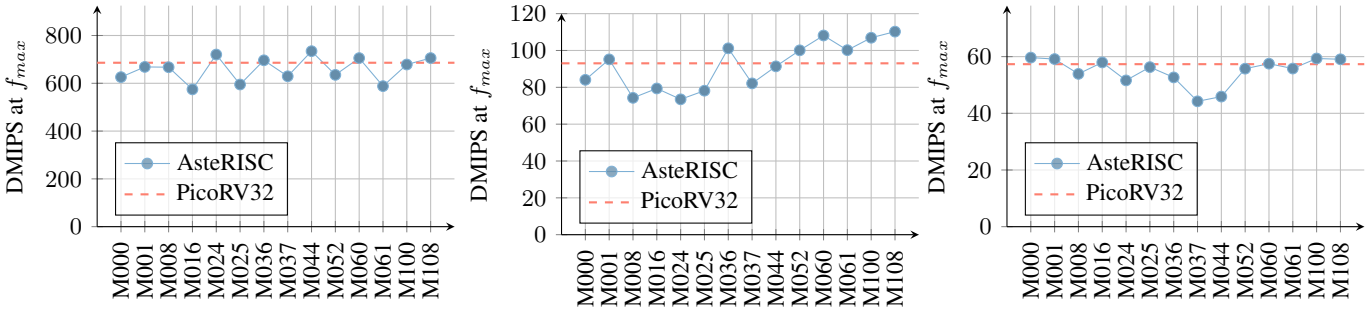
Table 1. Résumé des principales configurations et de leur score Dhrystone

| Arch. | Paramètres | | | | | | Performance |
|---|---|---|---|---|---|---|---|
| | *buf1* | *buf2* | *buf3* | *buf4* | *buf5* | *buf6* | *DMIPS/MHz* |
| M000 | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | 0.678 |
| M001 | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | 0.462 |
| M008 | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | 0.599 |
| M016 | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | 0.630 |
| M024 | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | 0.561 |
| M025 | ✗ | ✗ | ✓ | ✓ | ✓ | ✗ | 0.405 |
| M036 | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ | 0.462 |
| M037 | ✗ | ✓ | ✓ | ✗ | ✗ | ✓ | 0.351 |
| M044 | ✗ | ✓ | ✗ | ✗ | ✓ | ✓ | 0.425 |
| M052 | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ | 0.439 |
| M060 | ✗ | ✓ | ✗ | ✓ | ✓ | ✓ | 0.405 |
| M061 | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | 0.317 |
| M100 | ✓ | ✓ | ✗ | ✗ | ✗ | ✓ | 0.453 |
| M108 | ✓ | ✓ | ✗ | ✗ | ✓ | ✓ | 0.416 |
| PicoRV32 | Voir section 4.1 | | | | | | 0.305 |

Il peut être relevé que, globalement, plus on alloue de registres optionnels, plus le score est faible. En effet, le score est inversement proportionnel au CPI moyen pendant l'exécution du benchmark. Cependant, ce chiffre seul n'est pas significatif. L'implémentation sur une cible technologique donne une fréquence maximale de fonctionnement. À partir de cette fréquence, il est possible de déduire une puissance de calcul en DMIPS. Le PicoRV32 a un score plus faible, mais vise une fréquence maximale plus élevée.

## 4.3 Impact de la cible sur le chemin critique

Des scripts ont été développés pour tester les différentes configurations sur diverses cibles technologiques en cherchant automatiquement la fréquence maximale. La Figure 8 compare les résultats de performance de ces configurations sur plusieurs cibles. Il est notable que les trois tendances sont différentes. Cela signifie que certaines configurations sont plus adaptées que d'autres pour une cible donnée. En fait, la meilleure configuration n'est pas la même pour les circuits FGPA et pour les ASIC. En effet, bien qu'elle soit la plus performante sur ASIC 28 nm, la configuration M044 est la deuxième plus mauvaise sur le FPGA xc7a100t-csg324-1. Ces résultats peuvent être expliqués par la nature différente d'un circuit FPGA et




(a) ASIC (ST 28 nm FDSOI CMOS)

(b) FPGA (xc7k70t-fbg676-2)

(c) FPGA (xc7a100t-csg324-1)

FIGURE 8 : Résultats des performances en DMIPS de plusieurs configurations d'AsteRISC avec une taille de mémoire de 2048x32

d'un ASIC. Il est à souligner que les différentes configurations ne donnent pas les mêmes résultats sur deux circuits FPGA différents, même de la même famille, comme c'est le cas ici.

On peut souligner qu'il existe toujours des configurations qui donnent de meilleurs résultats que le PicoRV32. Cependant, la configuration qui offre les meilleures performances est différente dans les trois cas. Cela met en évidence l'intérêt de l'architecture configurable proposée.

### 4.4 Impact de la taille des mémoires sur le chemin critique

Selon l'application visée, la taille des mémoires peut varier considérablement. De plus, lorsque la taille des mémoires augmente, leur temps d'accès augmente également.

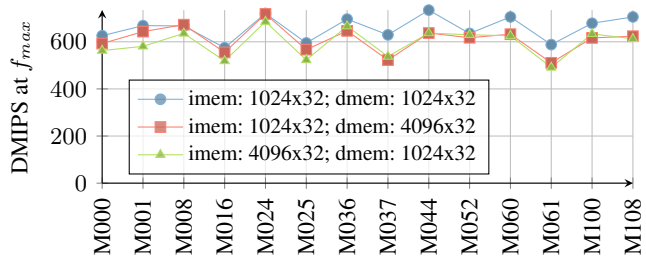



FIGURE 9 : Résultats de performance sur ASIC[1] (ST 28 nm CMOS FDSOI)

L'impact de la taille des mémoires sur les performances du circuit ASIC est illustré par la Figure 9. Différentes tailles de mémoire conduisent à différents classements en termes de performance. Par exemple, dans le premier cas (—●—), la configuration M044 donne les meilleures performances. Au contraire, pour un autre cas (—■—), cette configuration est moins intéressante. Dans ce cas, c'est la configuration M024 qui donne les meilleures performances.

### 4.5 Utilisation des ressources

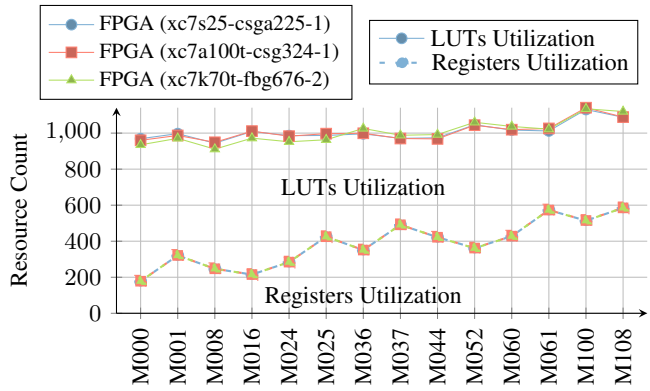



FIGURE 10 : Assignation des ressources sur circuit FPGA[2] pour une taille totale de mémoires de 2048x32

En plus des résultats de performance, une métrique importante est l'utilisation des ressources. Comme on peut le voir sur la Figure 10, les différentes configurations sont similaires en termes de nombre de LUTs. Logiquement, les configurations avec plus de registres optionnels utilisent plus de registres physiques. Nous pouvons noter que les résultats sont très proches pour les 3 FPGAs. En effet, même s'ils sont de gammes différentes, ils sont de la même famille. Il y a un facteur 3 entre la configuration qui alloue le moins de registres et celle qui en alloue le plus. Le défi pour le concepteur est alors d'estimer si le gain de performance vaut le coût supplémentaire en nombre de registres, en fonction des contraintes applicatives.

[1]Les résultats ASIC ont été obtenus avec Design Compiler H-2013.03

[2]Les résultats sur circuits FPGA ont été obtenus avec Vivado 2020.2

## 5 Conclusion

L'objectif de cet article est d'illustrer le potentiel des architectures flexibles par rapport aux architectures dédiées. En effet, la flexibilité de l'architecture AsteRISC permet de trouver un compromis entre puissance de calcul et utilisation des ressources qui s'adapte au mieux aux contraintes du cadre applicatif et de la cible technologique. L'exploration de l'espace de conception rendue possible par AsteRISC permet de générer des cœurs RISC-V aux performances supérieures à celles du PicoRV32 équivalent. Il est à noter que la flexibilité du chemin de données n'empêche pas l'ajout de coprocesseurs ou d'autres formes de flexibilité, comme le séquençage de certaines opérations en plusieurs cycles. Dans la suite de ces travaux, de nouvelles fonctionnalités seront ajoutées, telles qu'un prefetch buffer, une gestion de cache et un support du multi-cœur. En outre, il est prévu de fournir une flexibilité similaire du chemin de données sur une architecture pipeline. Cela élargira d'autant plus l'espace de conception que les utilisateurs de RISC-V peuvent explorer.

## Références


[1] S. BANDARA, A. EHRET, D. KAVA et M. KINSY : *BRISC-V : An Open-Source Architecture Design Space Exploration Toolbox*. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2019.

[2] A. DÖRFLINGER, M. ALBERS, B. KLEINBECK, H. Michalik Y. GUAN, R. KLINK, C. BLOCHWITZ, A. NECHI et M. BEREKOVIC : *A comparative survey of open-source application-class RISC-V processor implementations*. 18th ACM International Conference on Computing Frontiers (CF '21), 2021.

[3] R. HÖLLER, D. HASELBERGER, D. BALLEK, P. RÖSSLER, M. KRAPFENBAUER et M. LINAUER : *Open-Source RISC-V Processor IP Cores for FPGAs — Overview and Evaluation*. 2019 8th Mediterranean Conference on Embedded Computing (MECO), 2019.

[4] P. MANTOVANI, R. MARGELLI, D. GIRI et L. P. CARLONI : *HL5 : A 32-bit RISC-V Processor Designed with High-Level Synthesis*. 2020 IEEE Custom Integrated Circuits Conference (CICC), 2020.

[5] A. WATERMAN, Y. LEE, D. PATTERSON et K. ASANOVIĆ : *The RISC-V Instruction Set Manual, Volume I : User-Level ISA Version 2.1*. Technical Report UCB/EECS-2016-118, EECS Department, University of California, Berkeley, 2016.

[6] C. WOLF : *PicoRV32 - A Size-Optimized RISC-V CPU*. https://github.com/YosysHQ/picorv32, 2015.

[7] H. WONG, V. BETZ et J. ROSE : *Comparing FPGA vs. custom cmos and the impact on processor microarchitecture*. 19th ACM/SIGDA international symposium on Field programmable gate arrays (FPGA '11), 2011.