ENSEIRB-MATMECA

BORDEAUX Enseich-Material Enseich-Material

MISE EN ŒUVRE DU SOPC SUR COMPOSANTS FPGA INTEL ET XILINX

Patrice NOUEL Patrice KADIONIK kadionik.enseirb-matmeca.fr

TABLE DES MATIERES

1. B	But des travaux pratiques	4
2. T	TP 1 : grand TP. Mise en œuvre du SoPC avec Intel Cyclone V	6
2.1.	Introduction	6
2.2.	Carte cible Intel DE10-Standard	6
2.3.	Présentation du processeur softcore NIOS II d'Intel	10
3. E	EX 1 Intel : construction du design de référence	14
3.1.	Introduction	14
3.2.	Ajout du processeur NIOS II et de ses périphériques	17
3.3.	Ajout des connexions de signaux	31
3.4.	Ajout des interruptions	34
3.5.	Définition du mapping mémoire	34
3.6.	Définition des vecteurs d'exception	35
3.7.	Exportation des signaux externes au circuit FPGA	36
3.8.	Génération HDL du système SoPC	37
3.9.	Synthèse du système SoPC	39
3.10). Programmation du circuit FPGA	43
4. E	EX 2 Intel : Hello World	47
5. E	EX 3 Intel : création du BSP	51
6. E	EX 4 Intel : Hello World μC/OS Π	55
7. E	EX 5 Intel : tests des périphériques sous μC/OS II	56
8. E	EX 6 Intel : miniprojet : chronomètre	57
9. T	TP 2 : grand TP. Mise en œuvre du SoPC avec Xilinx Zynq	58
9.1.	Introduction	58
9.2.	Carte cible Digilent ZedBoard	58
10.	EX 1 Xilinx : génération du RAM disk et intégration d'une application	61
<i>11</i> .	EX 2 Xilinx : mise en œuvre de Linux embarqué sur la carte cible	64
12.	EX 3 Xilinx : intégration d'un périphérique matériel Libre	66
12.1	Introduction	66
12.2	2. Intégration d'un périphérique. <i>Timer</i> 64 bits	69
12.3	8. Tests logiciels	82
*	 Test de la mesure de temps Test de l'incrémentation 	. 82 83
· 13.	EX 4 : création du RAM disk pour le novau Linux standard	84
14	FX 5 · mesure de temps de latence avec le novau I inux standard	85
17,	122 5 . mesure ac temps ac atonee aree it noyaa Linas standard	00

15.	EX 6 : création du RAM disk pour le noyau Linux Xenomai	86
16.	EX 7 : compilation du noyau Linux Xenomai	88
17.	EX 8 : mesure de temps de latence avec le noyau Linux Xenomai	
17	1. Outils standards	
17	2.2. Outils graphiques	
18.	Conclusion	94
<i>19</i> .	Références	
<i>20</i> .	Annexe 1 : fichier source tsttimer64.c	
<i>21</i> .	Annexe 2 : fichier source jitter.c	
22.	Annexe 3 : fichier source hello_xenomai.c	
<i>23</i> .	Annexe 4 : fichier source jitter_xenomai.c	
<i>24</i> .	Annexe 5 : configuration réseau hôtes et cibles	

1. BUT DES TRAVAUX PRATIQUES

Ces Travaux Pratiques ont pour but de présenter une approche au problème de la conception des SoC (*System On Chip*) dans ce qu'il a de particulier : mener conjointement le développement matériel et logiciel d'un projet.

Pour des raisons évidentes de souplesse d'utilisation, la plateforme matérielle est basée sur un circuit programmable FPGA transformant notre SoC en SoPC (*System On Programmable Chip*). Ceci permet d'obtenir des prototypes fonctionnels dans le minimum de temps dont on dispose.

Mais que doit-on trouver dans un enseignement sur les SoC ? Le système est composé d'éléments standards non originaux et bien connus : un processeur, des mémoires, des périphériques, une interface Ethernet... Tous ces éléments ont déjà fait l'objet d'enseignements spécifiques : cours microprocesseur, cours sur les réseaux, cours VHDL. Dans tous ces enseignements, le matériel était parfaitement connu lorsqu'il s'agissait d'y associer du logiciel.

On le voit, ce qui va caractériser le SoC est en premier lieu sa capacité d'optimiser une solution en choisissant ce qui doit revenir au matériel et ce qui restera au logiciel et l'art de passer de l'un à l'autre. Créer ses propres périphériques ou ses propres instructions en VHDL, voilà en particulier ce que permet le SoC. On peut partir d'un processeur intrinsèquement peu performant et obtenir un système aux performances remarquables !

Les premiers TP de *codesign* ont été créés initialement en 2004 par Patrice Nouel (†1944-2022),



enseignant chercheur à l'ENSEIRB-MATMECA avec des cartes cibles Altera Stratix 1S10. Ils ont été ensuite mis à jour avec des cartes cibles Intel DE10-Standard. De nouveaux TP ont été créés avec des cartes cibles Xilinx dans le but d'avoir un « grand » TP alliant l'intégration d'un périphérique matériel Libre dans un SoPC complété par la mise en œuvre de Linux embarqué pour le processeur *hardcore* Cortex-A9 du circuit FPGA Zynq. Un pilote de périphérique sous Linux sera alors développé pour pouvoir écrire l'application de test du périphérique sous Linux embarqué. De même, Xenomai sera mis en œuvre sur la cible pour mesurer des temps de latence sur système non chargé et système chargé.

On le voit, tout cela permet de réaliser LA synthèse de différents modules proposés dans l'option Systèmes Embarqués SE.

La mise en œuvre d'un système SoPC sur une carte Xilinx à base de circuit FPGA Zynq et sur une carte Intel à base de circuit FPGA Cyclone V a fait l'objet d'un sujet de « projets avancés » de l'option Systèmes Embarqués SE. Je tiens ainsi à remercier Maxime Gernet, Jean-Christophe Meyer, Ayoub Benyahya et Samir Mammeri de la promotion SE 2016-2017 et Souleymane Soumah, Anis Yagoub et Fatima Ennaciri de la promotion SE 2022-2023 pour leur travail et leur contribution à l'amélioration constante de l'enseignement de l'option SE...

Mots clés : SoPC, Intel, Altera, Xilinx, Quartus Prime, Vivado, *hardcore*, *softcore*, Cyclone V, NIOS II, Zynq, ARM, Cortex-A9, Linux, Linux embarqué, Xenomai, μ C/OS II, langage C, VHDL

2. TP 1 : GRAND TP. MISE EN ŒUVRE DU SOPC AVEC INTEL CYCLONE V

2.1. Introduction

Il s'agit de mettre en œuvre l'environnement de développement SoPC d'Intel (ex Altera) qu'il s'agisse de *Platform Designer* pour la création d'un système SoPC (*System on Programmable Chip*), de *Quartus Prime* (édition standard) comme IDE (*Integrated Development System*) de placement routage et de synthèse ou bien de l'environnement *Eclipse* pour développement de la partie logicielle en langage C embarqué (ou mode dit *bare metal*).

Ce premier « grand » TP se décompose en :

- La création pas à pas d'un système SoPC à base du processeur *softcore* NIOS II pour circuit FPGA Cyclone V® avec les outils Intel *Quartus Prime* et *Platform Designer*.
- La mise en œuvre du langage C avec *Eclipe* sur le processeur NIOS II du circuit FPGA Cyclone V.
- Le développement du BSP (*Board Support Package*) avec le langage C embarqué pour piloter de façon simple les périphériques du système SoPC.
- Le développement d'applications avec le langage C en mode *bare metal* et mise en œuvre du noyau Temps Réel μC/OS II.

2.2. Carte cible Intel DE10-Standard

La carte cible mise en œuvre pour le SoPC sur circuit FPGA Cyclone V d'Intel est une carte d'évaluation Intel (société Terasic) DE10-Standard.



Carte cible Intel DE10-Standard

La carte DE10-Standard intègre un circuit FPGA Cyclone V qui incorpore un processeur *hardcore* ARM dans la partie HPS (*Hard Processor System*) et une zone de programmation logique PL (*Programmable Logic*). Cette approche duale existe aussi chez Xilinx avec son circuit FPGA Zynq.

La carte possède les caractéristiques suivantes pour la partie PL :

- Circuit FPGA Intel Cyclone V SE 5CSXFC6D6F31C6N.
- Mémoire pour la configuration de la partie PL EPCS128.
- Sonde USB-Blaster II pour la programmation avec le mode JTAG.
- 64 Mo de SDRAM.
- 4 boutons poussoir.
- 10 switchs.
- 10 leds rouges.
- 6 afficheurs 7 segments.
- Codec audio 24 bits avec line-in, line-out et entrée microphone.
- Convertisseur CNA 8bits (x3) avec sortie VGA.
- Décodeur TV (NTSC/PAL/SECAM) et entrée TV.
- Connecteur PS/2.
- Emetteur/récepteur IR.
- Convertisseur CAN avec interface SPI.

La carte possède les caractéristiques suivantes pour la partie HPS :

- Processeur double cœur ARM Cortex-A9 à 925 MHz.
- 1 Go de SDRAM DDR3.
- 1 interface Ethernet Gb/s.
- 2 ports USB Host.
- 1 socket micro SD.
- 1 accéléromètre avec interface I2C.
- 1 connecteur UART ver USB Mini-B.
- Boutons *warm reset* et *cold reset*.
- 1 bouton utilisateur et 1 led utilisateur.
- 1 module LCD 128x64 points.

Il faut noter que l'on ne peut pas atteindre directement des périphériques de la partie HPS depuis la partie PL et inversement sans la mise en œuvre de ponts (*bridge*). Par exemple, on ne peut pas piloter directement depuis la partie PL le module LCD 128x64 points.

La figure suivante présente l'ensemble des périphériques accessibles sur la carte cible DE10-Standard. Cela correspond à un *design* de référence d'un système SoPC appelé *Computer System* par Intel mettant en œuvre 2 processeurs *softcore* NIOS II et le processeur ARM.

Nous n'allons pas partir de ce *design* de référence trop complexe et tout fait mais nous allons développer notre propre *design* SoPC à base d'un seul processeur NIOS II pour apprendre à maîtriser les outils Intel.



Design de référence Computer System pour la carte cible Intel DE10-Standard



Périphériques de la carte cible Intel DE10-Standard

2.3. Présentation du processeur softcore NIOS II d'Intel

Le processeur NIOS II (2^{ème} génération du processeur NIOS) est un processeur RISC *softcore* entièrement synchrone, son architecture interne étant de type Harvard. Il possède au maximum 6 niveaux de *pipeline*, cadencé à quelques dizaines de MHz, avec une largeur de bus de 32 bits. Ses performances vont jusqu'à 250 MIPS (*Million Instructions per Second*).

Les caractéristiques du processeur NIOS II sont :

- Architecture RISC.
- Jeu d'instructions 32 bits.
- 32 registres généraux.
- 32 sources d'interruption.
- Instruction assembleur pour multiplications et divisions entières 32x32 bits pour un résultat 32 bits.
- Instructions pour multiplications 64 et 128 bits
- Instructions optionnelles pour opérations sur nombres réels simple précision.

- Accès à une variété de périphériques *on-chip* et interfaces vers les périphériques et la mémoire *off-chip*.
- Module de *debug* matériel.
- MMU (Memory Management Unit) optionnelle.
- MPU (*Memory Protection Unit*) optionnelle.



Architecture du processeur NIOS II

Il est possible d'accélérer certains traitements en ajoutant des instructions personnelles ou *Custom Instructions* (décrites en langage VHDL ou Verilog) au processeur NIOS II. De cette manière, il est possible de réaliser la surcharge d'opérateurs ou simplement d'étendre le jeu d'instructions.



Instruction personnalisée avec le processeur NIOS II

Lors de la configuration du processeur NIOS II avec l'outil *Platform Design* (ex outil *SoPC Builder* ou *Qsys*), il est possible de choisir entre 2 versions du processeur NIOS II : une première version *Economy* qui utilise moins de surface de silicium du composant FPGA et une version *Fast* qui est la plus rapide mais plus consommatrice de ressources.

	NIOS II /f	NIOS II /e	
Pipeline	6 niveaux	Non	
Multiplication Matériel	1 cycle	Par logiciel	
Branch Prediction	Dynamic	Non	
Cache d'Instructions	Configurable	Non	
Cache de données	Configurable	Non	
Instructions Personnalisés < 256			

Les 2 versions du processeur NIOS II

Le tableau suivant présente les performances en DMIPS (*Dhrystone Million Instructions per Second*) à l'aide du *benchmark* Dhrystone du processeur NIOS II sur les différentes familles de composants FPGA d'Intel (Agilex, Stratix V, Arria 10, Cyclone V, MAX 10...).

Métrique	NIOS II/f	NIOS II/e
DMIPS/MHz	0.753	0.107

Performances du processeur sa	oftcore	NIOS II
-------------------------------	---------	---------

Pour comparaison, le tableau suivant donne les performances de quelques plateformes matérielles à base de processeurs *hardcore* (mesurées par l'auteur) à l'aide du *benchmark* Dhrystone 2.1 :



Exemples de performances de processeurs *hardcore*

Notons que dans la phase de construction du circuit avec l'outil *Platform Designer*, il est possible d'inclure différents périphériques standards en utilisant le bus multimaître *Avalon* du processeur NIOS II :

- Mémoire.
- Timer.
- Liaison série JTAG/UART.
- Interface écran VGA.
- E/S parallèles.
- Interface Ethernet.
- JTAG.
- ...

3. EX 1 INTEL : CONSTRUCTION DU DESIGN DE REFERENCE

3.1. Introduction

Il s'agit d'utiliser l'environnement de développement SoPC d'Intel qu'il s'agisse de *Quartus Prime* et de *Platform Designer* pour la construction du *design* de référence donc du système SoPC.

Après synthèse, le circuit SoPC sera programmé dans le circuit FPGA de la carte DE10-Standard.

La figure suivante récapitule la démarche de conception typique d'un système SoPC :



Conception d'un système SoPC

Les différentes phases sont (analogues à la conception d'un système numérique) :

- Description du système SoPC : on utilisera l'outil graphique *Platform Designer*. Le code généré sous-jacent est soit du VHDL ou soit du Verilog.
- Synthèse : le système SoPC est analysé puis synthétisé pour viser un circuit FPGA, ici le circuit FPGA Cyclone V.

- Simulation fonctionnelle : vérification des délais de propagation et métastabilité.
- Placement/routage (*fitting*) : analyse temporelle : les délais de propagation dans les circuits sont analysés afin d'amener des indications sur la performance du circuit.
- Simulation temporelle : après placement/routage, le circuit est simulé en prenant en compte cette fois-ci les contraintes temporelles.
- Programmation : le circuit FPGA est programmé avec la sonde USB-Blaster à partir du fichier de programmation .sof(.bit chez Xilinx).

Intel fournit un outil Windows *DE10_Standard_SystemBuilder* qui permet de générer les fichiers nécessaires pour démarrer le projet *Quartus Prime*.

Le *design* est référence qui est un sous-ensemble du *design Computer System* est présenté sur la figure suivante.

DE10 Standard V1.0.1					\times
terasic			System Configurat	tion	
www.terasic.com			Project Name: D	E10_Standard	
DE10 Standa	ard FPGA Boa	rd	Top File Type: 🔽	erilog	•
AA C.D.	DE DURA	AAM	CLOCK	🔽 7-Segment x 6	
		Land	₽ LED×10	🔽 Switch x 10	
			🔽 Button x 4	IR TX/RX	
			Г VGA	⊏ Video-In	
	The second		□ Audio	I ADC	
			🔽 SDRAM, 32M	B 🗆 PS2	
			F (HPS)		
	CELO-Standard		GPIO Header (3.3∨)	
			None		
			Prefix Name:		
			HSMC Header		
Load Setting	Generate	HSMC 4	IO Voltage: 2.5	5∨(Default) 💽	
		5 6 2.5V(DEFAULT 1 8 3.3V 1	None	<u> </u>	
Save Setting	Exit		Prefix Name:		
		JP3 I			

Design de référence DE10_Standard_golden_top du système SoPC

Outre le processeur NIOS II (version *fast*), le système SoPC contient les périphériques externes au circuit FPGA suivants de la carte cible DE10-Standard :

- Mémoire SDRAM de 64 Mo.
- 10 leds.
- 4 boutons poussoir.
- 6 afficheurs 7 segments.
- 10 switchs.

On rajoutera aussi les périphériques internes au circuit FPGA suivants :

- Liaison série JTAG/UART.
- *Timer* 32 bits.

On aura ainsi au démarrage 5 fichiers pour notre projet Quartus Prime :

- Fichier .qpf (Quartus Project File) : fichier projet à ouvrir avec Quartus Prime.
- Fichier .qsf (*Quartus Setting File*): fichier de configuration avec les paramètres et l'affectation des broches du projet.
- Fichier .v : fichier *Top-Level* du plus haut niveau hiérarchique en langage Verilog.
- Fichier .sdc (Synopsis Design Constraints) : fichier des contraintes temporelles.
- Fichier .htm: fichier HTML donnant sous forme de tableaux les assignements des broches du circuit FPGA et le nom des signaux du *design* SoPC.

On notera que le nom des signaux Verilog (notamment dans le fichier *Top-Level*.v) est le lien dans tout le projet SoPC notamment sur l'usage des broches du circuit FPGA.

Le nom de notre projet *Quartus Prime* est *DE10_Standard_golden_top*.

On aura donc les 5 fichiers suivants :

- Fichier DE10_Standard_golden_top.qpf: projet Quartus Prime.
- Fichier DE10_Standard_golden_top.qsf.
- Fichier DE10_Standard_golden_top.v: fichier Verilog *Top-Level*.
- Fichier DE10_Standard_golden_top.sdc.
- Fichier DE10_Standard_golden_top.htm.

Le contenu du fichier Verilog DE10_Standard_golden_top.v est le suivant :

module DE10_Standard_golden_top(

```
/////// CLOCK ////////
                  CLOCK2 50,
input
input
                  CLOCK3_50,
input
                  CLOCK4_50,
input
                  CLOCK_50,
/////// KEY ////////
       [ 3: 0]
input
                  KEY,
/////// SW ////////
input
        [ 9: 0]
                  SW,
/////// LED ////////
output
       [ 9: 0]
                  LEDR,
/////// Seg7 ////////
        [ 6: 0]
output
                  HEX0,
        [ 6: 0]
output
                  HEX1,
output
        [ 6: 0]
                  HEX2,
output [ 6: 0]
                  HEX3,
output [ 6: 0]
                  HEX4,
output [ 6: 0]
                  HEX5,
/////// SDRAM ////////
output
                  DRAM_CLK,
                  DRAM_CKE,
output
```

out	out	[12:	0]	DRAM_ADDR,
out	out	[1:	0]	DRAM_BA,
ino	ut	[15:	0]	DRAM_DQ,
out	put			DRAM_LDQM,
out	put			DRAM_UDQM,
out	out			DRAM_CS_N,
out	put			DRAM_WE_N,
out	put			DRAM_CAS_N,
out	put			DRAM_RAS_N

```
endmodule
```

);

Le nom du projet Quartus Prime est bien DE10_Standard_golden_top.

Les périphériques extérieurs au circuit FPGA sont connectés aux broches du circuit FPGA par leur nom :

- KEY : 4 boutons poussoir. Bus de 4 signaux.
- SW : 10 switchs. Bus de 10 signaux.
- LEDR : 10 leds rouges. Bus de 10 signaux.
- HEXn : 6 afficheurs 7 segments. 6 bus de 7 signaux.
- CLOCKx_xx : horloges.
- DRAM_ : signaux de contrôle de la SDRAM externe.

Si l'on regarde le fichier DE10_Standard_golden_top.qsf, on retrouve l'association (nom_du_signal <-> numéro de broche).

Par exemple, le bouton poussoir KEYO est connecté à la broche du circuit FPGA AJ4 (set_location_assignment PIN_AJ4 -to KEY[0]).

3.2. Ajout du processeur NIOS II et de ses périphériques

Par la suite, on adoptera les conventions suivantes : Commande Linux PC hôte : host% commande Linux Commande Linux PC hôte pour le développement Intel : [NiosII EDS]\$

- Démarrer le PC sous Linux. Se connecter sous le nom se01, mot de passe : se01 ☺ pour le groupe 1 et sous le nom se02, mot de passe : se02 ☺ pour le groupe 2.
- Se placer dans son répertoire de travail : host% cd
- Recopier le fichier tp-de10.tgz sous /home/kadionik/: host% cp /home/kadionik/tp-de10.tgz .
- Se placer ensuite dans le répertoire de10/. L'ensemble du travail sera réalisé à partir de ce répertoire ! Les chemins seront donnés par la suite en relatif par rapport à ce répertoire... host% cd de10

- Se placer dans le répertoire design/. Nous allons maintenant construire notre design de référence DE10_Standard_golden_top: host% cd design host% ls DE10_Standard_golden_top.htm DE10_Standard_golden_top.sdc DE10_Standard_golden_top.qpf DE10_Standard_golden_top.v DE10_Standard_golden_top.qsf ip/
- Se placer dans l'environnement de développement Intel puis lancer Quartus Prime : host% n2sdk
 [Vilian EDK10] meantres

```
[Xilinx EDK]$ quartus
```

On obtient la figure suivante qui est l'interface graphique de Quartus Prime :



Interface graphique de *Quartus Prime* (1)

• Ouvrir le projet *Quartus Prime DE10_Standard_golden_top* par le menu *File > Open Project...* (ouverture du fichier DE10_Standard_golden_top.qsf). On obtient la figure suivante :

8	Quartus Prime Standard Edition - /home/kadionik/Altera/me357/design/DE10_Standard_golden_top - DE10_Standard_golden_to	p	_ = ×
<u>File Edit View Project Assignments Processing Tools Window</u>	Help		Search Intel FPGA
🗋 🐱 🖶 🤟 🗂 🗂 🔿 🕐 DE10_Standard_golden_top 🔹	✓ Ø Ø Ø ► ¥ K Ø Ø & Ø Ø		
Project Navigator 🔥 Hierarchy 👻 🔍 🖲 🛞		IP Catalo	g 🖉 🗷 🗷
Entity:Instance			× =,
 A Cyclone V: 5CSXFCE06P31C6 ▶ DE10_standard_golden_top /b 		- ₩ - ₩ - ₩ - ₩ - ₩	statlief (P got Offector Avalatie bray Basic Functions DSP Interface Protocols Memory Interfaces and Controllers Processon and Perphenals University Program arch for Partner (P
Task Compliation * ID ID Task * Comple Design * * Analysis Synthesis * > Analysis Synthesis *<		Keen Quartus Prime Information Decementation	
4 F		Notification Center + Add	
Image: System Processing	00End		

Projet *DE10_Standard_golden_top* (2)

• Lancer ensuite *Platform Designer* par le menu *Tool > Platform designer*. On obtient la figure suivante :



Interface graphique de *Platform Designer* (3)

• Sauvegarder le projet *Platform Designer* en le nommant nios2.qsys par le menu *File* > *Save*. On obtient la figure suivante :

	Save: unsaved _	• *
Enregistrer <u>d</u> ans	: 🗖 design 💌 🖬 🛱 🗖 🔡	
📑 db 📑 ip		
<u>N</u> om du fichier :	nios2 qsys	
<u>Type</u> de fichier :	Platform Designer System Files (*.qsys)	-
	Enregistrer Annuler	

Sauvegarde du projet *Platform Designer* nios2.qsys (4)

- Sélectionner et effacer le bloc IP (*Intellectual Property*) clock_0.
- Dans la fenêtre *IP Catalog*, rechercher les occurrences contenant la chaîne de caractères pll. On obtient la figure suivante :

P Catalog 🐰	17 5	ustem Conte	nts 🖄 Address F	Van 🖄 Interconnect B	equirements	23							
		- I III e.	nta or Address i	sup of interconnectio	equienenca	~							- 0 0
🔍 pll 🛛 🗶 🏠		1 - 3	stem. mosz							100			
Plat: Plat: Bas: Functions Plat: Plat: Plat:		Use Co	Name	Description		Export	Clock	Base	End	IPQ	Tags	Opcode Name	
		₩ ₩ 🔺 🦉	Current filter:										
	ŏ≣ M	lessages 🛛											- d =
	T	ype Pa	th					Mes	sage				
0 Emors 0 Warpings													Generate HDI Finish

Blocs IP contenant l'occurrence pll (5)

• Sélectionner le bloc IP *System and SDRAM Clocks for DE-series Boards*. On obtient la figure suivante :

	System and SDRAM Clocks for DE-series Boards - sys_sdram_pll_0	- • ×
System and SDRAM Close altera_up_avalon_sys_sdram_pll	cks for DE-series Boards	Documentation
* Block Diagram	* Settings	4
Show signals	Reference clock: 50.0 MH2 Desired System clock: 50.0 MH2 DE-Series Board: DE10-Standard •	
Info: sys_sdram_pil_0: Refclk Freq: 50.0		
		Cancel Finish

Configuration du bloc IP System and SDRAM Clocks for DE-series Boards (6)

On ajustera le champ *Desired System clock* à 100.0 MHz. On ne touchera pas à la valeur par défaut des autres champs. Puis on cliquera sur le bouton *Finish*.

• Changer le nom du bloc IP *sys_sdram_pll_0* en *pll*. Pour cela, on se positionne sur le label *sys_sdram_pll_0*, puis avec le clic droit de la souris, on accède au menu contextuel et au choix *Rename*. On obtient la figure suivante :



Bloc IP *pll* (7)

La sélection et la configuration du premier bloc IP ont été détaillées. Pour les suivants, on adoptera le même principe.

• Ajouter le bloc IP *NIOS II*. Dans la fenêtre *IP Catalog*, rechercher les occurrences commençant par nios. On choisira le bloc IP *Nios II Processor*. On choisira la version *fast*. On obtient la figure suivante :

	Nios II Processor - nios2_gen2_0 _ =
Nios II Processor	
altera_nios2_gen2	Documentation
V Black Disease	
Chew signals	Main Vectors Caches and Memory Interfaces Arithmetic Instructions MMU and MPU Settings JTAG Debug Advanced Features
onow signals	* Select an Implementation
nios2 gen2 0	Nios II Core: O Nios II/e
cik , data master.	Nios IVf
reset avaien instruction master	Nios II/e Nios II/f
debug mem slave debug reset request custom instruction master	Summary Resource-optimized 32-bit RISC Performance-optimized 32-bit RISC
there, introduced and the second states, introduced and the second states, interformation and the second states and the second state	Features TLAC Debug TAG Debug ECC RAM Protection Hardware Multiply/Divide Instruction/Data Caches Tightly-Coupled Masters ECC RAM Protection External Interrupt Controller Stellow Mediater Sets MMU
	RAM Usage 2 + Options 2 + Options
Error. nios2.gen2,0 instruction Cache is larger than the instruction Address. Please D Error. nios2.gen2,0. Reset slaws is not specified. Please salect the reset slaw D Error. nios2.gen2,0. Exception slaws is not specified. Please select the exception slaw Error. nios2.gen2,0. Exception slaws is not specified. Please select the exception slaw D Error. nios2.gen2,0. Exception slaws is not specified. Please select the exception slaw D Error. nios2.gen2,0. Exception slaws is not specified. Please select the exception slaw D Error. nios2.gen2,0. Exception slaws is not specified.	reduce the instruction Cache Size. Current Tag Size is 0 ve
	Cancel Finish

Bloc IP Nios II Processor (8)

• Renommer le bloc IP *nios2*. On obtient la figure suivante :

			Platf	orm Designer - nio	s2.qsys* (/home/kadionik/Altera/me357/de	sign/nios2.qsys)				_	- ° ×
Eile Edit System Generate View Iools Help											
📂 IP Catalog 😫 🗕 🗖 🗖	₽ s	ystem Cor	ntents 💠 Address I	Map 🖾 Interconnec	t Requirements 💠						- d 🗆
🔍 sdram 🛛 🗙 🔯		= 4 艘	System: nios2								
Project		Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Tags
Mew Component sdram J28mb sdram 32mb sdram 32mb tibrary Memory Interfaces and Controllers	(H) [[[] × []			⊉ pil ref_clk ref_reset sys_clk sdram_clk reset_source	System and SDRAM Clocks for DE-series Boards Clock Input Reset Input Clock Output Clock Output Reset Output Reset Output	Double-click to Double-click to Double-click to Double-click to Double-click to	unconnected pll_sys_clk pll_sdram_clk				
Henroy Interfaces with AdMemPHY Henroy Interfaces with AdMemPHY and IPDO Henroy Interfaces with AdMemPHY and IPDO Henroy Interfaces with UnPHY relei FPDA IP OLOSE SORAM Controller with UnPHY relei FPDA IP OLOSE SORAM CONTORING	H 4			S nios2 cik reset instruction_master debug_rese_request debug_mem_slave custom_instruction	Nios I Processor Clock Input Clock Input Avalon Memory Mapped Master Interrupt Receiver Reset Output Availon Memory Happed Slave Custom Instruction Mester	Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to	arcomected [ck] [ck] [ck] [ck] [ck] [ck] [ck]	IRQ 0 ∉ 0x0800	IRQ 33 OxOfff	(
¶t, Hearday 30 Device Family 30 — d" — ¶t nties2 (nties2, esyst) ⊕ ¶t nties2 (nties2, esyst) ⊕ ¶t nties2 ⊕ Ωt nt											
		•			11						11
	1	₩ -# + ▼	Current filter:								
	ŏ≣ M	lessages	82								- đ 🗆
	T	ype	Path				Message				K
	P 🖸	7 E	irrors								-
		8 nic	os2.nios2	instruction Cache is la	rger than the instruction Address. Please reduce th	ne Instruction Cache Si	ze. Current Tag Size is 0				
	- 1	🕴 nio	os2.nios2	Reset slave is not spe	cified. Please select the reset slave						
	- 1	🕴 nic	os2.nios2	Exception slave is not	specified. Please select the exception slave						1
		8 nic	os2.pll	pll.ref_clk must be co	innected to a clock output						
		😆 nic	os2.nios2	nios2.clk must be con	nnected to a clock output						
		😆 nic	os2.pll	pll.ref_reset must be	connected to a reset source						
		🕴 nic	s2.nios2	nios2.reset must be	connected to a reset source						-
7 Errors, 0 Warnings									G	enerate H	DL Finish

Bloc IP nios2 (9)

• Ajouter le bloc IP *SDRAM*. Dans la fenêtre *IP Catalog*, rechercher les occurrences commençant par sdram. On choisira le bloc IP *sdram_64mb*. On ne touchera pas à la valeur par défaut des autres champs. On renommera le bloc IP *sdram*. On obtient la figure suivante :

P		Plat	form Designer - nios2	.qsys* (/home/kadionik/Altera/me357/desi	gn/nios2.qsys)					- * *
Eile <u>E</u> dit <u>S</u> ystem <u>G</u> enerate <u>V</u> iew <u>T</u> ools <u>H</u> elp										
💾 IP Catalog 💠 🗕 🗗 🗖	🖬 🎞 Sys	stem Contents 🛛 Address	Map 💠 Interconnect P	Requirements 🕴						-ರೆಂ
sdram X	x	🔺 🝿 System: nios2 Pa	th: sdram							
Project	+ U	Ise Connections	Name	Description	Export	Clock	Base	End	IRQ	Tags
- ங New Component		E .	⊟®ipli	System and SDRAM Clocks for DE-series Boards						
 sdram_128mb 	×		ref_cik	Reset Input		unconnected				
 sdram_32mb sdram_64mb 			sys_clk	Clock Output	Double-click to	pll_sys_clk				
Library			sdram_clk	Clack Output	Double-click to	pll_sdram_clk				
Memory Interfaces and Controllers			reset_source	Reset Output	Double-click to					
Memory interfaces with AltMemPHy intel FPG4		a	clk	Clock Input	Double-click to	unconnected				
Memory Interfaces with UniPHY			reset	Reset Input	Double-click to	[clk]				
 DDR2 SDRAM Controller with UniPHY Intel FPGA IP 			data_master	Avalon Memory Mapped Master	Double-click to	[clk]				
 DDR3 SDRAM Controller with UniPHY Intel FPGA IP IRDDR3 SDRAM Controller with UniPHY Intel FPGA II 			instruction_master	Avaion Memory Mapped Master	Double-click to Double-click to	[CIK]	TPO (TRO 21		
University Program			debug_reset_request	Reset Output	Double-click to	[clk]				
9 Clock		••	debug_mem_slave	Avalon Memory Mapped Slave	Double-click to	[clk]	@ 0x0800	OxOfff		
 System and SDRAM Clocks for DE-series Boards 	100	w]	custom_instruction	. Custom Instruction Master	Double-click to			-		
		6.6.	clk	Clock Input	Double-click to	unconnected				
			reset	Reset Input	Double-click to	[clk]				
			sl	Avalon Memory Mapped Slave	Double-click to	[clk]	16			
						111	1			
L Herarchy 30 Device Family 30 d' ⊂ 10 nters: [nters: peys*] + 00 nos: + 00 nos:	1									
	4			1						•
	1	- 作 💎 🛒 Current filter:								
	o≣ Me	ssages 🖾								-ರೆಂ
	Тур	e Path				Message				~
	P 😫	9 Errors								-
	6	nios2.nios2	Instruction Cache is large	er than the Instruction Address. Please reduce the	Instruction Cache Size	a. Current Tag Size is 0				
	6	nios2.nios2	Reset slave is not specifi	ed. Please select the reset slave						4
	6	nios2.nios2	Exception slave is not sp	ecified. Please select the exception slave						
	6	3 nios2.pll	pll.ref_clk must be conn	ected to a clock output						
	6	3 nios2.nios2	nios2.clk must be conne	acted to a clock output						
	6	nios2.sdram	sdram.clk must be conn	nected to a clock output						
	6	a nios2.pll	pll.ref reset must be co	onnected to a reset source						-
	•			R.						•
								<i>r</i>		

Bloc IP sdram (10)

• Ajouter le bloc IP *Timer*. Dans la fenêtre *IP Catalog*, rechercher les occurrences commençant par timer. On choisira le bloc *IP Interval Timer Intel FPGA IP*. On ne touchera pas à la valeur par défaut des autres champs. On renommera le bloc IP *timer*. On obtient la figure suivante :



Bloc IP timer (11)

• Ajouter le bloc IP *JTAG/UART* (port série par l'interface JTAG). Dans la fenêtre *IP Catalog*, rechercher les occurrences commençant par jtag. On choisira le bloc IP *JTAG UART Intel FPGA IP*. On ne touchera pas à la valeur par défaut des autres champs. On renommera le bloc IP *jtag_uart*. On obtient la figure suivante :



Bloc IP jtag_uart (12)

• Ajouter le bloc IP *PIO* (port parallèle). Dans la fenêtre *IP Catalog*, rechercher les occurrences commençant par pio. On choisira le bloc IP *PIO* (*Parallel I/O*) *Intel FPGA IP*. On configurera le bloc IP en entrée (*input*), avec une largeur de 4 bits, avec un registre de capture sur front descendant et capture synchrone (*edge capture register*, *synchronously capture*) et avec génération d'interruptions sur front (*generate IRQ*, *IRQ type edge*). On ne touchera pas à la valeur par défaut des autres champs. On renommera le bloc IP *bp*. On obtient la figure suivante :



Bloc IP bp (13)

• Ajouter le bloc IP *PIO*. Dans la fenêtre *IP Catalog*, rechercher les occurrences commençant par pio. On choisira le bloc IP *PIO (Parallel I/O) Intel FPGA IP*. On configurera le bloc IP en entrée (*input*), avec une largeur de 10 bits. On ne touchera pas à la valeur par défaut des autres champs. On renommera le bloc IP *switchs*. On obtient la figure suivante :



Bloc IP switchs (14)

• Ajouter le bloc IP *PIO*. Dans la fenêtre *IP Catalog*, rechercher les occurrences commençant par pio. On choisira le bloc IP *PIO (Parallel I/O) Intel FPGA IP*. On configurera le bloc IP en sortie (*output*), avec une largeur de 10 bits. On ne touchera pas à la valeur par défaut des autres champs. On renommera le bloc IP *leds*. On obtient la figure suivante :



Bloc IP leds (15)

• Ajouter enfin le bloc IP *7SEG_IF* (7 segments). Dans la fenêtre *IP Catalog*, rechercher les occurrences commençant par seg. On choisira le bloc IP *7SEG_IF*. On configurera le bloc IP avec *SEG7_NUM* égal à 6. On ne touchera pas à la valeur par défaut des autres champs. On renommera le bloc IP *seg7*. On obtient la figure suivante :



Bloc IP seg7 (16)

Nous avons intégré tous les blocs IP dont nous avons besoin dans notre système SoPC.

Bien sûr, le travail n'est pas terminé car nous avons encore des erreurs à corriger en ce qui concerne la connexion de l'horloge, du reset, des bus de données et d'instructions, des interruptions, de l'initialisation de la table des vecteurs d'interruption et du lien avec le fichier Verilog DE10_Standard_golden_top.v.

Si tout va bien, on obtient la figure suivante :

	5	ystem: r	ilos2 Path: seg7										
Jse		Connectio	ons Name	Description	Export	Clock	Base	_	En	ıd	IRQ	Tags	Opcode Nam
R			E E pli	System and SDRAM Clocks for DE-series Boards Clock loput		unconnected							
			ref_cik	Reset Input		unconnecteu							
			sys clk	Clock Output		oll sys clk							
			sdram clk	Clock Output		pll sdram clk							
			reset source	Reset Output									
×			田壇 nios2	Nios II Processor		5274							
	¢ ¢		clk	Clock Input	Double-click to	unconnected							
			reset	Reset Input	Double-click to	[clk]							
		(data_master	Avalon Memory Mapped Master	Double-click to	[Clk]							
			instruction_mas	Interrupt Receiper	Double-click to	[CIK]		TPO O		TP0 21			
			debug reset re	Reset Output	Double-click to	Icki		THA O		TIM OT			
		++	debug mem sla	ave Avalon Memory Mapped Slave	Double-click to	[c]k]	- 0x0800		xofff				
			custom instruc	ti Custom Instruction Master	Double-click to	50.05							
R			🗄 sdram	sdram_64mb									
			cik	Clock Input	Double-click to	unconnected							
			reset	Reset Input		[clk]							
			sl	Avaion Memory Mapped Slave		[clk]	10 ¹						
120			wire El timor	Internal Timer Intel ERGA IR	Double-click to	[CIK]							
			Cike .	Clock Input	Double-click to	unconnected							
			reset	Reset Input	Double-click to	[c]k]							
		00	sl sl	Avalon Memory Mapped Slave	Double-click to	[clk]	10						
		¢	irq	Interrupt Sender	Double-click to	[clk]							
r			🖽 jtag	ITAG UART Intel FPGA IP									
			clk	Clock Input		unconnected							
			reset	Reset input		[CIK]							
			avaion_ltag_sia	listerrupt Conder		[CIK]							
			Elbp	PIO (Parallel I/O) Intel FPGA IP	Dominie Circk (D	(civ)					1 T		
Adda at	\rightarrow	_	clk	Clock Input	Double-click to	unconnected							
	1		reset	Reset Input	Double-click to	[clk]							
		00	sl	Avalon Memory Mapped Slave	Double-click to	[clk]	al						
			external_conne	c Conduit	Double-click to								
1.22		0-	Ing	Interrupt Sender		[clk]					-0		
2			E switchs	Clock logut		unconnected							
			reset	Reset Input		Iciki							
			sl	Avalon Memory Mapped Slave		Iclk1	all .						
			external conne	c Conduit	Double-click to								
V			🗄 leds	PIO (Parallel I/O) Intel FPGA IP									
	¢-¢-		clk	Clock Input	Double-click to	unconnected							
		11	reset	Reset Input	Double-click to	[clk]							
		ŢŢ,	sl	Avaion Memory Mapped Slave	Double-click to	[CIK]	16.1						
		-	external_conne	SEG7 IE	Doume-click to			-		_			
2			avalon slave	Avalon Memory Manned Slave	Double-click to	Iclock sinkl	w.						
			conduit end	Conduit		tore or Danied							
			clock_sink	Clock Input	Double-click to	unconnected							
			clock_sink_rese	t Reset Input	Double-click to	[clock_sink]							
			conduit_end clock_sink clock_sink_rese	Conduit Clock input t Reset input	Double-click to Double-click to Double-click to	unconnected [clock_sink]							

Système SoPC *DE10_Standard_golden_top* (17)

Nous en avons fini avec l'addition de blocs IP dans le système SoPC.

3.3. Ajout des connexions de signaux

Nous devons connecter les signaux d'horloge, de reset, de bus de données à chaque périphérique mais aussi connecter le bus d'instructions à la mémoire SDRAM.

Dans la fenêtre *Platform Designer*, il y a dans la colonne *Connections* un réseau (X, Y) de connexions de signaux. Il suffit de cliquer à l'interconnexion de 2 signaux pour les interconnecter.

• Interconnecter le signal sortant *sys_clk* du bloc IP *pll* à l'entrée *clk* des autres blocs IP *nios2, sdram, timer, jtag_uart, bp, switchs, leds* et *seg7 (clk_sink* pour le bloc IP *seg7)*. On obtient la figure suivante :

Eile Edit System Generate View Tools Help				Flation Designer - mosz	qsys (nome/kautomk/Artera/mess//desi	gn/niosz.qsys/					×
P Catalog 🕴 – 🗗 🖸		System	Contents 🕴 Add	iress Map 💠 Interconnect F	equirements 💠						- 6 0
			System: nios2	Path: 7segs.clock_sink							
Desile at	1.4	Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Tac
Project		K		回國 pll	System and SDRAM Clocks for DE-series Boards						-
 sdram 128mb 				ref_clk	Clock Input	Double-click to	unconnected				
 sdram_32mb 	-			ref_reset	Reset Input	Double-click to					
 sdram_64mb 				sys_clk	Clock Output		pll_sys_clk				
← System				sdram_cik	Report Output		pil_sdram_cik				
 Terasic Technologies Inc. 				E D nios2	Nios II Processor	DOUDIC+CIICK (D					
Basic Functions			• •		Clock Input	Double+click to	pll sys clk				
► DSP			00	reset	Reset Input	Double+click to	[clk]				
- Interface Protocols				data_master	Avalon Memory Mapped Master	Double-click to	[clk]				
Memory Interfaces and Controllers				instruction_master	Avalon Memory Mapped Master	Double-click to	[CIR]	100	0 100		
 Processors and Peripherals 				debug reset request	Reset Output	Double_click to	(cik)	THU	C THU	21	
 Qsys Interconnect 				→ debug mem slave	Avalon Memory Mapped Slave	Double-click to	[clk]	- 0x0800	OxOfff		
 Tri-State Components 			200	custom_instruction	Custom Instruction Master	Double-click to			1000306300		
- Onwersity Program		×		🗄 sdram	sdram_64mb						
			1		Clock Input		pll_sys_clk				
			1	el	Avalon Mamoni Manned Slave		[CIK] [cik]	<i>a</i>			
				wire	Conduit		(cik)				
		~		🗄 timer	Interval Timer Intel FPGA IP						
			• •		Clock Input	Double-click to	pll_sys_clk				
			0 0	reset	Reset Input	Double-click to	[cik]				
New Edit + Add			0-0-0-	SI Ico	Avaion Memory Mapped Slave	Double-click to	[CIK]	ad		-	
		V		🖽 itag uart	ITAG UART Intel FPGA IP		Teach				
1 Hierarchy II Device Family II			÷ · · · · · · · · · · · · · · · · · · ·		Clock Input	Double-click to	pll_sys_clk				
	8		0 0	reset	Reset Input	Double-click to	[clik]				
clock sink reset			0.0	avalon_jtag_slave	Avalon Memory Mapped Slave		[CIK]	<i></i>			
coduit end	_			Ebn	PIO (Parallel I/O) Intel EPGA IP	compar-caucie to	[Cik]			TY I	
• ⊕ bp			• •		Clock Input	Double-click to	pll sys clk				
🔶 🖿 clk			0	reset	Reset Input	Double-click to	[clk]				
external_connection			0-0	s1	Avalon Memory Mapped Slave	Double-click to	[clik]	40°			
← ➡ irq				external_connection	Conduit Internet Conder	Double-click to	falls1				
o- ► reset				E switchs	PID (Parallel I/O) Intel EPGA IP		(Cik)				-
a Dittan uart		-		- alk	Clack Input	Double click to	all eve elk				-
A page dans			1 (1997)								
← ➡ clk		P~ 5	💙 🛒 Current fil	ter:							
o- ➡ irq	1 X		20								
🔶 🖬 reset	0.5	Messa	ges								
P ⊕ leds		Туре	Path			Me	sage				8
o- ► cik	9	8	14 Errors								<u>^</u>
external_connection		8	nios2.7segs M	lodule name cannot start with a	i digit						-
eser si		8	nios2.nios2 In	struction Cache is larger than	he Instruction Address. Please reduce the Instruc	tion Cache Size. Curre	nt Tag Size is 0				
• 🖷 nios2		8	nios2.nios2 B	eset slave is not specified. Plea	se select the reset slave						
← ➡ clk			nioc2 nioc2	sention size is not specified	Bloose select the exception slove						
custom_instruction_master		~	11052.11052	Aception slave is not specified.	riease select the exception slave						
• 🛥 data_master			niosz.pil p	II.rer_cik must be connected t	o a clock output						
• • debug_mem_slave		8	nios2.7segs 7	segs.clock_sink_reset must l	e connected to a reset source						
e instruction master		8	nios2.bp b	p.reset must be connected to	a reset source						
- met detter											•
14 Errors, 15 Warnings									Ge	nerate HDL	Finish

Connexion du signal d'horloge sys_clk (18)

• Interconnecter le signal sortant *reset_source* du bloc IP *pll* à l'entrée *reset* des autres blocs IP *nios2*, *sdram*, *timer*, *jtag_uart*, *bp*, *switchs*, *leds* et *seg7* (*clk_sink_reset* pour le bloc IP *seg7*). On obtient la figure suivante :

					Platform Designer - nios	2.qsys* (/home/kadionik/Altera/me357/desi	ign/nios2.qsys)					
File Edit System Generate View Loois He	eip 		urtom	Contonte 🕅 Ade	Interconnect	Requiremente 😚						
r catalog to		••••	ystem	concerns to Add	Dethy Zeree	Requirements (%						
	× 🕸			System: mos2	Path: /segs			1		1	Long II	-
Project New Component sdram 128mb sdram 32mb			Use	Connections	name P P PI ref_clk ref reset	Description System and SDRAM Clocks for DE-series Boards Clock Input Reset Input	Double-click to Double-click to	unconnected	Base	End	IRQ	Taç -
sdram_64mb System Terasic Technologies Inc.			-		sys_clk sdram_clk reset_source	Clock Output Clock Output Reset Output	Double-click to Double-click to Double-click to	pll_sys_clk pll_sdram_clk				
Basic Functions DSP Drafface Protocols Low Power Memory Interfaces and Controllers Processors and Peripherals Opys Interconnect Trisfate Components		H H			→ clk reset data_master instruction_master irq debug_reset_reques debug_mem_slave	Clock Input Reset Input Avalon Memory Mapped Master Inferrupt Receiver Reset Output Avalon Memory Mapped Slave	Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to	pil_sys_clk [clk] [clk] [clk] [clk] [clk] [clk]	IRQ (# 0x0800	3 OxOfff	IRQ 31	
			X		custom instruction. sdram clk reset s1 wire	Lustom Instruction Master sdram_64mb Clock Input Reset Input Avalon Memory Mapped Slave Conduit	Double-click to Double-click to Double-click to Double-click to Double-click to	pll_sys_clk [clk] [clk] [clk]				
New			V	••••••••••••••••••••••••••••••••••••••	clk reset s1 irq	Interval Imerintell PPGA IP Clock Input Avalon Memory Mapped Slave Interrupt Sender TAG LIABT Indel EPGA IP	Double-click to Double-click to Double-click to Double-click to	pll_sys_clk [clk] [clk] [clk]	di l		,¢	
Hierarchy Device Family Device Fami	- d' ⊏	1		• • • • •	→ clk → reset avalon_jtag_slave	Clock Input Reset Input Avalon Memory Mapped Slave	Double-click to Double-click to Double-click to Double-click to	pll_sys_clk [clk] [clk]	-			
← ➡ clock_sink ← ➡ clock_sink_reset ← ➡ conduit_end			×	•••••	⊖ bp clk reset	PIO (Parallel VO) Intel FPGA IP Clock Input Reset Input Valon Memory Mapped Slave	Double-click to Double-click to Double-click to	pll_sys_clk [clk]				
 ➡ clk ➡ external_connection ➡ irq 	-		×	o	external_connection irq B switchs	Conduit Interrupt Sender PIO (Parallel VO) Intel FPGA IP	Double-click to Double-click to	[clk]				
o- ■ reset o- ■ s1 p- ■ itag_uart c- ■ avaion itag slave			• - + 井	👻 🗑 Current fil	ter:	1						•
clk	-	ôs N	lessag	jes 23								- 5
e- ■ reset		2 23	the	6 Errors			Me	วรจนิด				
9 - Eleds			0	nios2.7segs M	Iodule name cannot start with	a digit						
external connection			0	nios2.nios2 In	struction Cache is larger than	the Instruction Address. Please reduce the Instruc-	tion Cache Size. Curre	ent Tag Size is 0				_
🕶 🖿 reset			8	nios2.nios2 R	eset slave is not specified. Ple	ase select the reset slave						
o- ► sl			8	nios2.nios2 E	eception slave is not specified.	Please select the exception slave						
e 🖬 nios2			8	nios2.pll n	II.ref dk must be connected	to a clock output						
- d custom instruction master			0	nios2.pll p	Il.ref reset must be connected	ed to a reset source						
🕶 🛥 data_master			-	15 Warpings								
► debug_mem_slave	*			a o traininga								
6 Errors, 15 Warnings											Generate H	DL Finisl

Connexion du signal de reset reset_source (19)

• Interconnecter le signal sortant *data_master* du bloc IP nios2 à l'entrée *s1* des autres blocs IP *sdram, timer, jtag_uart (avalon_jtag_slave* pour le bloc IP *jtag_uart), bp, switchs, leds* et *seg7 (avalon_slave* pour le bloc IP *seg7)*. On obtient la figure suivante :



Connexion du signal data_master (20)

• Interconnecter le signal sortant *instruction_master* du bloc IP *nios2* à l'entrée *s1* du bloc IP *sdram* uniquement. En effet, la mémoire SDRAM contient les données et les instructions d'un programme à exécuter. On obtient la figure suivante :

					Platform De	signer - nios	.qsys* (/home/kadionik/Altera/me357/des	ign/nios2.qsys)				
<u>Eile E</u> dit <u>S</u> ystem <u>G</u> enerate <u>V</u> iew <u>T</u> ools <u>H</u> e	lp											
🖰 IP Catalog 🛛	- d' =		System	Contents 🕺	Address Map 🛛 🕅	Interconnect	Requirements 🕸					- ದೆ
0.	26 (25)			System: n	os2 Path: sdrai	m.sl						
Paral and		-	Use	Connectio	ns	Name	Description	Export	Clock	Base	End	IRQ Tar
sdram_128mb		×	×		B∰ pll ref_c ref r	:lk eset	System and SDRAM Clocks for DE-series Boards Clock Input Reset Input	Double-click to Double-click to	unconnected			
sdram_64mb system Terasic Technologies Inc.					sys_ sdra	clk m_clk t_source	Clock Output Clock Output Reset Output	Double-click to Double-click to Double-click to	pll_sys_clk pll_sdram_clk			
Library Basic Functions CDSP Interface Protocols			×	•	→ Clk rese	t mastar	Nios II Processor Clock Input Reset Input	Double-click to Double-click to Double-click to	pll_sys_clk [clk]			
Low Power Memory Interfaces and Controllers Processors and Peripherals					instr irq debi	uction_master	Avalon Memory Mapped Master Interrupt Receiver Reset Output	Double-click to Double-click to Double-click to	[clk] [clk] [clk]	IRQ	0 IRQ 31	7
Gyys Interconnect Gyse Interconnect Tri-State Components University Program			V		debi cust	ug_mem_slave om_instruction n	Avalon Memory Mapped Slave Custom Instruction Master sdram_64mb	Double-click to Double-click to	[clk]	= 0x0000_0800	0x0000_0fff	
				••			Clock Input		pll_sys_clk			
					rese	t	Aralon Memory Manned Slave	Double-click to	(CIK)	0×0000 0000	0x0264 4444	
					· wire		Conduit	Double-click to	[cik]	0,0000_0000	oxoshi_hhh	
				•••	→ timer clk		Interval Timer Intel FPGA IP Clock Input	Double-click to	pll_sys_clk			
New Edit	- Add			••	s1 irq	vart	Avalon Memory Mapped Slave Interrupt Sender	Double-click to Double-click to	[clk] [clk]	e 0x0000_0000	0x0000_001f	-0
Hierarchy 💠 Device Family 🕸				•	clk rese	t on itan slave	Clock Input Reset Input Avalon Memory Manned Slave	Double-click to Double-click to Double-click to	pll_sys_clk (clk) (clk)	- 0x0000 0000	6x6000_0007	
🗢 📭 Irq				~	irq El bp		Interrupt Sender PIO (Parallel I/O) Intel EPGA IP	Double-click to	[clk]			-0
e leds e ■ clk				•••••••	clk rese	t	Clock Input Reset Input	Double-click to Double-click to	pll_sys_clk [clk]			
 external_connection reset e1 				0	exte	rnal_connection	Avaion Memory Mapped Slave Conduit Interrupt Sender	Double-click to Double-click to Double-click to	[cik]	=, 0X0000_0000	0x0000_0001	-0
• 00 nios2			K		🗆 swite	hs	PIO (Parallel I/O) Intel FPGA IP					
• ► clk			•	•	and the second		Clock Input	Double click to	foll over elk			
- data master			n~ #t	V Curren	t filter:							
• • debug_mem_slave		X.	Messa	ges 23								- 5
 debug_reset_request instruction_master 	-		Type		Path	1			Message			
🗢 🛥 irg		9 6	•	19 Errors								
• Freset			8	nios2.7segs		Module name	cannot start with a digit					
Clock_bridge			0	nine2 nine2		Reset slove is	not energied. Please select the reset slove					
e e cpu			õ	plac3 plac3		Exception sta	a is not exactled. Blance calest the accention of					
- U reset_bridge		1		mosz.mosz		Exception stav	e is not specified. Please select the exception slav	e				
• III ol			8	nios2.nios2.d	ata_master	timer.s1 (0x0	uxit) ovenaps sdram.sl (0x00x3#####)					
• D sdram			8	nios2.nios2.d	ata_master	jtag_uart.av	alon_jtag_slave (0x00x7) overlaps timer.s1 (0x0)0x1f)				
• • clk			8	nios2.nios2.d	ata_master	bp.sl (0x00	d) overlaps jtag_uart.avalon_jtag_slave (0x00)	(7)				
🕶 🖿 reset			8	nios2.nios2.d	ata master	switchs.sl ()	0x00xf) overlaps bp.sl (0x00xf)					
← ■ 51	-		-				J.					
												La LIDI

Connexion du signal instruction_master (21)

Nous en avons fini avec les interconnexions de signaux du système SoPC.

3.4. Ajout des interruptions

Les blocs IP *timer*, *jtag_uart* et *bp* peuvent travailler sous interruption. On doit donc préciser leur numéro d'interruption, de 0 à 31.

• Dans la fenêtre *Platform Designer*, il y a dans la colonne IRQ le réseau de connexions des interruptions. Pour ces 3 périphériques, il suffit de cliquer sur le petit losange pour définir le numéro d'interruption. Peut importe la valeur. On obtient la figure suivante (avec pour les valeurs d'IRQ : 0 à 2 ici) :



Ajout des interruptions (22)

Nous en avons fini avec les interruptions du système SoPC.

3.5. Définition du mapping mémoire

Il faut définir le mapping mémoire des périphériques du système SoPC pour qu'il n'y ait pas de recouvrement mémoire.

- Choisir le menu *System > Assign Base Addresses*. On observe que le nombre de messages d'erreurs en rouge a drastiquement diminué.
- Mapper la mémoire SDRAM à partir de l'adresse 0x0000_0000. Pour cela, on cliquera dans la colonne *Base* de la fenêtre *Platform Designer* sur la valeur qui a été attribuée par défaut (0x0400_000) pour l'ajuster à 0x0000_0000. On obtient la figure suivante :

<u>Eile Edit System G</u> enerate <u>V</u> iew <u>T</u> ools <u>H</u> elp											
💾 IP Catalog 🔤 🗕 🖬 🗖	- 12	Syste	m Contents 🕺 Addre	is Map 💠 Interconnect P	lequirements 🔅						- d c
s ר			System: nios2	ath: sdram.sl							
Project	1	Use	Connections	Name E 🖳 pll	Description System and SDRAM Clocks for DE-series Boards	Export	Clock	Base	End	IRQ	Taç
• sdram 128mb				ref_clk	Clock Input	Double-click to	unconnected				
 sdram_32mb 				ref_reset	Reset Input		oll out elle				
• sdram_64mb	12			sdram clk	Clock Output		pll_sys_cik				
← Terasic Technologies Inc.				<pre>reset_source</pre>	Reset Output	Double-click to					
Library				日壇 nios2	Nios II Processor						
← Basic Functions				clk	Clock Input	Double-click to	pll_sys_clk				
← Interface Protocols	T			data master	Avalon Memory Mapped Master	Double-click to	(cik)				
🗢 Low Power	-			 Instruction_master 	Avalon Memory Mapped Master	Double-click to	[clk]				
 Memory Interfaces and Controllers 				+ Irq	Interrupt Receiver	Double-click to	[clk]	IRQ	0 IRQ	31	
 Processors and Peripherals Osys Interconnect 				debug_reset_request	Reset Output	Double-click to	[cik]	0.0000 0000	0.0000 0444		
 Tri-State Components 				custom instruction	Custom Instruction Master	Double-click to	[Cik]	- 0X0800_0800	0x0800_0111		-
 University Program 				⊟ sdram	sdram 64mb	Double-click to					
			• • • • • • • • • • • • • • • • • • • •	→ clk	Clock Input	Double-click to	pll_sys_clk				
			• •	+ reset	Reset Input	Double-click to	(clic)		0.0000.0000		
			~	si wine	Conduit	Double-click to	(cik)	0X0000_0000	UXUSTT_TTTT		
		~		🖯 timer	Interval Timer Intel FPGA IP	a carbre carete co	[1000]				
			•••	 clk 	Clock Input	Double-click to	pll_sys_clk				
			• • • •	reset	Reset Input	Double-click to	[cik]				
New Edit + Add				f SI	Avaion Memory Mapped Slave	Double-click to	[CIK]	= 0x0800_1020	0X0800_103T	-6	
				🗄 jtag uart	JTAG UART Intel FPGA IP		fearl			Ĩ	
🤨 Hierarchy 🛞 Device Family 🛞 🗕 🗗 🗖			+	+ clk	Clock Input	Double-click to	pll_sys_clk				
	8		• <u></u>	+ reset	Reset Input		{clk}		0.0000 1077		
• ■ 51				avaion_tag_slave	Avaion Memory Mapped Slave		(CIK)	= 0X0800_1010	0X0800_10//	<u>_</u>	
or ■ clk		2		E bp	PIO (Parallel I/O) Intel FPGA IP		10001			Ϋ́	
- d custom instruction master			+ + + + + + + + + + + + + + + + + + + +	+ clk	Clock Input	Double-click to	pll_sys_clk				
🗢 🛋 data_master			• • • • •	+ reset	Reset Input	Double-click to	[clik]	0.0000 1000	0.0000 1004		
► debug_mem_slave				external connection	Conduit	Double-click to	[Cik]	= 0X0800_1060	0X0800_1061		
- debug_reset_request			•	< irq	Interrupt Sender	Double-click to	[clik]			-2	
e ing				🗄 switchs	PIO (Parallel I/O) Intel FPGA IP						-
er ⊫reset		•		al alk	Clack Innut	Inauhla eliek to	all over elk				
- 🗢 clock_bridge		nly 4	Current filter								
🗢 🖨 cpu		DUCK	Current inter	•							
• • reset_bridge	XXX	Mess	ages 23								
Connections		Type	Path	1			Messane				
• D sdram	0	C	5 Errore				heessage				
🔶 🛏 cik		- 0	plac2 Team	Madula name	connet start with a dialt						
← ► reset			mosz./segs	Module Harie I	annot start with a digit						
← ▶ 51			niosz.niosz	Reset slave is	not specified. Please select the reset slave						
o wre		8	nios2.nios2	Exception slaw	e is not specified. Please select the exception slave	e					
		8	nios2.pll	pll.ref_clk mu	st be connected to a clock output						
- external connection		8	nios2.pll	pll.ref_reset	must be connected to a reset source						
∽ ► reset	9	1	5 Warnings								
► s1		À	nios2.7segs	7segs.condu	it end must be exported, or connected to a match	ning conduit.					-
P Detimer		_	and the state of the second	and the second second	and the second sec						•
5 Errors, 5 Warnings									Ge	nerate HD	DL Finish

Mapping mémoire (23)

Nous en avons fini avec le mapping mémoire du système SoPC.

3.6. Définition des vecteurs d'exception

Le processeur NIOS II a 2 vecteurs d'exception :

- le vecteur de reset qui précise le point d'entrée du programme exécuté au reset du processeur.
- le vecteur d'interruption en cas d'occurrence d'une interruption. Il précise le point d'entrée de la routine d'interruption ISR (*Interrupt Sub Routine*). Cette routine est commune aux 32 sources d'interruption possibles et il faudra donc dans la routine d'interruption rechercher qui est la cause de l'interruption.
- Dans la fenêtre *Platform Designer*, on double clique sur le bloc IP *nios2*. Dans la fenêtre *Parameters* qui apparait à droite, on choisit l'onglet *Vectors*.
- On choisit pour le champ *Reset vector memory* la valeur *sdram.s1*.
- On choisit pour le champ *Exception vector memory* la valeur *sdram.s1*.

Les deux vecteurs d'exception sont installés dans la mémoire SDRAM aux adresses respectives 0x0000_0000 et 0x0000_0020. On laisse les autres valeurs par défaut. On obtient la figure suivante :

<u>Eile Edit System Generate View Tools Help</u>								
📂 IP Catalog 🕴 🗕 🖬 🗖	12:	System	Contents 🕅 A	ddress Map 💠 Interconnec	ct Requirements 🚳		- d' D	Na Parameters 💠 – 🗗 🗖
			System: nios	2 Path: nios2				System: nios2 Path: nios2
The local data	F	Use	Connections	Name	Description	Export	Clc	Nios II Processor
Project P		V V V V V V V V V V V V V V V V V V V	Connections Connections	Alame Alame Alame C	Description System and SDMM Clocks for DE-series Boards Clock Input Reset Input Reset Input Reset Tropic News IP Processor Clock Toput News IP Processor Clock Toput News IP Processor Clock That I Processor Autom Memory Happed Master News IP Processor Autom Memory Happed Stave Custom Instruction Naster Autom Memory Happed Stave Custom Instruction Naster Autom Memory Happed Stave Conduit meet PCA P Clock Input Reset Tropic Autom Memory Happed Stave Memory Happed Stave Memory Happed Stave Internut Starder PIC Parallel (0) Prot PCA P Clock Input News IP PCA P Clock Input Reset Tropic Tabulater Treat PCA P Clock Input News IP PCA P Clock Input Net PCA PCA PCA P Clock Input Net PCA PCA P Clock Input Net PCA PCA PCA PCA P Clock Input Net PCA	Expert DIF of (k) DIF of (k)	Ctc exporte pll_sdraw pll_sdraw pll_sdraw pll_sdraw	Nios II Processor Details Intrimute Instructions Molu and MPU Settings TAG Debug Adamed Feature Intrimute Instructions Molu and MPU Settings TAG Debug Adamed Feature Intrimute Instructions Molu and MPU Settings TAG Debug Adamed Feature Intrimute Instructions Molu and MPU Settings TAG Debug Adamed Feature Intrimute Instructions Instructions Instructions Carbo Reset vector Offeet: 0x0000000 Instructions Instructions Exception vector offeet: 0x00000200 Instructions Instructions Fast: TLB Miss Exception vector offeet: 0x0000020 Instructions Instructions Fast: TLB Miss Exception vector offeet: 0x00000000 Instructions Instructions Instructions Fast: TLB Miss Exception vector offeet: 0x00000000 Instructions Instructions Instructions Fast: TLB Miss Exception vector offeet: 0x00000000 Instructions Instructions Instructions Fast: TLB Miss Exception vector offeet: 0x00000000 Instructions Instructions Instructions Fast: TLB Miss Exception vector
🗢 🕼 timer		0	nios2.jtag	TAG UART IP input clock need	to be at least double (2x) the operating frequence	y of JTAG TCK on boa	rd	
Connections		0	nios2.pll	Refclk Freq: 50.0	in the second seco	, ,		
		0	nios2.switchs	PIO inputs are not hardwired in	n test bench. Undefined values will be read from F	10 inputs during simi	ulation.	
	•)
0 Errors 0 Warnings								Generate HDI Cinish
o unora, o manninga								Generate HDL Philsi

Vecteurs d'exception (24)

Nous en avons fini avec les vecteurs d'exception du système SoPC.

3.7. Exportation des signaux externes au circuit FPGA

Certains signaux du système SoPC sont externes au circuit FPGA car ils sont connectés aux périphériques externes comme la mémoire SDRAM, les boutons poussoir, les switchs, les leds et les afficheurs 7 segments. Il y a aussi l'horloge externe et le reset.

Dans la fenêtre *Platform Designer*, il y a la colonne *Export* pour l'exportation des signaux externes au circuit FPGA.

- Dans la fenêtre *Platform Designer*, on double clique sur le champ *Export* du signal *ref_clk* du bloc IP *pll*. Il apparait la valeur *pll_ref_clk* que l'on gardera par défaut.
- On double clique sur le champ *Export* du signal *ref_reset* du bloc IP *pll*. Il apparait la valeur *pll_ref_reset* que l'on gardera par défaut.
- On double clique sur le champ *Export* du signal *sdram_clk* du bloc IP *pll*. Il apparait la valeur *pll_sdram_clk* que l'on gardera par défaut.
- On double clique sur le champ *Export* du signal *wire* (*Conduit*) du bloc IP *sdram*. Il apparait la valeur *sdram_wire* que l'on changera en *sdram*.
- On double clique sur le champ *Export* du signal *external_connection* (*Conduit*) du bloc IP *bp*. Il apparait la valeur *bp_external_connection* que l'on changera en *bp*.
- On double clique sur le champ *Export* du signal *external_connection* (*Conduit*) du bloc IP *switchs*. Il apparait la valeur *switchs_external_connection* que l'on changera en *switchs*.
- On double clique sur le champ *Export* du signal *external_connection* (*Conduit*) du bloc IP *leds*. Il apparait la valeur *leds_external_connection* que l'on changera en *leds*.
- Enfin, on double clique sur le champ *Export* du signal *conduit_end* (*Conduit*) du bloc IP *seg7*. Il apparait la valeur *seg7_conduit_end* que l'on changera en *seg7*.

On notera qu'il n'y a plus de messages d'erreurs.

Nous en avons fini avec l'exportation des signaux externes du système SoPC.

On obtient alors le système SoPC suivant :

stem	content	addn	Interconnect i	Requirements 🗠							
	Sys	tem: nios2	Path: seg7.conduit_end								
	Co	nnections	Name	Description	Export	Clock	Base	End	IRQ	Tags	Opcode Name
	H		B B pH - ref_clk - ref_reset ≺ sys_clk - sdram_clk ← setar_source	System and SDRAM Clocks for DE-series Boards Clock Input Reset Input Clock Output Clock Output Beast Output	pll_ref_clk pll_ref_reset pll_sdram_clk	e <i>xported</i> pll_sys_clk pll_sdram_clk					
			B B B is ros2 clk clk reset data_master instruction_master irq debug_reset_request debug_mem_slave custom_instruction	Nios II Processor Clock Input Reset Input Avalon Memory Mapped Master Interrupt Receiver Reset Output Reset Output Avalon Memory Mapped Slave Custom Instruction Master	Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to	pll_sys_clk [clk] [clk] [clk] [clk] [clk] [clk] [clk] [clk]	IRQ (# 0x0800_0800	0 IRC 0x0800_0fff	31←		
×	•	• •	⊟ sdram → clk → reset → s1 ↔ wire	sdram_64mb Clock input Reset Input Avalon Memory Mapped Slave Conduit	Double-click to Double-click to Double-click to sdram	pll_sys_clk (clk) (clk) (clk)	= 0x0000_0000	0x03f1_1111			
×	•	•••	⇒ clk → reset → s1 → irq	interval Timer Intel FPGA IP Clock Input Reset Input Avalon Memory Mapped Slave Interrupt Sender	Double-click to Double-click to Double-click to Double-click to	pll_sys_clk [clk] [clk] [clk]	# 0x0800_1020	0x0800_103f)—la		
×	•	•••	<pre></pre>	(TAG UART Intel FPGA IP Clock Input Reset Input Avalon Memory Mapped Slave Interrupt Sender	Double-click to Double-click to Double-click to Double-click to	pll_sys_clk [clk] [clk] [clk]	# 0x0800_1070	0x0800_1077			
×	•	••	→ clk → reset → s1 ← external_connection	PIO (Parallel I/O) Intel FPGA IP Clock Input Reset Input Avalon Memory Mapped Slave Conduit	Double-click to Double-click to Double-click to bp	pll_sys_clk [clk] [clk]	# 0x0800_1060	0x0800_106f			
2	•	•	→ irq ⇒ switchs → ck → reset → s1 ↔ external connection	Interrupt Sender PIO (Parallel VO) Intel FPGA IP Clock Input Reset Input Avalon Memory Mapped Slave Conduit	Double-click to Double-click to Double-click to Double-click to switchs	(clk) pli_sys_clk (clk) (clk)	e 0x0800_1050	0x0800_105f	-2		
Y	•	•		Pio (Parallel VO) Intel FPGA IP Clock Input Reset Input Avalon Memory Mapped Slave Conduit	Double-click to Double-click to Double-click to leds	pll_sys_clk [clk] [clk]	# 0x0800_1040	0x0800_104f			
			⇒ seg7 avalon_slave ⊂ conduit_end	SEG7_IF Avalon Memory Mapped Slave Conduit	Double-click to	(clock_sink)	# 0x0800_1000	0x0800_101f			
			CIOCK_SINK	clock input		pii_sys_cik					

Système SoPC DE10_Standard_golden_top (25)

3.8. Génération HDL du système SoPC

Dans la fenêtre *Platform Designer*, on clique sur le bouton *Generate HDL* pour générer les sources HDL du système SoPC.

On obtient la fenêtre suivante :

	Generation _ = ×
* Synthesis	
Synthesis files are used to	compile the system in a Quartus project.
Create HDL design files for	synthesis: Verilog 🔻
Create timing and resou	irce estimates for third-party EDA synthesis tools.
Simulation	(1997)
The simulation model conta Simulation scripts for this of Follow the guidance in the use the <i>ip-setup-simulation</i> the IP in your design. Create simulation model:	ains generated HDL files for the simulator, and may include simulation-only features. omponent will be generated in a vendor-specific sub-directory in the specified output directory. generated simulation scripts about how to structure your design's simulation scripts and how to and <i>ip-make-simscript</i> command-line utilities to compile all of the files needed for simulating all of None
 Output Directory 	
Path:	/home/kadionik/Altera/me357/design/nios2
l	Generate Cancel

Génération HDL du système SoPC DE10_Standard_golden_top (26)

- Cliquer sur le bouton *Generate*.
- Après génération, cliquer sur le bouton *Finish* pour sortir de *Platform Designer*.

La fenêtre d'indications suivante apparait :



Indication sur l'inclusion du fichier .qip (27)

Elle signale qu'il faudra intégrer le fichier . qip dans son projet *Quartus Prime* pour intégrer le système SoPC dans la synthèse.

Notre système SoPC est sauvegardé dans le fichier nios2.qsys.

3.9. Synthèse du système SoPC

On revient maintenant à l'outil Quartus Prime.

Le projet *DE10_Standard_golden_top* est toujours ouvert.

• Ajouter le fichier nios2.qip qui est sous design/nios2/synthesis/ au projet *Quartus Prime*. On choisit le menu *Project > Add/Remove Files in Project*. On obtient la fenêtre suivante :

	Settings - D	E10_Standard_golden_top			1 403
ategory:					Device/Board
General	Files				
Files Libraries • IP Settings	Select the design files y directory to the project	you want to include in the project. (Click Add All to add	d all design files in	the project
IP Catalog Search Locations Design Templates	<u>F</u> ile name:				Add
 Operating Settings and Condition: Voltage 				×	Add All
Temperature	File Name	Туре	Libra	ary Design Entry/	Remove
 Compilation Process Settings Incremental Compilation EDA Tool Settings Design Entry/Synthesis 	DE10_Standard_go	DE10_Standard_golden_top.sdc_Synopsys Design Constraints File			
Simulation					Down
Verilog HDL Input Default Parameters Timing Analyzer Assembler Design Assistant Signal Tap Logic Analyzer Logic Analyzer Interface Power Analyzer Settings SSN Analyzer	-				
			1		
			OK Cano	el Apply	Help

Ajout du fichier nios2.qip (28)

• Ajouter le fichier nios2.qip en cliquant sur le bouton "…". On obtient la fenêtre suivante :

e	Settings - DE10_Stand	ard_golden_top			_ 0 3
Category:					Device/Board
General	Files				
Files Libraries V IP Settings	Select the design files you want to in directory to the project.	clude in the project. Click Add All	to add all design file	s in t	he project
Design Templates	<u>F</u> ile name:				Add
 Operating Settings and Conditions Voltage 				\times	Add A <u>l</u> l
Temperature	File Name	Туре	Library Design Ent	try/S	Remove
Incremental Compilation * EDA Tool Settings Design Entry/Synthesis Simulation Board-Level * Compiler Settings VHDL Input Verilog HDL Input Default Parameters Timing Analyzer Assembler Design Assistant Signal Tap Logic Analyzer Logic Analyzer Interface Power Analyzer Settings SSN Analyzer	DE10_Standard_golden_top.sdc	: Synopsys Design Constraints File	e <none></none>		Up Down Properties
	4	N		•	
K		ОК	Cancel App	ly	Help

Fichier nios2.qip ajouté (29)

Il faut enfin aussi modifier le fichier Verilog *Top-Level* DE10_Standard_golden_top.v pour faire le lien entre les signaux exportés (*Conduit*) du système SoPC et les signaux externes du niveau *Top-Level*.

L'outil Platform Designer a généré un canevas Verilog pour nous faciliter la tâche.

Sous *Quartus Prime*, on ouvrira le fichier *Top-Level* DE10_Standard_golden_top.v mais aussi le fichier canevas sous design/nios2/nios2_inst.v.

Le fichier nios2_inst.v contient :

nios2 u0 (.pll_ref_clk_clk .pll_ref_reset_reset .pll_sdram_clk_clk .sdram_ba .sdram_cas_n .sdram_ccs_n .sdram_cke .sdram_cs_n .sdram_dq .sdram_dqm .sdram_ras_n .sdram_ras_n .bp_export .switchs_export	<pre>(<connected-to-pll_ref_clk_clk>), (<connected-to-pll_ref_reset_reset>), (<connected-to-sdram_clk_clk>), (<connected-to-sdram_addr>), (<connected-to-sdram_cas_n>), (<connected-to-sdram_cke>), (<connected-to-sdram_cke>), (<connected-to-sdram_dq>), (<connected-to-sdram_dq>), (<connected-to-sdram_qdp>), (<connected-to-sdram_ras_n>), (<connected-to-sdram_we_n>), (<connected-to-bp_export>), (<connected-to-switchs_export>),</connected-to-switchs_export></connected-to-bp_export></connected-to-sdram_we_n></connected-to-sdram_ras_n></connected-to-sdram_qdp></connected-to-sdram_dq></connected-to-sdram_dq></connected-to-sdram_cke></connected-to-sdram_cke></connected-to-sdram_cas_n></connected-to-sdram_addr></connected-to-sdram_clk_clk></connected-to-pll_ref_reset_reset></connected-to-pll_ref_clk_clk></pre>		<pre>pll_ref_clk.clk pll_ref_reset.reset pll_sdram_clk.clk sdram.ba .addr .cas_n .cke .cs_n .dq .dqm .ras_n .we_n bp.export switchs.export</pre>
.switchs_export .leds_export	<pre>(<connected-to-switchs_export>), (<connected-to-leds_export>),</connected-to-leds_export></connected-to-switchs_export></pre>	 	switchs.export leds.export
.seg7_export);	(<connected-to-seg7_export>)</connected-to-seg7_export>	//	seg7.export

Il faudra copier ce canevas dans le fichier DE10_Standard_golden_top.v qu'il faudra ensuite modifier pour raccrocher les signaux externes du système SoPC aux signaux externes du circuit FPGA.

Le fichier DE10_Standard_golden_top.v devient alors :

```
module DE10_Standard_golden_top(
```

```
/////// CLOCK ////////
                   CLOCK2_50,
input
                   CLOCK3_50,
input
input
                   CLOCK4_50,
                   CLOCK_50,
input
/////// KEY ////////
        [ 3: 0]
input
                  KEY,
/////// SW ////////
        [ 9: 0]
input
                   SW,
/////// LED ////////
output [ 9: 0]
                  LEDR,
/////// Seg7 ////////
       [ 6: 0]
output
                   HEX0,
output
         [ 6: 0]
                   HEX1,
        [ 6: 0]
output
                   HEX2,
        [ 6: 0]
[ 6: 0]
output
                   HEX3,
output
                   HEX4,
output
        [ 6: 0]
                   HEX5,
/////// SDRAM ////////
                   DRAM_CLK,
output
                   DRAM_CKE,
output
         [12: 0]
output
                   DRAM_ADDR,
output
         [ 1: 0]
                   DRAM_BA,
         [15: 0]
                   DRAM_DQ,
inout
                   DRAM_LDQM,
output
                   DRAM_UDQM,
output
output
                   DRAM_CS_N,
                   DRAM_WE_N,
output
                   DRAM_CAS_N,
output
                   DRAM_RAS_N
output
```

HEXOP; wire HEX1P; wire HEX2P; wire wire HEX3P;

);

```
HEX4P:
wire
wire HEX5P;
nios2 u0 (
.pll_ref_clk_clk
.pll_ref_reset_reset
                             (CLOCK_50),
                             (1'b0),
.pll_sdram_clk_clk
                             (DRAM_CLK)
.sdram_addr
                             (DRAM_ADDR),
                             (DRAM_BA),
.sdram_ba
.sdram_cas_n
                             (DRAM_CAS_N),
                             (DRAM_CKE)
.sdram_cke
.sdram_cs_n
                             (DRAM_CS_N),
.sdram_dq
                             (DRAM_DQ),
                             ({DRAM_UDQM, DRAM_LDQM}),
.sdram_dqm
                             (DRAM_RAS_N),
.sdram ras n
                             (DRAM_WE_N),
.sdram_we_n
.bp_export
                             (KEY),
.switchs_export
                             (SW),
                             (LEDR)
.leds_export
                ({HEX5P, HEX5, HEX4P, HEX4, HEX3P, HEX3, HEX2P, HEX2, HEX1P, HEX1, HEX0P, HEX0})
.seg7_export
);
```

```
endmodule
```

On note l'ajout des signaux supplémentaires HEXOP à HEX5P qui sont les anodes communes des 6 afficheurs 7 segments nécessaires au bloc IP 7SEG_IF.

Il ne reste plus qu'à synthétiser le système SoPC pour générer le fichier de programmation du circuit FPGA .sof.

- Choisir le menu *Processing > Start Compilation*.
- Après synthèse, quelles sont les ressources consommées et combien dans le circuit FPGA ? Après synthèse, on obtient la figure suivante :

Fit Discussion Discusion Discussion Discussion	8		Qı	artus Prime Standard Edition - /h	ome/kadionik/Altera/me357/	design/DE10_Standard_golden_top - DE10_Standard_golden_top		_ 0 ×
Image: The set of the set o	Elle Edit View Project	Assignments Processing	Tools Window He	lp.				Search Intel FPGA
Writer Nordganz	00000	つで DE10 Standard	golden top 👻 🏒					
Entry and contracts Part of contracts	Project Navigator	A Hierard	τγ ▼ Q.₽ØØ	CE10 Standard colden tony II	Compilation Report - DE10 Sta	ndard eolden top X	IP Catalog	
Option W. K2337500971102 Image and any option Image any option <td< td=""><td></td><td>Entityclostance</td><td></td><td>Table of Contents</td><td>Elow Summary</td><td></td><td></td><td>* =</td></td<>		Entityclostance		Table of Contents	Elow Summary			* =
Dit 10 Standard, golden, top A Part Standard, golden,	Cyclone V: 5C5XFC6D6F31	IC6		Table of Contents	ccFilter>>			
Task Complation IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII	、 開 DE10_Standard_golder	η, τορ - Υπ		Implementation Implementation Flow You-Charland Global Settings Implementation Implementation<	Figure Status Quartus Pimes Version Revision Name Fog-Aveil Entity Name Family Device Device Device Total groups Total Piologi Piologi Total Version Piolo	Succentul - Wed File 22 130/33 2023 21.11 Build 50 06/2022 25 43/audd Giftion DDD Subid _ picken, top Cyclore V ScSPCCD01 - picken ScSPCCD01 - picken 100 / 40 (2 %) 0 0 0 / 40 (2 %) 0 / 10 (5 %) 0 / 5 / 5 %) 0 / 5 / 5 % 0 / 5 %	Projection System Basic Fun Superior Basic Fun Dop Superior Search for P Search for P	tory ctions Protocols rs and Peripherals y Program artner IP
Two Two Two * * P Complex Design 000514 * * P Complex Design 000511 * * P Complex Design 000510 * * P Complex Design 000600 * * * P Complex Design 000000 * * * * * * * * * * * * * * * * * * *	• Tasks	Compilation	• • = ₽ ⊗ 8		Total HSSI TX PCSs Total HSSI PMA TX Serializers Total PLLs	0/9(0%) 0/9(0%) 1/15(7%)		
• • • Comple Design 000388 • • • • • • • • • • • • • • • • • • •		Task	Time		Total DLLs	0/4[0%)		
✓ → Analysis Synthesis 000121 ✓ → Filter (lives Should 000221 ✓ → Filter (lives Should 000221 ✓ → Temp Analysis 000121 ✓ → Temp Analysis 000221 ✓ → Temp Analysis 000240 ✓ > Extended Witer ● ● Program Device Open Programmed) ● ●	 Compile Design 		00:05:38					
Program Control (Depth Toppaming) Program Cont (Depth Toppaming) Program Control (Depth Toppaming) Pr	 Analysis & Sy 	mthesis	00:01:21					
A Anamber Generator generatory table, is 7,511	 Fitter (Place 8 	& Route)	00:02:21					
	 Assembler (G 	Senerate programming files	00:01:10					
Pogum Docke Open Programmed Pogum Docke Open Programed Pogum Docke Open Programed Pogum Docke Open Program	 Timing Analy 	isls	00:00:46					
Constrained of the straine of the straine of the strained of the straine of	+ 🕨 EDA Netlist V	Vriter						
Program Decke (Dpen Fregammer)	Edit Settings							
Image: Contract of the second of the seco	Program Device	(Open Programmer)						
Image: State							+ Add.	
Type ID Message > 0.32324 Mort-case recovery slick is 7.511 > 0.32324 Mort-case microsyl slick is 7.511 > 0.32324 Mort-case microsyl slick is 1.600 > 0.3232 Mort Mestadility: Found 25 synchronizer claims > 0.324 Mort Mestadility: Found 25 syncho	8 AL O A A A	Sector Contraction of the sector of the s		66 Eind 66 Find Negt				
<pre> • District -case recovery lake is 7.31 * District -case recovery lake is 6.38 * District -case recovery lake is 7.38 * District -case recovery lake recovery lake is 7.38 * District -case recovery lake is 7.38 *</pre>	Type ID Message							
Suffer Proceeding (53)	 332146 Worst-ca 332146 Worst-ca 332146 Worst-ca 332146 Worst-ca 33214 Report M 332102 Design i Quartus 293000 Quartus 	ise recovery slack is ise removal slack is use minimum pulse wid letastability: Found is not fully constrai Prime Timing Analyze Prime Full Compilati	7.511 0.388 th slack is 1.800 23 synchronizer cl ned for setup requ ned for hold requ r was successful. on was successful.	hains. uirements irements 38 warnings . 0 errors, 38 warnings . 0 errors, 406 warnings				-
E System Processing (512)	8							×.
Σ <u>στού</u> ποσεσιήζεται 1000 - 1000	System Processing (5	12)						10096 00:05:39

Synthèse du système SoPC (30)

3.10. Programmation du circuit FPGA

A partir de Quartus Prime, on peut programmer le circuit Cyclone V de la carte DE-Standard.

• Choisir le menu *Tool > Programmer*. La fenêtre suivante apparaît :

welled the star lab		-	r NVCCSNCS		1			235
Hardware Setu	IP DE-SoC [2-1.6]	Mode:	JTAG		 Progre 	ess:	(Faile	ed)
Enable real-tim	ie ISP to allow backgroun	d programming wi	hen available					
▶ [™] Start	File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank- Check	Examine
M Stop	output_files/DE10_S	5CSXFC6D6F31	01453502	01453502	V			
Auto Detect								
X Delete								
Add File								
Change File								
Save File								
Add Device								
1 ND Up	ты							
1ª Down								
	5CSXFC6D6	5F31						
	TDO							

Fenêtre Programmer (31)

• Cliquer sur le bouton *Auto Detect* puis on choisira le *Device* 5CSXFC6D6. On obtient la figure suivante :



Sélection du circuit FPGA (32)

• Sélectionner ensuite le composant 5CSXFC6D6 puis on cliquera sur le bouton *Change File...* pour choisir le fichier .sof.

• Dans le répertoire output_files/, choisir le fichier DE10_Standard_golden_top.sof. On cochera la case *Program/Configure* pour ce composant puis on cliquera sur le bouton *Start* pour programmer le circuit FPGA. On obtient la figure suivante :



Programmation du circuit FPGA (33)

Nous en avons alors fini avec la partie matérielle de construction de notre système SoPC.

Il nous reste la deuxième partie du codesign qui correspond à la programmation logicielle.

On pourra toujours reprogrammer le circuit FPGA depuis Eclipse.

On peut enfin fermer Quartus Prime ...

4. EX 2 INTEL : HELLO WORLD

Nous allons réaliser un premier test logiciel fonctionnel en implémentant sur la carte cible DE10-Standard le programme « *Hello World* ».

On utilisera par la suite *Eclipse* comme atelier de développement logiciel IDE.

On utilisera pour cela le *shell script* goeclipe qui lancera *Eclipse* avec les bons paramètres.

Il faut aussi savoir qu'*Eclipse* utilise un espace de travail (*Workspace*) qui est généralement le répertoire software/ que l'on a sous de10/design/.

- Se placer ensuite dans le répertoire de10/design/. host% cd de10/design
- Se placer dans l'environnement de développement Intel si ce n'est pas déjà fait puis lancer *Eclipse*. On ajustera le *Workspace* à de10/design/software/. On a la fenêtre suivante :

host% n2sdk
[Xilinx EDK]\$ goeclipse

e	Workspace Launcher		- • *
Select a wo	rkspace		
Eclipse store Choose a wo	s your projects in a folder called a workspace. rkspace folder to use for this session.		
Workspace:	/home/kadionik/Altera/me357/design/software	•	Browse
Use this	as the default and do not ask again Cance	I	ОК

Définition du Workspace Eclipse

L'allure d'*Eclipse* est donnée sur la figure suivante et a un fonctionnement similaire à l'IDE *Netbeans* :



IDE Eclipse

On vérifiera aussi que l'on a aussi dans le terminal de lancement d'*Eclipse* les traces suivantes :

```
Byte Stream Device: jtaguart_0

Path: /connections/DE-SoC on localhost (2-

1.6)/5CSEBA6(.|ES)|5CSEMA6|..02/(110:128 v1 #0)/jtaguart_0

Processor: nios2_0

Path: /connections/DE-SoC on localhost (2-

1.6)/5CSEBA6(.|ES)|5CSEMA6|..02/(70:34 v3 #0)/nios2
```

Cela signifie que l'on a bien accès au processeur NIOS II par le JTAG.

Si l'on n'a pas ces traces, il faudra programmer le circuit FPGA par le menu Eclipse *NIOS II > Quartus Prime Programmer*, sortir d'Eclipse et le relancer...

- Créer un nouveau projet *Eclipse hello* par le menu *File > New > NIOS II Application and BSP from Template*. On renseignera le champ *SOPC Information File name* avec le fichier de10/design/nios2.sopcinfo. Le fichier .spocinfo est un fichier verbeux de type XML qui donne toutes les informations sur le système SoPC et ses blocs IP.
- Choisir *hello* comme nom de projet pour le champ *Project name*.
- Choisir pour le *Project template* le *template Hello Word*.
- Cliquer sur le bouton *Next* puis sur le bouton *Finish*.

• A quoi correspond le projet *hello_bsp* ? On a la figure suivante :

geenaranara	
SOPC Information File name:	/home/kadionik/Altera/me357/design/nios2.sop
CPU name:	nios2 👻
oplication project	
Project name: hello	
Project template	Template description
Float2 Functionality Float2 GCC Float2 Performance Hello Freestanding	Hello World prints 'Hello from Nios II' to STDOUT. This example runs with or without the MicroC/OS-II RTOS and requires an STDOUT device in your system's hardware.
Hello MicroC/OS-II	For details, click Finish to create the project and refer to the readme.txt file in the project
Hello World	

Projet Eclipse Hello

- On a donc 2 projets : *hello* et *hello_bsp*. Dans le projet *hello*, on ouvrira le fichier hello_world.c qui est le programme principal qui sera exécuté par le processeur NIOS II du système SoPC.
- Analyser le source du fichier hello_world.c.

- Compiler le projet *hello* par le menu *Project > Build Project*. On peut le faire aussi par le menu contextuel avec le clic droit de la souris sur le projet *hello* dans la fenêtre *Project Explorer*.
- Télécharger le code exécutable (fichier .elf) par le menu contextuel du projet *hello* (clic droit de la souris sur le projet *hello*) en choisissant le menu *Run As > 3 Nios II Hardware*

ou en cliquant sur le bouton vert avec le triangle blanc $^{\textcircled{0}}$.

• Remarquer les traces d'exécution dans la fenêtre *Nios II Console* qui correspond à ce que le PC de développement reçoit sur la liaison JTAG/UART :



Traces d'exécution du projet Eclipse Hello

• Par le projet hello, quels blocs IP du système SoPC a-t-on fonctionnellement validé ?

5. EX 3 INTEL : CREATION DU BSP

Quand nous avons créé le projet *hello*, *Eclipse* nous a créé automatiquement le projet *hello_bsp*.

Le projet *hello_bsp* est un BSP support qui correspond à la couche d'abstraction matérielle de base appelée HAL (*Hardware Abstraction Layer*).

Cela correspond à un ensemble de fonctions C générées de façon automatique qui permet d'interagir avec les blocs IP du système SoPC. On retrouve dans ces routines printf() par la liaison JTAG/UART par exemple.

On peut avoir un aperçu de la HAL en regardant dans le répertoire HAL > src du projet hello_bsp.

Un fichier important du projet *hello_bsp* est le fichier C system.h qui est la traduction en langage C du fichier .sopcinfo. On le voit sur la figure suivante, c'est une suite de #define qui donne les caractéristiques essentielles pour chaque bloc IP.

Sur la figure, on voit par exemple les caractéristiques du bloc IP *bp*. On voit que l'adresse de base des registres du bloc IP *bp* correspond à l'étiquette BP_BASE.



Contenu du fichier system.h du projet Eclipse Hello

L'idée est de construire notre propre BSP au dessus de la HAL pour le pilotage des blocs IP *bp, switchs, leds* et *seg7*.

On utilisera pour cela les routines d'E/S de la HAL.

- Créer un projet nommé *bsp* à partir du *template Hello Word*.
- Importer les fichiers ressources bsp.h, bsp.c et hello_world.c du répertoire de10/ressources/ dans le projet *bsp*. On peut faire cela avec l'explorateur de fichiers sous Linux et faire un copier/coller des 3 fichiers dans le projet *bsp* par un *drag and drop*. On obtient la figure suivante :

0	Nios II - bsp/bsp.h - Eclipse		_ # X
File Edit Source Refactor Navigate Search Project Run N	ios II Window Help		
······································	Q + Ø ∅ 𝔅 + 𝔅 + 𝔅 + 𝔅 + 𝔅 + 𝔅 +		Quick Access
Image: Second	<pre>Subsph ti</pre>		Quick Access Image: Control of the second secon
		Writable Smart Insert 1:1	

Projet Eclipse bsp

Pour réaliser notre BSP, on va utiliser 2 fonctions de base fournies par la HAL pour la lecture et l'écriture des registres de nos périphériques.

On a à disposition dans le fichier io.h les fonctions : IORD(BASE, REGNUM); IOWR(BASE, REGNUM, DATA);

IORD () renvoie le contenu 32 bits du registre numéro REGNUM à partir de l'adresse de base BASE.

IOWR () écrit la donnée 32 bits DATA dans le registre numéro REGNUM à partir de l'adresse de base BASE.

Notons aussi, que l'on relit exactement ce que l'on a écrit précédemment...

Les adresses de base de nos périphériques sont définies dans le fichier system.h de la HAL et sont pour nous :

- LEDS_BASE pour les leds.
- SWITCHS_BASE pour les switchs.
- BP_BASE pour les boutons poussoir.
- SEG7_BASE pour les afficheurs 7 segments.

Le registre 0 correspond à la donnée à lire ou à écrire pour les leds, switchs ou boutons poussoir.

Le bit 0 du registre 0 correspond à la led 0, le switch 0 ou le bouton poussoir 0

Ces 3 périphériques sont basés sur le même bloc IP et fonctionnent donc de la même façon.

Pour les afficheurs 7 segments, le registre 0 correspond au masque à écrire pour l'afficheur 0 jusqu'au registre 6 qui correspond au masque à écrire pour l'afficheur 6.

Les bits du registre d'un afficheur 7 segments sont connectés comme indiqué sur la figure suivante. Le point décimal n'est pas câblé :



Câblage des afficheurs 7 segments

Ce périphérique est basé sur un bloc IP différent de celui des leds par exemple.

Dans le fichier bsp.c, on a défini la fonction : SEG7_SET(seg7_nbr, seg7_mask);

Cette fonction permet d'écrire simplement la valeur du masque seg7_mask dans le registre 0 de l'afficheur numéro seg7_nbr. Il y a SEG7_NUM afficheurs soit ici 6.

Ces explications sont alors suffisantes pour écrire notre BSP.

Les fonctions du BSP à développer sont :

- void BSP_init(void): initialisation de la carte. Extinction des leds et des afficheurs 7 segments.
- void BSP_setLED(unsigned char lite): allumage de la led numéro lite.
- void BSP_clrLED (unsigned char lite) : extinction de la led numéro lite.
- unsigned int BSP_readSW() : lecture de la valeur courante des 10 switchs.
- unsigned int BSP_readBP(unsigned char nbr): valeur courante (on ou off) du bouton poussoir numéro nbr.
- void SEG7_on(void) : allumage complet des segments des 6 afficheurs 7 segments.
- void SEG7_off (void) : extinction des 6 afficheurs 7 segments.
- void SEG7_Decimal (unsigned int value) : affichage sur les afficheurs 7 segments de la valeur décimale value.
- Modifier le fichier bsp.c pour implémenter le BSP.
- Modifier le fichier principal hello_world.c pour tester séquentiellement (pas de multitâche pour l'instant) chaque périphérique.
- Le système SoPC est-il finalement fonctionnel ?

6. EX 4 INTEL : HELLO WORLD μ C/OS II

On va s'intéresser maintenant au noyau Temps Réel μ C/OS II qui est supporté par le processeur NIOS II et accessible avec *Eclipse*.

Outre le support du Temps Réel dur, µC/OS II nous apporte aussi le multitâche.

On va d'abord créer le projet μ C/OS II « *Hello World* » qui consiste à la création de 2 tâches périodiques qui écrivent sur la liaison JTAG/UART.

- Créer un nouveau projet *Eclipse hello_ucos* en utilisant le *template Hello MicroC/OS-II*.
- Tester. On obtient la figure suivante :



Projet *hello_ucos*

• Quel périphérique (bloc IP) est essentiel pour faire fonctionner μ C/OS II ?

7. EX 5 INTEL : TESTS DES PERIPHERIQUES SOUS μ C/OS II

On désire créer un projet *Eclipse* qui teste en parallèle l'ensemble des périphériques sous μ C/OS II.

- Créer le projet Eclipse *tst_all* à partir du *template Hello MicroC/OS-II*. On importera bien sûr les fichiers bsp.h et bsp.c de notre BSP dans le projet.
- Créer :
 - Une tâche qui réalise un chenillard sur les 10 leds.
 - Une tâche qui renvoie l'état courant des 10 switchs.
 - Une tâche qui détecte l'appui sur un bouton poussoir.
 - Une tâche qui affiche sur les 6 afficheurs 7 segments une valeur décimale qui s'incrémente chaque seconde.
- Tester.

8. EX 6 INTEL : MINIPROJET : CHRONOMETRE

On désire créer un chronomètre au dixième de seconde qui affiche le temps sur les 6 afficheurs 7 segments.

L'afficheur 1 affichera les dixièmes de seconde (l'afficheur 0 le plus à droite reste à 0) alors que les afficheurs 1 à 5 affichent les secondes.

Le bouton poussoir 0 est le start/stop du chronomètre.

Le bouton poussoir 3 est le reset du chronomètre.

- Créer le projet *Eclipse miniprojet* à partir du *template Hello MicroC/OS-II*.
- Développer le miniprojet.
- Tester. Remarquer l'existence de rebonds sur les boutons poussoir si l'on ne fait rien pour les supprimer.

Question ouverte :

• Réaliser une horloge qui affiche heures, minutes, secondes sur les 6 afficheurs 7 segments.

9. TP 2 : GRAND TP. MISE EN ŒUVRE DU SOPC AVEC XILINX ZYNQ

9.1. Introduction

Il s'agit d'utiliser l'environnement de développement SoPC de Xilinx qu'il s'agisse de Vivado comme outil de synthèse VHDL et de placement routage ou de l'environnement Linux comme environnement de développement croisé pour la partie logicielle.

Ce deuxième « grand » TP se décompose en :

- L'intégration d'un périphérique (fonction compteur d'un *timer* 64 bits simplifié) dans un SoPC pour circuit FPGA Zynq avec l'outil Xilinx Vivado.
- La mise en œuvre de Linux embarqué sur le processeur ARM Cortex-A9 du circuit FPGA Zynq.
- Le développement d'un pilote du périphérique sous Linux embarqué en mode utilisateur *(user mode driver)*.
- Le développement de l'application de test du périphérique sous Linux embarqué.
- La mesure de temps de latence du noyau standard et du noyau Xenomai.

Nous allons voir dans un premier temps la mise en œuvre de Linux embarqué sur la carte cible ZedBoard.

9.2. Carte cible Digilent ZedBoard

La carte ZedBoard est une carte d'évaluation développée par la société Digilent et met en œuvre le circuit FPGA Zynq Z-7020 de Xilinx.

Le circuit FPGA Zynq Z-7020 permet de réaliser un système SoPC et est constitué de logique programmable couplée à un processeur *hardcore* double coeur ARM Cortex-A9.

Dans un circuit FPGA Zynq, on appelle PS (*Processing System*) la partie processeur et ses périphériques associés.

La partie PS inclut :

- Les deux cœurs ARM Cortex-A9.
- Le bus AMBA-AXI (Advanced Microcontroller Bus Architecture-Advanced eXtensible Interface).
- les E/S GPIO et les liens série I2C, UART, CAN et SPI.
- Le contrôleur des mémoires QuadSPI, NAND et NOR.
- Le contrôleur de mémoire vive DDR3.

La partie PL (Programmable Logic) correspond à la partie logique programmable.

Le circuit FPGA utilisé par la carte ZedBoard est le circuit Xilinx XC7Z020-CLG484 (*Zynq-7020 AP SOC*). Le circuit Zynq Z-7020 est un circuit milieu de gamme dans la série et reprend la même logique programmable que les circuits Xilinx Artix. Les circuits Zynq haut de gamme Z-7030 et Z-7045 reprennent quant à eux la logique programmable des circuits Xilinx Kintex.

La carte Zedboard possède ainsi les éléments suivants :

- Processeur ARM Cortex-A9 à 533 MHz avec 32 Ko de cache L1 et 512 Ko de cache externe L2.
- 512 Mo de RAM DDR3.
- 32 Mo de mémoire Flash Quad SPI.
- Slot mémoire SD.
- Sorties HDMI (audio et vidéo).
- Sortie VGA.
- Ethernet (10/100/1000 Mb/s).
- Port de débug ARM DAP (Debug Access Port).
- Port USB 2.0 OTG *device* et *host*.
- Port USB-JTAG.
- Port USB-UART.
- Connecteur FMC (FPGA Mezzanine Connector).
- 5 ports d'extension Pmod.
- 9 leds utilisateur.
- 8 switchs.
- 7 boutons poussoir.

L'image suivante présente la carte cible ZedBoard :



Carte cible ZedBoard

Le choix du périphérique de *boot* de la carte ZedBoard se fait par la configuration de 3 *jumpers* suivant le tableau suivant :

	MIO[5]	MIO[4]	MIO[3]
JTAG	0	0	0
NOR	0	0	1
NAND	0	1	0
QSPI	1	0	0
SD	1	1	0

Configuration des *jumpers* MIO[3] à MIO[5] pour le choix du mode de boot

Nous utiliserons pour la suite le *boot* depuis la carte SD, ce qui permettra d'utiliser le *bootloader u-boot*.

10. EX 1 XILINX : GENERATION DU RAM DISK ET INTEGRATION D'UNE APPLICATION

Nous allons voir comment compiler le noyau Linux embarqué avec son RAM disk exécuté par le processeur Cortex-A9 de la carte cible ZedBoard.

Par la suite, on adoptera les conventions suivantes : Commande Linux PC hôte pour le développement croisé : host% commande Linux Commande Linux PC hôte pour le développement Xilinx : [Xilinx EDK]\$ Commande Linux embarqué sur la carte cible ZedBoard :

ZedBoard: # commande Linux embarqué

Commande *u-boot* sur la carte cible ZedBoard :

U-Boot> commande u-boot

Nous allons voir comment rajouter une application dans le RAM disk utilisé par le noyau Linux standard exécuté par le processeur Cortex-A9 de la carte cible ZedBoard. Nous allons dans un premier générer notre propre RAM disk.

- Démarrer le PC sous Linux. Se connecter sous le nom se01, mot de passe : se01 [©] pour le groupe 1 et sous le nom **se02**, mot de passe : **se02** ⁽²⁾ pour le groupe 2.
- Se placer dans son répertoire de travail : host% cd

Se connecter à la carte ZedBoard (cible) en utilisant l'outil minicom : host% minicom -b 115200 -D /dev/ttyACM0 Pour sortir de minicom, il suffit de taper la combinaison de touches : CTRL A, Z pour accéder au menu et taper q pour quitter. On arrêtera le compte à rebours de 3 secondes d'*u-boot* en appuyant sur la touche espace du clavier...

- Recopier le fichier tp-ZedBoard.tgz sous /home/kadionik/: host% cp /home/kadionik/tp-ZedBoard.tgz .
- Décompresser et installer le fichier tp-ZedBoard.tgz : host% tar -xvzf tp-ZedBoard.tgz
- Se placer ensuite dans le répertoire ZedBoard/. L'ensemble du travail sera réalisé à partir de ce répertoire ! Les chemins seront donnés par la suite en relatif par rapport à ce répertoire... host[®] cd ZedBoard

- Créer le système de fichiers *root* squelette root_fs pour la carte cible ZedBoard : host% cd ramdisk host% ./goskel
- Compiler busybox: host% cd ramdisk host% cd busybox host% ./go
- Générer le système de fichiers *root* final root_fs pour la carte cible ZedBoard. Il est demandé à un moment donné de rentrer son mot de passe ([sudo] password for se01/se02 :): host% cd ramdisk host% ./gorootfs
- Se placer dans le répertoire tst/hello/ et modifier le fichier hello.c afin de créer le fameux « Hello World! »: host% cd tst/hello host% gedit hello.c
- Compiler l'application hello pour la carte cible ZedBoard : host% ./go
- Installer l'application hello dans le système de fichiers *root* qui servira de base au *RAM disk*:
 host% ./goinstall
- Se placer dans le répertoire rootfs/ pour regénérer le RAM disk. Le système de fichiers root est sous root_fs/ et notre application hello a été précédemment copiée sous root_fs/bin/:
 host% cd ramdisk
 host% ls root_fs/bin
- Générer le RAM disk. Il est demandé à un moment donné de rentrer son mot de passe ([sudo] password for se01/se02 :). Que fait le shell script goramdisk ? Comment s'appelle le fichier RAM disk ? host% cd ramdisk host% sudo ./goramdisk
- Installer le nouveau RAM disk dans le répertoire de téléchargement d'u-boot /tftpboot: host% ./goinstall

Astuce !

Pour éviter de regénérer à chaque fois le *RAM Disk* et recharger le noyau Linux, on fera une compilation croisée puis on recopiera sous /tftpboot l'exécutable ainsi produit :

```
host% ./go
host% cp mon_appli_exe /tftpboot
```

ou plus simplement avec les shell scripts fournis :

```
host% ./go
host% ./goinstall
```

On pourra alors télécharger par TFTP l'exécutable en utilisant la commande tftp et lancer l'application :

```
target# tftp -g -r mon_appli_exe @IP_host
target# chmod u+x mon_appli_exe
target# ./mon_appli_exe
```

11. EX 2 XILINX : MISE EN ŒUVRE DE LINUX EMBARQUE SUR LA CARTE CIBLE

Nous allons voir comment compiler le noyau Linux embarqué exécuté par le processeur Zynq de la carte cible ZedBoard.

- Compiler le noyau Linux standard pour la carte cible ZedBoard : host% cd linux host% ./go
- Installer le fichier du noyau Linux dans le répertoire de téléchargement d'u-boot /tftpboot: host% ./goinstall
- Recharger (optionnel) le design de référence system.bit dans le circuit FPGA de la carte cible ZedBoard. Que fait le shell script load-design ?
 host% cd ZedBoard
 host% mbsdk
 [Xilinx EDK]\$./load-design
- Depuis *u-boot* de la carte cible ZedBoard, lancer la commande suivante. Quels sont les 3 fichiers téléchargés depuis le PC hôte en RAM de la carte ZedBoard et quel est leur rôle ? U-Boot> run ramboot
- Observer les traces de boot du noyau Linux standard dans la fenêtre minicom : Starting kernel ...

```
Booting Linux on physical CPU 0x0
Linux version 6.1.0-xilinx-48445-gf9c8e14ae03c (kadionik@ipcchipik) (arm-
buildroot-linux-gnueabihf-gcc.br_real (Buildroot 2021.11-4428-g6b
6741b) 11.3.0, GNU ld (GNU Binutils) 2.38) #16 SMP PREEMPT Tue Mar 14
16:29:43 CET 2023
CPU: ARMv7 Processor [413fc090] revision 0 (ARMv7), cr=18c5387d
CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
OF: fdt: Machine model: xlnx,zynq-7000
earlycon: cdns0 at MMIO 0xe0001000 (options '115200n8')
printk: bootconsole [cdns0] enabled
Memory policy: Data cache writealloc
cma: Reserved 16 MiB at 0x1f000000
percpu: Embedded 15 pages/cpu s31872 r8192 d21376 u61440
Built 1 zonelists, mobility grouping on. Total pages: 130048
                                   console=ttyPS0,115200
Kernel
           command
                        line:
                                                            root=/dev/ram
ramdisk_size=131072 rw
earlyprintk earlycon
clocksource: jiffies: mask: 0xfffffff max_cycles: 0xffffffff, max_idle_ns:
1911
2604462750000 ns
futex hash table entries: 512 (order: 3, 32768 bytes, linear)
pinctrl core: initialized pinctrl subsystem
NET: Registered protocol family 16
DMA: preallocated 256 KiB pool for atomic coherent allocations
cpuidle: using governor menu
hw-breakpoint: found 5 (+1 reserved) breakpoint and 1 watchpoint registers.
hw-breakpoint: maximum watchpoint size is 4 bytes.
```

P. Kadionik

zynq-ocm f800c000.ocmc: ZYNQ OCM pool: 256 KiB @ 0x(ptrval) e0001000.serial: ttyPS0 at MMIO 0xe0001000 (irq = 24, base_baud = 3125000) is a xuartps printk: console [ttyPS0] enabled printk: console [ttyPS0] enabled printk: bootconsole [cdns0] disabled printk: bootconsole [cdns0] disabled vgaarb: loaded Xilinx Zvng CpuIdle Driver started sdhci: Secure Digital Host Controller Interface driver sdhci: Copyright(c) Pierre Ossman sdhci-pltfm: SDHCI platform and OF driver helper mmcO: SDHCI controller on e0100000.sdhci [e0100000.sdhci] using ADMA ledtrig-cpu: registered to indicate activity on CPUs clocksource: ttc_clocksource: mask: 0xffff max_cycles: 0xffff, max_idle_ns: 5375 38477 ns . . . RAMDISK: gzip image found at block 0 mmc0: new high speed SDHC card at address 0007 mmcblk0: mmc0:0007 SD04G 3.71 GiB mmcblk0: p1 EXT4-fs (ram0): mounted filesystem with ordered data mode. Opts: (null) VFS: Mounted root (ext4 filesystem) on device 1:0. devtmpfs: mounted Freeing unused kernel memory: 1024K Run /sbin/init as init process Hostname : ZedBoard Kernel release : Linux 6.1.0-xilinx-48445-gf9c8e14ae03c Kernel version : #16 SMP PREEMPT Tue Mar 14 16:29:43 CET 2023 : [SUCCESS] Mounting /proc Mounting /sys : [SUCCESS] Mounting /dev : [SUCCESS] Mounting /dev/pts : [SUCCESS] Enabling hot-plug : [SUCCESS] : [SUCCESS] Populating /dev Mounting other filesystems : [SUCCESS] : [SUCCESS] Starting telnetd Network configuration : [SUCCESS] System initialization complete. Please press Enter to activate this console. ZedBoard:/# uname -r 6.1.0-xilinx-48445-qf9c8e14ae03c ZedBoard:/#

12. EX 3 XILINX : INTEGRATION D'UN PERIPHERIQUE MATERIEL LIBRE

12.1. Introduction

L'objectif de ce TP est d'intégrer un périphérique matériel Libre, en l'occurrence un *timer* 64 bits utilisé dans sa fonctionnalité de comptage qui pourra communiquer avec le microprocesseur *hardcore* Cortex-A9 du circuit FPGA Zynq de la carte cible via le bus AXI (*Advanced eXtensible Interface*) du processeur.



AXI Write : $dataoutX \le slv_regX;$ X \in [0,3] AXI Read : reg_data_out $\le datainX;$

Timer 64 bits et son interfaçage sur le bus AXI

Ce *timer* 64 bits conçu par nos soins s'interface directement sur le bus AXI du processeur Cortex-A9 du circuit FPGA Zynq. Il a été grandement simplifié et ne propose que la fonction de comptage du nombre de périodes d'horloge du circuit FPGA depuis son reset. La fréquence est de 100 MHz soit 10 ns de période. Il est donc possible de faire des mesures de temps avec une précision de 10 ns avec ce *timer*... Une seconde de temps écoulée correspond ainsi à l'augmentation de la valeur courante du compteur de 100 millions...

Les fichiers sources VHDL du *timer* sont directement fournis pour gagner du temps mis à part un ajout simple à faire dans le fichier source. Ce *timer* possède des registres de contrôle et de données mappés dans l'espace d'adressage du processeur et directement accessibles par le bus AXI :

- Un registre de capture sur 64 bits à concaténer par 2 **lectures** de 32 bits (le bus AXI est un bus 32 bits) aux adresses base et base+4.
- Un registre de contrôle accessible en écriture seulement seulement à l'adresse de base :
 - Ecriture de 1 dans le bit 0 du registre : on enregistre la valeur courante du compteur et on la transfère vers le registre de capture du *timer*.
 - Ecriture de 1 dans le bit 1 du registre : on réinitialise à 0 le compteur du *timer*.

Nous pouvons résumer cela par la figure suivante :



Accès registres en lecture

Registres de contrôle et de données du timer 64 bits

Le *timer* fonctionne met en œuvre 3 fichiers sources VHDL :

- Fichier mon_ip_v1.0_S00_AXI.vhd : ce fichier VHDL correspond à l'implantation de l'interface esclave AXI. Il est généré par Vivado.
- Fichier timer64.vhd: ce fichier VHDL est le compteur 64 bits en lui-même.
- Fichier mon_ip_v1.0.vhd : ce fichier VHDL correspond au bloc IP *timer64* incluant les 2 fichiers précédent. Il est généré par Vivado.

Le fichier source timer64.vhd est le suivant :

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity timer64 is
  port(
    clk, reset: in std_logic;
    start: in std_logic;
    capture_high: out std_logic_vector(31 downto 0);
    capture_low: out std_logic_vector(31 downto 0)
    );
end timer64;
architecture arch of timer64 is
  signal timer64: unsigned((2*32)-1 downto 0);
  signal capture_register: unsigned((2*32)-1 downto 0);
begin
  core: process(clk, reset)
  begin
    if (reset = '1') then
      timer64 <= (others=>'0'); -- Asynchrone
    elsif (clk'event and clk='1') then
       -- A compléter . . .
    end if;
  end process core;
  capture: process(clk, start)
  begin
    if (clk'event and clk='1') then
      if (start = '1') then
        capture_register <= timer64;</pre>
      end if;
    end if;
  end process capture;
-- output
-- Compteur 32 bits poids fort @
  capture_high <= std_logic_vector(capture_register((2*32)-1 downto 32));</pre>
-- Compteur 32 bits poids faible @+4
  capture_low <= std_logic_vector(capture_register(32-1 downto 0));</pre>
end arch;
```

Fichier source timer 64. vhd du timer 64 bits

Le fichier est incomplet et est à compléter dans le processus VHDL *core*. L'entité/architecture s'appelle timer64 (arch).

On peut noter que :

- Le signal reset réinitialise le compteur à 0.
- Le signal start lance un *snaphot* de la valeur courante du compteur 64 bits.
- Les signaux capture_high et capture_low capturent les 32 bits de poids fort et les 32 bits de poids faible du *snapshot*.

Si l'on regarde le fichier mon_ip_v1.0.vhd vers la ligne 151, on a le code VHDL suivant :

Extrait du fichier source mon_ip_v1.0.vhd du timer 64 bits

Le signal reset est bien relié au bit 1 du registre @base (dataout0) alors que le signal start (capture) est bien relié au bit 0 du registre @base (dataout0) du périphérique *timer* 64 bits.

De même, on lit bien le *snapshot* via le registre @base (datain0) pour les 32 bits de poids fort et le registre @base+4 (datain1) pour les 32 bits de poids fort du périphérique *timer* 64 bits.

12.2. Intégration d'un périphérique. *Timer* 64 bits

- Se placer dans le répertoire ZedBoard/ : host% cd ZedBoard
- Se placer ensuite dans le répertoire design/design_ZedBoard/ et l'on se placera dans l'environnement Xilinx avec le *shell script* mbsdk : host% cd design/design_ZedBoard/ host% mbsdk
- Lancer l'outil Xilinx Vivado de création de SoPC : [Xilinx EDK]\$ vivado
- Ouvrir le projet Vivado project_1.xpr se trouvant dans le répertoire courant par :
 - Open Project.
 - Sélection de *project_1 > project_1.xpr*.

File Flow Tools Window Help Q- Quick Access	Vivado 2018.2	_ 0
VIADO HLx Editions		€ XILINX.
Quick Start create Project > open Example Project >	Recent Projects project_1 rhome hadwork/training hmm_thufst_wood project_1 project_1 rhome hadwork/training hmm_thufst_wood/shrifted2/project_1 project_1 rhome hadwork/training/mmm_thuffreeys4ddr/fibo2/project_1 project_1 rhome hadwork/training/mmm_thuffreeys4ddr/fibo2/project_1	
Tasks Manage IP > Open Hardware Manager > XillinxTcl Store >		
Learning Center Documentation and Tutorials > Quick Take Videos > Release Notes Guide >		
Tcl Console		

Projet SoPC project_1.xpr

- Ouvrir le *Block Design* :
 - Flow Navigator > IP Integrator > Open Block Design.

A	project_1 - [/home,	kadionik/tp/ZedBoard/design/design_ZedBoard/project 1/project 1.xpr] - Vivado 2018.2 _ σ ×
<u>Eile Edit Flow Tools Reports</u>	<u>Window</u> Layout <u>View</u> <u>Help</u> <u>Q* Quick Access</u>	Synthesis and Implementation Out-of-date details 🔸
■,Ⅲ ★ ★ 目 助 × ●	• ≌ ▶, ₩ ✿ Σ ½ ∅ ¥	III Default Layout V
Flow Navigator 🗧 🔍 🗕	BLOCK DESIGN - design_1	? ×
 PROJECT MANAGER Settings 	Sources Design × Signals Board ? _ []	Diagram x Address Editor x ? □ ⊠ 0 0 55 51 0 7 0
Add Sources Language Templates & IP Catalog	design_1 Sum External interfaces Sum interface Connections Sum Nets	
✓ IP INTEGRATOR	> processing_system7_0 (ZYNQ7 Processing System:5.5)	
Create Block Design		
Open Block Design Generate Block Design		
 SIMULATION Run Simulation 		
✓ RTL ANALYSIS		TTC: WARLOUT
> Open Elaborated Design	Properties ? _ 🗆 🖾 ×	
	+ + \$	FCLK_RESETO_N
SYNTHESIS		ZYNOZ Processing System
 Open Synthesized Design 		Line / Hocessing System
	Select an object to see properties	
 Open Implemented Design 		
 PROGRAM AND DEBUG 		
👫 Generate Bitstream		
> Open Hardware Manager	Q X + Console × Messages Log Reports Design	1 Runs ? _ D C
	Saming sources Finishes scaning sources 1969: 10-10-10-2014 Refreshing IP repositories 1969: 10-10-10-2014 Houser Vr repositories 1969: 10-10-10-2014 Houser Vr repositories 1969: 10-10-10-10-2014 Adding cells -10-2014 Houser Vr repository 4 dding cells - xilax.com hyprecessing yerker?.5 0 Soccessionly read dayma design 1- free BD file < update_company_conter-likest environ_1	r:ff.ad //opt/X1]m/YB:seds/2018.2/Metrx/p'. //opt/X1]m/YB:seds/2018.2/Metrx/p'. //opt/X1]m/YB:seds/2018.2/Metrx/project_l.srcs/sources_l/Ad/Mesign_l/Mesign_l.bd> //orcessing.pt:seds/arr/design/design_2deBaard/project_l.srcs/sources_l/Ad/Mesign_l/Mesign_l.bd>

Ouverture du Block Design

- Créer un nouveau périphérique matériel. On suivra pas à pas les différentes recopies d'écran suivantes. Sauf indications contraires, on gardera les valeurs par défaut :
 - Tools > Create and Package New IP.

A	Create and Package New IP _			
	Create and Package New IP			
HLx Editions	This wizard can be used to accomplish following tasks:			
	Package a new IP for the Vivado IP Catalog This wizard will guide you through the process of creating a new Vivado IP using source files and information from your current project, block design or specified directory.			
	Create a new AXI4 Peripheral This wizard will guide you through the process of creating a new AXI4 peripheral which includes HDL, driver, software test application, IP Integrator VIP simulation and debug demonstration design.			
£ XILINX.	Click Next to continue			
?	< <u>B</u> ack <u>Next</u> <u>Finish</u> Cancel			

Création d'un périphérique (1)

4	Create and Package New IP	- • ×
Create P Please sel	eripheral, Package IP or Package a Block Design ect one of the following tasks.	4
Packa	iging Options	
۲	Package your current project Use the project as the source for creating a new IP Definition.	
0	Package a block design from the current project Choose a block design as the source for creating a new IP Definition.	
	Select a block design: design_1 🗸	
0	Package a specified directory Choose a directory as the source for creating a new IP Definition.	
Creat	e AXI4 Peripheral	
0	Create a new AXI4 peripheral Create an AXI4 IP, driver, software test application, IP Integrator AXI4 VIP simulation and debug demonstration design.	
?	< <u>B</u> ack <u>N</u> ext > <u>F</u> inish Can	el

Création d'un périphérique (2)

- Sélectionner :
 - Create a new AXI4 Peripheral.

	Create and Package New IP	-
ripheral Deta	ils	
ecity name, versi	ion and description for the new peripheral	
Name:	mon_ip	8
Version:	1.0	8
Display name:	mon_ip_v1.0	8
Description:	My new AXI IP	⊗
IP location:	/home/kadionik/tp/ZedBoard/design/design_ZedBoard/ip_repo	
Overwrite e	isting	

Création d'un périphérique (3)

On choisira les valeurs suivantes :

- *Name* : **mon_ip**.
- Le reste inchangé.
| dd Interfaces
Id AXI4 interfaces supported by yo | our peripheral | | 4 |
|---|----------------|--|--------|
| Enable Interrupt Support | + - | Name S00_AXI Interface Type Lite | ©
~ |
| | | Interface Mode Slave
Data Width (Bits) 32
Memory Size (Bytes) 64 | *
* |
| S00_AXI
mon_ip_v1.0 | > | Number of Registers 4 | [4512] |
|) | | < <u>B</u> ack <u>N</u> ext > <u>E</u> inish | Cancel |

Création d'un périphérique (4)

<u>}</u>	Create and Package New IP _ = *
	Create Peripheral
HLx Editions	Peripheral Generation Summary
	1. IP (user.org:user:mon_ip:1.0) with 1 interface(s)
	2. Driver(v1_00_a) and testapp more info
	3. AXI4 VIP Simulation demonstration design more info
	4. AXI4 Debug Hardware Simulation demonstration design more info
	Peripheral created will be available in the catalog :
	/home/kadionik/tp/ZedBoard/design/design_ZedBoard/ip_repo
	Next Steps:
	Add IP to the repository
	O Edit IP
	O Verify Peripheral IP using AXI4 VIP
	O Verify peripheral IP using JTAG interface
E XILINX.	Click Finish to continue
•	< <u>B</u> ack <u>N</u> ext > <u>Einish</u> Cancel

Création d'un périphérique (5)

On choisira les valeurs suivantes :

- Next Steps : Add IP to the repository.
- Cliquer sur le bouton *Finish*.
- Le périphérique est créé mais ce n'est qu'une coquille vide !
- Dans la zone *Diagram*, cliquer sur le bouton « *Add IP* » et ajouter l'IP mon_ip :

4	project_1 - [/home;	kadionik/tp/ZedBoard/design/design_ZedBoard/project_1/project_1.xpr] - Vivado 2018.2	- ° ×
<u>E</u> lle <u>E</u> dit Flow <u>T</u> ools Rep <u>o</u> rts	<u>Window Layout View Help</u> <u>Q+ Quick Access</u>	Synthesis and Implementation Out-of-	-date details 🤞
■, H + + B B × #		I Default	Layout 🗸
Flow Navigator 🗧 🗘 ? 🗕	BLOCK DESIGN - design_1		? ×
 PROJECT MANAGER Settings Add Sources Language Templates 	Sources Design × Signals Board ? _ □ Ľ Q ቿ ⅓ ↓	Diagram × Address Editor × Q Q X ∑ ○ Q ≍ ○ + ↓ ▶ ⊠ ★ C ⊈ ⊕	2 C 2 \$
P IP Catalog	> 🗁 Interface Connections > 🗁 Nets		
V IP INTEGRATOR	Processing_system7_0 (ZYNQ7 Processing System:5.5)		
Create Block Design Open Block Design Run Simulation KITL ANALYSIS Open Eliaborated Design SYNTHESIS Open Synthesis Open	Properties ? _ □ □ × + ⇒ Select an object to see properties	Processing_system7_0 DOR Processing_system7_0	
Generate Bitstream			
> Open Hardware Manager	Tcl Console × Messages Log Reports Desig Q X ♦ II E III II	Runs	? _ 🗆 🖸
	Scaning sources Finished scaning sources Finished scaning sources IPGN (IPFIex D=1704) No user IP repositories IPGN (IPFIex D=1704) No user IP repository 0 points, drive (Theor Astimut Atp/2608mc/desay) 0 sourcest(U) read Lagars action_12 for Data Sourcest(U) read Lagars action_12 for Data Sourcest(U) read Lagars action_12 for Data C	ified //opt/Xilinx/Ysvado/2018.2/data/jp'. xign_Zaddbard/projet_l_hroiget_l_arcs/sources_l/Ad/design_l/design_l.bd) htmas/kadlonlk/fp/ZedBoard/design_ZedBoard/project_l/project_l.srcs/sources_l/Ad/design_l/design_l.bd>	, , ,



Ajout de l'IP mon_ip

• Cliquer sur le texte « *Run Connection Automation »* en haut à droite dans la zone *Diagram* pour interfacer le bloc IP sur le bus AXI.

2 素 ♠	Description	
✓ All Automation (1 out of 1 selected) ✓ ♥ ♥ mon_ip_0 ✓ ♥ 500 AXI	Connect Slave interface (/mon_ip_0/S00_AXI space.) to a selected Master address
	Options	
	Master	/processing_system7_0/M_AXI_
	Bridge IP	New AXI Interconnect 🛛 🗸
	Clock source for driving Interconnect IP	Auto
	Clock source for Master interface	/processing_system7_0/FCLK_0
	Clock source for Slave interface	Auto

Ajout des connections du périphérique mon_ip au bus AXI

• Valider le *design* en appuyant sur la touche F6 et sauvegarder le *Block Design* (CTRL + S).

• On obtient alors le *design* project_1.xpr suivant :

A	project_1 - [/home	rkadionik/tp/ZedBoard/design/design_ZedBoard/project_1/project_1.xpr] - Vivado 2018.2 _ ه م
<u>Eile Edit Flow Iools Reports</u>	Window Layout View Help Q+ Ouick Access	Synthesis and implementation Out-of-date details 🔸
■, H ★ + B B × #	≤ ▶, ₩ ✿ Σ % ∅ ×	III Default Layout 🗸 🗸
Flow Navigator 🗧 🔍 🗧 🚽	BLOCK DESIGN - design_1	? ×
	Sources Design × Signals Board ? □ C Q X I Q Adayn,1 Q Adayn,2 A dasyn,1 > External interfaces > > Non- > > > External interfaces ><	Diagram × Address Editor × Q Q X Q Q Image: S Q Q X Q Q Q X Q Q Q X Q Q Q Q X Q
 Generate Bitstream Open Hardware Manager 	Tcl Console x Messages Log Reports Designation Q ★ ♦ ┃ ● ■ ■	n Runs ? _ D 🛙
	⇒ goly bå urbassion -roll arliar cen åd, mylla ard. DBN: [Dect/1260] bic Castalib Barid Tateriar i USA: DSN: DSN: DSN: DSN: DSN: DSN: DSN: DSN	config (Cl satur (processing_systes7_0FLL_040 (100 Hcl)) Clk_sleve (Auto) Clk_star (Auto) Hester (processing_systes7_0FLAT_GPD) Slave (pro_1g_0500_ACI) intc_ip or found, Band Tab or created in consiste cont o <pre>cyprocessing_systes7_0FDatas at </pre></pre></pre></pre></pre></pre></pre></pre>

Projet Vivado avec le bloc IP mon_ip

• Dans la fenêtre *Diagram*, cliquer sur le bloc IP mon_ip_0 et par appui sur la touche de droite de la souris (menu contextuel), choisir le menu « *Edit in IP Packager* » :

<u>A</u>	Edit in IP Packager	_ 0 ×
Choose a project nar	ne and location for editing.	4
<u>P</u> roject name:	mon_ip	
Project <u>l</u> ocation:	/home/kadionik/tp/ZedBoard/design/design_ZedBoard/	···
	be created at/design/design_2edbbard/mon_ip	
	ОК	Cancel

Création du projet Vivado mon_ip

On choisira les valeurs suivantes :

- *Project name* : mon_ip.
- *Project Location* : **ZedBoard/design/design_ZedBoard**. Attention à bien respecter ce paramètre !!!
- Cliquer sur le bouton OK. OK to overwrite ? Yes !
- Fermer aussitôt le projet Vivado ainsi créé !!!!

- Ouvrir le projet Vivado mon_ip.xpr se trouvant dans le répertoire courant par :
 - File > Open Project.
 - Sélection de mon_ip > mon_ip.xpr.

A	Open Project			- • ×
Look in: 🕒	non_jp	~	🕯 🏠 🖵 🗏 👗 🖾 🗙 C	III
a mon_ip.ca mon_ip.ip. mon_ip.ip. mon_ip.spr ≱ mon_ip.xpr	che user_files 1	Recent Directories /home/kadionik/tp/ZedB File: mon_lp.xpr Directory: /home/kadionik/tp/ZedBoa Created: Today at 13:18 Modified: Today at 13:18 Size: 7.3 kB Size:	oard/design/design_ZedBoard rd/design/design_ZedBoard/mo РМ Э РМ РМ	∽ in_ip
		<		>
File <u>n</u> ame:	mon_ip.xpr			
Files of <u>t</u> ype:	Vivado, PlanAhead, and ISE Project Files (.xpr, ppr, xise)			~
			ОК Са	ancel

Sélection du projet Vivado mon_ip.xpr

- Laisser le projet project_1.xpr ouvert.
- Remplacer les fichiers VHDL précédemment générés par l'ajout du périphérique avec ceux correspondant au *timer*. Le fichier timer64.vdh sera importé au projet puis modifié directement depuis Vivado :

```
host% cd design/design_ZedBoard
host% cp ../ip/mon_ip_v1_0_S00_AXI.vhd ip_repo/mon_ip_1.0/hdl/
host% cp ../ip/mon_ip_v1_0.vhd ip_repo/mon_ip_1.0/hdl/
```

- On remarquera qu'il manque le fichier VHDL qui définit l'entité timer64 (arch).
- Ajouter le fichier VHDL timer64.vhd:
 - Flow Navigator > Project Manager > Add Sources.
 - Add or create design sources.
 - Add Files.
 - Dans le répertoire design/design_ZedBoard/mon_ip/, choisir le fichier VDHL timer64.vhd.
 - Cliquer sur le bouton *Finish*.

				Add Sources			
dd or Cre	e ate Desi g	jn Sources k Design, and IP	files, or directo	ories containing thos	e file types to add to	o vour project. Cre	eate a 🔥
ew source fil	le on disk a	nd add it to you	r project.		51		
+	+ +						
	Add Files	arias					
	<u>C</u> reate File						
		Use 4	Add Files, Add [Directories or Create	File buttons below		
					Consta File	1	
		<u>A</u>	dd Flies	Add Directories	<u>Create File</u>		
Scan an	id add RTL <u>i</u> r	nclude files into	project				
✓ Add sou	urces into p irces from si	ubdirectories					
?)				< B	ack Next >	Finish	Cancel
?				< <u>B</u>	ack Next >	<u> </u>	Cancel
?				< <u>B</u>	ack Next >	Einish	Cancel
?				< <u>B</u> Add Sources	ack Next >	Einish	Cancel
? .dd or Cre pecify HDL, r ew source fil	e ate Desi ç netlist, Blocl le on disk a	jn Sources < Design, and IP nd add it to you	files, or directo r project.	Add Sources ories containing those	ack <u>N</u> ext >	Einish	Cancel
dd or Cre pecify HDL, r ew source fil	e ate Desi q netlist, Blocl le on disk a	gn Sources < Design, and IP nd add it to you	files, or directo r project.	< B Add Sources	ack Next >	<u>Einish</u>	Cancel
add or Cre pecify HDL, r ew source fil +, =	eate Desig netlist, Blocl le on disk a	jn Sources < Design, and IP nd add it to your	files, or directo r project.	Add Sources	ack Next >	your project. Cre	Cancel
add or Cre pecify HDL, r ew source fil +_ =	eate Designetlist, Block le on disk a Index 1	gn Sources < Design, and IP nd add it to your Name timer64.vhd	files, or directo r project. Library xil_defaultlib	Add Sources ories containing those Location /home/kadionik/tp/	ack Next > e file types to add to ZedBoard/design/de	your project. Cre	Cancel
?	eate Designetlist, Block le on disk a Index 1	gn Sources < Design, and IP nd add it to your Name timer64.vhd Anclude files into project	files, or director r project. Library xil_defaultlib	Add Sources ories containing those Location /home/kadionik/tp/	ack Next > te file types to add to ZedBoard/design/de <u>Create File</u>) <u>E</u> inish	Cancel eate a
?	eate Designetlist, Bloch le on disk a Index 1	gn Sources < Design, and IP nd add it to your Name timer64.vhd timer64.vhd Anclude files into project ubdirectories	files, or directo r project. Library xil_defaultlib	Add Sources ories containing those Location /home/kadionik/tp/	ack Next > te file types to add to ZedBoard/design/de <u>Create File</u>) Einish	Cancel

Ajout du fichier timer64.vhd au projet Vivado mon_ip.xpr

- Editer le fichier timer64.vhd et compléter le code source à la ligne 25 pour ajouter la fonction d'incrémentation du compteur 64 bits.
- Lancer la synthèse du projet mon_ip.xpr pour vérification fonctionnelle par la commande :

Eile Edit Flow Tools Reports	Window Layout View Help Q+ Quick Access						Synthesis Complete 🚽
■, + + E B X ▶,	ti φ Σ % // ¥						💷 Default Layout 🗸 🗸
Flow Navigator 🗧 🗧 🤋	PROJECT MANAGER - mon_ip						? ×
 PROJECT MANAGER 	Sources 2	15 X Project Summary x timer64.vb	d v Package IP - mon	in v			D K
Settings		A	- Virgenaler vier				
Add Sources	C Design Sources (2)	Packaging Steps	Identification				
Language Templates	v startes (z) v mon_ip_v1_0(arch_imp) (mon_ip_v1_0.vhd)	(2) dentification	Vendon	urar ara			0
P IP Catalog	mon_ip_v1_0_S00_AXI_inst : mon_ip_v1_0_S	SOD_AX	Ubreak.	user.org			
Package IP	core : timer64(arch) (timer64.vhd)	 Compatibility 	Library:	user			0
N IN INTEGRATOR	> 🖙 Constraints	✓ File Groups	Name:	mon_ip			0
Create Block Design	> 🗁 Simulation Sources (1)	Customization Parameters	Version:	1.0			8
Open Black Design		Ports and Interfaces	Display name:	mon_ip_v1.0			0
Generate Block Design		- 1010 010 1101000	Description:	My new AXI IP			0
anne ann ann an an		 Addressing and Memory 	Vendor display name	0:			
V SIMULATION	Hierarchy Libraries Compile Order	Customization GUI	Company url:				
Run Simulation	Review an	Review and Package	Root directory:	/home/kadionik/tp/Zed8	Board/design/design_Ze	dBoard/ip_rep	o/mon_ip_1.0
	Source File Properties ? _ D	L ×	Xml file name:	/home/kadionik/tp/ZedB	Board/design/design_Ze	edBoard/ip_rep	o/mon_ip_1.0/component.xml
KIL ANALYSIS Open Elaborated Design	🔵 timer64.vhd 🗰 👄	0	Categories				
> open clausiated besign	2 Enabled			1			
 SYNTHESIS 	Lacotion formaticalizative (7adBoard Masian	uter	AVI Rerinheral	e ()			
Run Synthesis	none/cadonk(cp/2edboard/design	indez	Avi_renprierai				
> Open Synthesized Design	type: VHDL						
	Library: xil_defaultlib						
	Size: 1.1 KB						
Run implementation	Modified: Today at 13:24:12 PM	, [~]					
 Open implemenced Design 	General Properties						
V PROGRAM AND DEBUG							
👪 Generate Bitstream	Tci console Messages Log Reports De	sign kuns ×					7 - 0 8
> Open Hardware Manager	Q ≟ ⊕ 14 ≪ ▶ ≫ + %	Lines Land Lines Lines Lines			Territor Transaction		
	Name Constraints Status	WNS TNS WHS THS TPWS Total P	Power Failed Routes LU	AT FF BRAMS URAM	DSP Start 0 2/3/201/24 Ph	Elapsed	Run Strategy Vivado Synthesis Defaults (Vivado Synthesis 2018)
	> impl_1 constrs_1 Not started						Vivado Implementation Defaults (Vivado Implementation 2018)
	6.0						>

• Flow Navigator > Synthesis > Run Synthesis.

Ajout du fichier timer64.vhd au projet Vivado mon_ip.xpr

- Si la synthèse est OK, on *repackage* le bloc IP :
 - Flow Navigator > Project Manager > Package IP.
- On passera en revue toutes les étapes de « *Packaging Steps* » :
 - Files Groups > Merge changes from File Groups Wizard.
 - Customization Parameters > Merge changes from Customization Parameters Wizard.
 - *Review and Package > Re-Package IP.*
- La réalisation du périphérique *timer* est terminée. Fermer le projet :
 - File > Close Project.
- Revenir au projet Vivado project_1.xpr. On réalisera les actions dans l'ordre suivant :
 - Appuyer sur la touche F6 pour revalider le *design*.
 - Cliquer sur « Refresh IP Catalog ».
 - Cliquer sur « *Upgrade Selected* ».
- On regénère le *Block Design* :
 - Flow Navigator > IP Integrator > Generate Block Design.

🚴 🛛 Generate	Output Products	- • ×
The following output prod	ucts will be generated.	A .
Preview		
Q ¥ ♦		
✓	C per IP)	
🏐 Synthesis		
Implementation	ı	
Simulation		
Synthesis Options		
O <u>G</u> lobal		
 Out of context per 	IP	
Out of context per	Block Design	
	_ ,	
Run Settings		
On <u>l</u> ocal host:	Number of jobs: 8	~
○ On <u>r</u> emote hosts	Configure <u>H</u> osts	
O U <u>s</u> e LSF:	Con <u>f</u> igure LSF	
Apply	G <u>e</u> nerate	S <u>k</u> ip

Génération du Block Design

- Lancer enfin la synthèse pour générer le fichier .bit :
 - Flow Navigator > Program and Debug > Generate Bitstream.
- Sortir de Vivado. Recopier le nouveau fichier de programmation system.bit à la place de l'ancien: host% cd ZedBoard host% mv system.bit system.bit.org host% cp design/design_ZedBoard/project_1/project_1.runs/impl_1/design_1 _wrapper.bit system.bit
- Programmer le fichier system.bit dans le circuit FPGA de la carte cible ZedBoard. Il faudra faire cette opération à chaque reset de la carte cible :
 host% cd ZedBoard
 host% mbsdk
 [Xilinx EDK]\$./load-design

12.3. Tests logiciels

Pour tester le *timer*, il suffira d'interagir avec lui via son registre de contrôle et ses deux registres de données accessibles via le bus AXI et mappés dans l'espace d'adressage.

Pour les tests logiciels, on écrira un pilote de périphérique en mode utilisateur (*user mod driver*) qui évitera d'écrire un vrai pilote de périphérique. Un pilote de périphérique en mode utilisateur ouvre le périphérique /dev/mem en tant que superutilisateur et mappe une page mémoire de 4 Ko (ou plus) dans l'espace d'adressage du processus sur l'adresse de base des registres du périphérique via l'appel système mmap(). Dès lors, de simples lectures et écritures de 32 bits permettent d'interagir avec le périphérique...

Test de la mesure de temps

- Se placer dans le répertoire tst/tsttimer64/: host% cd tst/tsttimer64
- Analyser le fichier squelette tsttimer64.c donné en annexe et compléter le pour mesurer le delta toutes les secondes du compteur 64 bits. On notera que la constante XPAR_MON_IP_0_S00_AXI_BASEADDR définie dans le fichier xparameters.h correspond à l'adresse de base des registres du *timer* fixée sous Vivado : host% gedit tsttimer64.c
- Compiler le fichier tsttimer64.c et installer l'application tsttimer64 dans le système de fichiers root: host% ./go host% ./goinstall
- Regénérer le RAM disk : host% cd ramdisk host% sudo ./goramdisk host% ./goinstall
- Relancer le noyau Linux depuis *u-boot* : U-Boot> run ramboot
- Tester l'application tsttimer64. Quelle valeur théorique doit-on avoir en hexadécimal pour la valeur delta pour une seconde ? Pourquoi les valeurs delta mesurées diffèrent-elles de la valeur théorique ? Que se passe-t-il si l'on charge le système ?

Test de l'incrémentation

- Se placer dans le répertoire tst/tstinc/: host% cd tst/tstinc
- Analyser le fichier squelette tstinc.c et compléter le pour réaliser un *snapshot* toutes les secondes du compteur 64 bits.
- Compiler le fichier tstinc.c et installer l'application tstinc dans le système de fichiers *root*.
- Regénérer le RAM disk.
- Relancer le noyau Linux depuis *u-boot* et tester l'application tstinc. Au bout de combien de temps aura-t-on le débordement du compteur 64 bits ?

13. EX 4 : CREATION DU RAM DISK POUR LE NOYAU LINUX STANDARD

Nous allons regénérer le *RAM disk* qui sera utilisé par le noyau Linux standard exécuté par le processeur Cortex-A9 de la carte cible ZedBoard. Nous allons aussi y intégrer tous les utilitaires nécessaires pour tester les performances Temps Réel du noyau Linux standard.

- Se placer dans le répertoire tst/jitter/: host% cd tst/jitter
- Analyser le fichier squelette jitter.c donné en annexe et compléter le pour réaliser la mesure en continu du *jitter*, c'est-à-dire la mesure de la plus grande différence entre la mesure mesurée de l'incrémentation du *timer* et son incrémentation théorique sur une seconde (valeur mesurée sur une seconde moins 100 millions en dizaines de ns) :
- Compiler le fichier jitter.c et installer l'application jitter dans le système de fichiers *root*.
- Générer les utilitaires de tests cyclictest, stress... Les installer dans le système de fichiers *root* :

```
host% cd tst
host% cd stress
host% ./go
host% ./goinstall
host% cd tst
host% cd schedutils
host% ./go
host% ./goinstall
host% cd tst
host% cd rt-tests
host% ./go
host% ./goinstall
```

- Regénérer le *RAM disk*.
- Télécharger le noyau Linux standard et son *RAM disk* dans la carte cible ZedBoard et tester l'application jitter.

14. EX 5 : MESURE DE TEMPS DE LATENCE AVEC LE NOYAU LINUX STANDARD

Nous allons mesurer des temps de latence du noyau Linux standard dans le cas d'un noyau non stressé puis dans le cas d'un noyau stressé.

Pour stresser le noyau, on utilisera l'utilitaire stress.

Pour mesurer les temps de latence, on utilisera l'utilitaire cyclictest.

 Dévalider le throttling : ZedBoard: # echo -1 > /proc/sys/kernel/sched_rt_runtime_us

Noyau standard non stressé :

- Lancer jitter. Noter le temps de latence maximum au bout de 5 minutes de tests : ZedBoard:# jitter
- Lancer cyclictest. Noter le temps de latence maximum au bout de 5 minutes de tests : ZedBoard:# cyclictest -n -p 99 -i 5000

Noyau standard stressé :

- Stresser le noyau avec stress. Que fait le programme stress ? ZedBoard:# stress -c 50 -i 50 &
- Lancer jitter. Noter le temps de latence maximum au bout de 5 minutes de tests : ZedBoard:# jitter
- Lancer cyclictest. Noter le temps de latence maximum au bout de 5 minutes de tests : ZedBoard:# cyclictest -n -p 99 -i 5000

15. EX 6 : CREATION DU RAM DISK POUR LE NOYAU LINUX XENOMAI

Nous allons voir comment créer le système de fichiers *root* qui sera utilisé par le noyau Linux Xenomai exécuté par le processeur de la carte cible ZedBoard. Nous allons aussi y intégrer tous les utilitaires nécessaires pour tester les performances Temps Réel du noyau Linux Xenomai.

• Créer le système de fichiers *root* squelette root_fs pour la carte cible ZedBoard : host% cd ramdisk

host% ./goskel
host% ./gorootfs

• Générer les utilitaires de tests de Xenomai cyclictest, latency...:

host% cd xenomai
host% ./goconfig
host% ./go
host% ./goinstall

• Générer les utilitaires de tests cyclictest, stress...

```
host% cd tst
host% cd stress
host% ./go
host% ./goinstall
host% cd tst
host% cd schedutils
host% ./go
host% ./goinstall
host% cd tst
host% cd rt-tests
host% ./go
host% ./go
host% ./goinstall
```

- Se placer dans le répertoire tst/hello_xenomai/: host% cd tst/hello_xenomai
- Analyser le fichier hello_xenomai.c donné en annexe qui est le fameux «*Hello World!* » sous forme d'une tâche Xenomai périodique. Quelle API Xenomai a-t-on utilisé ?
- Compiler le fichier hello_xenomai.c et installer l'application hello_xenomai dans le système de fichiers *root*.
- Se placer dans le répertoire tst/jitter_xenomai/: host% cd tst/jitter_xenomai

- Analyser le fichier squelette jitter_xenomai.c donné en annexe et compléter le pour réaliser la mesure en continu du *jitter* sous forme d'une tâche Xenomai périodique, c'est-àdire la mesure de la plus grande différence entre la mesure mesurée de l'incrémentation du *timer* et son incrémentation théorique sur une seconde (valeur mesurée sur une seconde moins 100 millions en dizaines de ns). Quelle API Xenomai a-t-on utilisé ?
- Compiler le fichier jitter_xenomai.c et installer l'application jitter_xenomai dans le système de fichiers *root*.
- Regénérer le *RAM disk*.

16. EX 7 : COMPILATION DU NOYAU LINUX XENOMAI

- Appliquer le patch Xenomai sur le noyau Linux. Que fait le *shell script* go-ipipe? Quelle commande Linux utilise-t-on pour patcher un fichier ?: host% cd xenomai host% ./go-ipipe
- Compiler le noyau Linux Xenomai pour la carte cible ZedBoard : host% cd linux-xenomai host% ./go
- Installer le fichier du noyau Linux dans le répertoire de téléchargement d'u-boot /tftpboot: host% ./goinstall
- Recharger si besoin le désign de référence system.bit dans le circuit FPGA de la carte cible ZedBoard : host% cd ZedBoard host% mbsdk [Xilinx EDK]\$./load-design
- Depuis *u-boot* de la carte cible ZedBoard, exécuter la commande suivante : U-Boot> run ramboot
- Observer les traces de boot du noyau Xenomai dans la fenêtre minicom : Starting kernel ...

```
Uncompressing Linux... done, booting the kernel.
Booting Linux on physical CPU 0x0
Linux version 5.4.180-xilinx
                               (kadionik@ipcchipik) (gcc version 11.3.0
(Buildroot 2021.11-4428-q6b6741b)) #3 SMP PREEMPT Wed Mar 15
15:01:00 CET 2023
CPU: ARMv7 Processor [413fc090] revision 0 (ARMv7), cr=18c5387d
CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
OF: fdt: Machine model: xlnx, zynq-7000
bootconsole [earlycon0] enabled
CPU: ARMv7 Processor [413fc090] revision 0 (ARMv7), cr=18c5387d
CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
OF: fdt: Machine model: xlnx, zynq-7000
bootconsole [earlycon0] enabled
earlycon: cdns0 at MMIO 0xe0001000 (options '115200n8')
bootconsole [cdns0] enabled
. . .
Kernel
           command
                         line:
                                   console=ttyPS0,115200
                                                          root=/dev/ram
ramdisk_size=131072 rw
earlyprintk earlycon
. . .
I-pipe, 333.333 MHz clocksource, wrap in 12884 ms
clocksource: ipipe_tsc: mask: 0xffffffffffffffffffmax_cycles: 0x4ce07af025,
max_i
dle_ns: 440795209040 ns
timer #0 at (ptrval), irq=17
I-pipe, 333.333 MHz timer
Interrupt pipeline (release #8)
Console: colour dummy device 80x30
. . .
```

I-pipe, 333.333 MHz timer [Xenomai] scheduling class idle registered. [Xenomai] scheduling class rt registered. I-pipe: head domain Xenomai registered. [Xenomai] Cobalt v3.2.1 NET: Registered protocol family 17 Registering SWP/SWPB emulation handler mmc0: new high speed SDHC card at address 0007 mmcblk0: mmc0:0007 SD04G 3.71 GiB mmcblk0: p1 hctosys: unable to open rtc device (rtc0) ALSA device list: No soundcards found. RAMDISK: gzip image found at block 0 EXT4-fs (ram0): mounted filesystem with ordered data mode. Opts: (null) VFS: Mounted root (ext4 filesystem) on device 1:0. devtmpfs: mounted Freeing unused kernel memory: 1024K Run /sbin/init as init process Hostname : ZedBoard Kernel release : Linux 5.4.180-xilinx Kernel version : #3 SMP PREEMPT Wed Mar 15 15:01:00 CET 2023 Mounting /proc : [SUCCESS] : [SUCCESS] Mounting /sys Mounting /dev : [SUCCESS] Mounting /dev/pts : [SUCCESS] Enabling hot-plug : [SUCCESS] Populating /dev : [SUCCESS] Mounting other filesystems : [SUCCESS] Starting telnetd : [SUCCESS] Network configuration : [SUCCESS] System initialization complete. Please press Enter to activate this console. ZedBoard:/# uname -r 5.4.180-xilinx ZedBoard:/#

17. EX 8 : MESURE DE TEMPS DE LATENCE AVEC LE NOYAU LINUX XENOMAI

17.1. Outils standards

Nous allons mesurer des temps de latence sur le noyau Linux Xenomai dans le cas d'un noyau non stressé puis dans le cas d'un noyau stressé.

Pour stresser le noyau, on utilisera l'utilitaire stress.

- Dévalider le throttling : ZedBoard:# echo -1 > /proc/sys/kernel/sched_rt_runtime_us
- Dévalider l'anticipation sur la latence minimale de Xenomai : ZedBoard:# echo 0 > /proc/xenomai/latency

Noyau Xenomai non stressé. Outils standards :

• Lancer cyclictest. Noter le temps de latence maximum au bout de 5 minutes de tests : ZedBoard:# cyclictest -n -p 99 -i 5000

Noyau Xenomai stressé. Outils standards :

- Stresser le noyau avec stress: ZedBoard:# stress -c 50 -i 50 &
- Lancer cyclictest. Noter le temps de latence maximum au bout de 5 minutes de tests : ZedBoard:# cyclictest -n -p 99 -i 5000

Noyau Xenomai non stressé. Outils Xenomai :

On utilisera maintenant les outils Xenomai qui se trouvent dans le répertoire /usr/xenomai/.

- Lancer l'outil Xenomai cyclictest. Noter le temps de latence maximum au bout de 5 minutes de tests : ZedBoard:# /usr/xenomai/demo/cyclictest -n -p 99 -i 5000
- On utilise maintenant l'outil Xenomai latency dans 3 modes différents. A quoi correspondent ces 3 modes ? Noter pour les 3 modes le temps de latence maximum au bout de 5 minutes de tests :

ZedBoard:# /usr/xenomai/bin/latency -t0 -p 5000 ZedBoard:# /usr/xenomai/bin/latency -t1 -p 5000 ZedBoard:# /usr/xenomai/bin/latency -t2 -p 5000

 Lancer jitter_xenomai. Noter le temps de latence maximum au bout de 5 minutes de tests : ZedBoard:# jitter_xenomai

Noyau Xenomai stressé. Outils Xenomai :

- Stresser le noyau avec stress:
 ZedBoard:# stress -c 50 -i 50 &
- Lancer l'outil Xenomai cyclictest. Noter le temps de latence maximum au bout de 5 minutes de tests : ZedBoard:# /usr/xenomai/demo/cyclictest -n -p 99 -i 5000
- Lancer l'outil Xenomai latency dans les 3 modes. Noter pour les 3 modes le temps de latence maximum au bout de 5 minutes de tests : ZedBoard:# /usr/xenomai/bin/latency -t0 -p 5000 ZedBoard:# /usr/xenomai/bin/latency -t1 -p 5000 ZedBoard:# /usr/xenomai/bin/latency -t2 -p 5000
- Lancer jitter_xenomai. Noter le temps de latence maximum au bout de 5 minutes de tests : ZedBoard:# jitter_xenomai

17.2. Outils graphiques

Nous allons répéter les mesures avec des outils supplémentaires pour obtenir des graphiques. Nous allons uniquement exploiter cyclictest.

Nous aurons ainsi 2 types de graphiques :

• L'histogramme : ce graphique donne le nombre de fois que l'on obtient un temps de latence donné sur la durée de la mesure.

• La latence : ce graphique donne l'évolution du temps de latence au cours du temps. Si l'on a une mesure toutes les 1000 µs (1 ms), on en aura ainsi 1000 par seconde. On pourra alors visualiser l'évolution de la latence au cours du temps.

On ne produira ici que les graphiques dans le cas d'un noyau stressé.

Noyau Xenomai stressé. Outils standards :

- Stresser le noyau avec stress : RPi3:# stress -c 50 -i 50 &
- Lancer cyclictest pour une mesure pour 5 minutes de tests. A quoi correspond la valeur 300000?
 RPi3:# cyclictest -1 300000 -n -m -p 99 -i 1000 -v > ons.log
- Transférer le fichier ons.log (s pour standard) vers le PC hôte. Il faudra au préalable configurer l'interface réseau de la carte cible RPi (voir annexe 2):
 RPi3:# tftp -p -r ons.log @IP_host
- Recopier le fichier ons.log dans son répertoire de travail : host% cd tst host% cp /tftpboot/ons.log .
- Créer les graphiques histogramme et latence avec les shells scripts gohist et golat (sous /bin/) créés par l'auteur des TP: host% cd tst host% cp tftpboot/ons.log . host% gohist ons.log host% golat ons.log
- Commenter les résultats obtenus.

Noyau Xenomai stressé. Outils Xenomai :

- Stresser le noyau avec stress : RPi3:# stress -c 50 -i 50 &
- Lancer l'outil Xenomai cyclictest pour une mesure pour 5 minutes de tests: RPi3:# /usr/xenomai/demo/cyclictest -1 300000 -n -m -p 99 -i 1000 -v > onx.log
- Transférer le fichier onx.log (x pour Xenomai) vers le PC hôte : RPi3:# tftp -p -r onx.log @IP_host
- Recopier le fichier onx.log dans son répertoire de travail : host% cd tst host% cp /tftpboot/onx.log .

- Créer les graphiques histogramme et latence : host% cd tst host% cp tftpboot/onx.log . host% gohist onx.log host% golat onx.log
- Commenter les résultats obtenus. Les comparer aux résultats précédents.

18. CONCLUSION

On complètera le tableau suivant avec les mesures de temps de latence de la carte ZedBoard :

Outils Linux	ZedBoard	ZedBoard
Temps de latence en µs	Linux standard non stressé	Linux standard stressé
cyclictest		
jitter		

Outils Linux	ZedBoard	ZedBoard
Temps de latence en µs	Linux Xenomai non stressé	Linux Xenomai stressé
cyclictest		

Outils Xenomai	ZedBoard	ZedBoard
Temps de latence en µs	Linux Xenomai non stressé	Linux Xenomai stressé
cyclictest		
Xenomai		
latency -t0		
latency -t1		
latency -t2		
jitter_xenomai		

Que peut-on en conclure si l'on compare les résultats obtenus avec le noyau standard avec ceux obtenus avec le noyau Xenomai ?

Le matériel Libre *timer* 64 bits créé peut-il être utilisé comme outil de mesure de temps de latence ?

19. REFERENCES

- Site d'Intel : https://www.intel.com/
- Site de Xilinx : https://www.xilinx.com/
- Carte DE10-Standard : https://www.terasic.com.tw/cgibin/page/archive.pl?Language=English&No=1081&PartNo=4
- DE10-Standard Computer System with Nios II : https://ftp.intel.com/Public/Pub/fpgaup/pub/Intel_Material/18.1/Computer_Systems/DE10 -Standard/DE10-Standard_Computer_NiosII.pdf
- DE10-Standard User Manual : https://ftp.intel.com/Public/Pub/fpgaup/pub/Intel_Material/Boards/DE10-Standard/DE10_Standard_User_Manual.pdf
- Nios II Performance Benchmarks DS-1066 : https://cdrdv2public.intel.com/666568/ds_nios2_perf-683629-666568.pdf
- The Zynq Book. L. Crockett and al. Editions Strathclyde Academic Media. Version numérique téléchargeable ici : http://www.zynqbook.com/
- Carte cible ZedBoard : http://zedboard.org/product/zedboard
- Les latences de Xenomai. C. Blaess : https://www.blaess.fr/christophe/2012/07/23/les-latences-de-xenomai/
- mmap() sur le site Intel: https://community.intel.com/t5/FPGA-Wiki/Accessing-hardware-registers-from-userspace-programs/ta-p/735151

20. ANNEXE 1 : FICHIER SOURCE TSTTIMER64.C

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <signal.h>
#include <sys/mman.h>
#include "xparameters.h"
#define SIZE 4096
volatile int *ptr;
int main(int argc, char **argv) {
  int fd;
  unsigned int fort1, fort2;
  unsigned int faible1, faible2;
  int delta;
  delta = 0;
  fd=open("/dev/mem", O_RDWR | O_SYNC);
  if(fd < 0) {
    printf("Failed to open /dev/mem\n");
    exit(-1);
  }
  printf("/dev/mem open OK\n");
                                  PROT_READ PROT_WRITE,
  ptr
        =
              mmap(0,
                         SIZE,
                                                          MAP_SHARED,
                                                                           fd,
XPAR_MON_IP_0_S00_AXI_BASEADDR);
  if(ptr == (void *)-1) {
    close(fd);
    printf("mmap failed\n");
    exit(-1);
  }
  printf("mmap OK\n");
  printf("Test timer 64 bits. Delta value\r\n");
  printf("Delai = 1000 ms\r\n");
  *ptr = ???; // Reset
  while(1) {
    *ptr = ???; // Snapshot
    fort1 = ???;
    faible1 = ???;
    sleep(1);
    *ptr = ???; // Snapshot
    fort2 = ???;
    faible2 = ???;
    . . .
```

```
}
munmap((void *)ptr, SIZE);
close(fd);
exit(0);
}
```

21. ANNEXE 2 : FICHIER SOURCE JITTER.C

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <signal.h>
#include <sys/mman.h>
#include "xparameters.h"
#define SIZE 4096
// 1 seconde correspond àl'incrémentation du compteur de 100 millions
#define COUNT_1S 10000000
// 1 microseconde correspond à l'incrémentation du compteur de 100 (10 ns)
#define COUNT_1U 100
volatile int *ptr;
int main() {
  int fd;
  unsigned int fort1, fort2;
  unsigned int faible1, faible2;
  unsigned long long value1, value2;
  int delta;
  unsigned int jitter;
  unsigned int jitter_max;
  jitter_max = 0;
  fd=open("/dev/mem",O_RDWR | O_SYNC);
  if(fd < 0) {
    printf("Failed to open /dev/mem\n");
    exit(-1);
  }
  printf("/dev/mem open OK\n");
                                  PROT_READ PROT_WRITE,
                                                          MAP_SHARED,
  ptr
        =
             mmap(0,
                         SIZE,
                                                                           fd.
XPAR_MON_IP_0_S00_AXI_BASEADDR);
  if (ptr == (void *)-1) {
    close(fd);
    printf("mmap failed\n");
    exit(-1);
  }
  printf("mmap OK\n");
  *ptr = ???; // Reset
  while(1) {
    *ptr = ???; // Snapshot
    fort1 = ???;
    faible1 = ???;
    sleep(1);
```

```
*ptr = ???; // Snapshot
fort2 = ???;
faible2 = ???;
value1 = ((unsigned long long) fort1 << 32) | faible1;
value2 = ((unsigned long long) fort2 << 32) | faible2;
delta = value2-value1;
jitter = abs(delta - COUNT_1S);
...
}
munmap((void *)ptr, SIZE);
close(fd);
exit(0);
```

}

22. ANNEXE **3** : FICHIER SOURCE HELLO_XENOMAI.C

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <signal.h>
#include <sys/mman.h>
#include <alchemy/task.h>
// Periode de 1 s
#define TIMESLEEP 100000000
RT_TASK demo_task;
int end=0;
void catch_signal() {
  end=1;
}
void demo() {
  rt_printf("Starting Xenomai task...\n");
  // Configuration de la tache courante en mode periodique
  rt_task_set_periodic(NULL, TM_NOW, TIMESLEEP);
  while (end == 0) {
    // Attente de l'expiration de la periode
    rt_task_wait_period(NULL);
    rt_printf("Hello World from Xenomai!\n");
  }
}
int main() {
  char str[32];
  // Execution de la fonction catch_signal Ctrl+C (SIGINT) et
  // Sur un kill -9 (SIGTERM) pour tuer la tache Xenomai
  signal(SIGTERM, catch_signal);
  signal(SIGINT, catch_signal);
  // Avoids memory swapping for this program
  mlockall(MCL_CURRENT MCL_FUTURE);
  // Definition du nom de la tache Xenomai
  sprintf(str,"hello");
  // Creation et demarrage de la tache Xenomai
  rt_task_create(&demo_task, str, 0, 50, 0);
  rt_task_start(&demo_task, &demo, 0);
  // Attente appui sur clavier
  getchar();
```

```
rt_task_delete(&demo_task);
exit(0);
}
```

23. ANNEXE 4 : FICHIER SOURCE JITTER_XENOMAI.C

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <signal.h>
#include <sys/mman.h>
#include <alchemy/task.h>
#include "xparameters.h"
#define SIZE 4096
// 1 seconde correspond à l'incrémentation du compteur de 100 millions
#define COUNT_1S 10000000
// 1 microseconde correspond à l'incrémentation du compteur de 100 (10 ns)
#define COUNT_1U 100
// Periode de 1 s
#define TIMESLEEP 100000000
// Priorite de la tache
#define PRIO 99
RT_TASK demo_task;
int end=0;
volatile int *ptr;
void catch_signal() {
  end=1;
}
void demo() {
  unsigned int fort1, fort2;
  unsigned int faible1, faible2;
  unsigned long long value1, value2;
  int delta;
  unsigned int jitter;
  unsigned int jitter_max;
  jitter_max = 0;
  rt_printf("Starting Xenomai task...\n");
  // Configuration de la tache courante en mode periodique
  rt_task_set_periodic(NULL, TM_NOW, TIMESLEEP);
  while (end == 0) {
    *ptr = ???; // Snapshot
    fort1 = ???;
    faible1 = ???;
    rt_task_wait_period(NULL);
    *ptr = ???; // Snapshot
    fort2 = ???;
```

```
faible2 = ???;
    value1 = ((unsigned long long) fort1 << 32) | faible1;</pre>
    value2 = ((unsigned long long) fort2 << 32) | faible2;</pre>
    delta = value2-value1;
    jitter = abs(delta - COUNT_1S);
    . . .
  }
}
int main() {
 char str[32];
 int fd;
  signal(SIGTERM, catch_signal);
  signal(SIGINT, catch_signal);
 mlockall(MCL_CURRENT | MCL_FUTURE);
  fd=open("/dev/mem",O_RDWR | O_SYNC);
  if(fd < 0) {
    printf("Failed to open /dev/mem\n");
    exit(-1);
  }
 printf("/dev/mem open OK\n");
        =
                         SIZE,
                                  PROT_READ PROT_WRITE,
                                                            MAP_SHARED,
 ptr
             mmap(0,
                                                                            fd,
XPAR_MON_IP_0_S00_AXI_BASEADDR);
  if (ptr == (void *)-1) {
    close(fd);
    rt_printf("mmap failed\n");
    exit(-1);
  }
 printf("mmap OK\n");
  *ptr = 0x2; // Reset
 sprintf(str,"jitter");
  rt_task_create(&demo_task, str, 0, PRIO, 0);
 rt_task_start(&demo_task, &demo, 0);
  // Attente appui sur clavier
 getchar();
  rt_task_delete(&demo_task);
 munmap((void *)ptr, SIZE);
 close(fd);
  exit(0);
}
```

24. ANNEXE 5 : CONFIGURATION RESEAU HOTES ET CIBLES

	POSTE PC01	
	NOM	ADRESSE IP
HOTE	citron07	192.168.4.7
CIBLE	ml01 zb01	192.168.4.101

	POSTE PC02	
	NOM	ADRESSE IP
HOTE	citron08	192.168.4.8
CIBLE	ml02 zb02	192.168.4.102

	POSTE PC03	
	NOM	ADRESSE IP
НОТЕ	citron09	192.168.4.9
CIBLE	ml03 zb03	192.168.4.103

	POSTE PC04	
	NOM	ADRESSE IP
НОТЕ	citron10	192.168.4.10
CIBLE	ml04 zb04	192.168.4.104

	POSTE PC05	
	NOM	ADRESSE IP
НОТЕ	citron11	192.168.4.11
CIBLE	ml05 zb05	192.168.4.105

	POSTE PC06	
	NOM	ADRESSE IP
HOTE	citron12	192.168.4.12
CIBLE	ml06 zb06	192.168.4.106

Masque de sous réseau : 255.255.255.0

Exemple : configuration réseau de la carte cible ml01 : target# ifconfig eth0 192.168.4.101