

# ELE8307 – Laboratoire 2

## Introduction au processeur Nios II

---

### Introduction

L'utilisation de processeurs embarqués vient faciliter considérablement le prototypage rapide de circuits numériques. En effet, pour un système d'une complexité donnée, l'effort de conception et de vérification associée à une implémentation logicielle est nettement moindre que celui associé à une implémentation au moyen de matériel dédié. Cette facilité a toutefois un coût, la performance. Trop souvent, une application donnée ne peut pas être exécutée assez rapidement par une implémentation purement logicielle. D'un autre côté, les contraintes de temps, deadlines, etc. rendent souvent impossible une implémentation complète avec un circuit dédié. Le compromis consiste donc à concentrer les efforts sur les points chauds de l'algorithme pour lesquels des circuits spécialisés peuvent améliorer grandement les performances. La loi d'Amdahl explique que pour accélérer une application il faut s'attaquer aux sections qui consomment le plus de temps, et pour y parvenir, une bonne méthode consiste à utiliser des outils de profilage. Mais comme la conception matérielle requiert un effort supérieur, il est important de tirer d'abord le maximum possible du processeur au niveau logiciel. Après tout, peut-être que quelques optimisations logicielles seraient suffisantes à obtenir les performances requises !

Le processeur Nios II est un « soft processor », c'est-à-dire qu'il est implémenté à même les ressources disponibles sur les FPGA d'Altera. Grâce à l'outil Qsys, l'intégration de ce processeur avec différents périphériques est relativement simple, le système d'interconnexion étant généré automatiquement.

### Objectifs

À la fin de ce laboratoire, l'étudiant aura développé diverses compétences liées à l'utilisation d'un processeur dans un système embarqué :

- ✓ Conception d'un système embarqué avec Qsys
- ✓ Connaissance du processeur Nios II, son jeu d'instruction, ses registres, ...
- ✓ Utilisation du GNU Profiler pour le processeur Nios II
- ✓ Optimisation de code en langage machine (assembleur)
- ✓ Utilisation du debugger de l'environnement Eclipse pour le processeur Nios II

Afin d'atteindre ces objectifs, une application logicielle écrite pour le processeur Nios II vous est fournie. L'application modélise le système solaire sur un écran VGA dans le but (caché) de vérifier les prédictions de fin du monde à la fin de la session ! Le problème est que l'exécution est très lente. Pour prévoir l'avenir avant que celui-ci ne se produise, on aimerait bien accélérer cette application au maximum.

### Documentation

Deux documents sont pertinents à ce laboratoire (disponibles sur Moodle) :

- Tutoriel NIOS II : Introduction au processeur Nios II (environ 20 pages).
- Jeu d'instructions NIOS II : Guide de référence du jeu d'instruction du Nios II. À consulter au besoin pour connaître le détail d'une instruction assembleur en particulier.

## Survol rapide du Nios II

Le processeur Nios II est configurable, on peut ainsi choisir parmi 3 configurations de bases, soit la version économie (NiosII/e), standard (NiosII/s), ou performance (NiosII/f). Un compromis entre performance et utilisation de ressources est donc à portée de clic de souris. Toutefois, les versions moins performantes supportent moins d'options additionnelles, telles que l'utilisation d'une cache de données.

Le processeur est basé sur une architecture RISC (Reduced Instruction Set Computer) et fonctionne avec des mots de 32bits en mode d'adressage little-endian. Ce processeur contient également 32 registres de 32bits dont certains sont réservés pour usages spécifiques (pointeur de pile, de fenêtre, adresse de retour d'une fonction, ...). L'espace des adresses (32bits) est adressable par octet, l'architecture des instructions supporte 5 modes d'adressage différents (valeur immédiate, registres, adressage indirect d'un registre, ...), et il existe 3 formats d'instruction.

Il existe 11 catégories d'instructions, dont les principales sont celles de lecture/écriture, celles pour les opérations arithmétiques et logiques, celles de comparaisons, et celles de branchements et d'appel de fonctions. Il est important de savoir qu'en présence d'une cache, les instructions de chargement et d'écriture en mémoire passent par cette dernière. Afin de l'éviter, il est possible d'utiliser les instructions d'écriture et de lecture aux entrées et sorties (I/O).

Le compilateur du Nios II supporte également certaines directives en assembleur pour charger des données dans la mémoire programme où marquer la fin du code source (.end) par exemple. Ces directives sont distinguées par la présence d'un point à leur début. Tout le détail de ces différents aspects se retrouve dans le tutoriel NIOS II.

Voici un exemple de programme assembleur pour le Nios II calculant le produit scalaire entre deux vecteurs A et B de n éléments que l'on y retrouve :

```
.include "nios_macros.s"
.global _start
_start:
movia r2, AVECTOR    /* Register r2 is a pointer to vector A */
movia r3, BVECTOR    /* Register r3 is a pointer to vector B */
movia r4, N
ldw r4, 0(r4)        /* Register r4 is used as the counter for loop iterations */
add r5, r0, r0        /* Register r5 is used to accumulate the product */
LOOP: ldw r6, 0(r2)   /* Load the next element of vector A */
ldw r7, 0(r3)        /* Load the next element of vector B */
mul r8, r6, r7        /* Compute the product of next pair of elements */
add r5, r5, r8        /* Add to the sum */
addi r2, r2, 4        /* Increment the pointer to vector A */
addi r3, r3, 4        /* Increment the pointer to vector B */
subi r4, r4, 1        /* Decrement the counter */
bgt r4, r0, LOOP      /* Loop again if not finished */
stw r5, DOT_PRODUCT(r0) /* Store the result in memory */
STOP: br STOP
N:
.word 6                /* Specify the number of elements */
AVECTOR:
.word 5, 3, -6, 19, 8, 12 /* Specify the elements of vector A */
BVECTOR:
.word 2, 14, -3, 2, -5, 36 /* Specify the elements of vector B */
DOT_PRODUCT:
.skip 4
```

## Travail à réaliser

Ce laboratoire se divise en 6 parties. Dans un premier temps, il sera question de produire la plate-forme avec le processeur et de la programmer sur le FPGA. Dans un deuxième temps, une application sera compilée et chargée dans la mémoire programme du système embarqué sur FPGA. Par la suite, l'exécution de l'application peut être lancée, et ce moment est propice à l'introduction du debugger disponible dans Eclipse. La technique de profilage sera également abordée. Les deux dernières parties de ce laboratoire visent à développer des compétences en matière d'optimisation logicielle dans un premier temps en langage assembleur et, dans un deuxième temps, en langage C.

- ✓ Les fichiers nécessaires au laboratoire sont disponibles sur la page du laboratoire sur Moodle.

## 1. Création d'un système embarqué intégrant le Nios II avec QSYS

L'outil Qsys est intégré à Quartus II, commencez donc par ouvrir ce dernier. Une fois dans Quartus II, créez un nouveau projet et nommez le « systemesolaire » (ne mettez pas d'espace dans vos noms de dossiers car Eclipse ne l'acceptera pas). Prenez soin d'importer l'assignement des broches, de choisir le bon FPGA et de mettre les broches inutilisées en mode « As Input Tri-Stated ».

- ✓ Avant d'aller plus loin, copiez le répertoire « vga » (contrôleur VGA) dans le répertoire du projet.

Ajoutez ensuite un fichier schématique (systemesolaire.bdf) au projet (il vous faudra sans doute ajouter quelque chose dans le fichier avant d'être autorisé à l'enregistrer, un composant « input » peut suffire !). Lancez finalement Qsys :

Tools → Qsys

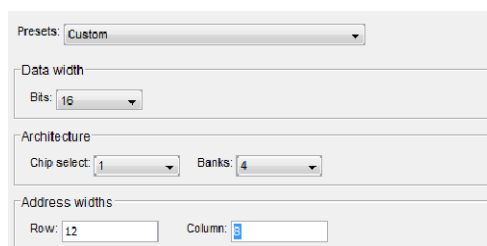
### A. Génération du système avec Qsys

Lors de l'ouverture, nommez votre système « sys\_systemesolaire », et laissez le target HDL en Verilog.

La fenêtre de Qsys est composée de plusieurs onglets :

- ✓ « System Contents » permet de spécifier les composantes du système et leurs interconnexions
- ✓ « System Generation » permet de générer le système.

Dans le premier onglet, le panneau de gauche « Component library » contient une gamme de périphériques pouvant être intégrés au système. Commencez par ajouter le **Nios II Processor** (catégorie « Processors ») en double-cliquant dessus. Une fenêtre d'assistance apparaîtra afin de le configurer, fermez-la en cliquant sur **Finish** pour l'instant. Le processeur aura besoin de mémoire, il y a pour cela sur la DE2 une DRAM de 8MB. Ajoutez son contrôleur dans **Memories and Memory Controller → SDRAM → SDRAM Controller**. Dans l'assistant, et ajustez le champ « Data width » à **16bits** avant de cliquer sur **finish**. Assurez-vous de connecter le "data master" et le "instruction master" du processeur au contrôleur mémoire. Il faudra finalement exporter l'interface *wire* de type *Conduit* du contrôleur mémoire. Pour ce faire, double-cliquer dans l'espace deux colonnes à droite de cette interface (colonne *Export* de l'onglet *System Contents*). Garder le nom donné par défaut.

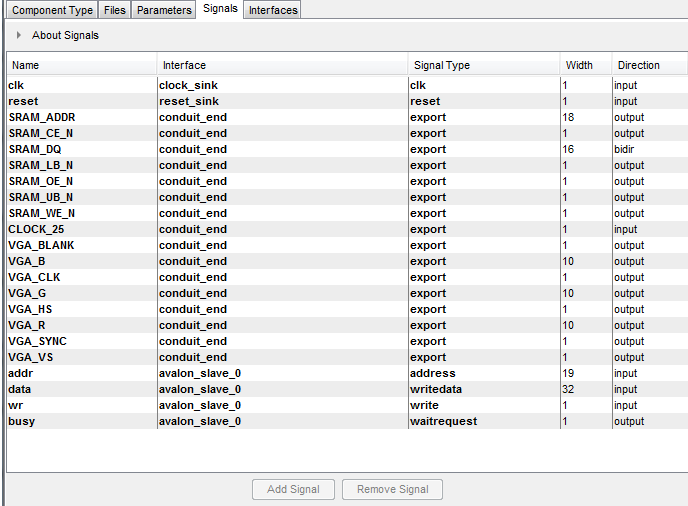


Retournez maintenant au processeur en double-cliquant sur le processeur déjà ajouté au système. Dans l'onglet « Core Nios II », prenez le temps de regarder les différences entre les différentes versions et choisissez la version **Nios II/f**. Cochez la case **hardware divide** et ajustez les champs **reset** et **exception vector** à la mémoire SDRAM. Dans l'onglet « Cache and Memory Interfaces », ajustez la taille de la cache de données à **4kB** par lignes de 32 Bytes, cliquez sur **finish**.

Pour récupérer le flux de sortie d'affichage, nous allons utiliser le lien USB-JTAG, donc ajoutez le composant **Interface Protocols → Serial → JTAG UART** et laissez les paramètres tels quels. Certaines fonctions requièrent

la présence d'un timer dans le système, donc ajoutez-en un en faisant **Peripherals ➔ Microcontroller Peripherals ➔ Interval Timer**, et conservez les paramètres initiaux (1ms). Le profileur GNU requiert également un 2<sup>e</sup> compteur, donc ajoutez-en un autre mais mettez sa résolution à 1us, et puis ensuite renommez ce dernier **timestamp\_timer**. Le dernier périphérique que nous ajoutons au système est le **Peripheral ➔ Debug and Performance ➔ System ID peripheral**, qui permet d'identifier le système matériel. Ce périphérique peut être utile si vous tentez de lancer une application qui était destinée à un autre système sur ce dernier. Assurez-vous que le nom de ce périphérique est **sysid**.

Ajoutez finalement le composant **vga** au système, que vous trouverez dans le dossier ELE8307. Si le dossier « ELE8307 » n'apparaît pas dans la « Component Library », il vous faudra cliquer sur **+Add...** puis aller dans **File** ➔ **Open** et naviguer dans votre dossier projet, dans le dossier **vga** et sélectionner **class.ptf**. Après le chargement et l'analyse du composant, il faudra apporter quelques petites modifications avant de le sauvegarder et de faire *Finish*. Dans l'onglet *Signals*, pour le signal clk, remplacez l'interface "clk" en cliquant dessus et en choisissant *new Clock Input...*, et type à *clk*. Pour le signal reset, remplacez l'interface "clk" par une nouvelle *new Reset Input...* (type reset). Il faut également remplacer l'interface de type *avalon\_slave\_0\_export* par une interface de type Conduit (*new Conduit...*).



Name	Interface	Signal Type	Width	Direction
clk	clock_sink	clk	1	input
reset	reset_sink	reset	1	input
SRAM_ADDR	conduit_end	export	18	output
SRAM_CE_N	conduit_end	export	1	output
SRAM_DQ	conduit_end	export	16	bidir
SRAM_LB_N	conduit_end	export	1	output
SRAM_OE_N	conduit_end	export	1	output
SRAM_UB_N	conduit_end	export	1	output
SRAM_WE_N	conduit_end	export	1	output
CLOCK_25	conduit_end	export	1	input
VGA_BLANK	conduit_end	export	1	output
VGA_B	conduit_end	export	10	output
VGA_CLK	conduit_end	export	1	output
VGA_G	conduit_end	export	10	output
VGA_HS	conduit_end	export	1	output
VGA_R	conduit_end	export	10	output
VGA_SYNC	conduit_end	export	1	output
VGA_VS	conduit_end	export	1	output
addr	avalon_slave_0	address	19	input
data	avalon_slave_0	writedata	32	input
wr	avalon_slave_0	write	1	input
busy	avalon_slave_0	waitrequest	1	output

Dans l'onglet *Interfaces*, faites *Remove Interfaces With No Signals* et spécifiez correctement l'horloge et le reset associée à chacune des interfaces restantes (*Clock Sink/Reset Sink*). Enregistrer, cliquer sur **Finish** pour finalement retourner dans la fenêtre principale de Qsys, et disposer de la librairie « ELE8307 » pour pouvoir ajouter le composant **vga**. Une fois cela fait, exporter à l'extérieur du système le conduit (interface de type Conduit) du composant "vga\_0" ajouté en double-cliquant sur l'espace "Double-click to export" situé deux colonnes à droite. Conserver le nom donné par défaut.

Dans la colonne « IRQ » (Interrupt Request), la priorité d'interruption associée à chaque périphérique est affichée dans un carré blanc, le nombre le plus faible étant le plus prioritaire. Mettez le timer (système) à la priorité **0**, le timestamp\_timer à **1**, et le JTAG à **2**. Dans la section de gauche, vous pouvez voir les interconnexions des chemins de données et d'instructions du processeur. Votre système devrait ressembler à ceci:

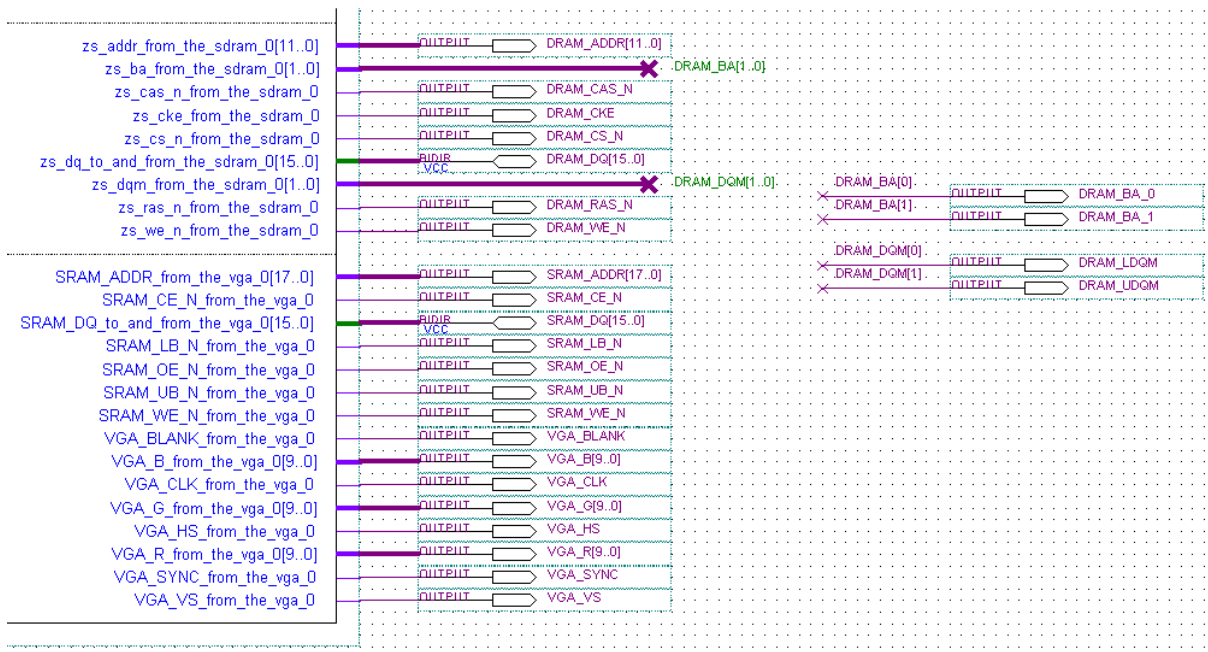
System Contents									
Address Map									
Clock Settings									
Project Settings									
Instance Parameters									
System Inspector									
HDL Example									
Generation									
Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	
<input checked="" type="checkbox"/>		clk_0	Clock Source	clk					
<input checked="" type="checkbox"/>		clk_in_reset	Clock Input	reset					
<input checked="" type="checkbox"/>		clk_reset	Reset Input	Double-click to export	clk_0				
<input checked="" type="checkbox"/>		clk_reset	Reset Output	Double-click to export					
<input checked="" type="checkbox"/>		nios2_qsys_0	Nios II Processor	Double-click to export	clk_0				
<input checked="" type="checkbox"/>		clk	Clock Input	Double-click to export	clk_0				
<input checked="" type="checkbox"/>		reset_n	Reset Input	Double-click to export					
<input checked="" type="checkbox"/>		data_master	Avalon Memory Mapped Master	Double-click to export	clk			IRQ_0	IRQ_31
<input checked="" type="checkbox"/>		instruction_master	Avalon Memory Mapped Master	Double-click to export	clk				
<input checked="" type="checkbox"/>		jtag_debug_module_reset	Reset Output	Double-click to export	clk				
<input checked="" type="checkbox"/>		jtag_debug_module	Avalon Memory Mapped Slave	Double-click to export	clk	0x1020_0800	0x1020_0fff		
<input checked="" type="checkbox"/>		custom_instruction_master	Custom Instruction Master	Double-click to export	clk				
<input checked="" type="checkbox"/>		new_sdr_controller_0	SDRAM Controller	Double-click to export	clk_0				
<input checked="" type="checkbox"/>		clk	Clock Input	Double-click to export	clk_0				
<input checked="" type="checkbox"/>		reset	Reset Input	Double-click to export	clk				
<input checked="" type="checkbox"/>		s1	Avalon Memory Mapped Slave	Double-click to export	clk	0x0800_0000	0x0fff_ffff		
<input checked="" type="checkbox"/>		wire	Conduit	new_sdr_controller_0_wire					
<input checked="" type="checkbox"/>		jtag_uart_0	JTAG UART	Double-click to export	clk_0				
<input checked="" type="checkbox"/>		clk	Clock Input	Double-click to export	clk				
<input checked="" type="checkbox"/>		reset	Reset Input	Double-click to export	clk				
<input checked="" type="checkbox"/>		avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to export	clk	0x1020_1048	0x1020_104f		
<input checked="" type="checkbox"/>		timer_0	Interval Timer	Double-click to export	clk_0				
<input checked="" type="checkbox"/>		clk	Clock Input	Double-click to export	clk				
<input checked="" type="checkbox"/>		reset	Reset Input	Double-click to export	clk				
<input checked="" type="checkbox"/>		s1	Avalon Memory Mapped Slave	Double-click to export	clk	0x1020_1020	0x1020_103f		
<input checked="" type="checkbox"/>		timestamp_timer	Interval Timer	Double-click to export	clk_0				
<input checked="" type="checkbox"/>		clk	Clock Input	Double-click to export	clk				
<input checked="" type="checkbox"/>		reset	Reset Input	Double-click to export	clk				
<input checked="" type="checkbox"/>		s1	Avalon Memory Mapped Slave	Double-click to export	clk	0x1020_1000	0x1020_101f		
<input checked="" type="checkbox"/>		sysid	System ID Peripheral	Double-click to export	clk_0				
<input checked="" type="checkbox"/>		clk	Clock Input	Double-click to export	clk				
<input checked="" type="checkbox"/>		reset	Reset Input	Double-click to export	clk				
<input checked="" type="checkbox"/>		control_slave	Avalon Memory Mapped Slave	Double-click to export	clk	0x1020_1040	0x1020_1047		
<input checked="" type="checkbox"/>		vga_0	VGA	Double-click to export					
<input checked="" type="checkbox"/>		avalon_slave_0	Avalon Memory Mapped Slave	Double-click to export	clk	0x1000_0000	0x101f_ffff		
<input checked="" type="checkbox"/>		conduit_end	Conduit	vga_0_conduit_end	clk				
<input checked="" type="checkbox"/>		clock_sink	Clock Input	Double-click to export	clk_0				
<input checked="" type="checkbox"/>		reset_sink	Reset Input	Double-click to export	clk_0				

Afin de générer correctement l'espace d'adressage de chaque périphérique, lancez simplement **System → Assign Base Addresses**. Vous pouvez maintenant générer le système en appuyant sur le bouton **Generate** de l'onglet « System Generation ». Si l'on vous demande de sauvegarder, utilisez le nom **sys\_systemesolaire** et sauvegardez dans votre répertoire projet directement.

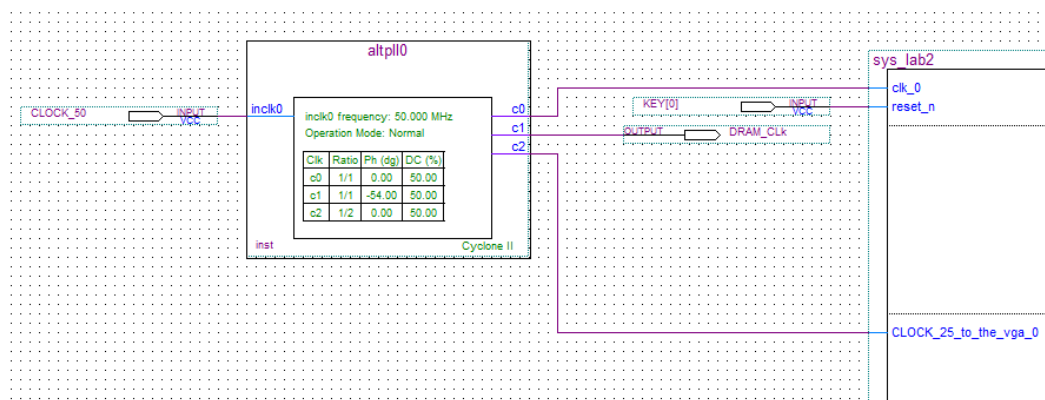
Lorsque la génération est terminée, sortez de Qsys. Si ce n'est pas fait automatiquement, ajouter le fichier **sys\_systemesolaire.qsys** à votre projet.

## B. Intégration du système dans Quartus II

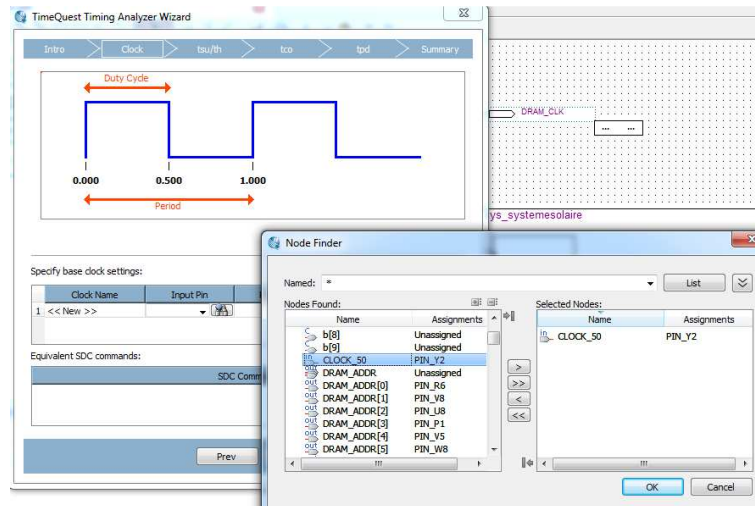
L'intégration d'un système produit avec Qsys dans un fichier schématique Quartus II est très similaire à l'intégration d'un module d'une librairie d'Altera. Cliquez sur le bouton **Symbol Tool** et dans le dossier projet, choisissez le module **sys\_systemesolaire** généré plus tôt. Il reste maintenant à connecter les ports d'entrées et sorties du système aux broches adéquates du FPGA. Le système embarqué utilise notamment une interface avec la mémoire SDRAM pour le programme, la mémoire SRAM comme mémoire vidéo, et un convertisseur numérique à analogique (DAC) pour l'interface VGA. Commencez par reproduire les connections tel qu'illustré à la figure suivante (**portez attention aux ports bidirectionnels**). Pour éviter de travailler pour rien, faites un clic droit sur l'instance, puis **Generate Pins for Symbol Ports**. Il vous suffit maintenant de changer les noms.



La mémoire DRAM requiert une horloge dont la phase est de **-3ns** par rapport à l'horloge du système, donc une PLL sera requise (altpll). Utilisez cette dernière pour produire une horloge à 50MHz (système), une deuxième à 50MHz avec -3ns de phase, à relier avec une nouvelle broche de sortie DRAM\_CLK, et puis une troisième horloge à 25MHz pour le contrôleur VGA. Reliez ensuite la 1<sup>re</sup> sortie de la PLL (à 50MHz) à l'entrée clk du système Qsys et une broche KEY[0] à l'entrée rst\_n (voir figure suivante). Lorsque tout est bien connecté, lancer la compilation du design et programmez la carte, la plate-forme matérielle est complétée.



Finalement, nous allons spécifier une contrainte de timing au moyen de l'assistant **TimeQuest Timing Analyzer Wizard** (Assignments → TimeQuest Timing Analyzer Wizard). Il est préférable d'exécuter l'analyse et la synthèse du design une fois avant de lancer l'assistant. Dans l'onglet Clock, spécifier la broche d'entrée CLOCK\_50, et spécifier une période de 20ns. Au dernier onglet, s'assurer que le fichier **.sdc** produit est ajouté au projet.



## 2. Création d'un projet logiciel avec Nios II SBT

L'environnement Software Build Tools Nios II - Eclipse permet de créer des projets logiciels ciblant des plateformes basées sur le processeur Nios II produites avec Qsys. Il faut d'abord lancer le logiciel :

Tools → Nios II Software Build Tools for Eclipse

L'utilisateur doit indiquer un emplacement pour l'espace de travail, choisissez simplement le répertoire de votre projet Quartus II. L'espace de travail pourra être modifié par la suite dans **File → Switch Workspace**.

### A. Une application « Hello-World » exécutée sur le Nios II

Créez un nouveau projet logiciel dans Eclipse:

File → New → Nios II Application and BSP from Template

Dans le champ « SOPC Information File name », sélectionnez le fichier **sys\_systemesolaire.sopcinfo** qui se trouve dans le répertoire du projet Quartus. Indiquez le nom du projet : **systemesolaire\_cpu0**. Cochez la case « Use default location ». Cliquez ensuite sur **Finish**. Le projet apparaît alors dans « Project Explorer ».

Dans le panneau Project Explorer vous verrez le dossier « systemesolaire\_cpu0 » qui contient les fichiers sources de votre application (hello\_world.c) tandis que le dossier « systemesolaire\_cpu0\_bsp » contient les bibliothèques, dont le fichier de description système system.h.

Avant de compiler l'application pour la première fois, il faut s'assurer que les paramètres du BSP sont adéquats. Pour ce faire, faites un clic-droit sur le dossier du projet et allez dans **Nios II → BSP Editor...** (Également disponible dans le menu déroulant, en haut). Dans l'onglet « Main » (Settings->Common), assurez-vous que les flux d'entrée et de sortie standard (stderr/stdin/stdout) correspond bien au jtag\_uart\_0 (ou jtag\_uart), et que les timers sont bien configurés (il faut spécifier le timestamp\_timer et le System clock timer). Ces options se retrouvent dans la section « Common » (à gauche). Dans la section « Advanced », désélectionnez seulement l'option *enable\_c\_plus\_plus* car le projet est écrit en C. Dans l'onglet « Linker Script », assurez-vous que la mémoire ciblée pour les différentes sections du programme corresponde à la SDRAM. Vous pouvez maintenant cliquer sur **Generate** dans le bas de la fenêtre puis quitter le BSP Editor.



Vous pouvez maintenant compiler l'application `soft_systemesolaire` : cliquez sur **Project-> Build All** (Ctrl+B). La compilation de la librairie système prend un certain temps, mais en principe elle ne doit être recompilée que lorsque la plate-forme (Qsys) est modifiée.

Afin d'exécuter l'application sur le Nios II, faites d'abord **Run->Run Configurations...** et puis dans la fenêtre qui s'ouvre vous serez invité à spécifier une configuration pour l'exécution. Double-cliquez sur Nios II Hardware, et puis sélectionnez votre projet. Assurez-vous que dans l'onglet « Target Connection » les paramètres correspondent au USB-Blaster, et puis faites **Run** pour lancer l'application.

Vous devriez voir le message « Hello from Nios II ! » s'afficher dans la console une fois l'exécution terminée. Notez bien que l'onglet **Nios II Console** représente la console du processeur Nios II, alors que l'onglet **Console** représente la console de votre ordinateur.

### B. L'application « SolarSystem3D »

Nous allons maintenant remplacer le code source de l'application `systemesolaire_cpu0` (`hello_world.c`) par le code source de l'application `SolarSystem3D`. Pour y parvenir, copiez les fichiers sources fournies directement dans le répertoire `<EspaceDeTravail>/software/systemesolaire_cpu0` qui contient déjà le fichier `hello_world.c` (que vous pouvez supprimer). Ensuite, dans Eclipse, faites un clic-droit sur le dossier « `systemesolaire_cpu0` » et choisissez **Refresh**. Vous devriez voir les sources apparaître.

L'application `SolarSystem3D` peut tourner aussi bien sur un ordinateur personnel avec OpenGL que sur une plate-forme Nios II, c'est pourquoi vous verrez des directives de pré-compilation `#ifdef` dans les fichiers `ecran2d.h` et `ecran2d.c`. Il existe maintenant deux fonctions « main » dans le projet et une seule doit être active pour la compilation, celle définie dans `ss_niosII_main.c`. Le plus simple est d'enlever l'autre du projet (`ss_win32_main.c`). Sinon, vous pouvez aussi la désactiver : faites un clic-droit sur le fichier et cliquez sur **Remove from Nios II Build**.

Prenez un petit moment pour parcourir les `.h` du projet car il sera important de comprendre certaines parties de l'application pour les prochains laboratoires. L'ANNEXE A explique certaines sections importantes de l'application.

## 3. Utilisation du débogueur de l'environnement intégré Eclipse.

Afin d'identifier les problèmes d'une application logicielle, un outil de débogage facilite grandement le travail. Un tel outil permet notamment l'exécution pas à pas, l'ajout de point d'arrêt (break points), et le suivi des valeurs des variables et registres au fil de l'exécution. Pour lancer l'application en mode debug, il suffit d'appeler ce mode :

Run → Debug

Lorsque l'on passe à ce mode, l'IDE vous invite à passer en perspective Debug qui est mieux adaptée. Cette perspective inclut ainsi un panneau avec les variables du programme et leurs valeurs, un panneau pour les registres du processeur, un panneau pour voir le code assembleur du programme, ainsi que des boutons pour l'exécution pas à pas tels « Step Into », « Step Over », et « Step Return ». À droite des ces boutons, le bouton « Instruction Stepping Mode » permet une exécution pas à pas des instructions assembleur plutôt que des « instructions » en C. Pour insérer un point d'arrêt, il est suffisant de double-cliquer la marge à l'endroit désiré

du programme. Il est possible d'observer le code assembleur de l'application, sous l'onglet « Disassembly ». Si l'un des onglets n'est pas présent, vous pouvez le rajouter en faisant **Windows → Show View**.

Notez que le changement de perspective peut être réalisé en cliquant le bouton Open Perspective en haut à droite de la fenêtre principale de l'IDE.

#### 4. Produire et intégrer du code assembleur à l'application

Afin d'accélérer une application écrite en C, il est possible d'aller réécrire certaines parties directement en langage processeur, ce qui peut donner de meilleurs résultats que le code produit par le compilateur. Il va sans dire toutefois qu'il est préférable d'optimiser d'abord l'algorithme à plus haut niveau.

Dans le cadre de ce travail, en vue de se familiariser avec le langage assembleur du Nios II, vous allez produire une version optimisée écrite en assembleur de la fonction `ss_planet_drawBall` qui trace un cercle plein d'une couleur (dans notre cas, la couleur noire). On l'utilise pour effacer les planètes lors d'une nouvelle itération. Assurez-vous de bien comprendre l'algorithme et le fonctionnement de la fonction incluse `ecran2d_setPixel` qui envoie les coordonnées ainsi que la couleur vers l'écriture sur l'écran VGA. Vous pourrez vous assurer que votre code assembleur fonctionne en regardant les planètes s'effacer lors d'une nouvelle itération. (Pour plus d'informations sur l'algorithme d'affichage, vous pouvez consulter le site suivant : <http://www.cs.unc.edu/~mcmillan/comp136/Lecture7/circle.html>)

Pour transmettre 4 arguments de fonction et moins, les registres **r4, r5, r6 et r7** sont utilisés dans l'ordre. Les registres **r2 et r3** sont utilisés pour les valeurs de retour. Créez un fichier **ss\_planet\_drawBall\_opt.s** qui contiendra votre code assembleur. Pour que l'appel de fonction puisse être réalisé depuis un fichier C, vous devrez déclarer comme global le nom de votre fonction. Le fichier aura ainsi le format suivant :

```
.section .text
.global _planete_drawball_opt
_planete_drawball_opt :
<votre code assembleur>
ret
.end
```

Dans votre code C, dans le .h (`ss_planet.h`), il faudra ensuite déclarer la fonction comme externe :

```
extern void _planete_drawball_opt( int x0, int y0, int radius, int color);
```

Vous n'aurez plus qu'à changer l'appel de la fonction dans le code pour utiliser votre code assembleur au lieu du code C. Puisqu'il n'y aura pas de valeurs de sorties (r2/r3) vers le code C, vous devrez utiliser l'instruction « `stwio` » qui permet d'envoyer de l'information vers un périphérique I/O (dans notre cas, vers la mémoire pour l'écran). Dans le code C, nous utilisons la fonction suivante (la valeur de `VGA_0_BASE` se retrouve dans `system.h`):

```
void ecran2d_setPixel( int x, int y, int color ) {
    IOWR(VGA_0_BASE, (y*640)+x,color);
}
```

Utilisez la version *Disassembly* du code lors d'un parcours en mode *Debug* pour vous inspirer afin d'envoyer les valeurs des pixels par votre code assembleur. Assurez-vous de bien commenter le code et de ne pas faire appel à la pile (*stack*) comme le fait la compilation du C. Vérifiez le bon fonctionnement.

Note : Pour vérifier le bon fonctionnement, remplacez également la fonction `ss_planet_drawBall13D` par votre code assembleur. Si le tout fonctionne bien, vous verrez alors les planètes d'une seule couleur (sans le

3D). Notez bien que cette manipulation ne sert que pour vérifier le bon fonctionnement de votre code; vous devez conserver la fonction d’affichage 3D telle quelle pour la suite du laboratoire.

## 5. Utilisation du GNU Profiler

Afin de déterminer les points chauds de l’application, l’environnement d’Eclipse facilite l’utilisation du profileur GNU. Comme les optimisations du code par le compilateur affectent l’exécution de l’application à plusieurs niveaux, il est toujours préférable d’effectuer le profilage sur une version de l’application la plus près possible de la version finale, soit avec un niveau d’optimisation **-O3** et sans instruction pour faciliter le déverminage. Pour notre projet logiciel, il faudra donc enlever toutes les options de debug pour le profilage. Pour ce faire, faites un clic droit sur le dossier projet `systemesolaire_cpu0` et allez dans **Properties**. Sélectionnez ensuite, dans la liste à gauche de la fenêtre, l’item **Nios II Application Properties**, et mettez le **Debug Level** à **Off** et le **Optimization level** à **Level 3**. N’oubliez pas que si, par la suite, vous désirez déverminer de nouveau, vous devrez remettre ces options à leur état d’origine.

Ensuite, retournez dans le BSP Editor et cochez l’option **enable\_gprof**. Assurez-vous également de mettre **-O3** dans le champ « `bsp_cflags_optimization` ». Régénérez et quittez. Encore une fois, n’oubliez pas de changer ces options pour revenir vers un projet en mode « debug ».

La seule modification au code source à réaliser est d’appeler la fonction ***exit(0)*** avant le return de la fonction « main », pour lancer l’écriture du fichier de résultats de profilage. En théorie, le return est suffisant, mais il semble y avoir un petit bug dans Eclipse. On ne prendra donc pas de chance ! Vous pouvez maintenant refaire un Build du projet. Afin de lancer le profilage, il suffit de lancer l’application (Pour que ce soit plus rapide, ne faites qu’une itération dans la fonction « main »).

Lorsque ce sera complété, le fichier ***gmon.out*** sera produit dans le dossier **software/systemesolaire\_cpu0**, vous pouvez ouvrir ce fichier texte depuis Eclipse. Le rapport produit par le profileur GNU contient une section nommée *flat graph*, qui indique par ordre décroissant les fonctions (enfant) appelées qui consomment le plus de temps d’exécution, et une section nommée *call graph* qui contient l’arbre des appels de fonctions et leurs enfants, triées par ordre décroissant de temps d’exécution.



Question :

- Outre les calculs en points flottants, quel est le point chaud de l’application sur lequel il faudrait s’attarder afin de l’accélérer ?

**Note sur des bugs courants :** Les fonctions de profilages paraissent très peu stables.

- Vous lancez l’application pour la profiler, rien ne se passe à l’écran VGA : pressez la touche reset (KEY0). Il vous faudra peut-être le faire plusieurs fois...
- La production du fichier de profilage ne fonctionne pas (l’action « Uploading GMON data » ne complète pas) : relancez le profilage de l’application jusqu’à son fonctionnement (cela peut prendre,

en moyenne, de 1 à 3 fois). Le profilage aura fonctionné lorsque vous verrez « Uploaded GMON data :... ».

Conseil : une procédure semble fonctionner dans la majeure partie des cas :

1. Lancer le profilage (L'application reste bloquée)
  2. Appuyez sur reset jusqu'à ce que l'application démarre
  3. Laissez-la terminer (l'upload de GMON DATA ne fonctionne pas)
  4. Relancez alors le profilage, qui devrait fonctionner !
- le fichier gmon.out ne peut être ouvert : faites un clic droit sur le dossier systemesolaire\_cpu0, puis **Nios II ➔ Nios II Command Shell...** À partir de là, faites la commande suivante afin de créer un fichier texte que vous pourrez lire dans l'éditeur de votre choix (les commandes « cd » et « dir » vous aideront à naviguer jusqu'au dossier où se trouve gmon.out) :

```
nios2-elf-gprof systemesolaire_cpu0.elf gmon.out > report.txt
```

Actualisez alors le contenu du projet pour voir le fichier texte apparaître.

- Rien ne se passe lorsque l'application quitte : vérifiez l'onglet « console », il doit indiquer « Running target program until exit » (affiché lorsque le profilage est actif) et non pas « Starting processor at address ... » (affiché lorsque le profilage est désactivé). Si ce n'est pas le cas, vérifiez que vous avez bien coché la case « enable gprof », générez à nouveau le BSP, quittez Eclipse, et relancez-le depuis Quartus. La procédure devrait alors fonctionner.
- Vous souhaitez désactiver le profilage, mais il reste actif : quittez Eclipse, éteignez puis allumez la carte de développement DE2, programmez-la à nouveau avec Quartus (pas besoin de recompiler). Vous pouvez alors relancer Eclipse et votre application sans le profilage.

## 6. Quelques optimisations logicielles en C

Dans certains cas, il peut être avantageux de pré-calculer certaines valeurs qui reviennent souvent. Il peut notamment être efficace de discrétiser une fonction complexe pour obtenir une approximation raisonnable plus rapidement. À chaque étape, vérifiez que tout fonctionne, profilez l'application, et notez les différences ainsi que, le cas échéant, les accélérations obtenues.

### A. Diminuer la précision des calculs

Une optimisation simple consiste à remplacer les fonctions mathématiques ( $\sin$ ,  $\cos$ ,  $\sqrt{\phantom{x}}$ , etc.) qui utilisent des nombres flottants à double précision (trop précis pour notre utilisation) par leurs équivalents en simple précision, en général largement suffisante. Ces nouvelles fonctions ont les mêmes noms, avec le suffixe « f » ( $\sinf$ ,  $\cosf$ , etc.). Faites la chasse aux « double » dans le code pour les transformer en « float ». Dans le rapport du profilage vous les trouverez par leurs noms (`__divdf3`, `__muldf3`, ... le « df » traduisant « double float », opposé à « sf » (simple float)). Enlevez-en le plus possible, vous observerez déjà une accélération !

### B. Pré calculer les valeurs

Les fonctions  $\sqrt{\phantom{x}}$ () et  $\arccos()$  de la fonction `ss_planet_getBall3DReferenceIntensity()` consomment un temps d'exécution considérable. Comme l'argument de la fonction  $\arccos()$  est compris entre 0 et 1, avec l'hypothèse raisonnable que la précision requise pour le calcul d'intensité lumineuse n'est pas cruciale, proposez et implémentez une optimisation basée sur le pré-calcul de l'expression  $\sin(\arccos(x))$  avec un tableau d'une centaine de valeurs (variable globale). Ne remplissez pas ce tableau à la main !

### C. Éliminer les calculs inutiles

Dans la fonction `ss_planet_getBall3DReferenceIntensity()`, afin d'éliminer la racine carrée on pourrait travailler avec  $r^2$  (où  $r = \sqrt{x^2 + y^2}$ ) plutôt qu'avec  $r$ . C'est donc dire qu'on fait l'approximation  $r/R \approx (r/R)^2 = r^2/R^2$ . On obtient alors  $\text{intensity} = 30 * \sin(\arccos((x*x + y*y) / (\text{double})\text{radius}*\text{radius}))$ . Puisque  $r/R$  est contenu entre 0 et 1,  $r^2/R^2$  sera aussi contenu entre 0 et 1. On pourrait donc pré-calculer un tableau `i_sinacos[100]` qui contient le pré-calcul de la fonction *intensity* approximée pour des ratios  $r^2/R^2$  (i.e.: 0, 0.01, 0.02, 0.03, ... , 0.98, 0.99).

$$r/R \approx (r/R)^2 = (r^2/R^2) \rightarrow \sin(\arccos((x*x + y*y)/R^2))$$

## Instructions pour la remise

Voir :

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

Vous devez remettre votre rapport ainsi que votre dossier de projet compressé sur Moodle.



## Grille d'évaluation

Ce laboratoire compte pour 6% de la note finale.

Partie 4 (code complet fonctionnel)	= 3/6
Partie 5 (réponse à la question)	= 1/6
Partie 6 (code complet fonctionnel)	= 2/6

Version du 19 septembre 2014

## ANNEXE A

Voici une petite description des différents fichiers sources que l'on retrouve dans l'application fournie :

**ss\_niosII\_main.c** : On y retrouve la fonction *main()* du projet, où l'on initialise le système solaire. On utilise également une boucle de 100 itérations pour faire marcher le système solaire. Il est possible d'augmenter ou de diminuer cette boucle pour changer le temps de simulation du système.

**ss\_solar\_system.h/.c** : On crée le système solaire complet (orbites et planètes) et on gère les interactions. On s'occupe également de commander les différents affichages.

**ss\_orbit.h/.c** : On s'occupe de l'affichage des différentes orbites à l'écran. Il est important de noter que les orbites sont calculées 1 seule fois au début du programme et ne changent pas par la suite, on s'occupe uniquement de les faire afficher dans le plan 2D de l'écran selon notre emplacement.

**ss\_planet.h/.c** : C'est à travers ces fichiers que l'on fait l'affichage des planètes. Leur représentation 3D nécessite beaucoup de calculs, comme vous le constaterez. Il est important de bien étudier ce fichier pour les fonctions `ss_planet_drawBall` et `ss_planet_drawBall3D`. Lorsque l'on affiche une planète à l'écran, la deuxième fonction va recevoir en paramètre la coordonnée du centre, le rayon, la couleur ainsi que les vecteurs de lumière. On va ensuite faire des transformations matricielles pour ramener le tout en format 2 dimensions pour aller chercher l'intensité de couleur de chaque point pour recréer le 3D sur le plan 2D. Après une itération, il faut absolument effacer les planètes à l'écran car s'il y a eu du mouvement, on retrouvera la planète « dédoublée ». On va donc refaire l'algorithme mais cette fois-ci, avec la fonction `ss_planet_drawBall` qui permet de faire l'affichage d'une seule couleur, la noire dans notre cas. Commencez par bien étudier cette fonction qui est une version simplifiée de l'affichage 3D.

**camera3D.h/.c** : Ces fichiers permettent de retrouver les différents vecteurs de positionnement et de commander les transformations matricielles.

**matrice4D.h/.c** : Ces fichiers performant les transformations matricielles pour passer du 3D vers la 2D.

**ecran2D.h/.c** : Permet l'affichage à l'écran VGA des différents points calculés.