Augmenter la fréquence du CVA6 sur FPGA

5e concours national d'étudiants RISC-V 2024-2025

Équipe ArmoRISC – IMT Atlantique/Lab-STICC

Marc KLING
IMT Atlantique
Plouzané, France
marc.kling@imt-atlantique.net

Tévchhorpoan KHIEU
IMT Atlantique
Plouzané, France
tevchhorpoan.khieu@imt-atlantique.net

João Vitor MENEGON XAVIER
IMT Atlantique
Plouzané, France
joao-vitor.menegon-xavier@imt-atlantique.net

Cristian Camilo QUEVEDO BELTRAN
IMT Atlantique
Plouzané, France
cristian.quevedo-beltran@imt-atlantique.net

***Résumé*—Ce rapport documente les travaux menés par l'équipe ArmoRISC dans le cadre du concours organisé par Thales, dont l'objectif était d'augmenter la fréquence de fonctionnement du processeur RISC-V CV32A6 lorsqu'il est implémenté sur un FPGA.**

L'optimisation a nécessité une approche multidimensionnelle combinant des modifications architecturales, une amélioration des stratégies de synthèse et un ajustement précis du domaine d'horloge. Ces efforts ont permis d'exploiter au mieux les capacités du FPGA tout en garantissant l'intégrité fonctionnelle du processeur.

Nous détaillons les différentes étapes de notre démarche, notamment l'analyse des goulots d'étranglement limitant la fréquence, les ajustements apportés à la conception matérielle et les choix stratégiques en matière de contraintes temporelles et d'optimisation des chemins critiques.

Nous avons réussi à augmenter la fréquence de $40\ MHz$ **à** $64.7\ MHz$ **soit une augmentation de** $61.75\ \%$**, avec une perte de performance de** $6.7\ \%$ **sur le benchmark Coremark.**

Nos résultats démontrent une augmentation significative de la fréquence par rapport à l'implémentation de référence, mettant en évidence l'efficacité des stratégies adoptées. Ce travail illustre également l'importance des méthodologies avancées d'optimisation matérielle pour améliorer les performances des processeur sur FPGA.

***Index Terms*—RISC-V, CV32A6, FPGA, optimisation de fréquence, accélération matérielle, timing closure**

I. Introduction

Le processeur RISC-V CV32A6 est un cœur open-source haute performance conçu pour des implémentations sur FPGA et ASIC. Il offre une architecture modulaire et extensible, adaptée aux besoins de calcul embarqué et de haute performance. Le CV32A6 est une version compacte et optimisée du cœur ARIANE, initialement développé par l'ETH Zürich. ARIANE est un cœur d'application open-source basé sur l'architecture RISC-V, capable d'exécuter des systèmes d'exploitation riches et intégrant des fonctionnalités avancées telles qu'une unité de gestion de mémoire (MMU) et plusieurs niveaux de privilèges.

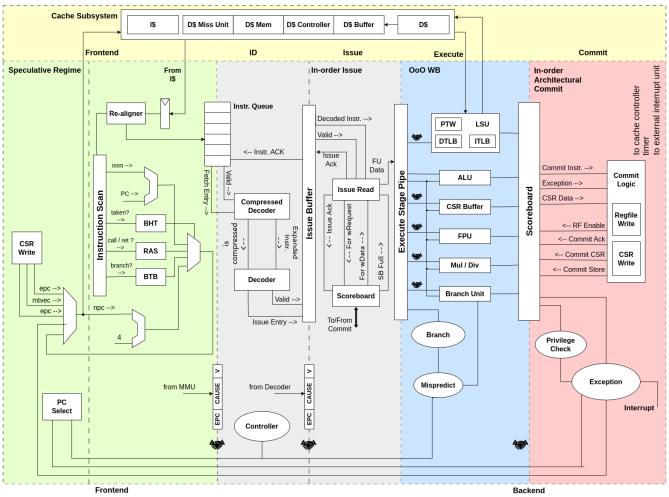


Figure 1. Architecture générale du CV32A6 [1]

Dans le cadre du concours organisé par Thales, le GDR SOC² et le CNFM, l'objectif principal est d'augmenter la fréquence de fonctionnement du processeur CV32A6 sans compromettre sa stabilité et sa fonctionnalité. Ce concours vise à encourager l'innovation dans le domaine des processeurs embarqués et à promouvoir l'utilisation de l'architecture RISC-V.

Son architecture se décompose en 6 étages de pipeline :

1) PC Gen : Génère l'adresse de la prochaine instruction.
2) IF (Fetch) : Récupère l'instruction en mémoire.
3) ID (Decode) : Décode l'instruction et lit les registres.
4) Issue : Planifie et distribue les instructions aux unités.
5) EX (Execute) : Exécute l'opération (ALU, mémoire, etc.).

6) Commit : Finalise l'instruction (écriture registre, exceptions).

Les différents modules intervenant dans ces étages sont illustrés dans la figure 1.

Cet enjeu technique implique une maîtrise avancée des techniques d'optimisation matérielle, notamment :

— L'analyse et l'optimisation des chemins critiques afin de réduire les délais de propagation des signaux.
— L'amélioration des stratégies de synthèse et de placement-routage pour maximiser l'exploitation des ressources du FPGA.
— L'optimisation du domaine d'horloge en ajustant la gestion des signaux d'horloge et des registres pour minimiser la latence et améliorer la convergence temporelle.

Plus particulièrement, dans le cadre de l'optimisation d'un processeur devant exécuter des binaires pré-compilés fournis par Thales, il a été nécessaire de porter une attention minutieuse à la conservation de l'intégrité fonctionnelle de l'architecture matérielle lors des modifications.

Ce rapport détaille les méthodologies employées pour relever ces défis, incluant des modifications du code en langage de description matérielle (HDL), l'optimisation des paramètres de synthèse du FPGA et l'analyse des contraintes temporelles.

II. Méthodologie

Pour le profilage des chemins critiques du processeur, nous avons utilisé le logiciel Vivado version 2024.1 de Xilinx. Après avoir effectué l'implémentation FPGA, Vivado nous a permis d'identifier les parties du processeur nécessitant une optimisation. Une fois les chemins critiques identifiés, nous avons évalué différentes méthodes pour déterminer la plus pertinente afin de résoudre les contraintes de timing tout en conservant l'intégrité fonctionnelle du processeur.

Nous avons appliqué des techniques spécifiques d'optimisation, telles que l'ajustement des paramètres de synthèse, l'optimisation du placement-routage, et la modification des chemins critiques. Après chaque modification, nous avons généré un bitstream pour confirmer, d'une part, la conservation de l'intégrité fonctionnelle en effectuant des tests sur carte avec les benchmarks Coremark et MNIST, et d'autre part, le gain en fréquence ainsi que la perte en performance sur le Coremark.

Nous avons opéré de manière itérative, en suivant ce processus rigoureusement. À chaque itération, nous avons observé des améliorations progressives de la fréquence, tout en surveillant attentivement les performances et la stabilité du processeur. Par exemple, à une étape intermédiaire, nous avons observé une augmentation de 10 MHz avec une perte de performance de 1 % sur le Coremark.

Cette approche itérative et méthodique nous a permis d'augmenter progressivement la fréquence du processeur tout en maintenant sa fonctionnalité et sa stabilité.

III. Changement des paramètres de placement routage

Pour améliorer les performances du processeur CV32A6, nous avons commencé par ajuster les scripts de placement et de routage dans Vivado. Initialement, nous utilisions les scripts de placement et de routage par défaut, qui étaient optimisés pour le temps d'exécution (`RuntimeOptimized`). Cependant, ces scripts ne permettaient pas d'atteindre les fréquences de fonctionnement souhaitées en raison de contraintes de timing non satisfaites.

Pour résoudre ce problème, nous avons décidé de passer à des scripts de placement et de routage plus agressifs, spécifiquement conçus pour optimiser les contraintes de timing. Nous avons donc modifié les scripts de placement pour utiliser `ExtraTimingOpt`, qui est conçu pour maximiser les performances de timing en ajustant les positions des cellules de manière plus précise. Pour le routage, nous avons opté pour `AggressiveExplore`, qui explore de manière plus exhaustive les possibilités de routage pour minimiser les délais de propagation des signaux.

Ces modifications ont eu un impact significatif sur les performances du processeur. En utilisant `ExtraTimingOpt` pour le placement et `AggressiveExplore` pour le routage, nous avons observé une amélioration notable des contraintes de timing, ce qui a permis d'augmenter la fréquence de fonctionnement du processeur de manière substantielle sans aucun coût sur la performance sur les benchmark MNIST et Coremark.

En résumé, le passage aux scripts `ExtraTimingOpt` pour le placement et `AggressiveExplore` pour le routage a permis d'atteindre des fréquences de fonctionnement plus élevées sans affecter l'intégrité fonctionnelle du système.

IV. Unité de Chargement

A. Description du Module

L'unité de chargement (*load unit*) est la partie du processeur responsable du chargement des données dans le banc de registres. Elle se trouve au sein de l'unité de chargement et stockage, illustrée dans la figure 2 dans l'étage d'exécution d'instruction.

B. Modification du Module

Pour charger des données dans le banc de registres, l'unité de chargement doit faire appel à l'unité de gestion de mémoire (*memory management unit* ou *MMU*) pour traduire les adresses virtuelles en adresses physiques. Cependant, nous avons remarqué que la routine exécutée lorsqu'une exception survenait pendant la phase de traduction d'adresse limitait la fréquence du processeur. Cette exception était déclenchée dans des conditions spécifiques où la traduction d'adresse échouait, entraînant un ralentissement du processeur.

Notre première modification a consisté à supprimer cette exception. Cette suppression a permis d'augmenter la fréquence de 40 MHz à 50 MHz sans perte de performance, que ce soit sur les benchmarks MNIST ou Coremark.

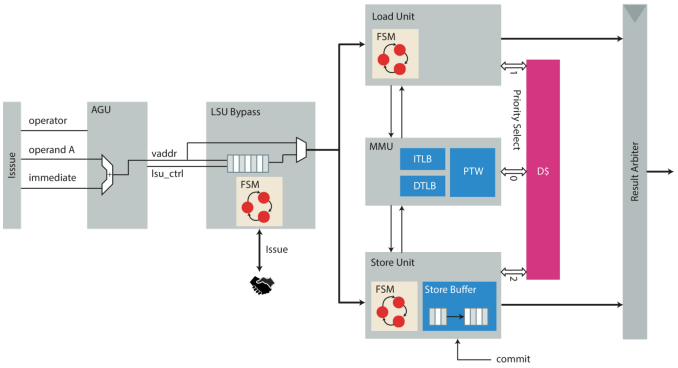


FIGURE 2. Unité de chargement et stockage [2]

Cette modification a été réalisée en ajustant le code de gestion des exceptions dans l'unité de chargement, ce qui a permis de réduire les délais de propagation des signaux et d'améliorer la fréquence opérationnelle.

V. TABLEAU DES SCORES

A. Description du Module

Le tableau des scores (*scoreboard*) est un module situé au sein de l'étage de distribution d'instruction, dont le modèle est illustré dans la figure 3. Il s'agit d'une file d'attente d'instructions qui ont été émises ou qui doivent l'être. Il contient des informations clés sur ces instructions, notamment les registres utilisés, ce qui permet de gérer les dépendances entre instructions. Ainsi, une instruction qui utilise des registres sur lesquels une instruction en cours d'exécution va écrire ne peut pas être émise.

B. Analyse des Chemins Critiques

En analysant les chemins qui ne respectaient pas les contraintes de timing avec Vivado, nous avons constaté que ceux avec un slack négatif important passaient par le tableau des scores.

C. Résolution des Contraintes de Timing

Nous avons remarqué que ce qui provoquait cet échec en termes de satisfaction des contraintes de timing était le temps pris par l'unité de division dans l'étage d'exécution pour rendre la donnée disponible sur les ports de writeback du scoreboard.

Dans un premier temps, nous avons supprimé les requêtes du scoreboard aux ports de writeback, ce qui a résolu les problèmes de timing et nous a permis d'augmenter la fréquence du processeur à plus de 60 MHz. Cependant, cette augmentation de la fréquence s'est accompagnée d'une baisse de performance. Sur le Coremark, nous avons observé une performance de 2.0 Coremark par MHz soit environ 12 % de perte en performance ce qui n'est pas acceptable dans le cadre du concours. En effet, en supprimant les requêtes du scoreboard à tous les ports de writeback, nous avons contraint le processeur à attendre

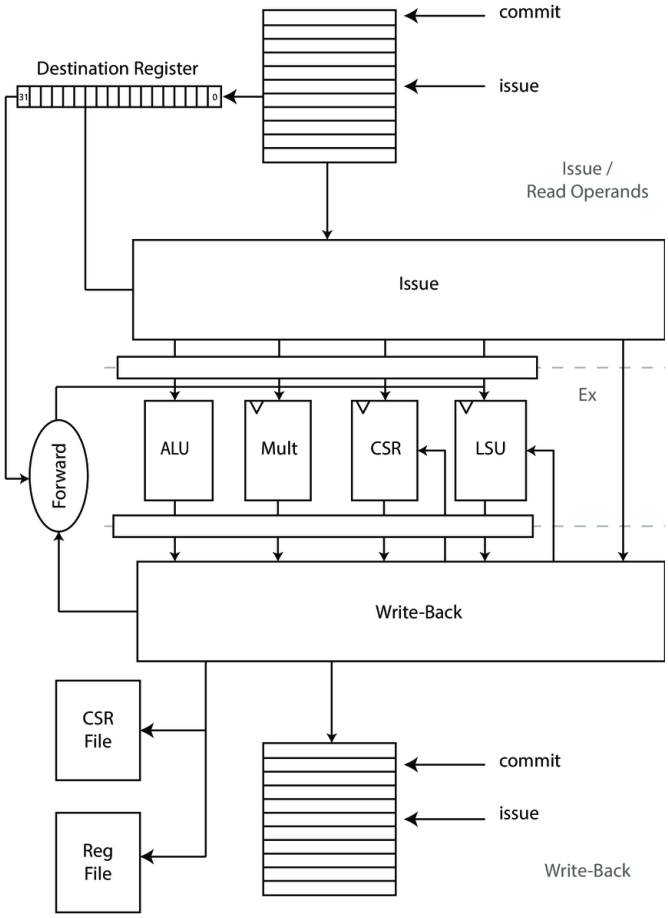


FIGURE 3. Étage de distribution d'instruction [3]

la fin de l'exécution d'une instruction antérieure avant de pouvoir émettre une nouvelle instruction utilisant les mêmes registres, afin d'assurer la cohérence des données.

Nous avons donc décidé de ne supprimer que les requêtes du tableau des scores aux ports de writeback associées à l'unité de division. Cela a permis de rallonger le nombre de cycles pris par une instruction de division, mais nous a permis de monter à une fréquence de 58 MHz quasiment sans perte de performance.

Subséquemment, nous avons examiné quel port était responsable de la baisse de performances et nous avons constaté que c'était le port de writeback associé à l'unité de stockage qui provoquait la plus grosse perte de performance. En ne laissant que ce port de writeback, nous avons pu constater une amélioration de l'ordre de quelques mégahertz et une baisse de performance de 6.77 % sur le benchmark Coremark.

VI. MODIFICATION DE LA CONFIGURATION DU CV32A6

Notre dernière modification a consisté à ajouter un étage de pipeline au sein de l'unité de stockage. Cette modification a été réalisée en changeant le paramètre

`CVA6ConfigNrStorePipeRegs` dans les packages SystemVerilog correspondant à la configuration utilisée. Cette modification a été motivée par la nécessité d'améliorer les contraintes de timing, bien que les derniers chemins critiques identifiés ne passaient pas directement par l'unité de stockage.

L'ajout de cet étage de pipeline a permis de répartir les opérations de stockage sur plusieurs cycles d'horloge, réduisant ainsi les délais de propagation des signaux dans cette unité. Bien que les chemins critiques initiaux ne passaient pas par l'unité de stockage, cette modification a indirectement contribué à une amélioration globale de la fréquence de fonctionnement. Nous supposons que l'ajout de cet étage a permis d'avoir des contraintes de timing plus laxes au niveau de l'unité de stockage, ce qui a facilité l'optimisation du placement et du routage des autres unités critiques.

Cette optimisation a permis d'améliorer la fréquence de fonctionnement du processeur, nous permettant finalement d'atteindre la fréquence retenue pour le concours, soit $64,7\ MHz$. Les résultats obtenus ont été validés par des simulations et des tests sur carte, confirmant l'efficacité de cette modification.

En résumé, l'ajout d'un étage de pipeline dans l'unité de stockage en modifiant le paramètre `CVA6ConfigNrStorePipeRegs` a été une étape cruciale pour atteindre la fréquence de fonctionnement souhaitée. Cette modification a non seulement amélioré les contraintes de timing au niveau de l'unité de stockage, mais a également permis une optimisation globale du placement et du routage, contribuant ainsi à l'amélioration des performances du processeur CV32A6.

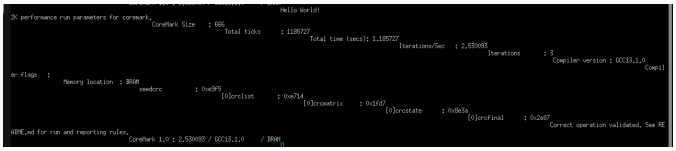
VII. Résultats

Dans cette partie nous présentons tous les résultats obtenus lors de nos travaux via des captures d'écran de la sortie UART et des captures d'écran des résultats des simulations sur Questa ainsi que le document.

Les tests sont fait comme le veulent les modalités du concours sur une carte Zybo Z7-20 et la version 10.7 du logiciel QuestaSim a été utilisé pour les simulations.

Tout d'abord la figure 4(a) montre une capture d'écran de la sortie UART de notre processeur modifié. Comme dit précedemment, nous avons constaté une performance de 2.53 $Coremark/MHz$ ce qui implique une perte de 6.77 % de performance sur le benchmark Coremark nous respectant ainsi les modalités du concours. De plus la figure 4(b) montre les résultats de notre méthode sur le benchmark MNIST. Ces deux captures d'écran montre les performances de notre processeur modifié ainsi que la conservation de l'intégrité fonctionnelle lors de l'implémentation sur FPGA.

On résume les ressources utilisés dans le tableau 5 :



(a) Sortie UART pour Coremark

(b) Sortie UART pour MNIST

Figure 4. Captures d'écran des sorties UART pour les benchmarks Coremark et MNIST à 64.7 MHz

Ressource	Avant modification	Après modification
LUT	24475	24015
LUTRAM	902	862
FF	14568	14637
RAM36	16	16
DSP	4	4

Figure 5. Tableau récapitulatif des ressources

Ces ressources ont été extraites du fichier `cva6_fpga.utilization.rpt` de l'instance du module `i_cva6`.

L'augmentation du nombre de Flip-Flop (FF) (de 14568 à 14637) peut être attribuée à l'ajout d'étages de pipeline, ce qui est nécessaire pour améliorer les performances en réduisant les délais de propagation des signaux. Le nombre de blocs de mémoire RAM36 reste inchangé, ce qui est positif car cela montre que les optimisations n'ont pas nécessité de ressources mémoire supplémentaires.

VIII. Conclusion

Dans ce rapport, nous avons présenté les travaux menés par l'équipe ArmoRISC dans le cadre du concours organisé par Thales, visant à augmenter la fréquence de fonctionnement du processeur RISC-V CV32A6 lorsqu'il est implémenté sur un FPGA. Nous avons adopté une approche multidimensionnelle combinant des modifications architecturales, une amélioration des stratégies de synthèse et un ajustement précis du domaine d'horloge.

Nos résultats montrent une augmentation significative de la fréquence de fonctionnement, passant de 40 MHz à 64.7 MHz, avec une perte de performance de 6.77 % sur le benchmark Coremark. Ces résultats démontrent l'efficacité des stratégies adoptées et illustrent l'importance des méthodologies avancées d'optimisation matérielle pour améliorer les performances des processeur sur FPGA.

Les modifications apportées, telles que l'ajout d'étages de pipeline et l'optimisation des paramètres de placement et de routage, ont permis d'améliorer les contraintes de timing et d'optimiser l'utilisation des ressources du FPGA. Les tests et simulations effectués ont confirmé la stabilité et la fonctionnalité du processeur modifié.

En résumé, ce travail montre que des optimisations matérielles avancées peuvent significativement améliorer les

performances des processeurs RISC-V sur FPGA, ouvrant la voie à des applications embarquées plus performantes et efficaces.

Références

[1] Florian Zaruba. *CVA6 Design Document*. URL : https://docs.openhwgroup.org/projects/cva6-user-manual/03_cva6_design/intro.html.

[2] Florian Zaruba. *CVA6 Design Document*. URL : https://docs.openhwgroup.org/projects/cva6-user-manual/03_cva6_design/ex_stage.html.

[3] Florian Zaruba. *CVA6 Design Document*. URL : https://docs.openhwgroup.org/projects/cva6-user-manual/03_cva6_design/issue_stage.html#scoreboard.