Conception des SOC/SOPC

Utilisation du SOPC Builder pour la conception de

SOPC (System On Programmable Chip)



Le SOPC Builder permet, entre autres, de concevoir des microcontrôleurs spécifiques à une application. Ces microcontrôleurs comportent donc une partie processeur à laquelle on associe des périphériques (PIO, Timers, UART, USB, composants propriétaires, ...) et de la mémoire. Cette dernière peut être embarquée dans le FPGA (on parle alors de RAM/ROM **On Chip**) ou à l'extérieur du composant FPGA. La partie microprocesseur proprement dite est le NIOSII de ALTERA (aujourd'hui devenu Intel), processeur de 32 bits qui se décline en trois versions : économique, standard, rapide. La version économique, la moins puissante, utilise le moins de ressources du FPGA. Bien sûr il est possible d'intégrer d'autres types de processeurs pour peu qu'on dispose de leurs modèles (VHDL, Verilog, ...).

La création d'une application SOPC comprend les étapes suivantes :

- 1) Création du composant matériel (processeur + périphériques) dans l'environnement Quartus II
- 2) Eventuellement simulation avec l'outil Modelsim
- 3) Téléchargement dans le composant FPGA (configuration) (environnement Quartus II)
- 4) Création du logiciel dans l'environnement NIOS2IDE, téléchargement dans le FPGA et débogage.

Partie 1 : Développement du matériel

Exemple : projet « sopc_compteur »

Pour illustrer la démarche on propose de créer un SOPC disposant de deux entrées et huit sorties logiques. Le logiciel d'application, développé en langage C, effectue périodiquement une incrémentation d'une variable de type octet et l'affiche sur huit leds de sortie. Les deux entrées (boutons poussoirs actifs à 0) permettent respectivement de remettre à 0 la variable et de bloquer le comptage. Dans cette application la mémoire utilisée est intégrée au composant FPGA : il s'agira donc de RAM/ROM **On Chip**.

Environnement de travail :

PC avec Windows 7 service Pack 1. Quartus II version 11.0, Nios 2 IDE version 11.0.

<u>Nota :</u> à partir de la version 11 de Quartus, un nouvel outil de conception de SOPC nommé QSYS a été intégré et remplace le SOPC Builder. Toutefois dans ce qui suit, nous continuerons à utiliser le SOPC Builder.

Démarrer Quartus II version 11.0 et créer un projet en spécifiant votre répertoire de travail, le nom de votre projet (sopc_compteur par exemple) et le composant FPGA utilisé : ici on utilise une carte Terasic DE0 nano équipée d'un FPGA cyclone 4 EP4CE22F17C6N.

,	<u></u>
	🐇 New Project Wizard
_	Directory, Name, Top-Level Entity [page 1 of 5]
	What is the working directory for this project?
	C:/quartus11_projets/de0_nano/sopc_compteur
	What is the name of this project?
	sopc_compteur
	What is the name of the top-level design entity for this project? This name is case sen:
	sopc_compteur
	Use Existing Project Settings

× ×		FBGA Any 6 anced devices	Package: Pin count: Speed grade:	▼			ne IV E	Family: Cyclor		
~		Any 6 anced devices	Pin count: Speed grade:	~		Device: All				
~		6 anced devices	Speed grade:	Devices: All						
		anced devices		Target device						
		anceu uevices	Chow adver		alget device					
			Jindw auvar			ter	e selected by the Fitt	Auto device		
		compatible only	HardCopy c		es' list	lable device	vice selected in 'Avai	Specific de		
<u>'LL GI 🗠</u>	PLL	bedded multiplier 9-bit elements	Bits Emb	Memo	User I/Os	LES	Core Voltage	Name		
10	2		30	276480	180	6272	1.2V	P4CE6F17C6		
10	2		46	423936	180	10320	1.2V	P4CE10F17C6		
20	4		112	516096	166	15408	1.2V	P4CE15F17C6		
20	4		112	516096	344	15408	1.20	P4CE15F23C6		
20	4		132	609256	104	22320	1.20	P4CE22F17C6		
20	4		132	608256	533	28848	1.27	P4CE30F20C6		
~ ~			102		000	20010	1.27			
>										
F	2 2 4 4 4 4	ibedded multiplier 9-bit elements	Bits Emb 30 46 112 112 112 132	Memo 276480 423936 516096 516096 608256 608256	User I/Os 180 180 166 344 154 329	LEs 6272 10320 15408 15408 22320 28848	Core Voltage 1.2V 1.2V 1.2V 1.2V 1.2V 1.2V 1.2V 1.2V	Name P4CE6F17C6 P4CE10F17C6 P4CE15F17C6 P4CE15F23C6 P4CE22F17C6 P4CE30F23C6		

Ouvrir une feuille de schéma (New => block diagram/schematic file). Y placer une broche d'entrée ou de sortie puis enregistrer le fichier avec le **même nom** que le projet (ici sopc_compteur) **en** faisant attention de bien l'enregistrer dans votre répertoire !!!

🐇 Quartus II - C:/quartus11_projets/de0_nano/s	sopc_compteur/s	opc_compteur - sopc_compteur	
File Edit View Project Assignments Processing	Tools Window He	elp 💎	
🗋 🎽 💭 🎒 🎒 👗 🖻 🛍 🗠 🖓 🔅	compteur	💌 💥 🐓 🥒 🏈 🎯 🕨 🕨	🔊 🔯
Project Navigator 🗗 🕈	× 🔁	sopc_compteur.bdf	
Entity	: 🗟 📐 🔍 🤇	ᢀᅀᅌᇂ᠂᠐᠋᠋᠋ᡞᡞ᠋ᠺ	
A Cyclone IV E: EP4CE22F17C6 → sopc_compteur B			
★ Hierarchy Files d ⁹ Design Units		pn_name	

Dans l'environnement Quartus 11.0, lancer le SOPC Builder (Tools=> SOPC Builder). Dans la fenêtre « Create new system », donner un nom au SOPC que l'on va concevoir (ici mon_sopc) puis sélectionner « VHDL » (ceci aura pour effet de générer des fichiers de description du sopc en langage vhdl).

_		
🖁 🖳 Create Nev	v System	
System Name: m	on_sopc	
Target HDL: 🔘 \	/erilog	
•	/HDL	
(OK Cancel	

Utilisation du SOPC Builder_1	J.L. Boizard
-------------------------------	--------------

Vérifier que l'horloge du processeur du sopc est bien de 50 MHz.

Dans la colonne de gauche, sélectionner « Processor » puis « Nios II processor » et faire « add ». Sélectionner la version économique parmi les trois proposées. Faire « next » et accepter les options par défaut (sélectionner le debugger niveau 1).

ures >	MMU and MPU Settings 🔰 JTAG [Сери
	ONios II/f	
he on ply e	RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide Barrel Shifter Data Cache Dynamic Branch Prediction	
	Up to 0 DMIPS	
	1400-1800 LEs	
ne are Divide	Inree M9Ks + cache	
ffset: 0×0		
ffs fse	set: 0x0 st: 0x20	set: 0x0

On obtient çà :

😃 Altera SOPC Builder									
File Edit Module System View T System Contents System Generation	Fools Nios II Help	,							
Component Library	Target		Clock S	Settings	1 -				
Library	Device Family:	Syclone IV E	Name		Source External		MHz 50,0		Add
Avalon Verification Suite Bridges and Adapters Debug Components									
-Digital Signal Processing -Interface Protocols	Use Conn	Name		Description		Clock	В	lase	End
Legacy Components Memories and Memory Contro Merlin Components Pripherals		cpu_0 instruction_mas data_master itag_debug_mo	ter dule	Nios II Processor Avalon Memory Map Avalon Memory Map Avalon Memory Map	ped Master ped Master ped Slave	[clk] clk_0 [clk] [clk]	-	IRQ 0 0x00000800	I 0x00000ff
⊕ PLL ⊕ Processor Additions Processors									

Dans la colonne de gauche, sélectionner la mémoire « on chip » puis faire « add ». Fixer la taille à 20ko et accepter la configuration proposée :

Component Library	Targ	et		Clock S	Settings		
	Devic	e Family:	Cyclone IV E 🛛 💌	Name	в	Source	MHz
Digital Signal Processing Interface Protocols Legacy Components Memories and Memory Controllers		- L		cik_0		External	50,0
	Use	Conn	Name		Description		Clock
- • Avalon-ST Dual Clock FIFO			🗆 cpu_0	.	Nios II Processor	, , , , , , , , , , , , , , , , , , , ,	[clk]
Avaion-S1 Mutti-Channel Shari			data_master	iter	Avaion Memory I Avaion Memory I	Mapped Master Mapped Master	[clk]
Avalon-ST Single Clock FIFO		$ \rightarrow \rightarrow$	jtag_debug_mo	dule	Avalon Memory I	Mapped Slave	[clk]
On-Chip FIFO Memory On-Chip Memory (RAM or RON		$ \bigcirc $	E sram s1		On-Chip Memory Avalon Memory I	(RAM or ROM) Mapped Slave	[clk1] clk_0
⊞SDRAM							1 – 1
Herlin Components							
i Peripherals							
i ⊕-PLL 🗸 🗸							
)

De la même manière, ajouter deux PIO (Parallel Inputs Outputs) (un de 2 bits en entrée et un de huit bits en sortie).

Nota : on peut les renommer en faisant un clic droit sur le composant.

Rajouter le composant JTAG UART qui permettra de communiquer avec le PC hôte, télécharger le logiciel dans le circuit et débugger le programme. Dans la colonne de droite des interruptions, lui assigner l'interruption n°16.

Revenir sur le processeur en double-cliquant dessus puis sélectionner « on-chip ram » dans les zones « reset vector » et « exception vector ».

Pour des raisons de sécurité on peut rajouter un composant « sysid » (System Identifier): celui-ci a pour rôle de donner un numéro d'identification au système que l'on a conçu et éviter ainsi de télécharger par erreur un programme qui ne correspondrait pas à l'application.

Faire ensuite « System » puis «assign base adress ». Ceci a pour effet d'assigner automatiquement une adresse logique à chaque ressource dans l'espace mémoire adressé par le processeur. A ce stade le SOPC est défini : type de processeur, horloge, périphériques utilisés, taille mémoire, adresses physiques des composants.

Devic	e Family: C	yclone IV E 🔽 🛛 Na	me	So	urce			MHz
		clk	_0	Exte	ernal			50,0
Use	Conn	Name	Description	Clock	Base	End	IRQ	Tags
~		⊟ cpu_0	Nios II Processor	[clk]				
		instruction_master	Avalon Memory Mapped Master	clk_0				
		data_master	Avalon Memory Mapped Master	[clk]	IRQ (D IRQ 3	ı←∖	
		jtag_debug_module	Avalon Memory Mapped Slave	[clk]	■ 0x00010800	0x00010fff		
~		🖃 sram	On-Chip Memory (RAM or ROM)	[clk1]				
		s1	Avalon Memory Mapped Slave	clk_0	■ 0x00008000	0x0000cfff		
 Image: A start of the start of		🖃 leds	PIO (Parallel I/O)	[clk]				
	∽	s1	Avalon Memory Mapped Slave	clk_0		0x0001100f		
~		boutons	PIO (Parallel I/O)	[clk]				
		s1	Avalon Memory Mapped Slave	clk_0	■ 0x00011010	0x0001101f		
V		sysid	System ID Peripheral	[clk]				
		control_slave	Avalon Memory Mapped Slave	clk_0		0x00011027		
~		🖃 jtag_uart_0	JTAG UART	[clk]				
	$ \rightarrow $	avalon_jtag_slave	Avaion Memory Mapped Slave	clk_0		0x0001102f	→ 1 6	

Une fois tous les composants rajoutés, sauvegarder puis cliquer sur « Generate » : ceci a pour effet de générer le fichier VHDL (ou Verilog suivant le choix qui a été fait) et le symbole graphique associés au SOPC que l'on vient de définir. A ce stade le composant a été créé. Ouvrir le fenêtre graphique précédemment créée et double-cliquer dedans (comme lorsqu'on fait une saisie de schéma classique). Dans le répertoire projet, récupérer le symbole du SOPC créé et le placer dans la feuille de travail. Rajouter les ports d'entrées-sorties puis enregistrer et compiler le projet.



Affectation des broches :

Ouvrir le « Pin Planner » puis affecter les broches comme suit (les affectations des broches sont récupérables dans le fichier De0_Nano.qsf):

)					2 3 4 5 6	7 8 9 10 11 12 13 14	15 16		
🖌 Named: * 🛛 🖌 🖉 🖉	idit: 🗙 🗸								
Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair
CLOCK_50	Input	PIN_R8	3	B3_N0	3.3-V LVTTL		8mA (default)		
KEY[1]	Input	PIN_E1	1	B1_N0	3.3-V LVTTL		8mA (default)		
KEY[0]	Input	PIN_J15	5	B5_N0	3.3-V LVTTL		8mA (default)		
LED[7]	Output	PIN_L3	2	B2_N0	3.3-V LVTTL		8mA (default)	2 (default)	
IED[6]	Output	PIN_B1	1	B1_N0	3.3-V LVTTL		8mA (default)	2 (default)	
IED[5]	Output	PIN_F3	1	B1_N0	3.3-V LVTTL		8mA (default)	2 (default)	
IED[4]	Output	PIN_D1	1	B1_N0	3.3-V LVTTL		8mA (default)	2 (default)	
LED[3]	Output	PIN_A11	7	B7_N0	3.3-V LVTTL		8mA (default)	2 (default)	
LED[2]	Output	PIN_B13	7	B7_N0	3.3-V LVTTL		8mA (default)	2 (default)	
LED[1]	Output	PIN_A13	7	B7_N0	3.3-V LVTTL		8mA (default)	2 (default)	
💿 LED[0]	Output	PIN_A15	7	B7_N0	3.3-V LVTTL		8mA (default)	2 (default)	
< <new node="">></new>									

Re-compiler le projet : on obtient un fichier de programmation sopc_compteur.sof qui permet de configurer le FPGA.

A ce stade la conception de la partie matérielle est terminée.

Partie 2 : Développement du logiciel

On peut lancer NIOS2IDE v11.0 depuis le SOPC Builder de Quartus (onglet nios II) ou directement depuis le menu programmes de windows. Les étapes sont les suivantes :

- définition d'un espace de travail (Workspace)
- création du projet
- création de la bibliothèque
- création du programme
- compilation, téléchargement et exécution

2.1) Définition de l'espace de travail (Workspace)

Au lancement de NIOS2IDE, un espace de travail est sélectionné par défaut. Sélectionner l'espace de travail correspondant au circuit qui a été créé (sopc_compteur). La fenêtre « Navigator » généralement située à gauche, répertorie tous les projets déjà existant dans cet espace de travail. Si on souhaite changer d'espace faire : File => Switch Workspace puis sélectionner le nouvel espace de travail.

🖨 Workspace Launcher	
Select a workspace Eclipse stores your projects in a folder called a workspace. Choose a workspace folder to use for this session.	
Workspace: C:\quartus11_projets\de0_nano\sopc_compteur C:\quartus11_projets\de0_nano\sopc_compteur	Browse
?	OK Cancel

2.2) Création du projet

Faire File => New => « NIOS II Application and BSP from Template » puis sélectionner le projet « hello world small » parmi ceux proposés. Sélectionner le SOPC sur lequel sera exécuté le programme (ici mon_sopc avec l'extension **sopcinfo**) et donner un nom (ex. : test_compteur) au projet logiciel.

Nios II Application and BSP from Template	
Nios II Software Examples Create a new application and board support package based on a software example template	
Target hardware information SOPC Information File name: C:\quartus11_projets\de0_nano\sopc_compteur\mon_sopc.sopcinfo CPU name: cpu_0 Application project Project name: test_compteur Vuse default location Project location: C:\quartus11_projets\de0_nano\sopc_compteur\software\test_compteur Project cocation: C:\quartus11_projets\de0_nano\sopc_compteur\software\test_compteur Hello World Small Template description Hello World Small Hello World Small Hello World Small This example runs with or without the MicroC/OS-II RTOS and requires an STDOUT device in your system's hardware. For details, click Finish to create the project and refer to the readme.txt file in the project directory.	
Web Server (RGMII) Web Server (RGMII) For information about how this software example relates to Nic II hardware design examples, refer to the Design Examples page of the Nios II documentation	n 🗸

2.3) Création de la bibliothèque

Faire next et sélectionner « create a new system library » puis "finish". A ce stade le dossier « test_compteur » apparaît dans le navigateur avec le dossier « test_compteur_bsp » (bsp : Board support Package). Faire un clic droit sur test_compteur et sélectionner « build project » dans le menu déroulant.

🖨 Nios II - Eclipse						
File Edit Navigate Sear	ch Project	Run	Nios II	Window	Help	
i 📬 • 🖫 🗟 🕒 🖥) 🕴 💼 🝷	<u>6</u> ° -	€ -	G • 1	参•	0 -
Project Explorer 🛛						
	🗆 🔁	69				
⊞~≌ test_compteur ⊞~≌ test_compteur_bsp) [mon_sopc]					

2.4) Création du programme exécutable

Avec le choix d'un projet existant (Hello_world_small) un programme par défaut a été généré. On peut le renommer en faisant « save as » puis en supprimant le précédent.



2.5) Compilation, téléchargement du programme et exécution sur la cible (NIOS II HW configuration)

Avant le téléchargement du programme, penser à configurer le FPGA (programmation avec le fichier sopc_compteur.sof). Le téléchargement et l'exécution du programme se font par la commande « run » ou « debug ». Cliquer droit sur le projet et sélectionner :

- Run as : pour télécharger et exécuter le programme immédiatement.
- Debug as : pour télécharger et exécuter le programme en mode debug.

Commande « Run as » :

Sélectionner Nios II Hardware et dans la boîte de dialogue, renseigner les rubriques « Name » et « Project » puis faire « Run ». Le programme est téléchargé dans la cible puis est automatiquement exécuté.

Commande « debug as » :

Le même type de boîte de dialogue que dans le mode « run as » s'ouvre. Sélectionner Nios II Hardware, renseigner les rubriques « Name » et « Project » puis faire « Debug ».

Debug Configurations	
Create, manage, and run config Nios II Hardware Tab Group	urations to the second s
Ype filter text C/C++ Application C/C++ Application C/C++ Application C/C++ Application C/C++ Postmortem Debugger Launch Group Nios II DPX Hardware Nios II DPX Hardware Nios II Hardware test_compteur Nios II Hardware	Name: test_compteur Nios II Hardware configuration Project Target Connection Debugger Source Common Project name: test_compteur Project ELF file name: C:\quartus11_projets\de0_nano\sopc_compteur\software\test_compteur.lef Project ELF file name: C:\quartus11_brojets\de0_nano\sopc_compteur\software\test_compteur.lef File system ELF file name: Advanced

Nota : Si l'onglet « Target Connexion » est barré d'une croix rouge, le sélectionner puis faire « refresh connexion »

La fenêtre de debug s'ouvre avec le pointeur positionné sur la première instruction du programme à exécuter :

- La commande « Resume » lance/relance l'exécution.
- La commande « Suspend » suspend l'exécution du programme.
- La commande « Terminate » arrête l'exécution et clôt la session.

Debug - compteur.c - Nios II IDE	
File Edit Refactor Navigate Search Run Project To	ols Window Help
📑 • 🖫 👜 🚠 🏇 • 💽 • 🏊 • 😕 🚿] ि •] 월 • 중 • ← 수 • → •
🅸 Debug 🕄 🙀	🕸 📭 🗉 🔳 🖉 🔍 👁 🕼 🤜 述 🖬 🎽 🗖
compteur Nios II HW configuration [Nios II Hardware Nios II Elf Debugger (01/12/10 10:42) (Suspended) Thread [0] (Suspended) S main() at\compteur.c:11 0x00004066 2 alt_main() at \cygdrive\c\altera\81\nid 1 _start() 0x00004058 I _ start() 0x00004058 I _ start() 0x00004058 Nios II Terminal Window (01/12/10 10:42) I _ nios2-gdb-server output (01/12/10 10:42) I _ nios2-elf-gdb (01/12/10 10:42)	d) Pas à pas 2eds\components\altera_hal\HAL\src\alt_main.c:163 0x000041d8 put (01/12/10 10:42) Suspend Terminate

Un double-clic dans la colonne de gauche de la fenêtre du programme insère un point d'arrêt. Un autre double-clic l'enlève.

Pour visualiser et/ou modifier une variable :

a) Suspendre l'exécution du programme (touche suspend)

 b) Dans la fenêtre « variables » faire un clic droit et sélectionner « add global variables ».
 Sélectionner les variables à visualiser. Celles-ci apparaissent dans la fenêtre, il est alors possible de sélectionner le format de la variable (clic droit sur la variable) et de modifier si besoin sa valeur (clic gauche dans « value »).



Remarque : l'adjonction de #include « system.h » dans le programme permet d'éviter de renseigner les adresses physiques des périphériques du système qui a été créé.

2.6) Téléchargement du programme et exécution sur l'ISS (NIOS II ISS configuration) (ISS : Instruction Set Simulator)

Cette possibilité permet de tester le programme lorsqu'on n'a pas de carte cible. Idem chapitre 5 sauf qu'on sélectionne NIOS II ISS configuration. Attention cependant car les entrées/sorties physiques ne sont pas simulées !!!!

Lors de la création du système (le micro contrôleur), il se peut que certaines ressources ne soient pas parmi celles proposées par le constructeur. Il devient nécessaire de :

- Intégrer un composant d'un fournisseur tiers
- Créer son propre composant

Dans ce chapitre nous allons voir la création d'un composant spécifique (un circuit de génération de PWM), son intégration et sa mise en œuvre dans le SOPC Builder. Notons toutefois qu'une alternative à la méthode décrite ci-dessous consiste à créer son propre composant logique et à connecter les entrées/sorties à autant de PIO que nécessaire. Cette méthode, simple, devient lourde pour des composants disposant de beaucoup d'entrées/sorties.

Création du circuit PWM : Le circuit comprend trois parties distinctes :

- La partie fonctionnelle propre à l'application ciblée
- L'interface avec le bus Avalon d'ALTERA

- La partie « registres » qui mémorise les signaux issus du bus Avalon (cette partie peut être intégrée à l'interface Avalon)



3.1) Création de la partie fonctionnelle

Un circuit générateur PWM comprend :

- Un compteur libre sur N bits piloté par une horloge de référence clk.

- Un comparateur sur N bits qui compare la sortie du compteur avec son modulo (FREQ : ce qui permet de fixer la fréquence de la PWM). La sortie du comparateur assure la remise à zéro du compteur.

- Un comparateur sur N bits qui compare la sortie du compteur avec le rapport cyclique désiré (DUTY). La sortie de ce comparateur génère la sortie pwm_out.

Le fichier VHDL ci-dessous décrit un tel circuit :

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity pwm_nano is
port (
clk, reset_n : in std_logic;
freq, duty : in std_logic_vector (15 downto 0);
out_pwm_nano : out std_logic
);
end entity;
```

```
ARCHITECTURE arch_pwm_nano of pwm_nano IS

-- signaux relatifs au circuit gestion pwm_nano

signal counter : std_logic_vector (15 downto 0);

signal pwm_nano_on : std_logic;
```

BEGIN

```
-- circuit de gestion de la pwm_nano
-- fixe la fréquence de la pwm_nano
divide: process (clk, reset_n)
   begin
   if reset n = 0 then
   counter \leq (others \geq '0');
   elsif clk'event and clk = '1' then
       if counter >= freq then
       counter \leq (others \geq '0');
       else
       counter \leq counter + 1;
       end if;
   end if:
end process divide;
                 ******
   --génère le rapport cyclique
__*************
                      ******
compare: process (clk, reset_n)
   begin
   if reset_n = '0' then
   pwm nano on \leq 0';
```

14

Utilisation du SOPC Builder_1 J.L. Boizard

END arch_pwm_nano ;

3.2) Création de l'entité et de l'interface Avalon (on appellera ce circuit : avalon_pwm)

Pour cela on utilisera les noms des signaux tels qu'apparaissant sur la gauche du schéma synoptique : ces noms sont reconnus par le SOPC Builder comme des signaux du bus Avalon et évitent au concepteur de redéfinir manuellement les correspondances entre signaux.

Entité du circuit:

llibrary IEEE; use IEEE.std_logic_1164.all; use IEEE.std_logic_unsigned.all; use IEEE.std_logic_arith.all; entity avalon_pwm is port (clk, chipselect, write_n, reset_n : in std_logic; writedata : in std_logic_vector (15 downto 0); readdata : out std_logic_vector (15 downto 0); address: std_logic_vector (1 downto 0); out_pwm : out std_logic); end entity;

Architecture du circuit :

15

```
ARCHITECTURE arch_avalon_pwm of avalon_pwm IS
signal freq : std_logic_vector (15 downto 0);
signal duty : std_logic_vector (15 downto 0);
signal counter : std_logic_vector (15 downto 0);
signal control : std_logic_vector (1 downto 0);
signal pwm_on : std_logic;
BEGIN
divide: process (clk, reset_n)
```

begin if control(0) = '0' then

Utilisation du SOPC Builder_1 J.L. Boizard

```
elsif clk'event and clk = '1' then
           if control(1) = 1' then
                 if counter >= freq then
                 counter \leq (others \geq '0');
                 else
                 counter \leq counter + 1;
                 end if;
           end if;
     end if;
end process divide;
compare: process (clk, reset_n)
     begin
     if control(0) = '0' then
     pwm on \leq 1';
     elsif clk'event and clk = '1' then
           if counter >= duty then
           pwm on \leq 0';
           elsif counter = 0 then
           pwm_on <= '1';
           end if;
     end if;
end process compare;
out_pwm <= pwm_on and control(0);
                                          --interface avalon
-- écriture registres
process_write: process (clk, reset_n)
     begin
     if reset n = 0 then
     freq \leq (others \geq '0');
     duty \leq (others \geq '0');
     control \leq (others = '0'):
     elsif clk'event and clk = '1' then
           if chipselect ='1' and write_n = '0' then
                 if address = "00" then
                 freq \leq writedata;
                 end if:
                 if address = "01" then
                 duty <= writedata;
                 end if:
                 if address = "10" then
                 control <= writedata (1 downto 0);
                 end if;
           end if;
     end if:
Utilisation du SOPC Builder 1 J.L. Boizard
                                                   16
```

counter \leq (others \geq '0'):

end process;

```
-- lecture registres

process_Read:

PROCESS(address, freq, duty, control)

BEGIN

case address is

when "00" => readdata <= freq ;

when "01" => readdata <= duty ;

when "10" => readdata <= duty ;

when "10" => readdata <= X"000"&"00"&control ;

when others => readdata <= (others => '0');

end case;

END PROCESS process_Read ;

END arch_avalon_pwm ;
```

3.3) Intégration dans le SOPC Builder

On commence par créer son micro contrôleur comme vu dans le début du document puis dans la fenêtre du SOPC Builder cliquer sur « new » en bas à gauche. Dans la fenêtre qui se présente faire « next » puis « add » pour aller chercher le circuit que l'on vient de décrire (ici avalon_pwm.vhd).

빌 Component E	ditor - avalon_pwm_hw.tcl*					X
File Templates						
Introduction HDL	Files Signals Interfaces Component Wi	zard				
About HDL Fil	es					
					r	
	File Name	Synth	Sim	Тор		
	avalon_pwm.vhd	✓	~	•		
HDL Files:						
	Add Remove	Reanalyz	ze HDL File	s		
	Create HDL Template			_		
Top Level File:	/quartus81/projets_eset/projet1/avalon_pvvr	n_jib/avalo	n_pwm.vh	nd		
Top Level Module:	avalon_pwm					

<u>Remarque 1 :</u> il est préférable de copier le dossier projet « avalon_pwm » dans le dossier projet du système que l'on est en train de créer (bug de quartus ?) sinon, à chaque fois qu'on viendra ouvrir le projet il faudra recommencer la création du circuit avalon_pwm !!

<u>Remarque 2</u>: on peut aussi rajouter manuellement le chemin d'accès du dossier dans les « settings » du projet.

Faire « next ». La fenêtre ci-dessous apparaît avec :

1	1		ELIOCK SETTIDAS			
	<u>u</u>	Component Editor - av	alon_pwm_hw.tcl*			×
	File	e Templates				
ſ	Int	roduction HDL Files Sig	nals Interfaces Component Wizard			
	Þ	About Signals				
		Name	Interface	Signal Type	Width	Direction
	72.	cik	clock_reset	cik	1	input
	\mathbb{Z}	chipselect	avalon_slave_0	chipselect	1	input
	write_n avalon_slave_0		avalon_slave_0	write_n	1	input
	\mathbb{Z}	address avalon_slave_0		address	1	input
	\mathcal{U}	reset_n	clock_reset	reset_n	1	input
	\mathbb{Z}	writedata	avalon_slave_0	writedata	32	input
	77.	readdata	avalon_slave_0	readdata	32	output
	\mathbb{Z}	out_pwm	conduit end 🗖	export 🗾	1	output

- Les signaux du circuit dans la première colonne

- Les signaux du bus Avalon dans la deuxième Si on a respecté la syntaxe des noms des signaux lors de la conception du composant, la correspondance est automatique. Sinon il y a lieu de préciser la correspondance entre les signaux du circuit et ceux du bus Avalon. Pour le signal de sortie « pwm_out » sélectionner « conduit » dans la colonne « interface » puis « export » dans la colonne « signal type » (ceci rajoutera un signal de sortie au niveau du système créé).

Faire « next » puis laisser la configuration proposée par défaut. Cliquer sur « next » de nouveau. On a la boîte de dialogue ci-dessous qui permet d'indiquer dans quel dossier on veut classer le composant créé (ici le groupe « other »).

Eile Templates	t Editor - ava <mark>lon_p</mark> w	m_hw.tcl*				2			
	/ IDL Files Signals Inte	erfaces Compo	nent Wizar	d					
About Com	nonent Wizerd								
Folder:	E:\doc_jlb\fpga\quartu:	s81\projets_esel	\projet1\av	alon_pwm_ji	b				
Class Name:	avalon_pwm								
Display Name:	avalon_pwm								
Version:	1.0		_						
Group:	Other			-					
Description:	circuit pwm		_	_					
Created by:									
loop:			-						
ICON.									
Datasheet URL:	no datasneet								
	Name	Default Value	Editable	Туре	Group	Tooltip			
	(No ton level Verilog (VHD) Instances)								
Parameters:									
	Add Parameter	Remove Par	ameter						
	Preview the GUI	1							

18

Utilisation du SOPC Builder_1 J.L. Boizard

Le circuit apparaît dans le groupe correspondant à « Other».

Intégration du composant au micro contrôleur :

Sélectionner le circuit avalon_pwm et faire « add ». La suite est la même que pour les autres composants. Attention !!!! Ne pas oublier de faire «assign base address» .

Faire « generate » pour créer le fichier VHDL du micro contrôleur complet puis fermer le SOPC Builder. Enfin, compiler le projet complet pour obtenir un fichier .sof qui sera téléchargé dans le FPGA.

A CE STADE LA PARTIE HARDWARE EST TERMINEE !!!!!

Exemple de programme utilisant le circuit avalon_pwm :

```
#define freq (unsigned int *) AVALON_PWM_0_BASE
#define duty (unsigned int *) (AVALON_PWM_0_BASE + 4)
int main()
{
    ....
    *freq = 0x0400; // divise clk par 1024
    *duty = 0x0200; // RC = 50%
.....
```

Pour la mise en œuvre des interruptions, se référer au document « Mise en œuvre du sopc builder ».

http://jeanlouis.boizard.free.fr/m1_isme/master_1/sopc_builder.pdf

Fin du document