VLSI

Rapport Partiel de projet :

Modélisation d'un cœur de processeur ARM

Louis Geoffroy Pitailler, Timothée Le Berre

December 2021

Sommaire

1 Introduction

Au cours de ce projet, on cherche à décrire le cœur d'un processeur basé sur une architecture ARM. Pour ce faire on utilise le langage de description matérielle VHDL.
Le CPU que l'on modélise est un processeur pipeliné sur 4 étages :
- Fetch : récupère l'instruction en mémoire et l'envoie à l'étage decode,
- Decode : récupère l'instruction chargé par Fetch et procède à son décodage afin de sélectionner les opérandes, registres et calculs nécessaires à son exécution,
- Exe : effectue les opérations arithmétiques de bases,
- Mem : effectue des accès mémoires si l'instruction exécutée en nécessite.
Voici un schéma simplifié du pipeline que l'on va modéliser :

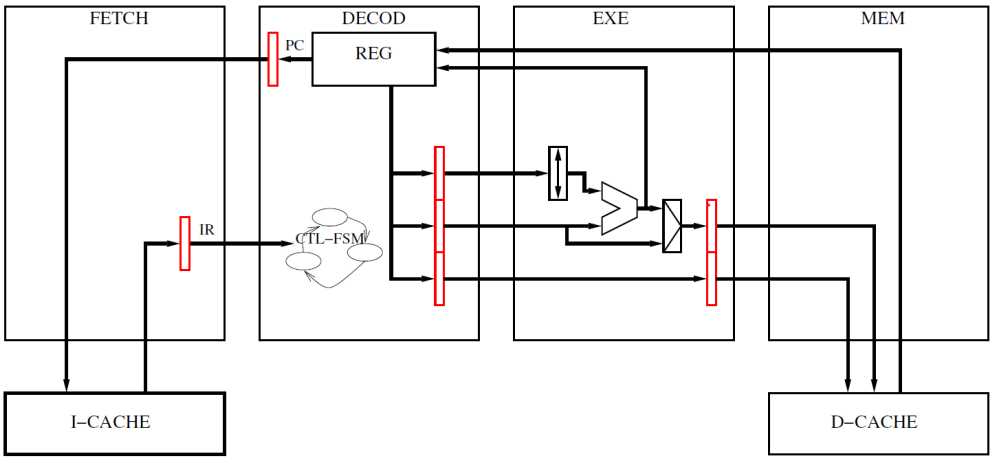

Figure 1: Schéma simplifié du pipeline

2 EXE

2.1 Introduction :

La première étape de notre modélisation a été l'étage EXE, c'est en effet le plus simple. Il est constitué de 2 parties primordiales : l'ALU (*arithemic logic unit*) et le shifter. Voici un schéma de son architecture :

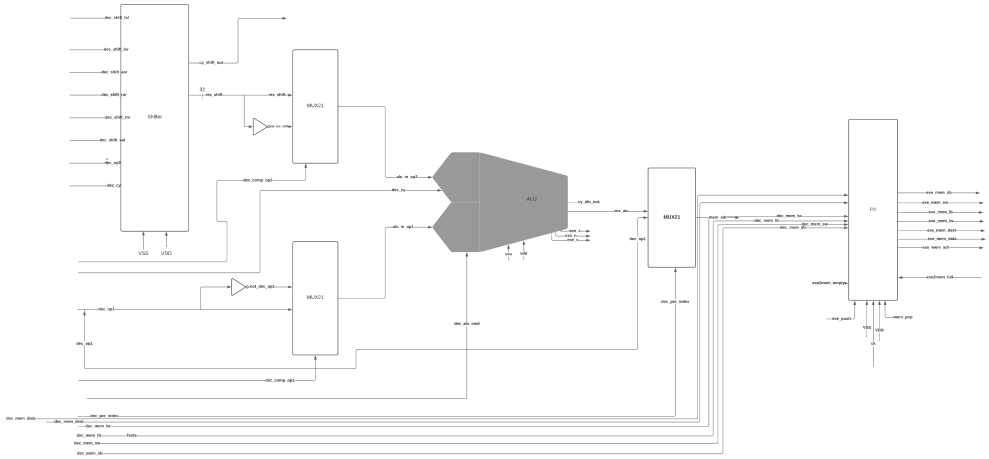

Figure 2: Etage EXE

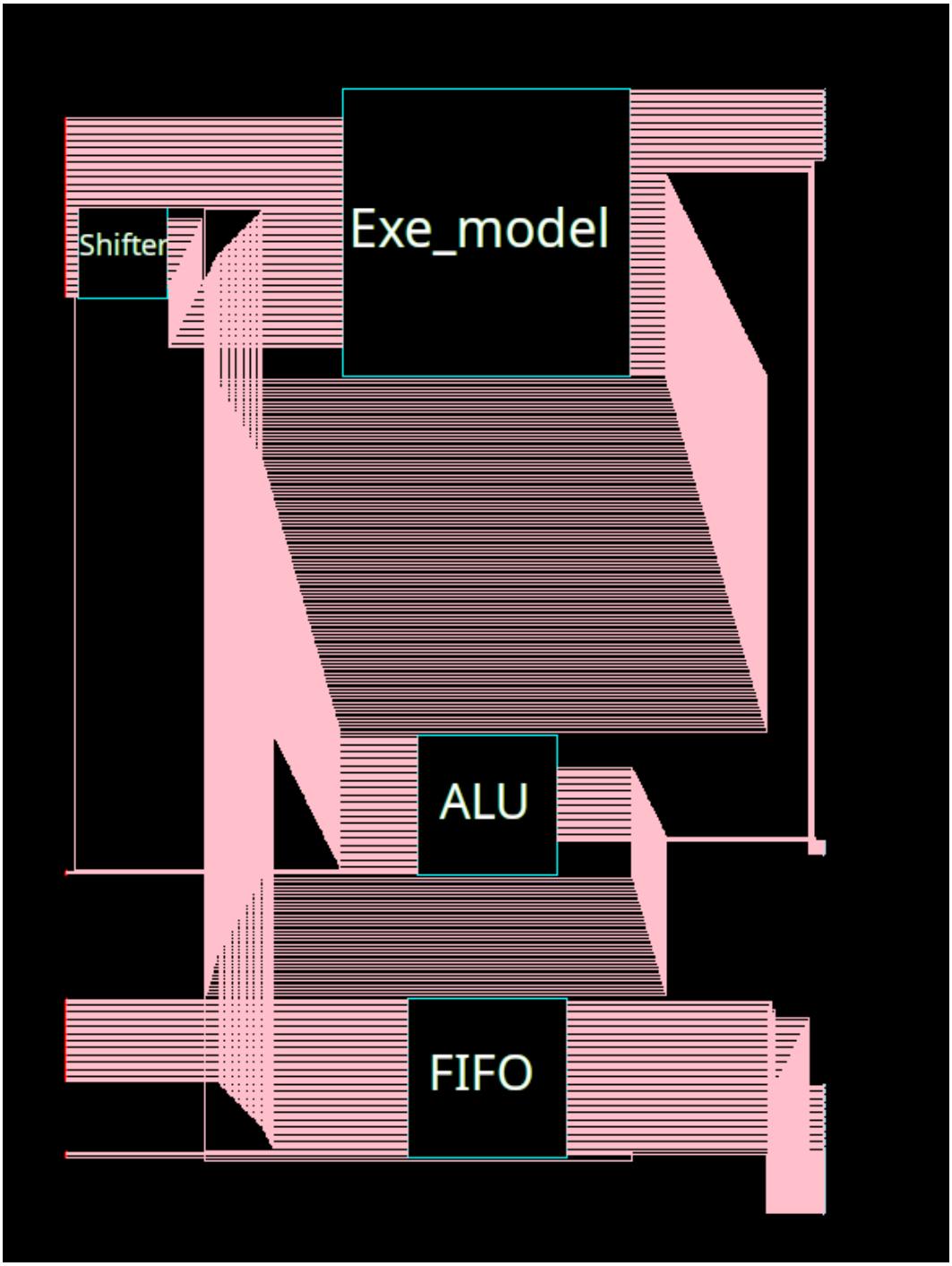

Figure 3: Etage EXE : circuit

2.2 ALU : Arithmetic logic Unit

La modélisation est assez simple, on envoie un signal de commande sur 2 bits à l'ALU et on sélectionne ainsi l'opération à exécuter : and, or, xor, add.
Pour effectuer ces opérations, nous avons utilisé les fonctions logiques fournis par le VHDL et une conversion d'entier signé afin de pouvoir utiliser l'opérateur "+". Dans l'architecture globale de EXE nous avons ajouté des inverseurs commandés afin d'être en mesure de faire le complément à 2 du signal et ainsi d'effectuer des soustractions.

2.3 Shifter

Le shifter est commandé par 4 bits indiquant le type de shift que l'on fait et 5 bits indiquant la valeur de shift.
Un shifter permet de faire des multiplications et des divisions par des puissances de 2. Un shift de 1 vers la droite est par exemple une division par 2 tandis qu'un shift de 1 vers la droite est une multiplication par 2.
Étant donné que l'on peut manipuler des entiers signés et non signés, on a besoin de shift conservant le signe du nombre calculé, c'est pourquoi un shift arithmetic est également nécessaire.
Pour réaliser notre shifter nous avons donc créé 3 entités : un shifter right, un shifter left et un shifter ror.
Pour chacun des types de shifter que nous avons créé le raisonnement est le suivant :
Si le bit n de la valeur de shift est à 1 cela signifie que l'on fait un décalage de 2^n.
Nous avons donc créé des entités effectuant des shits de 1,2,4,8 et 16 que l'on map en fonction de la valeur de shift que l'on veut.
En combinant ces shifters, on peut shifter n'importe quelle valeur entre 0 et 31.

2.3.1 Shifter right :

Cette entité va permettre de gérer les shifts right logique et arithmétique à l'aide d'une commande que l'on envoie en entrée de l'entité.
Les shifts sont simplement gérés via des concaténations. Par exemple pour un shift right logique de 1 on va récupérer le bit 0 que l'on stocke dans la carry et on décale tous les bits vers la droite en concaténant à gauche avec un zéro. Pour les shifts right arithmétiques la logique est la même, mais on garde le bit 31 à la même position afin de conserver le signe.

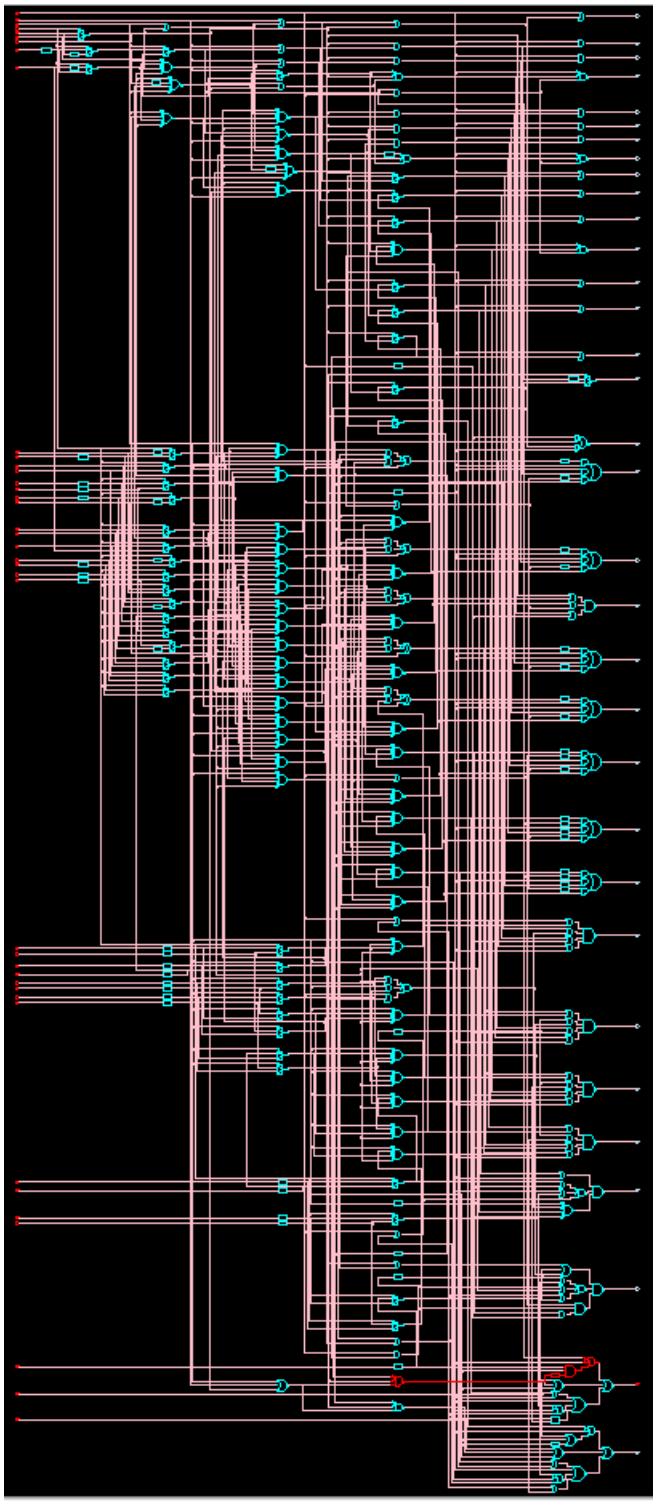

Figure 4: Circuit du shift right

2.3.2 Shift left

La mise en œuvre est exactement la même que pour le Shifter right à la différence près que la retenue n'est pas la même. En effet, dans ce cas, on ne prend pas le dernier bit de poids faible, mais le 1er bit de poids fort que l'on stocke dans la retenue.

2.3.3 Ror

Pour la rotation, on effectue une concaténation de n bits (n étant la valeur du shift value) vers la gauche.
Cette entitée permet également de gérer les shifts rrx, en effet un shift rrx n'est rien d'autre qu'un shift ror pour lequel la retenue est égale à 1. La gestion de cette retenue étant différente, on implémente directement la gestion de ce type de shift dans l'entité ror.

2.4 EXE Test Bench

Une fois toutes les entités mappées, on réalise un test bench pour l'ALU et pour le Shifter afin de vérifier que nos implémentations fonctionnent correctement.

3 DECOD :

L'étage DECOD est composé de deux parties principales : d'un côté, le banc de registre, et de l'autre, le décodage des instructions et le contrôle général du processeur.

3.1 REG : Registre Bank

Sur cette architecture ARM, le banc de registre fait partie de l'étage DECOD.

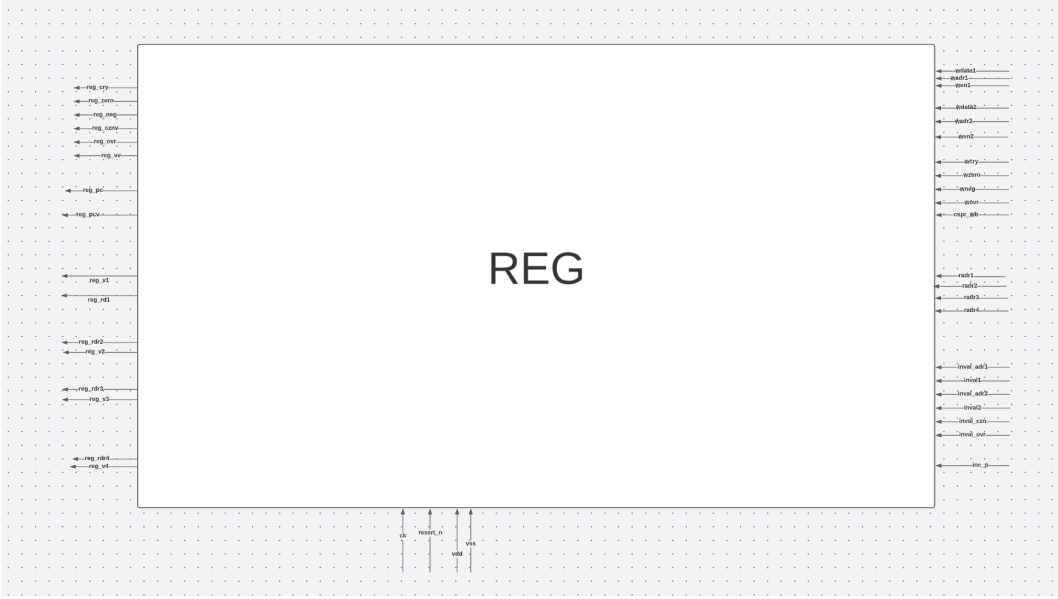

Figure 5: Banc de registre

Il contient 16 registres, ainsi que les flags C, Z, N et V. Le banc de registres peut gérer deux écritures (dont une prioritaire sur l'autre en cas d'adresse identique), 3 lectures 32-bits et une lecture 5-bits servant uniquement pour la lecture du registre codant la valeur d'un shift.
À chaque registre est associé un bit de validité. Un registre est invalidé par DECOD quand une instruction qui écrit dedans est lancée et redevient valide quand le résultat de l'instruction est écrit dans le registre.
Les 4 flags partagent un bit de validité.
En pratique, l'étage REG reçoit les registres à invalider en entrée et chaque registre redevient valide dès qu'une écriture est effectuée dessus. Les lectures sont accompagnées du bit de validité du registre correspondant.
En VHDL, notre banc de registre est représenté par un tableau de STD_LOGIC_VECTOR, et les bits de validité par un tableau de bits. Lors d'une lecture ou d'une écriture, les adresses sont converties en entiers pour être utilisées comme index pour les tableaux.
L'écriture dans les registres se fait à chaque front montant de l'horloge, en donnant bien priorité au premier registre écrit s'il y a un conflit d'adresse.

REG s'occupe aussi de gérer PC : si un flag de contrôle vaut 1, PC est incrémenté de 4, et sinon il garde sa valeur. On réalise la synthèse à l'aide des outils de la suite alliance et on affiche le résultat à l'aide de xsch. Voici un exemple de ce que l'on obtient lors de la synthèse de REG :

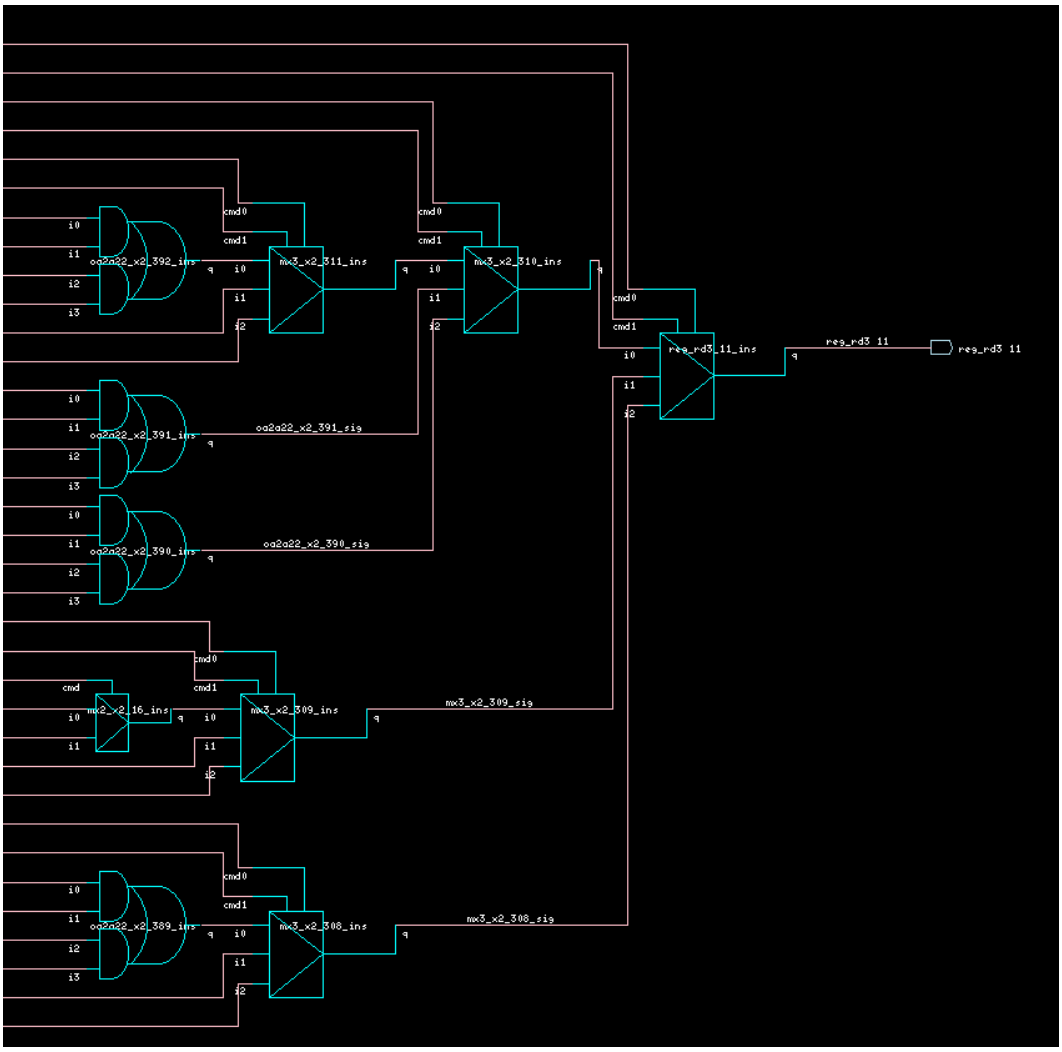

Figure 6: Lecture d'un bit dans un registre

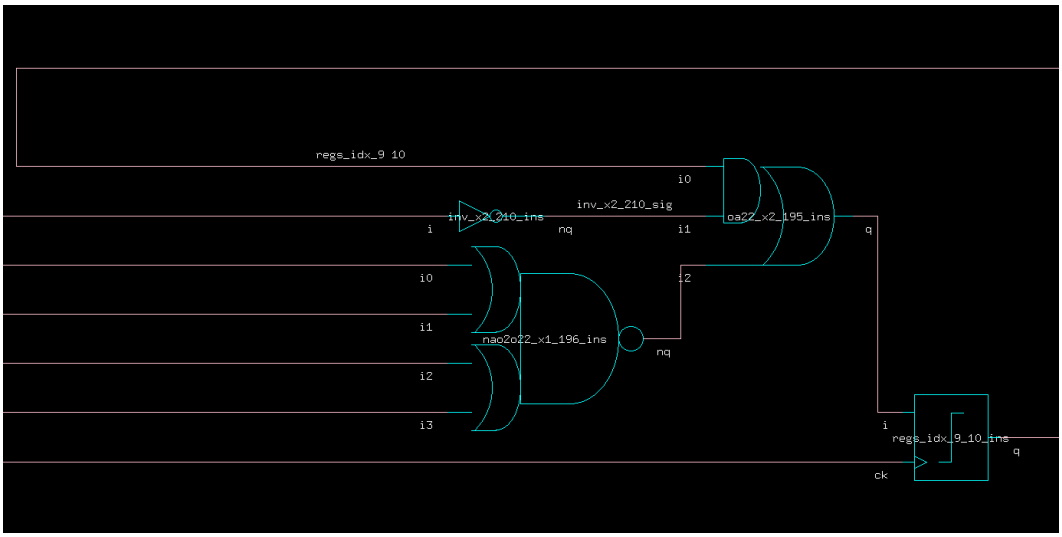

Figure 7: Écriture d'un bit dans un registre

4 Conclusion

Il nous reste à faire le branchement final des différents étages, et à réaliser un test bench pour vérifier que tout fonctionne correctement.
Nous avons réalisé une FIFO générique (donc d'une profondeur arbitraire), et même si les GENERIC ne sont pas synthétisables, il nous suffira de les remplacer par des entiers littéraux pour avoir une FIFO de longueur 4 ou 8 par exemple.