UNIVERSITÉ TOULOUSE III PAUL SABATIER — Université de Toulouse

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

Risc-V @ ZYnq

Synthétiser un processeur RISC-V sur une carte FPGA Zybo Z7-20

Cahier des charges

January 3, 2024

MORAL Alexandre
GUICHETEAU Axel
ASSIER Axel

Contents

1 Introduction

Il n'est, à ce jour, plus utile de revenir sur l'intérêt que représente l'architecture RISC-V et son jeu d'instructions open-source. Pour rappel, Alibaba a sorti un processeur 16 coeurs RISC-V@2.5GHz en 2020 et plus récemment, c'est Espressif (ESPs chips) qui lui a emboîté1 le pas avec ses ESP32-C3.
Cependant, même si les coeurs soft (i.e IP soft) NioS-V sont effectivement basés sur l'architecture RISC-V, il n'en reste pas moins qu'il n'existait pas, jusqu'à très récemment, de FPGA embarquant un ou plusieurs coeurs RISC-V sous la forme d'IP hard. En effet, les FPGAs qui intègrent un ou plusieurs coeurs ARM (e.g Xilinx Zynq, Altera CycloneV) existent mais le nouveau FPGA GoWin GW5AST intègre lui un IP hard RiscV AE350_SOC.

L'objectif de ce projet est de synthétiser un processeur RISC-V sur une carte Zybo Z7-20.
Le processeur devra pouvoir accéder à la mémoire DDR3 embarquée sur la carte et accéder aux E/S (e.g HDMI out, serial link ...).
De même, il devra être possible de télécharger des applications C/C++ afin de tester le bon fonctionnement et les performances du processeur.

L'implémentation RISC-V que nous avons décidé d'utiliser est le Pico-RV32.
Ce design est simple et n'utilise pas trop de place, ainsi, il est facile de l'implémenter sur des cibles FPGA.

Nous devons réussir à faire tourner du code baremetal sur ce processeur pour valider que l'interfaçage de notre coeur FPGA et des différentes entrées sorties de la carte.
A terme, nous ne devrons n'utiliser que le coeur RISC-V à la place du dual-core ARM de la carte.

Également, il nous a été demandé de créer une plaque de protection de la carte dans du PMMA 3mm sans bloquer les différents interrupteurs et entrées ou sorties de la carte.

Si les objectifs sont atteints, il sera probable que nous ajoutions sur la carte un codeur incrémental après avoir réussi a implémenter le processeur.

Comme le PicoRV32 que nous utilisons n'a pas de MMU, nous n'essairons pas de faire tourner un kernel linux sous ce dernier, mais si une fois de plus nous avons le temps, il se peut que nous essayons de faire tourner un linux sur le processeur dual-core ARM de la carte, ou que nous essayons d'implémenter un autre processeur RISC-V avec une MMU sur le FPGA.

2 Analyse de l'existant et des risques

Dans ce projet, plusieurs contraintes ont été identifiées.

2.1 Contraintes organisationnelles

- Contrainte de temps : Le projet doit être terminé avant Mars, ce qui est un délai plutôt court. Il est donc important de bien plannifier les tâches à l'aide d'un GANTT et des réunions d'avancement du projet.

- Contraintes de communication : Il est important de communiquer régulièrement avec le client et entre collègues, du travail distanciel sera surement mis en place. De plus, un git sera mis en place afin d'assurer la communication entre les membres du projet.

- Contraintes de matériel : Comme nous sommes trois à travailler sur le projet, nous aurons besoin d'assez de matériel pour chacun d'entre nous.

- Contraintes d'objectifs : Le projet est souvent soumis à des évolutions sur ce que nous devons faire, il sera important de ne pas vouloir se lancer dans des tâches trop ardues et être objectif.

2.2 Contraintes et risques techniques

- Complexité du projet : Le projet nous demande d'avoir des compétences en Vivado et en FPGA, il nécessite d'avoir également des connaissances en compilation croisée, en FPGA. Nous devrons être assez qualifiés pour le mener à bien, et communiquer avec le corps enseignant en cas de problème ou de choses que nous ne comprenons pas, ce projet étant encadré par la faculté Paul Sabatier III.

- Fiabilité ou problèmes du matériel : Les cartes Zybo utilisées sont fiables, mais il y a une chance de défaillance ou d'accident .

- Fiabilité du Pico-RV32 : L'implémentation du Pico-RV32 est normalement assez fiable car souvent implémentée par des débutants en FPGA, néanmoins il est possible que nous rencontrions des problèmes

- Compilation croisée : Nous devrons être capable de trouver et d'utiliser une chaîne de compilation croisée afin de pouvoir exécuter du code sur notre coeur

- Interfaçage de notre coeur avec la carte : Il est possible que l'interfaçage de notre coeur avec les éléments de la carte pose problème

- Qualité du code : Le code que nous créons pour implémenter le processeur doit être de bonne qualité et bien documenté afin de pouvoir assurer une reprise éventuelle du projet ou assurer la compréhensions du travail fourni.

2.3 Contraintes humaines

- Motivation : Le projet se déroulant sûrement en distanciel, il sera important de maintenir la motivation des équipes tout au long des trois mois

- Compétences : Il est possible que les membres de l'équipes n'ont pas les compétences pour mener à bien le projet, mais c'est également le but de ce projet d'apprendre et de les développer.

2.4 Contraintes financières

- Matériel : Le nombre de carte ainsi que les câbles étant limités et leur prix demandera de porter un attention particulières et de ne pas abîmer les cartes.

3 État de l'art

Pour notre projet nous avons commencé par faire un état de l'art afin savoir ce qui a déjà été réalisé sur le sujet et pouvoir se renseigner sur la manière dont nous allions procéder à sa réalisation.
Comme une des étapes de notre projet est d'implémenter un RISC-V sur notre carte Zybo il fallait se renseigner sur les différents cœurs de processeur open source et synthétisables sur FPGA pour RISC-V que nous aurions pu potentiellement appliquer à notre projet :

- PicoRV32 qui est écrit en Verilog et optimisé pour des cartes FPGA limitées en taille
- RISC-V mini écrit en Chisel
- SERV écrit en Verilog mais est plus petit que le PicoRV32
- Rocket-Chip écrit en Chisel
- NOEL-V écrit en VHDL et synthétisable FPGA mais aussi à destination des ASIC

Notre choix se porte sur le PicoRV32 car il a été imposé par notre client. Nous utiliserons un cœur Regular du PicoRV32 qui utilise 917 Slice LUTs, 48 LUTs as Memory et 583 Slice Registers ce qui sera amplement suffisant pour notre projet, notre carte ayant 53,200 LUTs, 106,400 Flip-Flop et 630Kb de bloc RAM.

Le PicoRV32 est un cœur implémentant les instructions RISC-V et peut être configuré en tant que cœur RV32E, RV32I, RV32IC, RV32IM ou RV32IMC.
De nos jours, la plupart des systèmes Linux sont équipés d'outils de compilation RISC-V, tels que le package gcc-riscv64-unknown-elf pour Ubuntu. Ces outils permettent de compiler du code C/C++ pour les processeurs RISC-V 64 bits, à condition que la toolchain soit bien définie.
Pour pouvoir nous en servir dans notre projet, nous avons besoin de construire la toolchain du RISC-V en 32 bits. Nous avons pu trouver comment l'implémenter sur le GitHub du PicoRV32.
Il faut commencer par installer les packages nécessaires en fonction de l'OS, disponibles sur le GitHub des tools du RISC-V.
Par la suite, nous avons la possibilité de build la toolchain pour un RISC-V 32 ou 64 bits. Dans notre cas, c'est 32 bits. Il faut ensuite l'installer dans le dossier correspondant.
Il ne reste plus qu'à faire un make sur la toolchain sélectionnée. Il est possible de faire un make pour tous les types de PicoRV32 ou uniquement pour la toolchain du type choisi.

Comme nous avions aussi à réaliser une découpe pour créer une plaque permettant l'accès à certains composants de notre carte, nous avons aussi dû mener une recherche sur les capacités et les mesures de notre carte : Référence carte Zybo Z7

Nous avons pu aussi, au fil de nos recherches, trouver des projets similaires qui montrent par exemple une implémentation d'un cœur RISC-V Rocket Core sur une carte Zybo :
Github zynq-fpga pkorolov

4 Analyse des besoins

Dans cette partie nous avons défini plus en détail les objectifs, les besoins de la solution.

Besoins fonctionnels :

- Le processeur implémenté doit pouvoir accéder à la mémoire DDR3 embarquée sur la carte
- Le processeur doit pouvoir accéder aux entrées sorties de la carte
- Le processeur doit pouvoir exécuter des codes C / C++ en baremetal
- Le processeur doit pouvoir être utilisé comme processeur principal de la carte Zybo

Besoins non fonctionnels :

- Le processeur implémenté doit pouvoir accéder à la mémoire DDR3 embarquée sur la carte
- Le processeur doit pouvoir accéder aux entrées sorties de la carte
- Le processeur doit pouvoir exécuter des codes C / C++ en baremetal
- Le processeur doit pouvoir être utilisé comme processeur principal de la carte Zybo

5 Description de la solution

5.1 Cas d'utilisation

Le cas d'utilisation principal de la solution est de permettre l'exécution de code C/C++ sur le coeur RISC-V de la carte Zybo Z-7020. Le processeur RISC-V doit être capable d'accéder à la mémoire DDR3 de la carte et aux différentes entrées/sorties.

5.2 Maquette de la solution

La maquette de la solution consistera en l'implémentation du processeur RISC-V PicoRV32 sur la carte et en différents codes C / C++ permettant de :

- Tester la capacité du processeur à accéder à la mémoire DDR3
- Tester la capacité du processeur à accéder aux entrées/sorties (voir 1)
- Tester la capacité du processeur à exécuter un code C classique (par exemple un Triple DES)

Il est possible que en plus des besoins identifiés plus haut le projet évolue et que l'on nous demande de faire autre chose. Ces besoins devront être identifés et pris en compte dans la description de la solution.

Type d'élément	Ports GPIO
Interrupteurs	G15, P15, W15, T16
Boutons-poussoirs	K18, P16, K19, Y16
Diodes électroluminescentes	M14, M15, G14, D18

Table 1: Définition des entrées et sorties de la carte à tester

5.3 Identification des risques

Nous avons dans cette partie identifié des risques comme le montre la table 2. Ces risques pourraient poser problème durant notre projet. Nous avons donc essayé d'identifier des solutions à ces problèmes.

5.4 Matériel et ressources à disposition

Notre projet nécessite donc l'utilisation de carte FPGAs et de connecteurs. Ainsi, nous allons avoir besoin de deux composants matériels:

- Cartes Zybo Z7-20
- Câbles USB-microUSB

De plus, notre projet se repose également sur d'autres ressources extérieures :

- Processeur PicoRV32 et chaînes compilation croisée: https://github.com/YosysHQ/picorv32

Risque	**Probabilité**	**Solution**
Problème de carte	>3%	Utiliser une autre carte
Problème de prise en main Vivado	40%	Poser des questions à M.Crozet ou aux enseignants, effectuer des recherches internet
Problème implémentation Pico-RV32	>3%	Faire une issue ou contacter quelqu'un qui a réussi à l'implémenter
Problème de connexion RISC-V / Éléments de la carte	20%	Étudier des projets similaires, demander de l'aide
Problèmes de connaissances	60%	Apprendre en travaillant, en faisant des recherches, en demandant au corps enseignant
Problème de temps	15%	Plannifier ce que l'on doit faire, tenir et respecter un diagramme de GANTT
Problème de communication	10%	Réaliser des réunions souvent, présenter ce que l'on a fait aux autres et s'assurer de leur compréhension
Problème d'objectifs et de redéfinition du projet	20%	Il est probable que l'on nous demande de faire quelque chose en plus, il sera important de bien définir ce que l'on nous demande et de bien évaluer le temps que ça nous prendra
Problème de compilation croisée	10%	Effectuer des recherches sur ce qui a été fait auparavent pour compiler du C vers une implémentation de PicoRV32
Problème d'interfaçage du coeur et des ressources de la carte	20%	Demander de l'aide, faire des recherches
Manque de documentation	40%	Garder un tracé qui explique ce qui a été fait, mettre des commentaires dans le code, faire de la documentation

Table 2: Risques et solutions

6 Modalités d'organisation

6.1 Rôles

Nous avons identifié les différents rôles des acteurs du projet (voir table 3).

Personne	Rôle
MORAL Alexandre	Chef de projet
GUICHETEAU Axel	Collaborateur
ASSIER Axel	Collaborateur
THIEBOLT François	Client et professeur encadrant
CASSE Hugues	Client et Professeur référent

Table 3: Rôles des différentes personnes du projet

6.2 Tâches

Nous avons découpé le projets en différentes tâches afin de pouvoir plannifier le travail à accomplir.

6.2.1 Analyse du projet

- Etude de la carte
- Réalisation d'un état de l'art
- Élaboration d'un cahier des charges

6.2.2 Réalisation de la plaque de protection

- Prise des mesures de la carte - ASSIER
- Réalisation d'un fichier AutoCAD - ASSIER
- Formation à la découpe laser - MORAL GUICHETEAU ASSIER
- Réalisation de prototypes - ASSIER
- Découpe du produit final et assemblage avec la carte - MORAL ASSIER

6.2.3 Implémentation du picoRV32

- Analyse des fichiers GitHub du picoRV32 - GUICHETEAU
- Implémentation du coeur - ASSIER MORAL GUICHETEAU
- Synthèse, placement et routage - ASSIER MORAL GUICHETEAU
- Initialisation mémoire - ASSIER MORAL GUICHETEAU

6.2.4 Interfaçage des Entrées/Sorties avec le cœur

- Identification des différentes entrées/sorties - ASSIER
- Définition et implémentation des E/S nécessaires - ASSIER MORAL GUICHETEAU

6.2.5 Tests et validation

- Écriture de tests pour les différentes entrées et sorties de la carte
- Test d'exécution de code C et C++ sur la carte - ASSIER MORAL GUICHETEAU
  - Test Unitaires
    - lecture/écriture dans la mémoire DDR3 - MORAL
    - boutons, interrupteurs(bouttons switch) et LEDs - ASSIER GUICHETEAU MORAL
  - Test d'Intégration
    - vérification de l'interaction des boutons, interrupteurs et LEDs avec le picoRV32. - ASSIER GUICHETEAU
    - communication entre le picoRV32 et la mémoire DDR3. - MORAL
  - Tests de Validation - TOUT LE GROUPE
    - vérification du démarrage du picoRV32 et des composants.
    - test de stress en appuyant rapidement sur les boutons et en redémarrant
    - contrôle de l'état des LEDs en réponse aux actions sur les interrupteurs/boutons.
    - utilisation d'appels systèmes pour interroger le nombre de coeurs disponibles et voir si le dual-core arm9 est bien désactivé.
    - Objectif : S'assurer que notre système répond aux besoins fonctionnels.
  - Jeux de test - TOUT LE GROUPE
    - Vérification des lectures et écritures à différentes adresses.
    - Simuler des actions avec des scénarios spécifiques (appui simultané, rapide, maintenu).
    - Exécution du même code C/C++ sur différentes machines et sur d'autres architectures risc-V .
    - Simuler des entrées non valides provenant d'autres composants.
    - Objectif : Avoir des tests pour couvrir différentes fonctionnalités et différents

scénarios d'utilisation.

- Tests de Performance - TOUT LE GROUPE
  - Mesure temps d'exécution.
  - Mesure du temps nécessaire pour effectuer des opérations de lecture/écriture.
  - Évaluation du débit mémoire.
  - Objectif : Évaluer les performances du picoRV32 en vue de les comparer avec le coeur Ariane.

6.3 Diagramme de GANTT

Nous avons donc prévu le planning de tâches suivant :

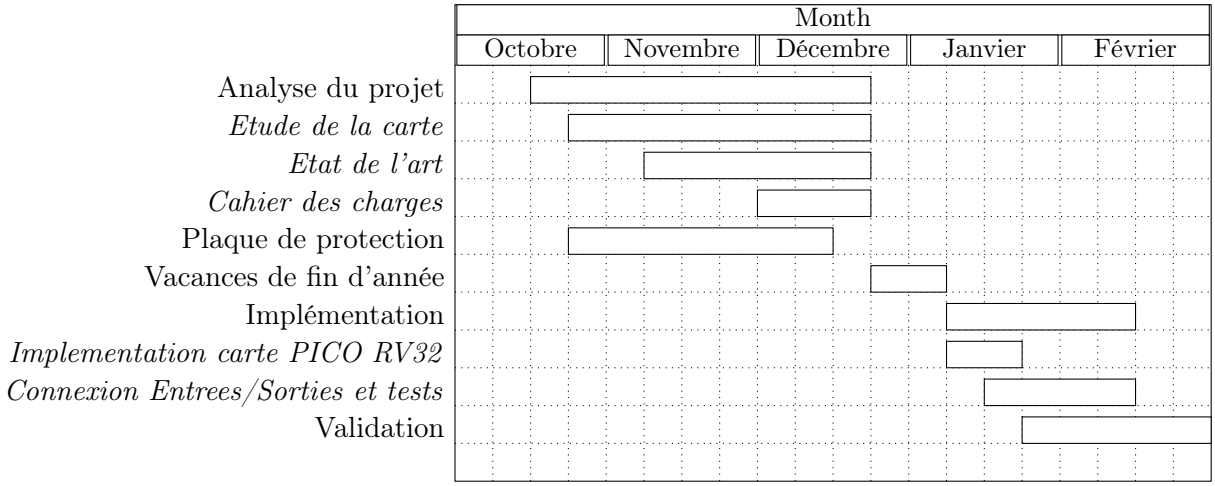

Figure 1: Diagramme de GANTT