ELE8307 – Laboratoire 3
Spécialisation du processeur Nios II

Introduction

Dans bon nombre d'applications, l'utilisation de processeurs à usage général est insuffisante à l'atteinte des objectifs fixés, en termes de performance comme en termes de consommation d'énergie. Bien qu'il soit possible d'utiliser un coprocesseur ou bien du matériel dédié, une solution intéressante est tout simplement de spécialiser le processeur à usage général avec des instructions matérielles dédiées qui viennent augmenter l'ALU typique du processeur.

Il s'agit ensuite de modifier le code de l'algorithme afin de faire usage des instructions spécialisées plutôt que des instructions classiques. Avec des instructions spécialisées, on conserve l'avantage d'avoir une application décrite en logiciel (évolutivité, facilitée de débogage). D'autre part, puisque l'on accède aux données depuis le processeur et que celui-ci assure une grande partie du contrôle, le travail de conception/vérification matérielle s'en trouve grandement simplifié.

Objectifs

À la fin de ce laboratoire, l'étudiant sera en mesure de spécialiser le processeur Nios II afin d'accélérer une application avec les types d'instructions suivantes:

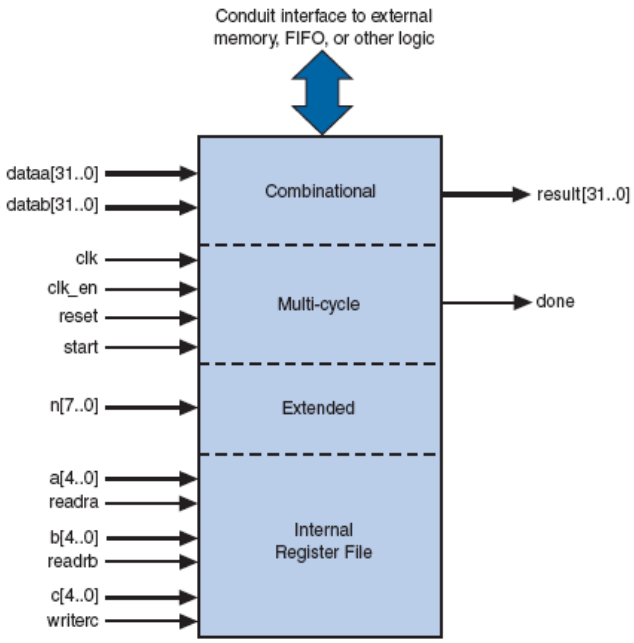
- ✓ Utilisation des instructions spécialisées pour l'arithmétique flottante
- ✓ Conception d'instructions spécialisées ayant une interface externe
- ✓ Conception d'instructions spécialisées combinatoires, multi-cycles et étendues

Documentation

La documentation de référence pour la conception d'instructions spécialisées sur le Nios II est disponible sur Moodle (NIOS II custom instructions). Les explications qui suivent en présentent un résumé, mais il est néanmoins fortement conseillé de bien lire et comprendre ce document qui explique tout le fonctionnement des instructions spécialisées et vous permettra de bien réaliser les futurs laboratoires.

Les instructions spécialisées du processeur Nios II sont juxtaposées à l'ALU dans le chemin de donnée. Ainsi, les instructions prennent 2 arguments de 32bits et produisent un résultat de 32bits. Le processeur Nios II supporte jusqu'à 256 instructions spécialisées.

Il existe 4 types d'instructions spécialisées : les instructions combinatoires, les instructions multi-cycles, les instructions étendues, et les instructions à accès au registre (qui ne seront pas traitées dans ce laboratoire). La figure suivante illustre les différents ports des instructions spécialisées en fonction de leur type. Pour chaque type d'instruction, il existe des modèles en VHDL qui peuvent être utilisés comme point de départ. Vous pouvez en éditez les ports afin de ne conserver que ce qui est requis, et ajouter l'implémentation matérielle voulue.



Les instructions combinatoires

Les instructions combinatoires produisent leur résultat en moins d'un cycle, à partir d'arguments dataA et dataB. La table suivante présente les ports de ce type d'instruction :

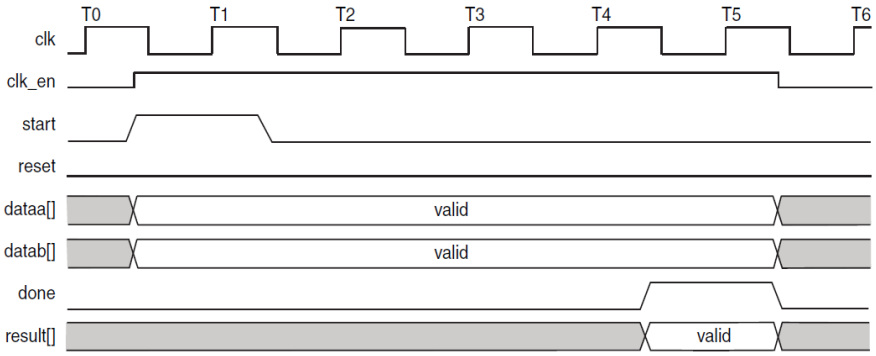
Port	Requis	Description
dataA(31 downto 0)	Non	Opérande A de l'instruction
dataB(31 downto 0)	Non	Opérande B de l'instruction
result(31 downto 0)	Oui	Résultat de l'instruction

Les instructions multi-cycles

Pour différentes raisons, il peut être très intéressant d'implémenter des instructions multi-cycles afin de réaliser des opérations de traitement plus complexes. Il faut alors gérer les signaux de contrôle start et done de l'interface lorsque le nombre de cycles est variable. Il est aussi possible de spécifier le nombre de cycles d'exécution à la génération de l'instruction.

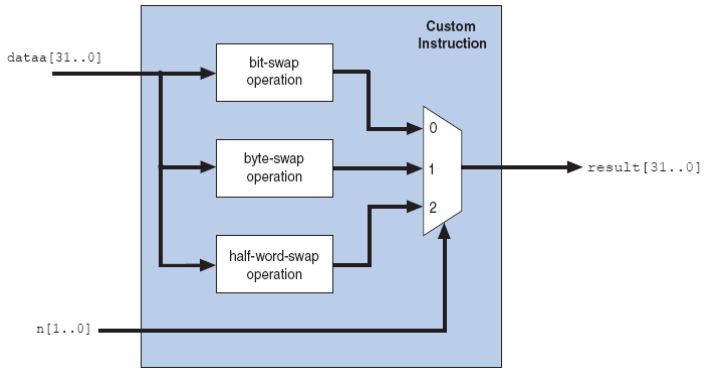
Port	Requis	Description
dataA(31 downto 0)	Non	Opérande A de l'instruction
dataB(31 downto 0)	Non	Opérande B de l'instruction
Result(31 downto 0)	Non	Résultat de l'instruction
clk	Oui	Horloge à la fréquence du processeur.
clk_en	Oui	Clock Enable.
reset	Oui	Reset de l'instruction
start	Non	Indique le début d'un appel d'instruction
done	Non	Indique la fin d'une instruction et la validité du résultat produit.

Voici un chronogramme expliquant le fonctionnement des instructions multi-cycles :



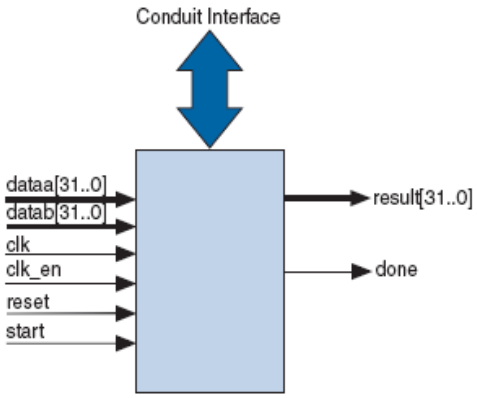
Les instructions étendues

Il est possible de créer une unité de calcul complexe réalisant plusieurs fonctions, que l'on sélectionne à l'aide du port n de 8 bits. Le port n agit comme sélecteur d'entrées d'un multiplexeur consistant en différent résultats possibles. Par exemple, sur la figure suivante, nous faisons 3 opérations différentes sur la même entrée, en fonction de la valeur de n.



Les instructions avec une interface externe

Parfois, il peut être intéressant qu'une instruction spécialisée puisse accéder à du matériel extérieur au processeur. Ceci est rendu possible par l'ajout d'une interface nommée Conduit. Lorsque l'on utilise un conduit, on doit lui spécifier l'interface d'horloge du système lors de la génération de l'extension d'instructions. On assigne tous les signaux qui doivent entrer ou sortir du système à l'interface Conduit. La figure suivante montre le schéma de l'utilisation d'une interface externe.



Travail à réaliser

Pour cette séance, nous repartirons du projet de simulateur de système solaire du laboratoire précédent. Copiez le dossier du projet du laboratoire précédent pour en garder une sauvegarde.

Ce laboratoire se divise en 3 parties qui seront des accélérations successives de l'application. Nous aborderons d'abord l'unité arithmétique pour nombres à virgule flottante disponible pour le Nios II. Nous concevrons ensuite des instructions spécialisées, d'abord pour l'affichage VGA et ensuite pour accélérer le calcul matriciel.

Partie 1 - Instructions spécialisées pour le calcul en point flottant

Il existe déjà une unité de calcul pour les nombres en point flottant pouvant être ajoutée au jeu d'instructions du processeur Nios II. Pour l'ajouter, dans ***Library -> Embedded Processors -> Floating Point Hardware***. Il faudra ensuite générer le système SOPC et recompiler le design matériel, avant de recompiler l'application logicielle. Faites un test et vous observerez une accélération obtenue très facilement !

Partie 2 - Instructions spécialisées pour l'affichage VGA

Dans cette partie nous allons tester deux méthodes pour accélérer l'accès à l'emplacement mémoire de chaque pixel.

1. Calcul d'adresse

Dans un premier lieu vous allez produire une instruction spécialisée combinatoire qui calcule en un seul cycle l'adresse d'un pixel (x, y) dans le buffer mémoire. L'adresse est donnée par :

$$ADDR = VGA\_0\_BASE/4 + x + 640*y$$

Vous devez implémenter ce calcul en utilisant seulement des décalages (**pas de multiplieur**). Écrivez votre code VHDL à partir d'un modèle (les modèles pour les quatre types d'instructions sont disponibles sur Moodle).

Une fois l'instruction écrite, dans SOPC Builder, ouvrez la configuration du processeur Nios II, et dans l'onglet « Custom Instructions », cliquez sur « Import… » (Il est possible que l'on vous demande d'insérer un nouveau composant avant cela, suivez les instructions). Vous serez alors invité, dans l'onglet « HDL Files », à spécifier le fichier VHDL et l'entité de haut niveau. Dans le panneau « Signals », il faut associer chaque signal à la bonne interface, « Nios II Custom Instruction Slave », qu'il vous faudra d'abord créer. Attention de ne la créer qu'une seule fois et de la réutiliser ensuite (**ne pas produire de nouvelle interface pour chaque signal**). Pour les instructions spécialisées Nios II, il faut faire la correspondance des signaux de l'interface d'extension d'instructions.

Dans l'onglet « Interfaces », vous pouvez enlever les interfaces sans signaux (bouton au bas de la fenêtre). Une fois terminé, vous pouvez ajouter l'instruction au processeur comme il a été fait avec l'unité arithmétique flottante (si votre instruction n'est pas là, fermez la configuration du processeur et ouvrez la fenêtre de nouveau). Lorsque vous ajoutez une instruction au processeur NIOS II, SOPC Builder peut signaler une erreur du type « Failed to analyze port width ». C'est un bug du logiciel, vous ne pouvez alors pas générer votre système. Il faut fermer (et sauvegarder) SOPC Builder et le ré-ouvrir. Il vous sera alors possible de générer votre système. N'oubliez pas de recompiler et de remettre sur la carte. Vous pourrez voir la macro d'appel de fonction dans le fichier system.h après régénération du BSP et de votre projet. Vérifiez que l'adresse obtenue est valide en testant le système. Votre code C pour la fonction ecran2d_setPixel devra maintenant être :

```
void ecran2d_setPixel( int x, int y, int color ) {
    int addr = ALT_CI_CALC_PIXEL_ADDR_INST(x,y); //nom provenant de system.h
    IOWR(0, addr, color);
}
```

2. Contrôle direct du contrôleur VGA

Pour cette partie, le contrôleur VGA sera retiré du système SOPC Builder et sera instancié dans un bloc séparé du système. Pour ce faire il vous faudra modifier votre système SOPC, en enlevant le composant VGA (n'oubliez pas de refaire System → Assign Base Addresses). Vous allez ensuite instancier le contrôleur VGA à l'extérieur du système SOPC. Pour y arriver, ouvrez (dans Quartus II) le fichier vga.vhd qui se trouve dans le répertoire du composant, et ajoutez-le au projet :

Project → Add Current File To Project

Ensuite générez le symbole :

File → Create/Update → Create Symbol Files for Current File

Vous pouvez alors ajouter le composant au fichier schématique de la même manière que vous avez ajouté votre système SOPC. Ainsi votre instruction spécialisée pourra/devra s'interfacer directement avec le contrôleur VGA. Il ne sera plus possible d'accéder au contrôleur VGA via le bus Avalon à l'aide de la fonction IOWR( ).

Branchez le reset sur le même bouton que pour votre SOPC. (N'oubliez pas de mettre une porte « not »!) Branchez également les horloges (50MHz et 25MHz) ainsi que les sorties. Vous devriez maintenant avoir les entrées addr, data, wr et la sortie busy de libre. Afin de communiquer avec le contrôleur VGA, nous utiliserons une interface externe avec une instruction spécialisée. Il faudra donc s'interfacer avec les signaux wr, busy, addr et data du contrôleur VGA. Notez que dans ce cas il n'y a pas d'adresse de base (VGA_BASE_0) et l'espace adressable commence à l'adresse 0 pour le premier pixel. L'adresse devient donc :

$$ADDR = x + 640*y$$

Commencez par faire une instruction multi-cycle setPixel() qui permettra de fixer la couleur d'un pixel à l'écran. Utilisez le fichier déjà préparé pour vous (**setpixel.vhd**). Puisque les coordonnées (x,y) de l'écran requièrent moins de 32bits, nous utiliserons les bits (15..0) de dataA pour X et les bits (31..16) pour Y. DataB contiendra la couleur du pixel. Notez qu'une instruction multi-cycle est requise pour gérer le signal busy. Modifiez le code de l'application pour remplacer les appels à IOWR par des appels à votre instruction spécialisée.

Lorsque vous brancherez les entrées/sorties, vous devrez reproduire la figure suivante :

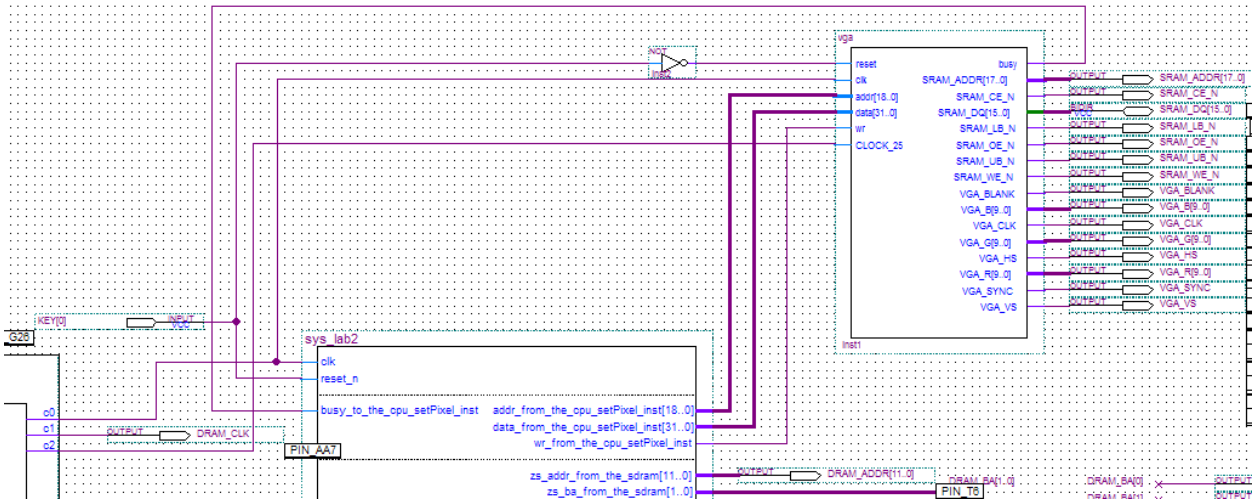
File Templates

Component Type | Files | Parameters | Signals | Interfaces

About Signals

Name	Interface	Signal Type	Width	Direction
addr	conduit_end	export	19	output
data	conduit_end	export	32	output
wr	conduit_end	export	1	output
busy	conduit_end	export	1	input
result	nios_custom_instruction_slave	result	32	output
datab	nios_custom_instruction_slave	datab	32	input
dataa	nios_custom_instruction_slave	dataa	32	input
done	nios_custom_instruction_slave	done	1	output
start	nios_custom_instruction_slave	start	1	input
clk_en	nios_custom_instruction_slave	clk_en	1	input
reset	nios_custom_instruction_slave	reset	1	input
clk	nios_custom_instruction_slave	clk	1	input

Également, voici ce à quoi devrait ressembler les changements dans le fichier .bdf :



Lorsque vous serez dans Éclipse, il vous faudra, encore une fois, faire appel à votre nouvelle instruction spécialisée dans la fonction ecran2d_setPixel. Assurez-vous de passer les bons paramètres ((y<<16)+x, color).

Partie 3 - Tracé de lignes obliques

La nouvelle configuration du contrôleur VGA externe permet d'offrir de nouvelles fonctions dans lesquelles les opérations sont parallélisées, plutôt que de laisser le processeur gérer les séquences d'opérations à effectuer. On peut par exemple tracer des primitives entières, ce qui peut prendre un certain temps, mais sans bloquer le processeur. Dans cette optique, transformez votre instruction multi-cycles en instructions multi-cycles étendue de sorte que l'affichage d'un pixel soit sélectionné par n=0. Pour n=1, une ligne oblique soit tracée entre les points (x1, y1) et (x2, y2).

Inspirez-vous de l'algorithme séquentiel utilisé pour tracer les orbites pour concevoir votre machine à état. Dans ce cas, nous aurons dataA qui contient le point (x1, y1), et dataB qui contient le point (x2, y2) et la couleur. La fonction à implémenter existe déjà en logiciel dans le fichier ss_orbit.c et se nomme ss_orbit_line_draw. Afin de

vous aider pour faire la transposition vers le matériel, vous allez la modifier un peu. Transmettez les arguments par valeur (x1, y1, x2, y2, color) et assurez-vous de ne plus faire appel aux fonctions abs et swap. Appelez votre nouvelle fonction dans ss_orbit_draw. Profilez la fonction logicielle avant d'en faire l'implémentation matérielle. **Pour cette partie, vous devez essayer d'utiliser le moins d'états possibles.** N'oubliez pas que vous devez toujours prendre en compte le busy lors de l'écriture vers le contrôleur VGA. Lorsque vous aurez complété cette instruction, écrivez un petit programme (tracer_lignes_main.c) qui trace plusieurs lignes et comparez les temps d'exécution entre les versions avec et sans l'instruction spécialisée pour le tracé de la ligne oblique.

Instructions pour la Remise

Voir :

- ✓ plan de cours
- ✓ page d'informations sur Moodle

Vous devez remettre votre rapport ainsi que votre dossier de projet (contenant tous les fichiers projets !) compressé. Envoyez le tout sur la page Moodle du laboratoire, sur laquelle vous pourrez répondre à deux questions très courtes (facultatif, cela nous aidera à améliorer les laboratoires pour les années suivantes).

N'OUBLIEZ PAS DE SAUVEGARDER VOTRE TRAVAIL

ET DE NETTOYER LE DISQUE TEMPORAIRE D !

IL EST ACCESSIBLE À TOUT LE MONDE

ET EST NETTOYÉ TOUTES LES NUITS !

Grille d'évaluation

Ce laboratoire compte pour 6% de la session :
Partie 1 : 0.5/6
Partie 2 : 2/6
Partie 3 : 3.5/6
Vous devez obligatoirement commenter votre code VHDL !