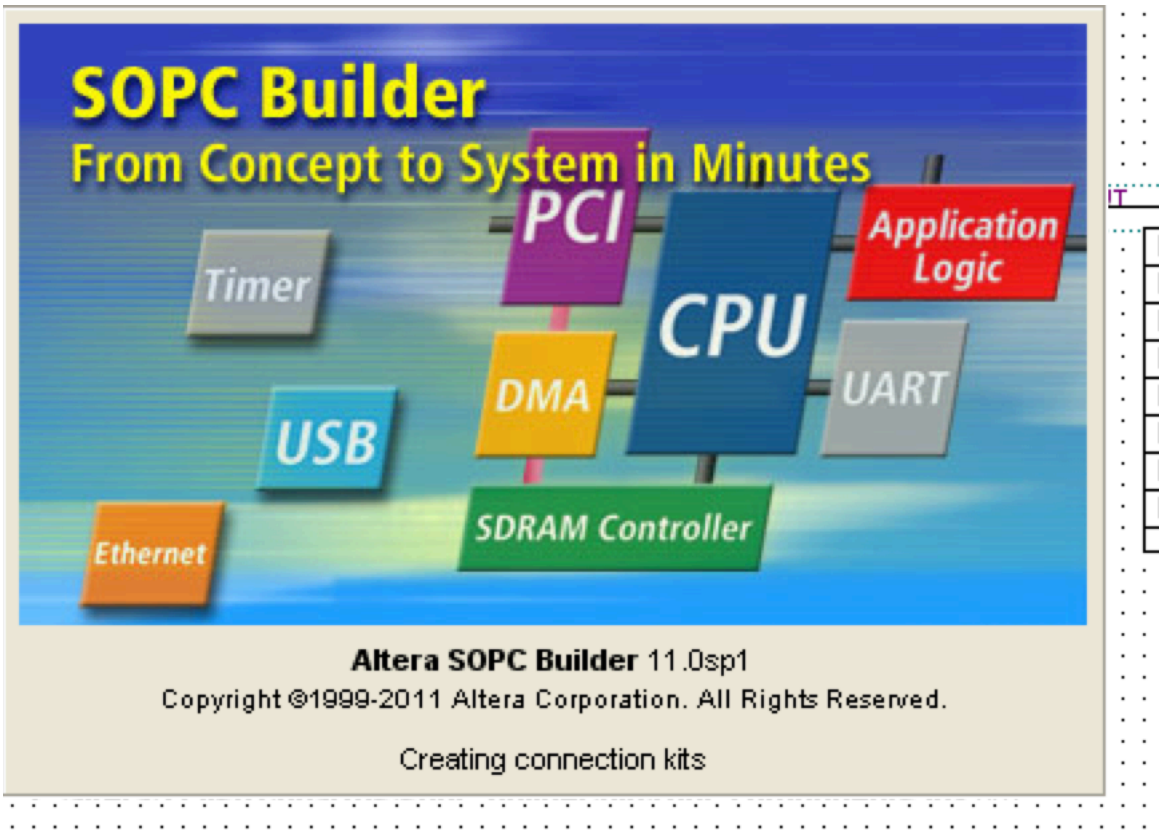
Conception des SOC/SOPC

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 »

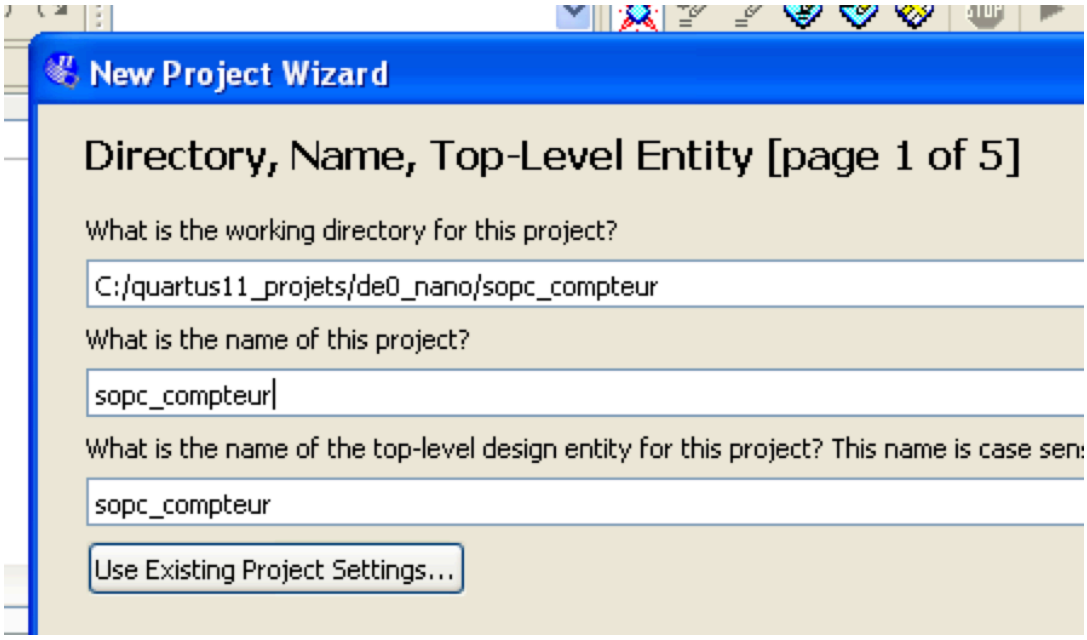
Pour illustrer la démarche on propose de créer un SOPC disposant de deux entrées et huit sorties logiques. Le logiciel d'application, développé en langage C, effectue périodiquement une incrémentation d'une variable de type octet et l'affiche sur huit leds de sortie. Les deux entrées (boutons poussoirs 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**.

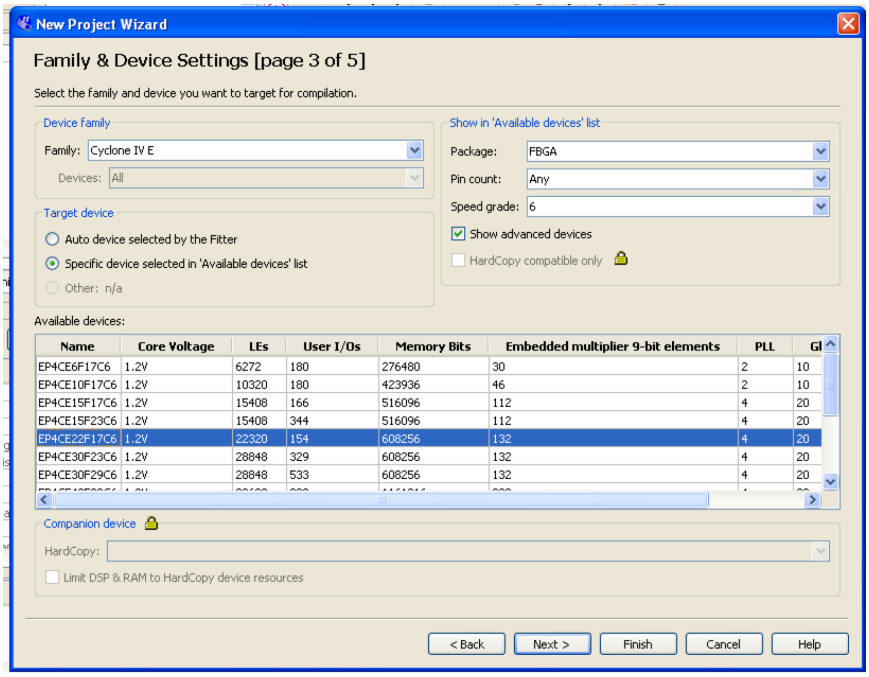
Environnement de travail :

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

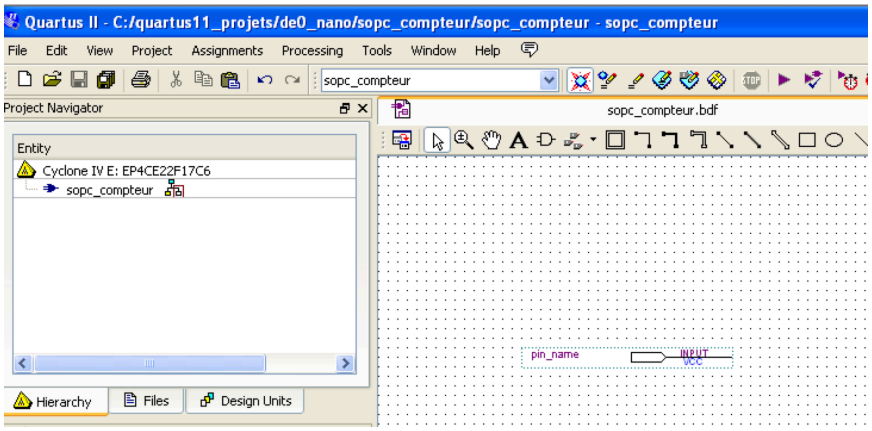
***Nota :* à partir de la version 11 de Quartus, un nouvel outil de conception de SOPC nommé QSYS a été intégré et remplace le SOPC Builder. Toutefois dans ce qui suit, nous continuerons à utiliser le SOPC Builder.**

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.

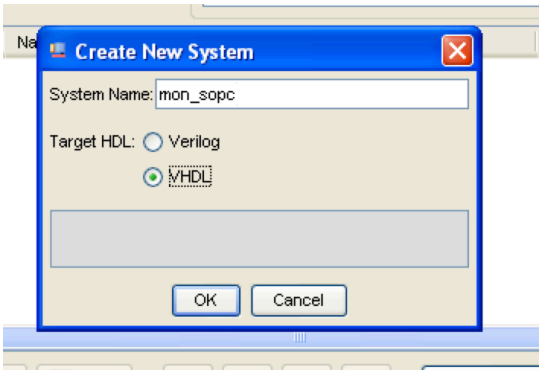




Ouvrir une feuille de schéma (New => block diagram/schematic file). Y placer une broche d'entrée ou de sortie puis enregistrer le fichier avec le **même nom** que le projet (ici sopc_compteur) **en faisant attention de bien l'enregistrer dans votre répertoire !!!**

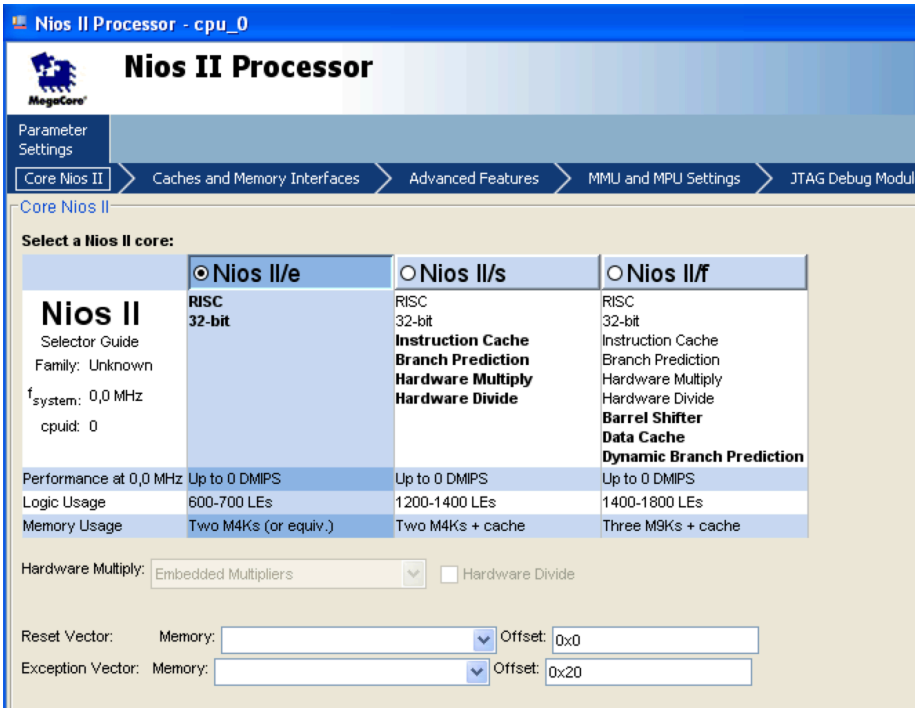


Dans l'environnement Quartus 11.0, lancer le SOPC Builder (Tools=> SOPC Builder). Dans la fenêtre « Create new system », donner un nom au SOPC que l'on va concevoir (ici mon_sopc) puis sélectionner « VHDL » (ceci aura pour effet de générer des fichiers de description du sopc en langage vhdl).

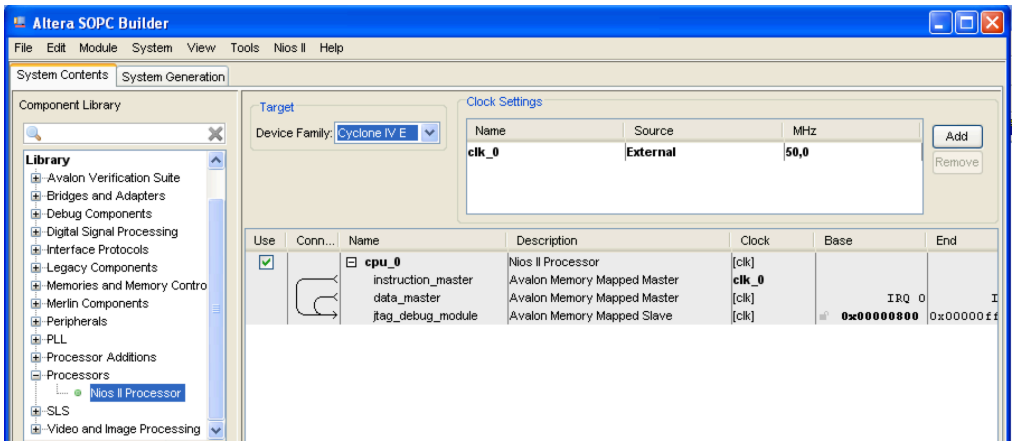


Vérifier que l'horloge du processeur du sopc est bien de 50 MHz.

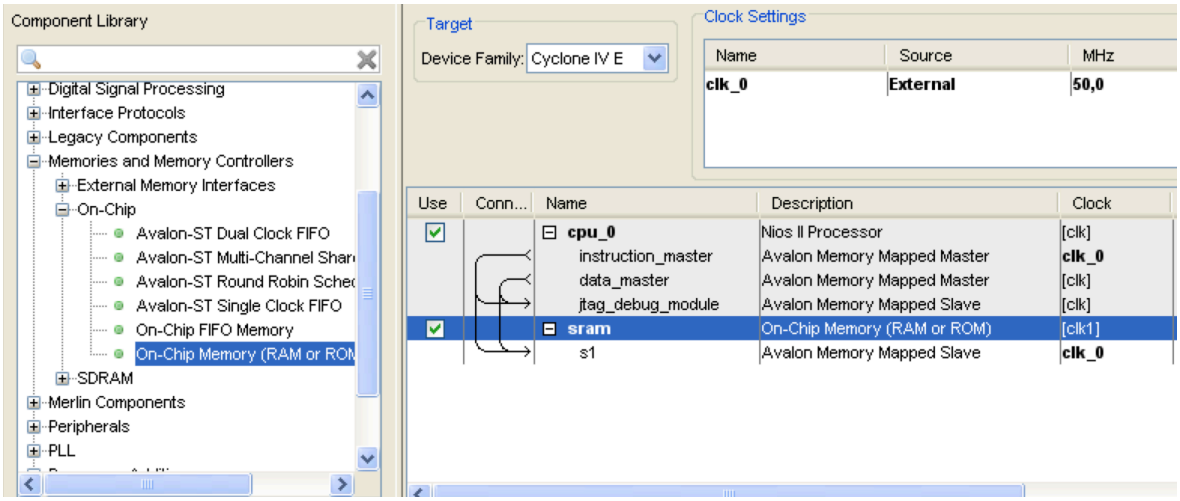
Dans la colonne de gauche, sélectionner « Processor » puis « Nios II processor » et faire « add ». Sélectionner la version économique parmi les trois proposées. Faire « next » et accepter les options par défaut (sélectionner le debugger niveau 1).



On obtient çà :



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



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

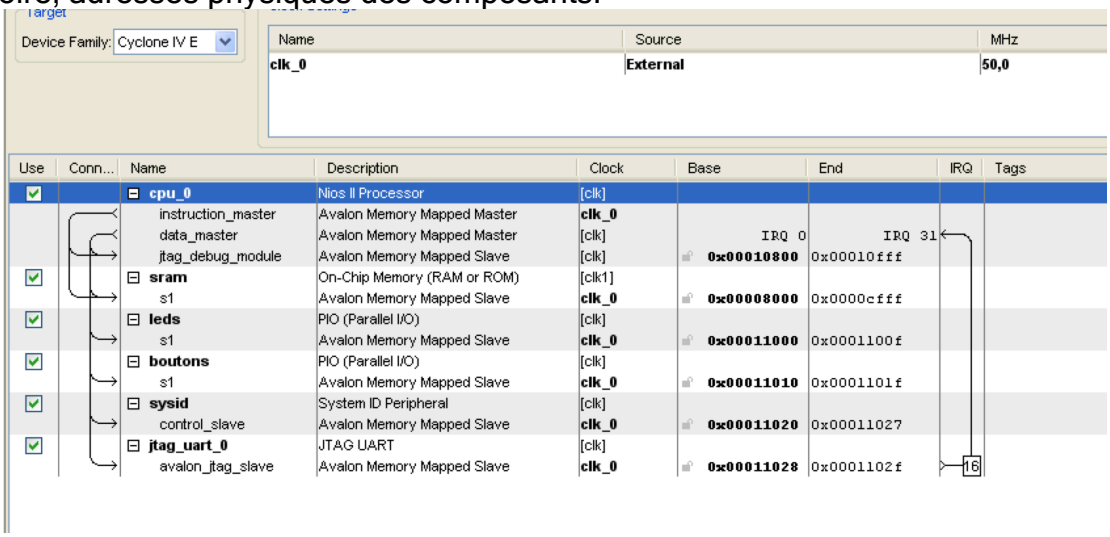
Nota : on peut les renommer en faisant un clic droit sur le composant.

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.

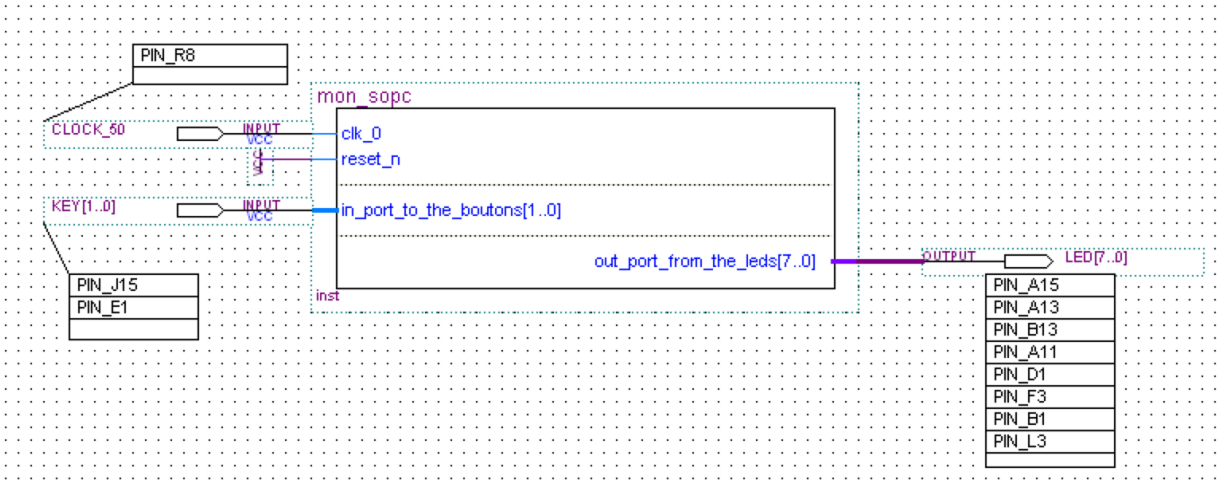
Revenir sur le processeur en double-cliquant dessus puis sélectionner « on-chip ram » dans les zones « reset vector » et « exception vector ».

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.



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):

Named: * Edit:

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)	
LED[6]	Output	PIN_B1	1	B1_N0	3.3-V LVTTL		8mA (default)	2 (default)	
LED[5]	Output	PIN_F3	1	B1_N0	3.3-V LVTTL		8mA (default)	2 (default)	
LED[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>>									

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.

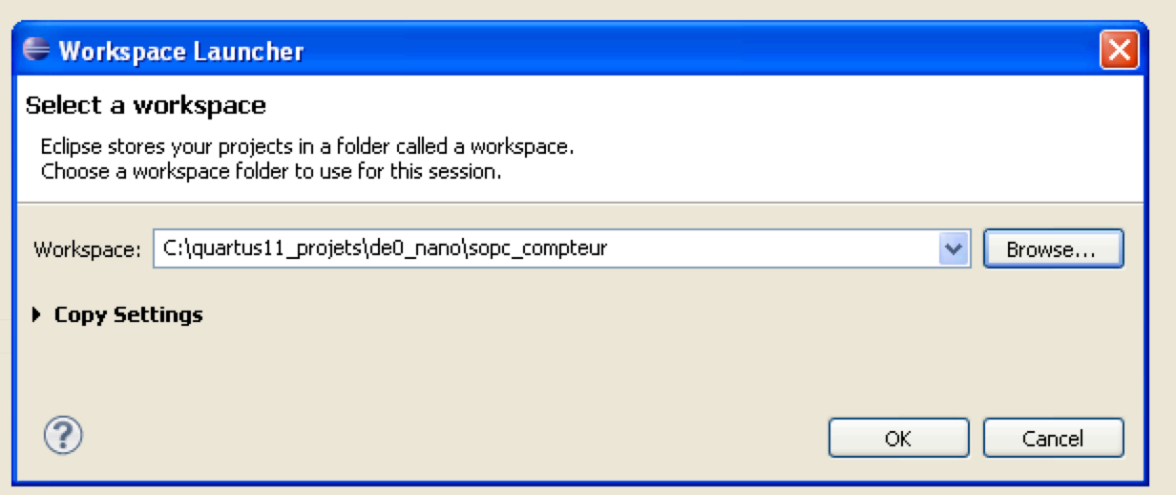
Partie 2 : Développement du logiciel

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

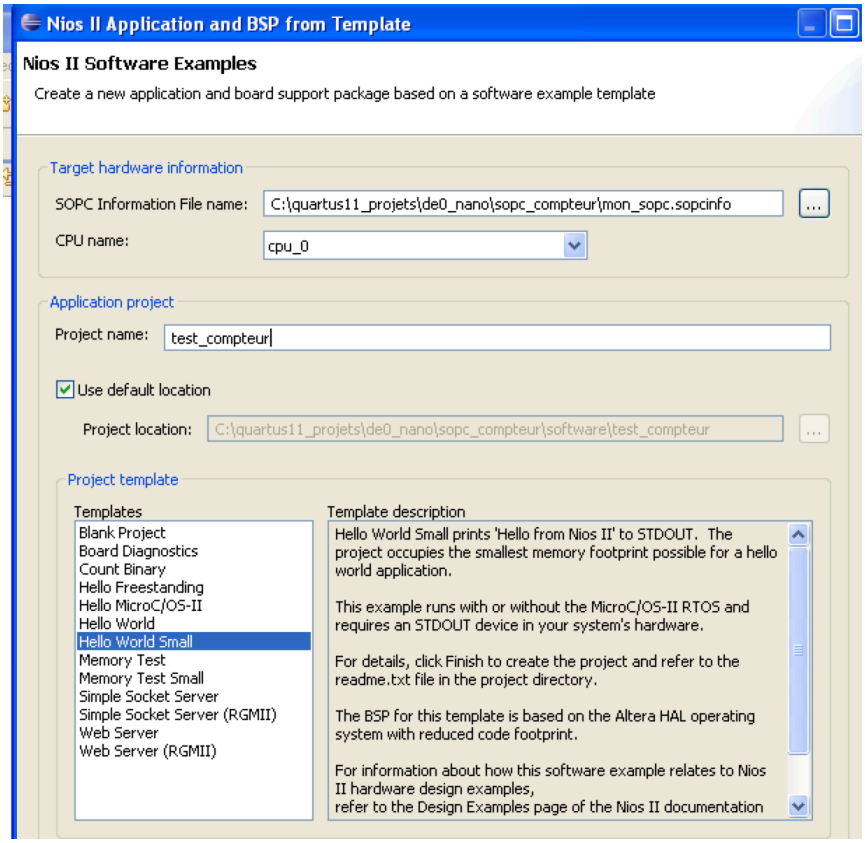
2.1) Définition de l'espace de travail (Workspace)

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.



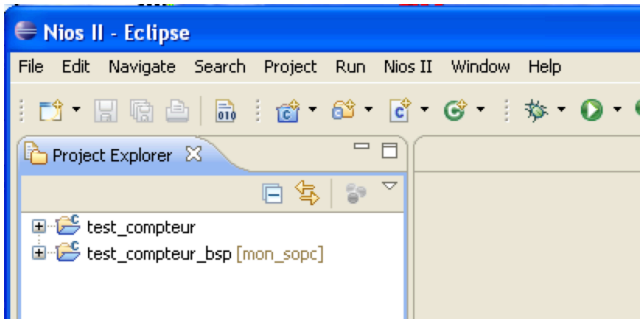
2.2) Création du projet

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.



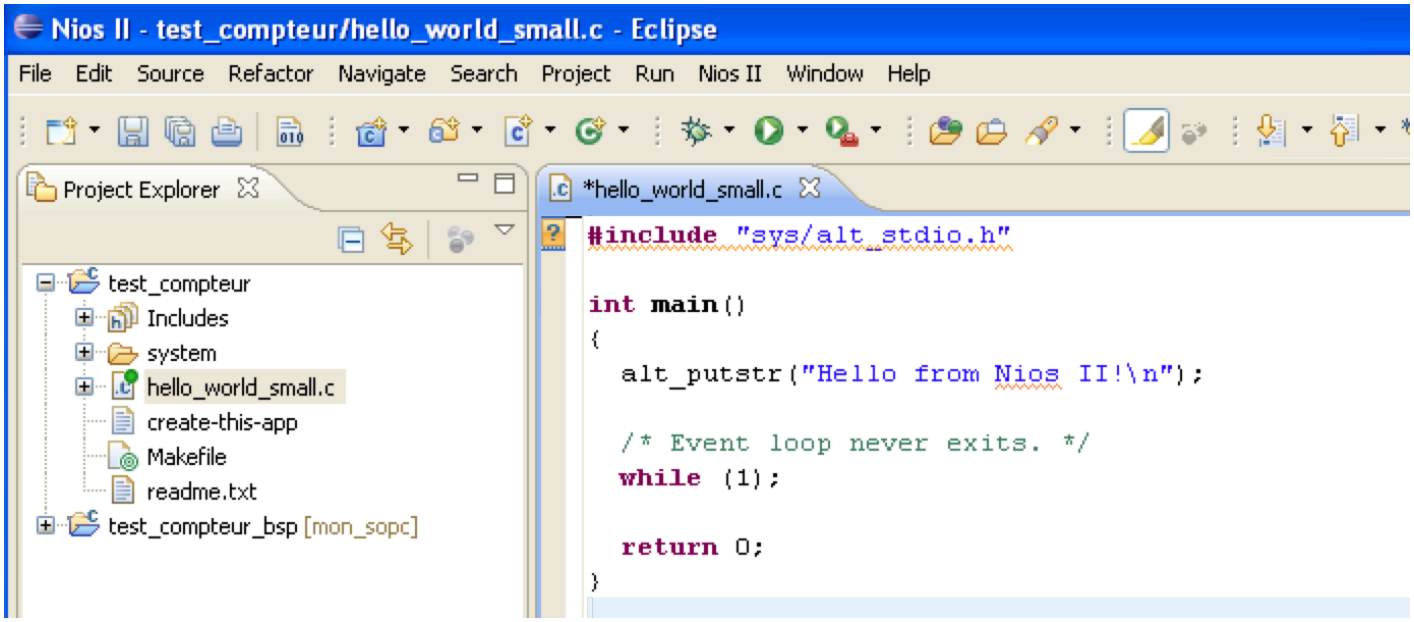
2.3) Création de la bibliothèque

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.



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 :

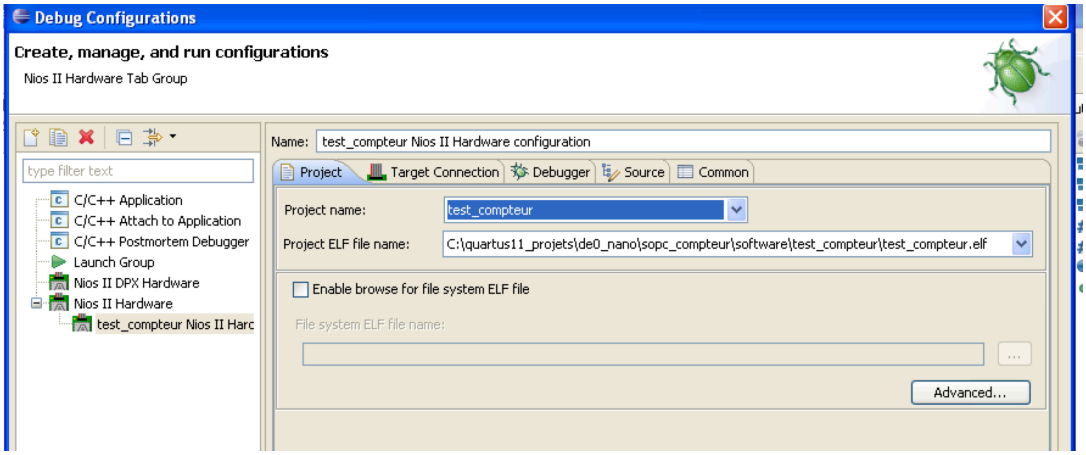
- Run as : pour télécharger et exécuter le programme immédiatement.
- Debug as : pour télécharger et exécuter le programme en mode debug.

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.

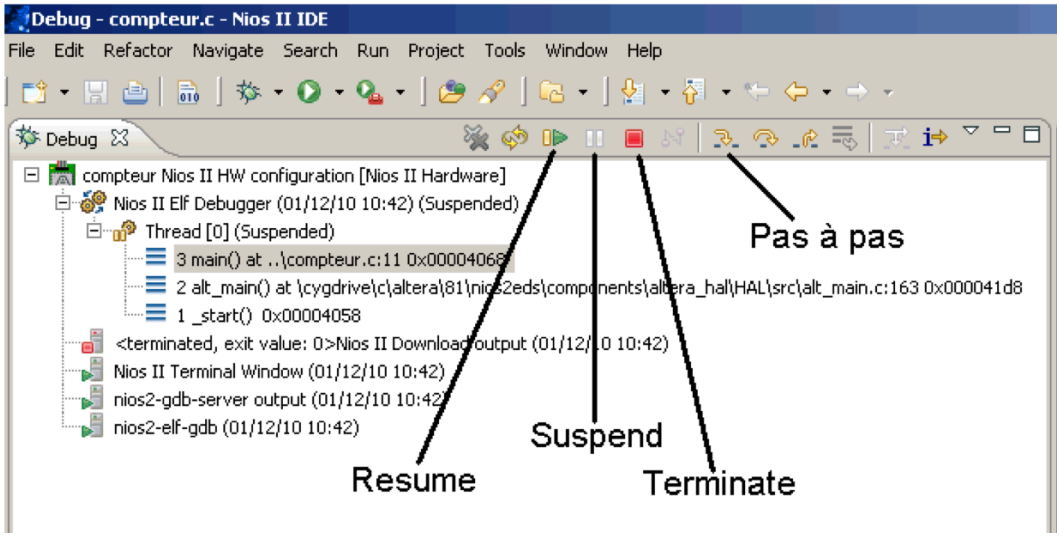
Commande « debug as » :

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 ».



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.

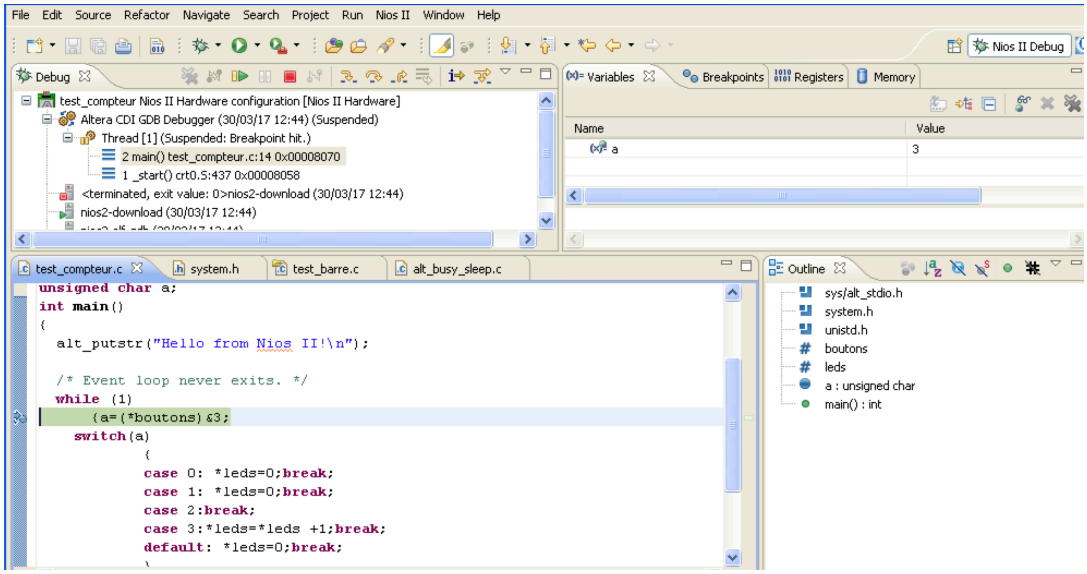


Un double-clic dans la colonne de gauche de la fenêtre du programme insère un point d'arrêt. Un autre double-clic l'enlève.

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 !!!!

Partie 3 : Intégration d'un composant propriétaire

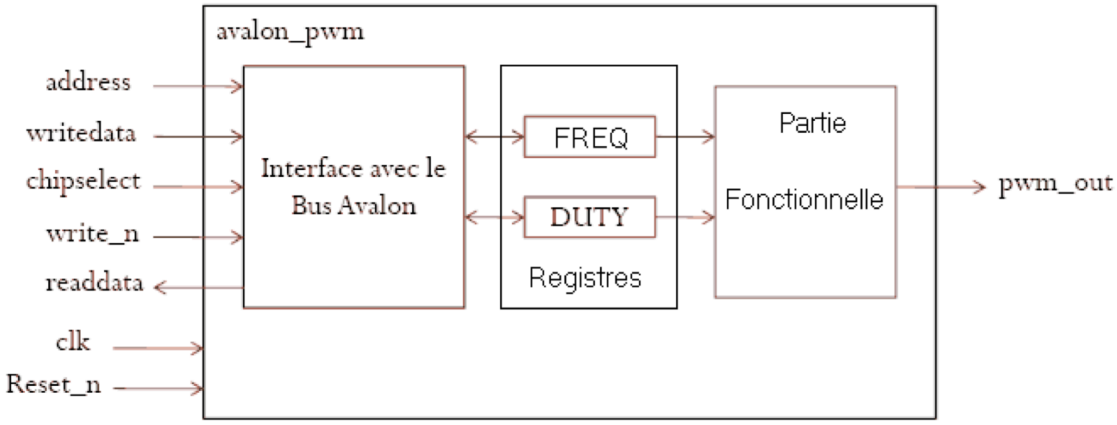
Lors de la création du système (le micro contrôleur), il se peut que certaines ressources ne soient pas parmi celles proposées par le constructeur. Il devient nécessaire de :

- 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

--00000000000000000000000000000000000000000000000000000000
-- 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 <= (others => '0');
        elsif clk'event and clk = '1' then
                if counter >= freq then
                counter <= (others => '0');
                else
                counter <= 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 <= '0';
```

```
        elsif clk'event and clk = '1' then
                if counter >= duty then
                pwm_nano_on <= '0';
                else
                pwm_nano_on <= '1';
                end if;
        end if;
end process compare;
--*******************************************************
--00000000000000000000000000000000000000000000000000000000
out_pwm_nano <=    pwm_nano_on;
--*************************************************************
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 :

```
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
```

```
        counter <= (others => '0');
        elsif clk'event and clk = '1' then
                if control(1) ='1' then
                        if counter >= freq then
                        counter <= (others => '0');
                        else
                        counter <= counter + 1;
                        end if;
                end if;
        end if;
end process divide;

compare: process (clk, reset_n)
        begin
        if control(0) = '0' then
        pwm_on <= '1';
        elsif clk'event and clk = '1' then
                if counter >= duty then
                pwm_on <= '0';
                elsif counter = 0 then
                pwm_on <= '1';
                end if;
        end if;
end process compare;
out_pwm <= pwm_on and control(0);
```

--interface avalon

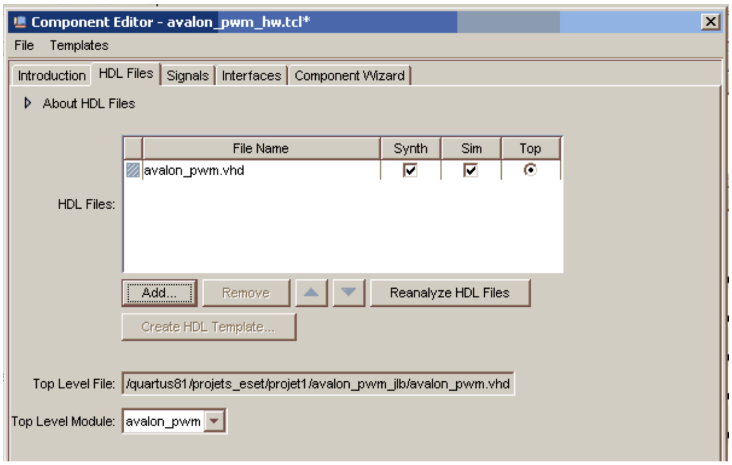
```
-- écriture registres
process_write: process (clk, reset_n)
        begin
        if reset_n = '0' then
        freq <= (others => '0');
        duty <= (others => '0');
        control <= (others => '0');
        elsif clk'event and clk = '1' then
                if chipselect ='1' and write_n = '0' then
                        if address = "00" then
                        freq  <= 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;
```

```
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 <= 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

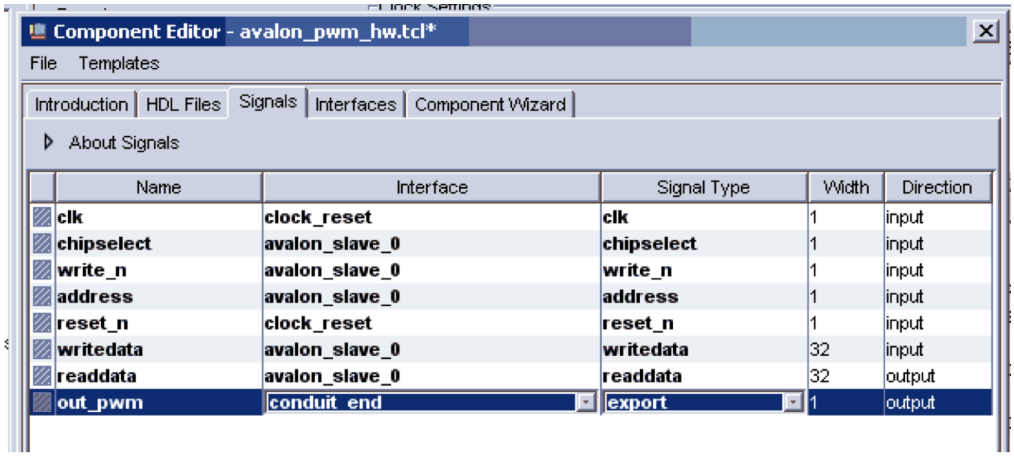
On commence par créer son micro contrôleur comme vu dans le début du document puis dans la fenêtre du SOPC Builder cliquer sur « new » en bas à gauche. Dans la fenêtre qui se présente faire « next » puis « add » pour aller chercher le circuit que l'on vient de décrire (ici avalon_pwm.vhd).



Remarque 1 : 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 !!

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

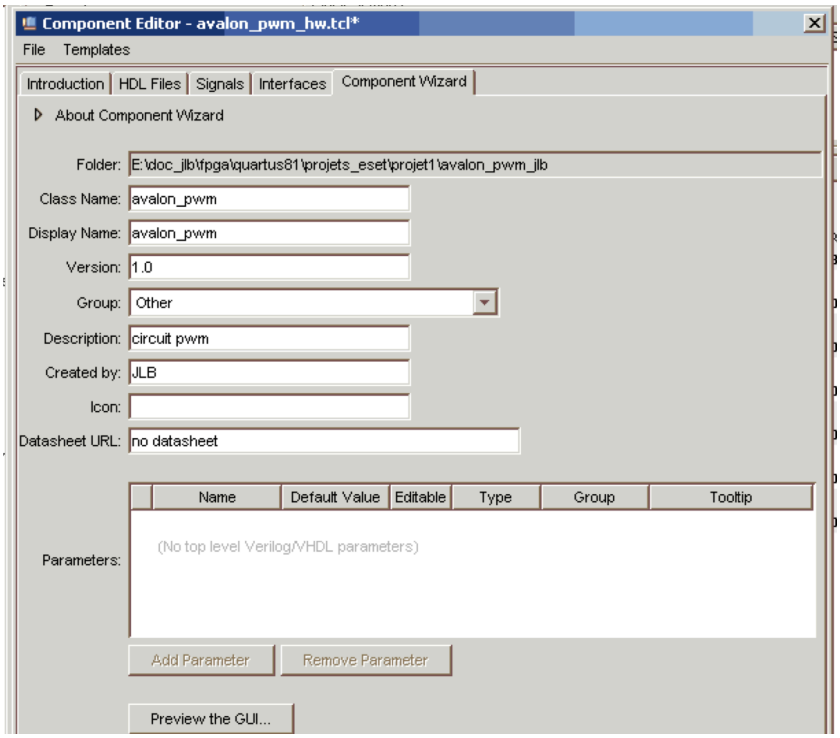
Faire « next ». La fenêtre ci-dessous apparaît avec :



- 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 »).



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