Rapport Architecture Systèmes

-

Processeur multi-cycle

Grégoire MAHON
Armand LELONG
EI2I-3 – II (Groupe B)

Introduction

Dans le monde dynamique de l'informatique, la conception et la mise en œuvre de processeurs sont d'une importance capitale. Les processeurs sont le cœur de tous les systèmes informatiques, exécutant les instructions qui permettent à nos ordinateurs, smartphones et autres appareils numériques de fonctionner. Dans le cadre de ce projet, nous avons exploré la conception et la mise en œuvre d'un processeur multi-cycle en utilisant le langage de description matériel VHDL (VHSIC Hardware Description Language).

Le VHDL est un langage de programmation utilisé pour décrire les systèmes numériques à différents niveaux d'abstraction. Il est largement utilisé dans l'industrie et l'académie pour la modélisation, la simulation et la synthèse de circuits numériques. Dans ce rapport, nous présentons notre travail sur la conception et la mise en œuvre d'un processeur multi-cycle en utilisant le VHDL.

Un processeur multi-cycle est un type de processeur où chaque instruction est exécutée sur plusieurs cycles d'horloge. Cela contraste avec les processeurs mono-cycle, où chaque instruction est exécutée en un seul cycle d'horloge. Les processeurs multi-cycle sont généralement plus complexes à concevoir que les processeurs mono-cycle, mais ils offrent une plus grande flexibilité et peuvent souvent atteindre une meilleure performance.

Il est important de noter que, contrairement à notre travail précédent sur le processeur mono-cycle, nous n'avons pas réalisé de testbenches pour ce processeur multi-cycle. Cela est dû à diverses contraintes, notamment le temps et les ressources. Cependant, nous avons fait de notre mieux pour vérifier la validité de notre conception à travers une analyse rigoureuse et une revue de code détaillée.

Multiplexeur 4 vers 1 - MUX_4to1.vhd

Lien vers le code du Multiplexeur 4 vers 1 (GitHub)

- *Entité : mux4to1*

Un multiplexeur, souvent abrégé en "mux", est un composant fondamental dans le domaine de l'électronique numérique. Il est utilisé pour sélectionner une ligne de données parmi plusieurs. Dans le cas du multiplexeur 4 vers 1 que nous examinons ici, il y a quatre lignes d'entrée et une ligne de sortie.

Le code VHDL fourni représente un multiplexeur 4 vers 1. Il prend quatre vecteurs de bits d'entrée de 32 bits (A, B, C, D) et un vecteur de bits d'entrée de 2 bits (COM). En fonction de la valeur de COM, il passe l'une des entrées à la sortie S.

La logique de sélection est la suivante :

- Si COM = "00", alors A est passé à S.
- Si COM = "01", alors B est passé à S.
- Si COM = "10", alors C est passé à S.
- Si COM = "11", alors D est passé à S.

Dans le cas où COM est une autre valeur, S est mis dans un état indéterminé (signalé par 'X').
Ce multiplexeur 4 vers 1 est un exemple parfait de la flexibilité et de la modularité en conception numérique. En utilisant le langage VHDL, nous pouvons créer des composants numériques qui sont à la fois flexibles et réutilisables. Le même code peut être utilisé pour créer des multiplexeurs qui manipulent des vecteurs de différentes tailles, ce qui permet de réutiliser le même code dans différentes parties d'un système numérique.

De plus, la conception du multiplexeur illustre l'importance de la modularité en conception de circuits numériques. Le multiplexeur est un composant simple qui effectue une seule tâche, mais il peut être combiné avec d'autres composants pour construire des systèmes numériques plus complexes. En concevant des composants modulaires qui peuvent être réutilisés et combinés de différentes manières, nous pouvons simplifier la conception de systèmes numériques complexes et rendre ces systèmes plus faciles à comprendre et à maintenir.

En conclusion, le multiplexeur 4 vers 1 est un composant essentiel dans de nombreux systèmes numériques. Sa conception et son fonctionnement sont un excellent exemple de l'efficacité et de la flexibilité de la conception numérique moderne.

Registre 32 bits - 32_Register.vhd

Lien vers le code du Registre 32 bits (GitHub)

- *Entité : reg32*

Un registre est un autre composant essentiel dans les systèmes numériques. Il est utilisé pour stocker des données qui peuvent être utilisées plus tard dans le système. Le code VHDL fourni représente un registre de 32 bits avec une fonction de réinitialisation.

Ce registre prend un vecteur de bits d'entrée de 32 bits (DATAIN), un signal d'horloge (CLK), et un signal de réinitialisation (RST). À chaque front montant de l'horloge, le contenu du registre est mis à jour avec la valeur de DATAIN. Lorsque le signal de réinitialisation est actif (haut, ou '1'), le registre est réinitialisé à 0.

La logique de fonctionnement est la suivante :

- Si RST = '1', alors DATAOUT est réinitialisé à 0.
- Si un front montant est détecté sur CLK, alors DATAOUT est mis à jour avec la valeur de DATAIN.

Ce registre de 32 bits est un exemple parfait de la manière dont les composants numériques peuvent être conçus pour être flexibles et réutilisables. En utilisant le langage VHDL, nous pouvons créer des composants numériques qui sont à la fois flexibles et réutilisables. Le même code peut être utilisé pour créer des registres de différentes tailles, ce qui permet de réutiliser le même code dans différentes parties d'un système numérique.

De plus, la conception du registre illustre l'importance de la modularité en conception de circuits numériques. Le registre est un composant simple qui effectue une seule tâche, mais il peut être combiné avec d'autres composants pour construire des systèmes numériques plus complexes. En concevant des composants modulaires qui peuvent être réutilisés et combinés de différentes manières, nous pouvons simplifier la conception de systèmes numériques complexes et rendre ces systèmes plus faciles à comprendre et à maintenir.

En conclusion, le registre de 32 bits est un composant essentiel dans de nombreux systèmes numériques. Sa conception et son fonctionnement sont un excellent exemple de l'efficacité et de la flexibilité de la conception numérique moderne.

Rapport sur la Mémoire 64x32 bits - 64x32_Memory.vhd

Lien vers le code de la RAM 64x32 (GitHub)

- *Entité : RAM_64x32*

La mémoire à accès aléatoire (RAM) est un autre composant essentiel dans les systèmes numériques. Elle est utilisée pour stocker des données qui peuvent être lues et écrites dans n'importe quel ordre, d'où le terme "accès aléatoire". Le code VHDL fourni représente une mémoire RAM de 64 mots de 32 bits.

Cette RAM prend une adresse de 6 bits (addr), un signal d'activation d'écriture (we), et une donnée d'entrée de 32 bits (DATAIN). À chaque front montant de l'horloge, si le signal d'activation d'écriture est haut, le mot à l'adresse spécifiée par 'addr' est mis à jour avec la valeur de 'DATAIN'. Indépendamment de l'opération d'écriture, à chaque cycle d'horloge, la valeur du mot à l'adresse 'addr' est sortie sur 'DATAOUT'.

La logique de fonctionnement est la suivante :

- Si un front montant est détecté sur CLK et que WE est '1', alors le mot à l'adresse spécifiée par ADDR est mis à jour avec la valeur de DATAIN.
- À chaque cycle d'horloge, la valeur du mot à l'adresse spécifiée par ADDR est sortie sur DATAOUT.

Cette RAM 64x32 est un exemple parfait de la manière dont les composants numériques peuvent être conçus pour être flexibles et réutilisables. En utilisant le langage VHDL, nous pouvons créer des composants numériques qui sont à la fois flexibles et réutilisables. Le même code peut être utilisé pour créer des mémoires RAM de différentes tailles, ce qui permet de réutiliser le même code dans différentes parties d'un système numérique.

De plus, la conception de la RAM illustre l'importance de la modularité en conception de circuits numériques. La RAM est un composant simple qui effectue une seule tâche, mais elle peut être combinée avec d'autres composants pour construire des systèmes numériques plus complexes. En concevant des composants modulaires qui peuvent être réutilisés et combinés de différentes manières, nous pouvons simplifier la conception de systèmes numériques complexes et rendre ces systèmes plus faciles à comprendre et à maintenir.

En conclusion, la RAM 64x32 est un composant essentiel dans de nombreux systèmes numériques. Sa conception et son fonctionnement sont un excellent exemple de l'efficacité et de la flexibilité de la conception numérique moderne.

Rapport sur le Contrôleur d'Interruption Vectorisé (VIC) - VIC.vhd

Lien vers le code du VIC (GitHub)

- *Entité : VIC*

Un Contrôleur d'Interruption Vectorisé (VIC) est un composant crucial dans les systèmes numériques, en particulier dans les systèmes embarqués et les microcontrôleurs. Il permet au programme principal d'être interrompu si l'une des deux demandes d'interruption (IRQ0 et IRQ1) est active. Le code VHDL fourni représente un tel contrôleur d'interruption vectorisé.

Le VIC prend deux signaux d'interruption (IRQ0 et IRQ1), un signal d'horloge (CLK), un signal de réinitialisation (RESET) et un signal d'acquittement d'interruption (IRQ_SERV). Il échantillonne les valeurs de IRQ0 et IRQ1 pour connaître les valeurs actuelles et précédentes de ces deux signaux. Si un front montant est détecté sur l'un de ces signaux, il met un signal correspondant IRQ0_memo ou IRQ1_memo à haut, signalant une demande d'interruption.

La logique de fonctionnement est la suivante :

- Si RESET = '1', alors IRQ0_prev, IRQ1_prev, IRQ0_memo et IRQ1_memo sont réinitialisés à '0'.
- Si un front montant est détecté sur CLK, alors IRQ0_prev et IRQ1_prev sont mis à jour avec les valeurs actuelles de IRQ0 et IRQ1 respectivement.
- Si un front montant est détecté sur IRQ0, alors IRQ0_memo est mis à '1'.
- Si un front montant est détecté sur IRQ1, alors IRQ1_memo est mis à '1'.
- Si IRQ_SERV = '1', alors IRQ0_memo et IRQ1_memo sont réinitialisés à '0'.
- Le signal d'interruption IRQ est la somme logique OR de IRQ0_memo et IRQ1_memo.
- Si IRQ0_memo = '1', alors VICPC est mis à 0x9.
- Si IRQ1_memo = '1', alors VICPC est mis à 0x15.
- Si aucune interruption n'est demandée, alors VICPC est forcé à 0.

Ce contrôleur d'interruption vectorisé est un exemple parfait de la manière dont les composants numériques peuvent être conçus pour être flexibles et réutilisables. En utilisant le langage VHDL, nous pouvons créer des composants numériques qui sont à la fois flexibles et réutilisables. Le même code peut être utilisé pour créer des contrôleurs d'interruption vectorisés avec différentes configurations, ce qui permet de réutiliser le même code dans différentes parties d'un système numérique.

De plus, la conception du VIC illustre l'importance de la modularité en conception de circuits numériques. Le VIC est un composant simple qui effectue une seule tâche, mais il peut être combiné avec d'autres composants pour construire des systèmes numériques plus complexes. En concevant des composants modulaires qui peuvent être réutilisés et combinés de différentes manières, nous pouvons simplifier la conception de systèmes numériques complexes et rendre ces systèmes plus faciles à comprendre et à maintenir.

Conclusion

Au cours de ce projet, nous avons acquis une compréhension approfondie de la conception et de la mise en œuvre de processeurs multi-cycle. Nous avons appris à utiliser le langage VHDL pour décrire des systèmes numériques à différents niveaux d'abstraction, et nous avons acquis une expérience pratique de la conception de composants numériques tels que les multiplexeurs, les registres, la mémoire RAM et les contrôleurs d'interruption vectorisés.

Nous avons également appris l'importance de la modularité en conception de circuits numériques. En concevant des composants modulaires qui peuvent être réutilisés et combinés de différentes manières, nous avons pu simplifier la conception de systèmes numériques complexes et rendre ces systèmes plus faciles à comprendre et à maintenir.

En outre, nous avons appris à vérifier la validité de notre conception à travers une analyse rigoureuse et une revue de code détaillée. Bien que nous n'ayons pas réalisé de testbenches pour ce processeur multi-cycle en raison de diverses contraintes, nous avons fait de notre mieux pour assurer la qualité de notre travail.

En somme, ce projet nous a permis d'acquérir des compétences précieuses en conception de systèmes numériques et en programmation VHDL. Nous sommes convaincus que ces compétences nous seront utiles dans nos futures carrières en ingénierie.

Sur le plan technique, nous avons approfondi notre compréhension des différences fondamentales entre les architectures de processeurs mono-cycle et multi-cycle. Nous avons appris comment les processeurs multi-cycle, grâce à leur capacité à exécuter chaque instruction sur plusieurs cycles d'horloge, peuvent offrir une plus grande flexibilité et souvent une meilleure performance que les processeurs mono-cycle.

Malheureusement, en raison du temps et des ressources limités, et de notre concentration sur le processeur mono-cycle, nous n'avons pas été en mesure de terminer entièrement le processeur multi-cycle. Cependant, le travail que nous avons accompli nous a permis d'acquérir une solide compréhension de la technologie des processeurs multi-cycle et de la manière dont ils sont conçus et mis en œuvre.

En conclusion, bien que nous n'ayons pas été en mesure de terminer le processeur multi-cycle, nous avons tiré de précieuses leçons de ce projet. Nous avons acquis une expérience pratique de la conception de systèmes numériques et nous avons approfondi notre compréhension de la technologie des processeurs multi-cycle. Nous sommes convaincus que ces compétences et ces connaissances nous seront d'une grande utilité dans nos futures carrières en ingénierie.