Proc-8-bits - Conception d'un processeur avec jeu d'instructions élémentaires - v1.0

David DEVANT - Aurélien TROMPAT

11 février 2019

Table des matières

Introduction

L'objectif de ce projet est de concevoir un processeur 8 bits disposant d'un jeu de 4 instructions élémentaires : Non-OU, Addition, Stockage mémoire, Saut d'adresse.

Il sera implémenté en VHDL dans l'optique de le déployer sur les cartes de développement Nexys 4 DDR.

1 Architecture

L'architecture de notre CPU est imposée par le cahier des charges. Elle se divise en 3 parties :

- Contrôleur (UC) : il est l'unité de contrôle qui génère des signaux internes de commandes.
- Calculateur (UT) : Il effectue des calculs sur des données d'entrée et retourne les résultats vers la mémoire.
- Mémoire : Elle stocke le programme et les variables.

2 Architecture CPU

Voici l'architecture de notre CPU tel qu'imposé par le cahier des charges :

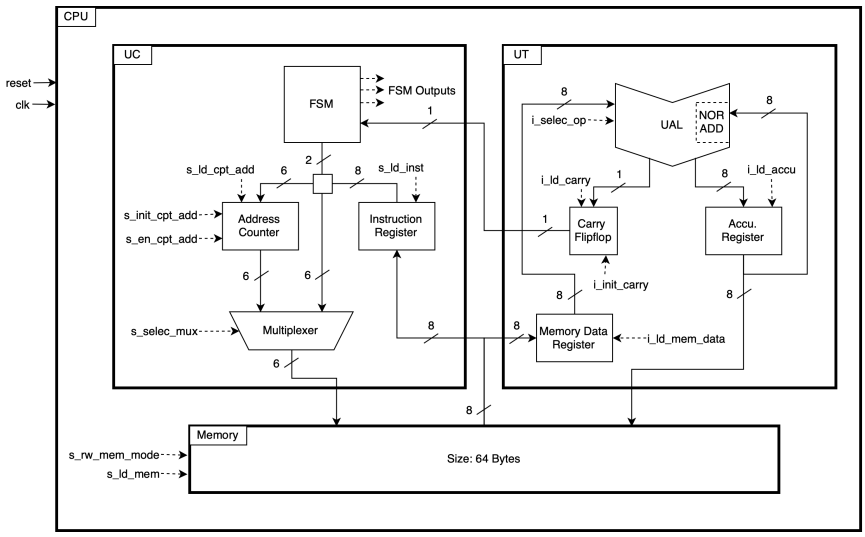

FIGURE 1 – Architecture du CPU (Clock et reset non représenté)

2.1 Contrôleur (UC)

2.1.1 Registre d'instruction

Ce registre 8 bits permet de stocker l'instruction qui est actuellement traitée par le CPU. Les instructions 8 bits sont issues de la mémoire.

2.1.2 Compteur d'adresse

Ce compteur permet de définir l'adresse mémoire de la prochaine instruction que devra exécuter le CPU. Un système de chargement permet de modifier la valeur courante du compteur pour permettre des sauts d'adresse (Instruction

JCC). Une entrée "init" permet de remettre à 0 le compteur pour redémarrer le programme à l'adresse 0.

2.1.3 Multiplexeur

Le multiplexeur est un composant purement combinatoire qui permet de sélectionner la source de l'adresse mémoire à lire/écrire. Si son signal de commande est à '0', alors c'est la sortie du compteur d'adresse qui est connecté à la mémoire. Ceci permet de lire les instructions du programme. Lorsque le signal de commande est positionné à '1', ce sont les 6 bits de poids faibles qui fixent l'adresse. Ceci permet d'effectuer les accès mémoire pour la partie calcul.

2.1.4 Machine d'état (FSM)

La FSM est l'ordonnanceur du CPU. Elle génère 12 signaux qui commandent les différents composants du CPU. Voici son schéma :

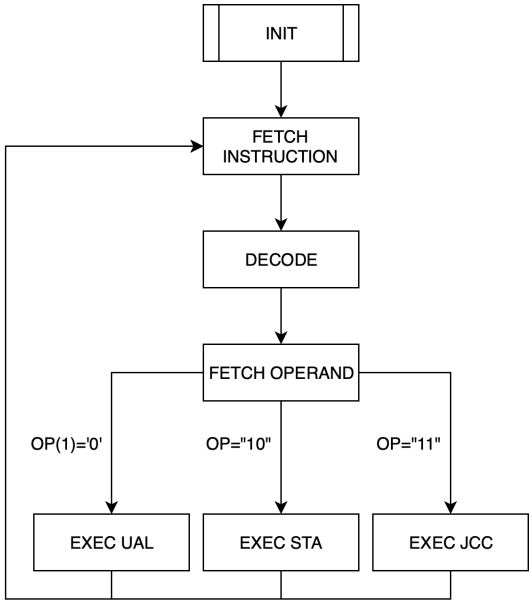

FIGURE 2 – Diagramme de la machine d'état du CPU

- INIT : Phase d'initialisation du CPU, cette étape n'est active qu'après un reset le temps d'une période d'horloge.
- FETCH INSTRUCTION : Chargement de l'instruction indiquée par le compteur d'adresse.
- DECODE : Décodage du code opération de l'instruction et anticipation du prochain chargement de mémoire.

- FETCH OPERAND : Chargement de la variable pointée par l'instruction de la mémoire vers la partie opérative (UT). Le choix de l'étape suivante est conditionné par le code d'opération.
- EXEC UAL (Code OP = "00" ou "01") : Défini l'opération à effectuer sur la partie opérative puis ordonne la sauvegarde du résultat dans le registre d'accumulation.
- EXEC STA (Code OP = "10") : Sauvegarde la valeur du registre d'accumulation en mémoire à l'adresse indiquée par l'instruction.
- EXEC JCC (Code OP = "11") : Effectue un test sur la valeur de la retenue pour déterminer si le saut d'adresse doit être exécuté. Le saut est effectué en plaçant les 6 bits de poids faible de l'instruction dans le compteur d'adresse grâce au système de chargement évoqué précédemment.

2.2 Contrôleur (UT)

2.2.1 Registre de données mémoire

Ce registre permet de stocker la dernière valeur lue en sortie de la mémoire. Il est directement lié à l'unité de calcul UAL.

2.2.2 Registre d'accumulation

Ce registre 8 bits stocke le résultat de la partie UAL. Sa valeur peut être envoyée vers la mémoire pour être sauvegardée ou bien réutilisée pour le calcul suivant.

2.2.3 Unité de calcul (UAL)

Cette unité de calcul dispose d'une entrée de sélection d'opération entre l'addition et le Non-OU. L'opération selectionnée est appliquée entre les deux valeurs 8 bits placées en entrée et le résultat est retourné vers le registre d'accumulation. Une seconde sortie permet de récupérer la retenue sur 1 bit pour être dirigée vers la flipflop. Notons que cette retenue n'est pas conservée après l'exécution d'une opération NOR.

2.2.4 Flipflop de retenue (Carry)

Cette bascule de type flipflop permet de stocker l'état de la retenue du dernier calcul de l'UAL. Sa sortie est utilisée par la FSM pour déterminer si le saut d'adresse de l'instruction JCC doit être effectué ou ignoré.

2.3 Mémoire

Cette mémoire de 64 octets est une mémoire simple accès avec un port de lecture et un second d'écriture. Elle est adressée sur 6 bits et manipule des valeurs sur 8 bits.

3 Tests

Chaque composant a été testé indépendemment à l'aide de "testbench". Une fois fonctionnels, nous les avons implémentés dans le top layer "CPU". Pour valider le projet, nous avons écrit les deux programmes de test vu en cours dans des fichiers "*.data". Ces fichiers sont chargés dans la mémoire juste avant la simulation. En observant les derniers accès mémoire du programme, on observe que les résultats sont cohérent avec les résultats attendus.

3.1 Additionneur N-bits

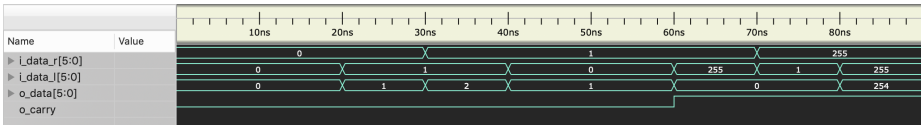

FIGURE 3 – Diagramme temporel de l'additionneur N-bits

Sur cette figure, on observe que les données "right" et "left" sont additionnées en sortie. On remarque que la carry est bien mise à '1' lorsque le résultat dépasse un codage sur 8 bits.

3.2 Flipflop

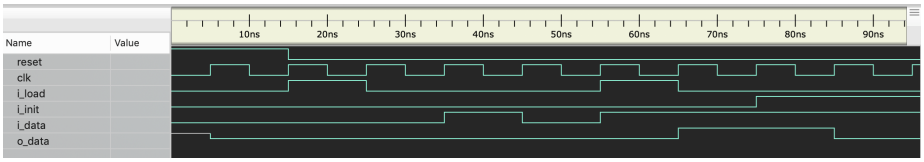

FIGURE 4 – Diagramme temporel de la flipflop de retenue

Ici nous avons un premier "load" qui recopie le '0' de "i_data" vers la sortie puis un changement sur "i_data" pour montrer que la sortie n'évolue pas sans la présence du signal "i_load". Enfin, nous avons un changement de la sortie à '1' car nous avons un chargement et un '1' en entrée. La bascule est finalement reset par le signal "s_init".

3.3 Multiplexeur à 2 entrées

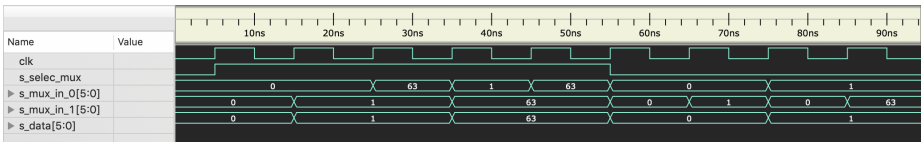

FIGURE 5 – Diagramme temporel du multiplexeur

Cette figure nous montre que les signaux "s_mux_in_0" et "s_mux_in_1" évolue en fonction du temps avec des valeurs sur 6 bits. Les extremums sont utilisés pour les bien du testbench. Le signal "s_selec_mux" permet de choisir quelle entrée recopier en sortie.

3.4 Unité de calcul

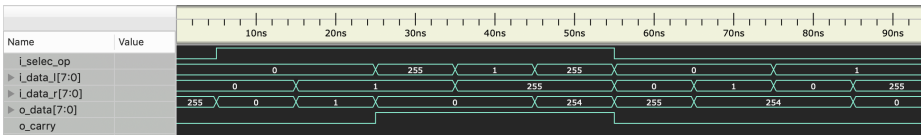

FIGURE 6 – Diagramme temporel de l'unité de calcul UAL

L'UAL dispose de deux opérations. L'addition a été testée dans un premier temps en définissant "i_selec_op" à '1' puis le Non-OU dans un second temps. Une fois de plus les entrées "left" et "right" prennent des valeurs spécifiques pour tester les limites du système.

On observe en premier lieu que la sortie est bien la somme des entrées avec la mise à jour de la retenue. Puis nous constatons que l'opération NOR fonctionne aussi.

3.5 CPU - Programme du PGCD

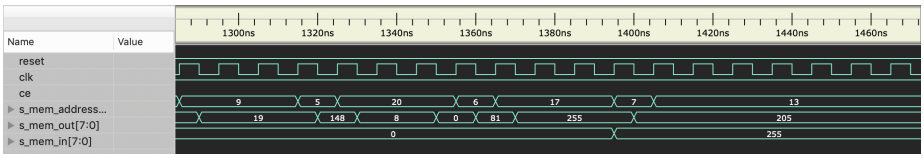

FIGURE 7 – Diagramme temporel du CPU avec programme de PGCD

Ce diagramme a été obtenu en combinant tous les blocs précédents. Il montre les accès mémoire lorsque le programme de calcul de PGCD arrive à sa fin. On observe que le signal "s_mem_out" passe de 8 à 0 aux alentours de 1340ns.

Ceci correspond à la dernière opération effectuée sur la variable 'a' à la fin de l'algorithme lorsque 'a' est soustrait pour la dernière fois. Les données de départ étant 40 et 24, notre résultat de 8 est donc cohérent.

3.6 FSM

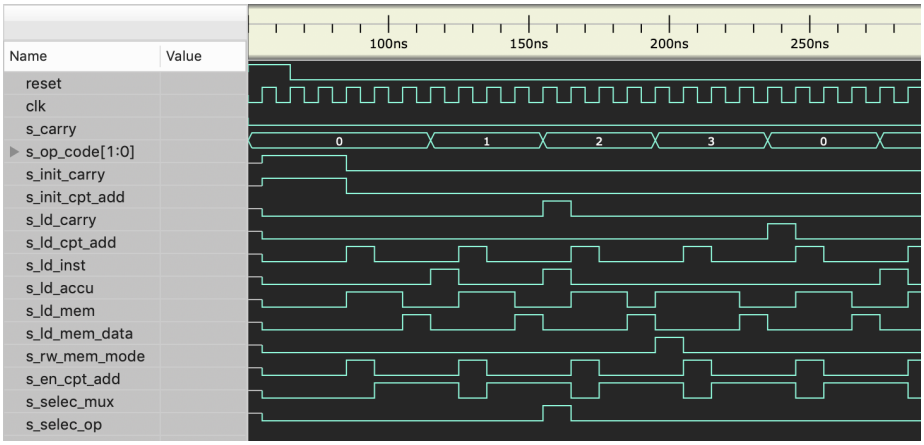

FIGURE 8 – Diagramme temporel de la FSM

Ce diagramme temporel nous montre l'évolution des signaux de sortie de la FSM en fonction des instructions reçues par le CPU. On remarque clairement que chaque instruction est exécutée en 4 périodes d'horloge.

4 Conclusion

Ce projet nous a fait découvrir les fonctionnements internes d'un processeur ainsi que les problématiques d'accès mémoire. L'implémentation en VHDL nous a permis de se remémorer les bonnes pratiques des langages de description. De plus, la réutilisation des nos anciens composants VHDL nous a montré qu'il était important de bien structurer et bien documenter nos fichiers pour être plus efficaces dans nos futurs projet.

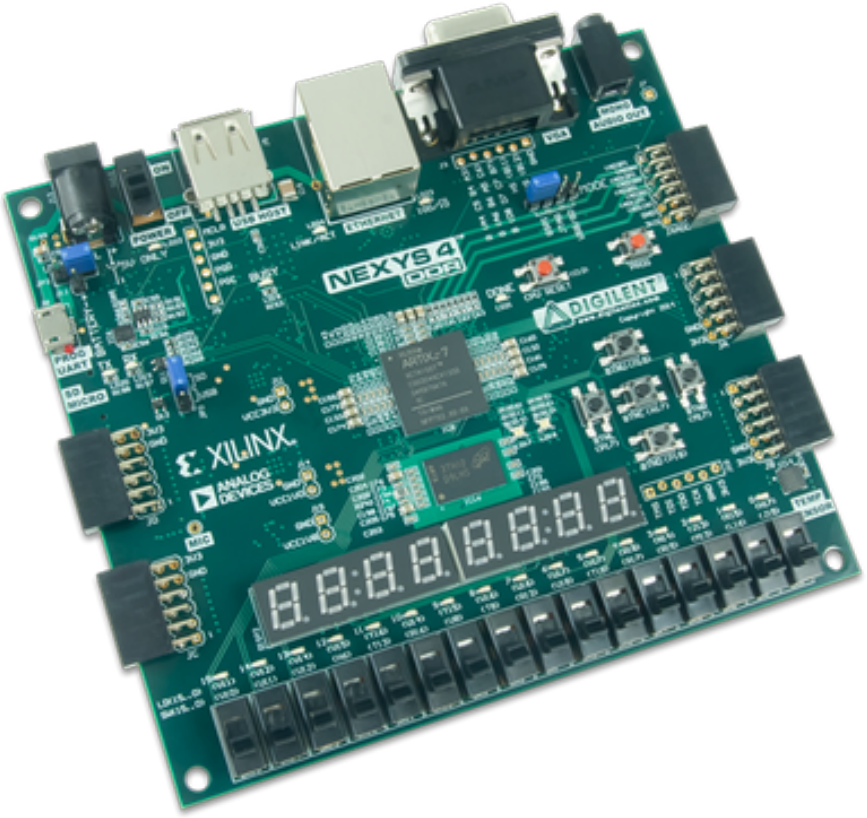

FIGURE 9 – Nexys 4 DDR