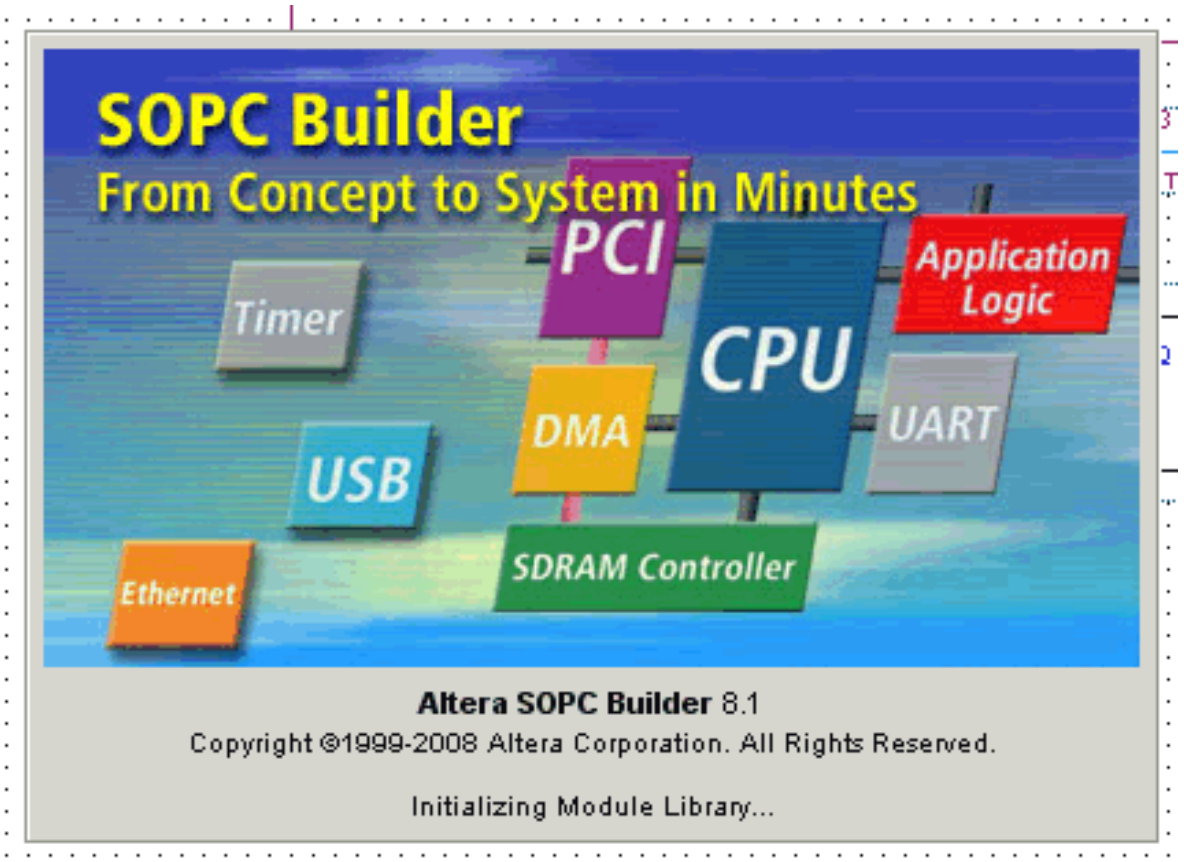
Utilisation du SOPC builder pour la conception de SOPC

(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 NIOS2 de ALTERA, 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 :

- Création du composant matériel (processeur + périphériques) dans l'environnement Quartus II
- Téléchargement dans le composant FPGA (configuration) (environnement Quartus II)
- Création du logiciel dans l'environnement NIOS2IDE et téléchargement dans le FPGA.

Exemple : projet « compteur »

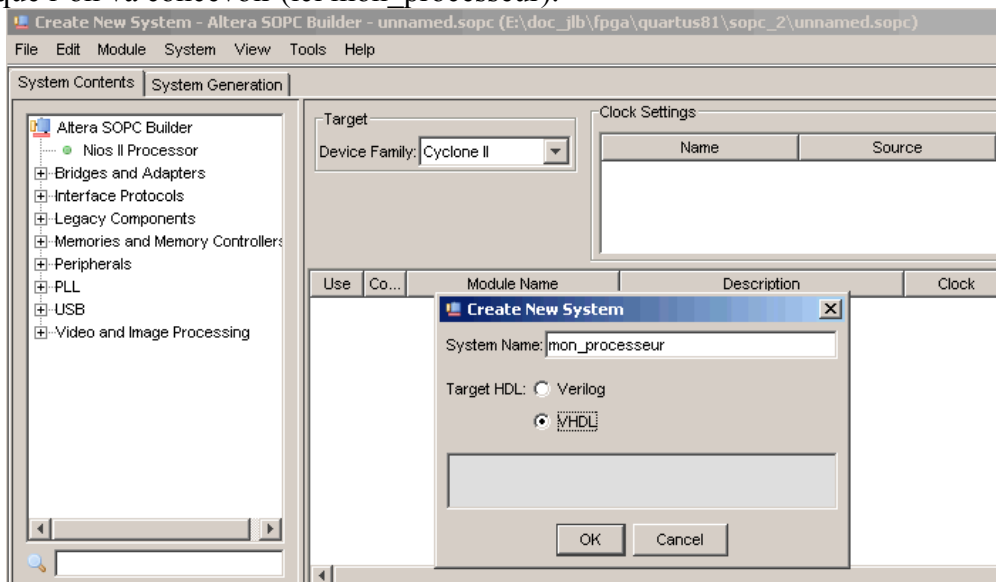
Pour illustrer la démarche on propose de créer un microcontrôleur disposant de huit entrées et huit sorties logiques. Le logiciel, développé en langage C, effectue une incrémentation et un affichage sur des leds d'une

variable via un port de sortie. Dans cette application la mémoire utilisée est intégrée au composant FPGA : il s'agira donc de RAM/ROM **On Chip**.

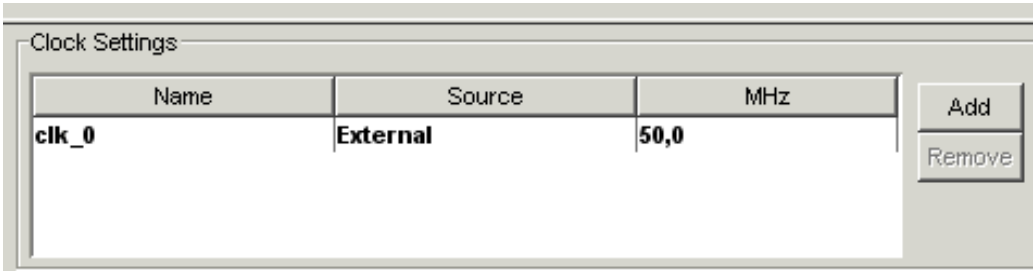
Environnement de travail :

PC avec Windows XP service Pack 2.
Quartus II version 8.1
Nios 2 IDE version 8.1

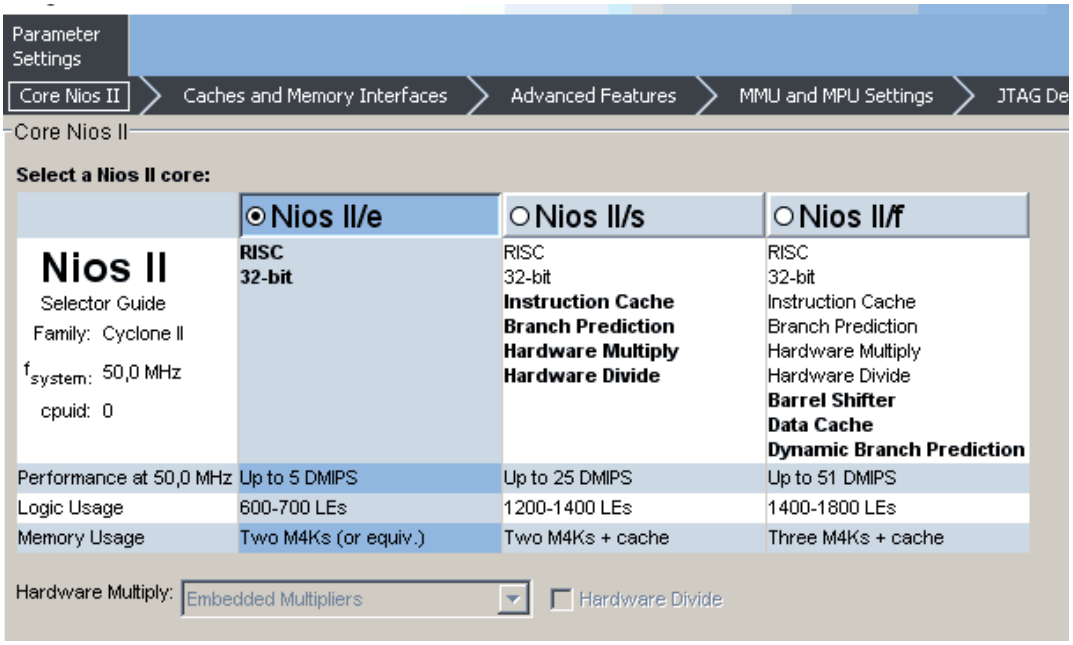
Démarrer Quartus II et créer un projet en spécifiant le répertoire de travail, le nom du projet (SOPC2 par exemple) et le composant FPGA utilisé : ici on utilise une carte Terasic DE1 équipée d'un FPGA cyclone 2 EP2C20F484C7.
Dans l'environnement Quartus lancer le SOPC Builder. Dans la fenêtre « Create new system », donner un nom au SOPC que l'on va concevoir (ici mon_processeur).



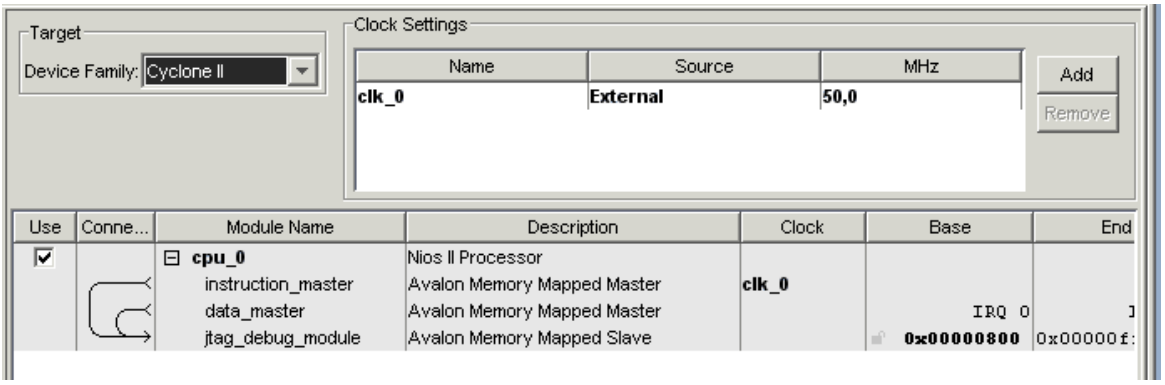
Définir l'horloge du processeur : l'horloge de la carte DE1 est de 50 MHz.



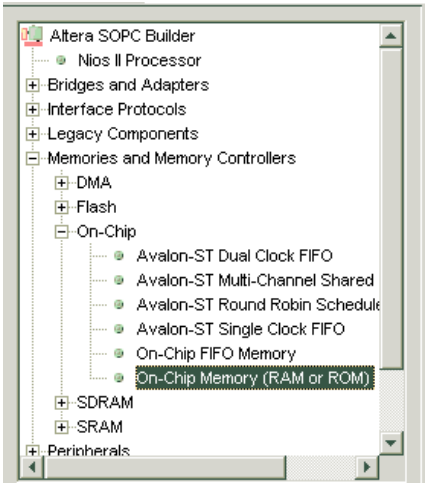
Dans la colonne de gauche, sélectionner « Nios II processor » et faire « add » puis sélectionner la version économique parmi les trois proposées. Faire « next » et accepter les options par défaut (sélectionner le debugger niveau 1).

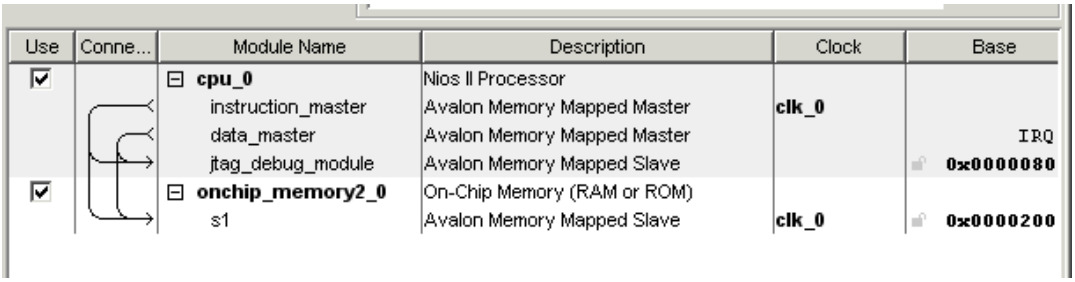


On obtient çà :

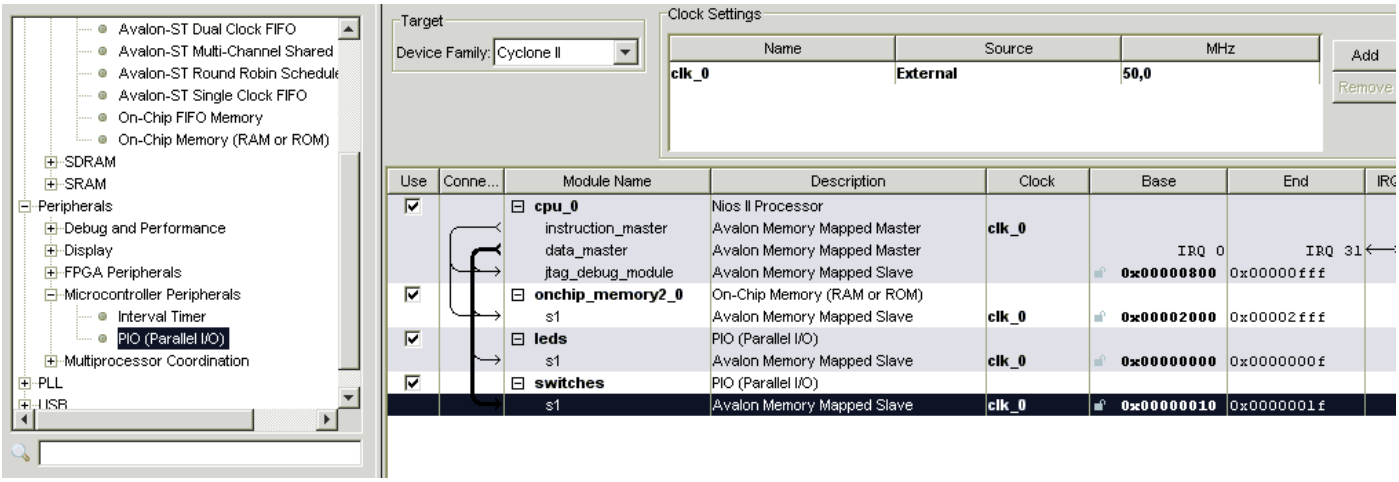


Dans la colonne de gauche, sélectionner la mémoire puis faire « add » et accepter la configuration proposée:

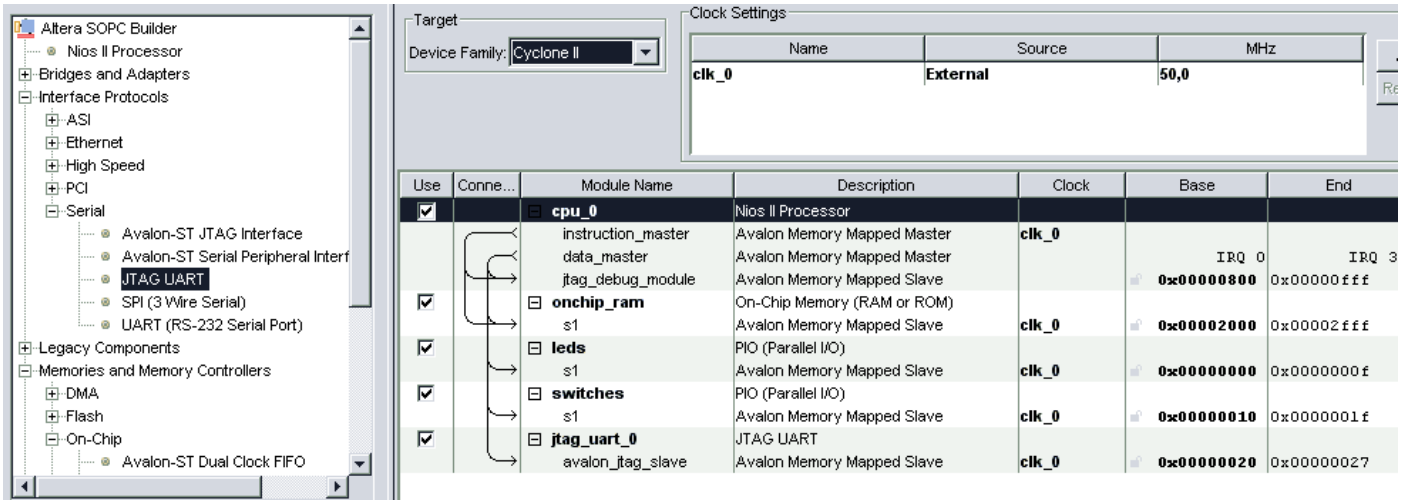




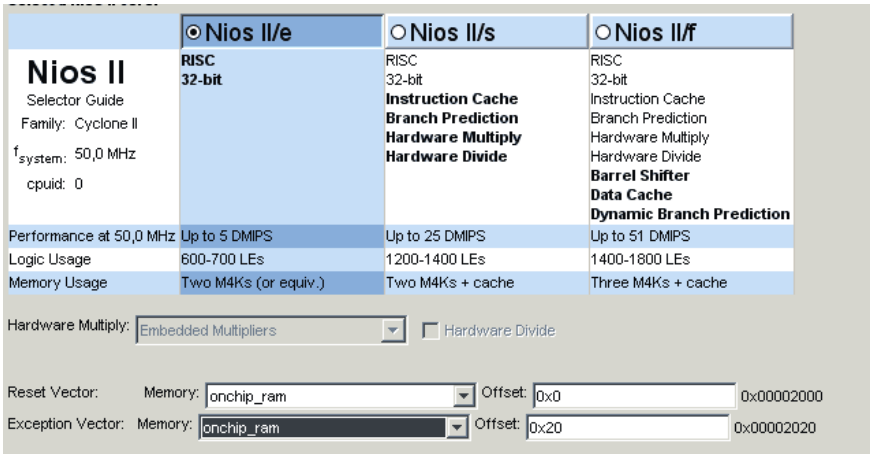
De la même manière, ajouter deux PIO de huit bits chacun (Parallel Inputs Outputs) : un en entrée et un en sortie. Nota : on peut les renommer en faisant un clic droit sur le composant.



Rajouter le composant JTAG UART qui permettra de communiquer avec le PC hôte et télécharger le logiciel dans le circuit.

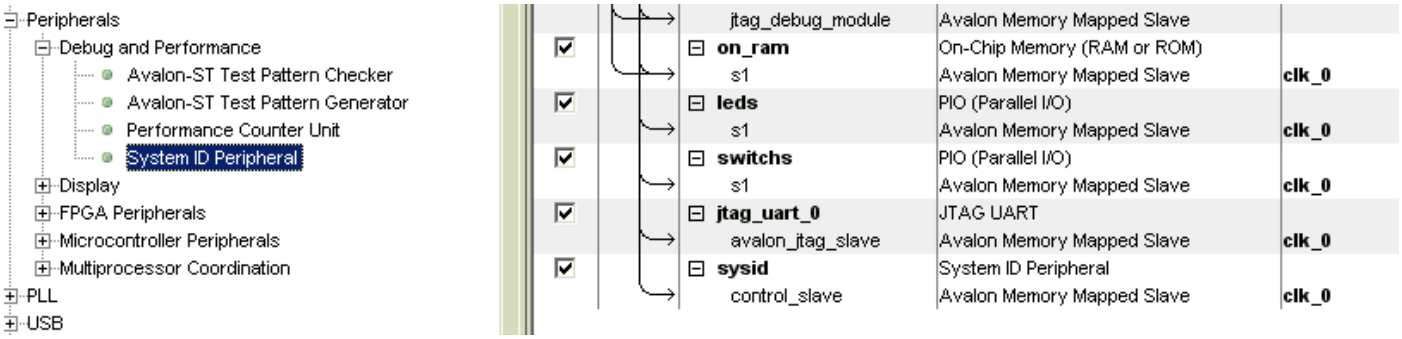


Revenir sur le processeur en double-cliquant dessus puis sélectionner « on-chip ram » dans les zones « reset vector » et « exception vector ».

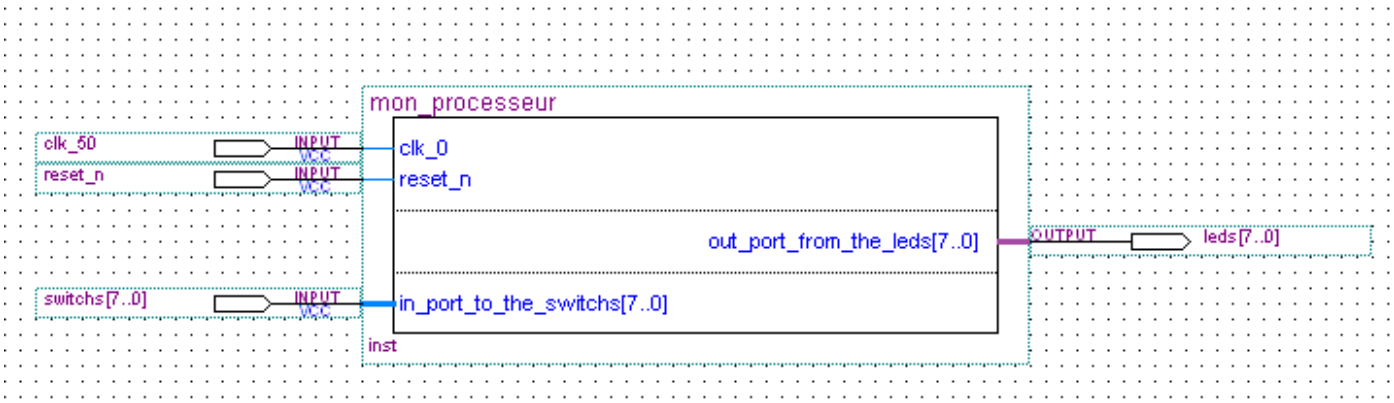


Faire ensuite « System » puis « auto-assign base adress ». Ceci a pour effet d'assigner automatiquement une adresse logique à chaque ressource dans l'espace mémoire adressé par le processeur. A ce stade le SOPC est défini : type de processeur, horloge, périphériques utilisés, taille mémoire, adresses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

Pour des raisons de sécurité on peut rajouter un composant « sysid » : 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.



Une fois tous les composants rajoutés, 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 une fenêtre graphique 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 et affecter les broches.



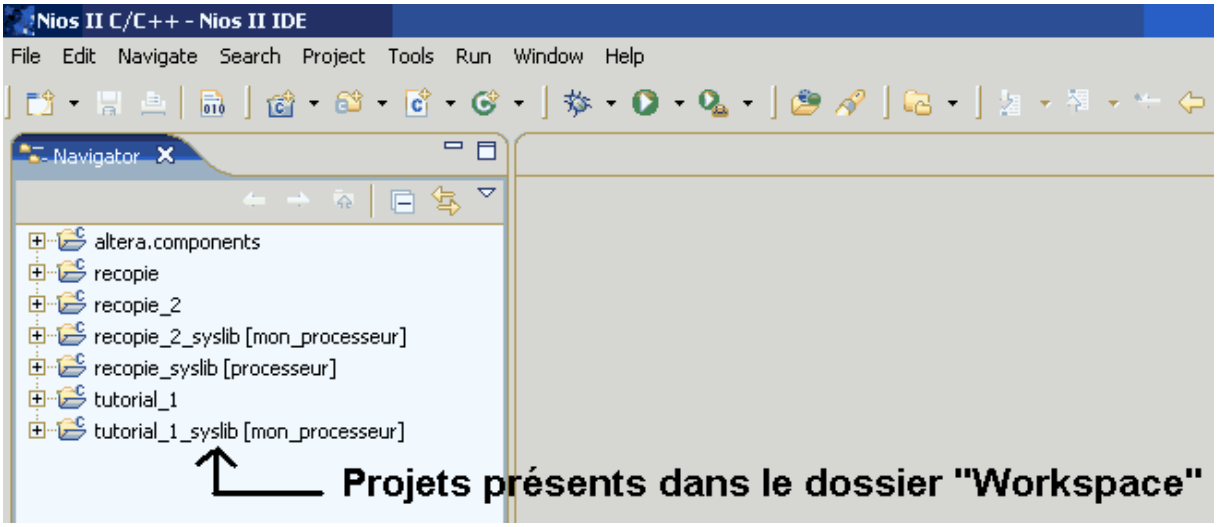
Compiler le projet : à ce stade on a un fichier de programmation SOPC2.sof qui permet de configurer le FPGA.

Développement du logiciel

On peut lancer NIOS2IDE 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

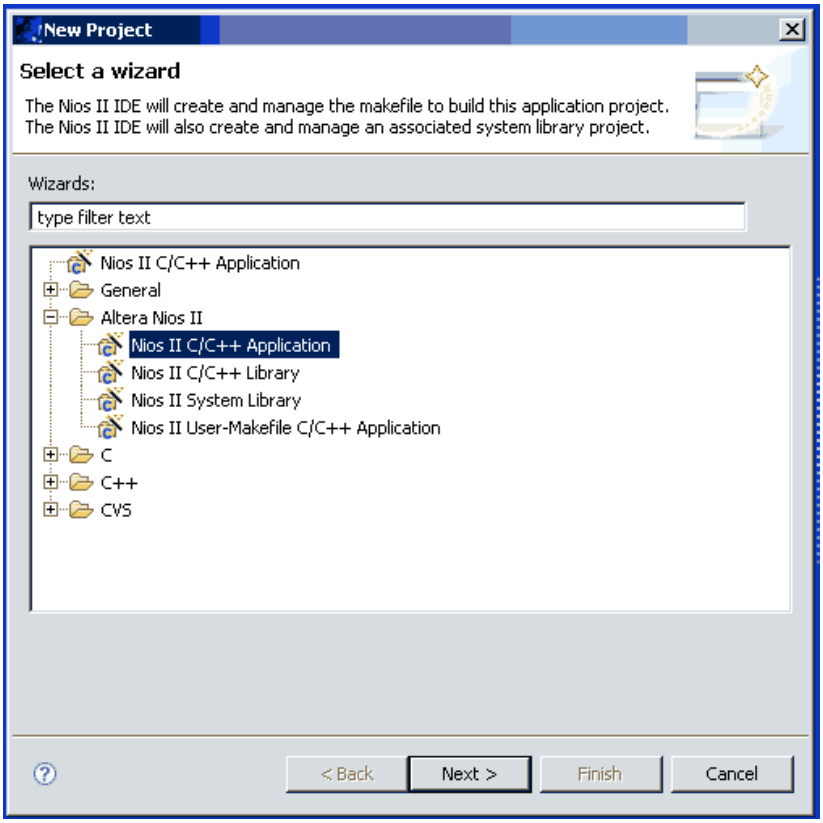
1) Définition de l'espace de travail (Workspace)

Au lancement de NIOS2IDE, un espace de travail est sélectionné par défaut. 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.

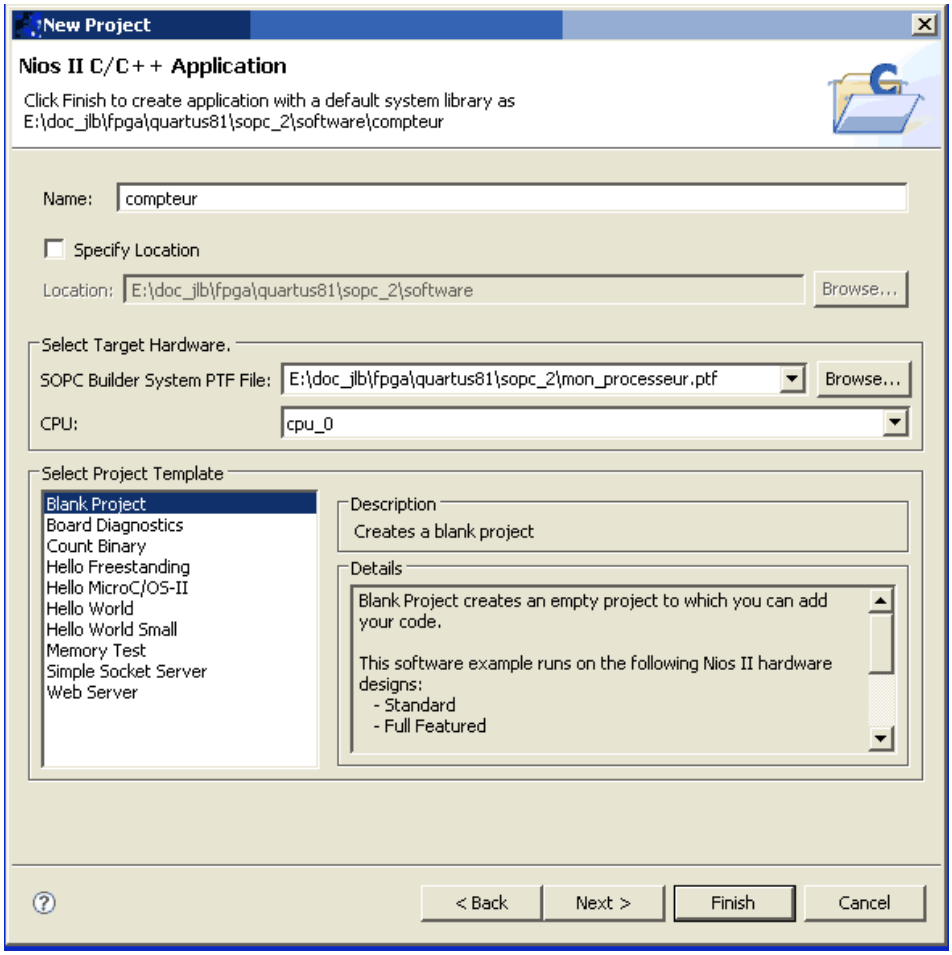


2) Création du projet

Faire File => New project, cliquer sur **NIOS II C/C++ application** ,faire « next »

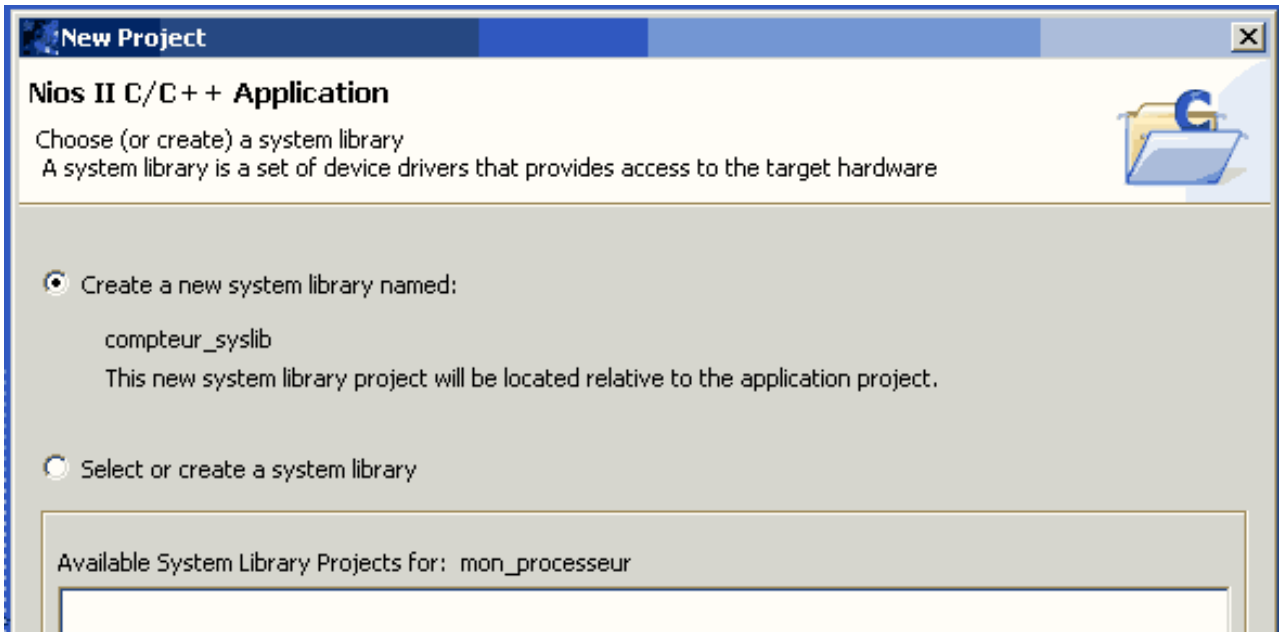


puis sélectionner un projet blanc et lui donner un nom (ex. : compteur). Dans la rubrique « select target hardware» sélectionner le système sur lequel s'exécutera le logiciel (ici c'est le système **mon_processeur** créé précédemment) ;

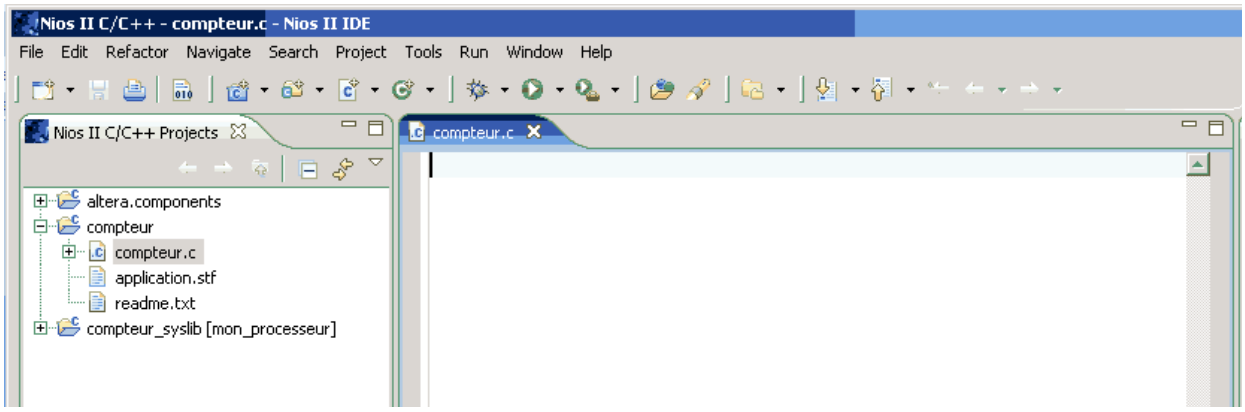


3) Création de la bibliothèque

Faire next et sélectionner « create a new system library » puis “finish”.

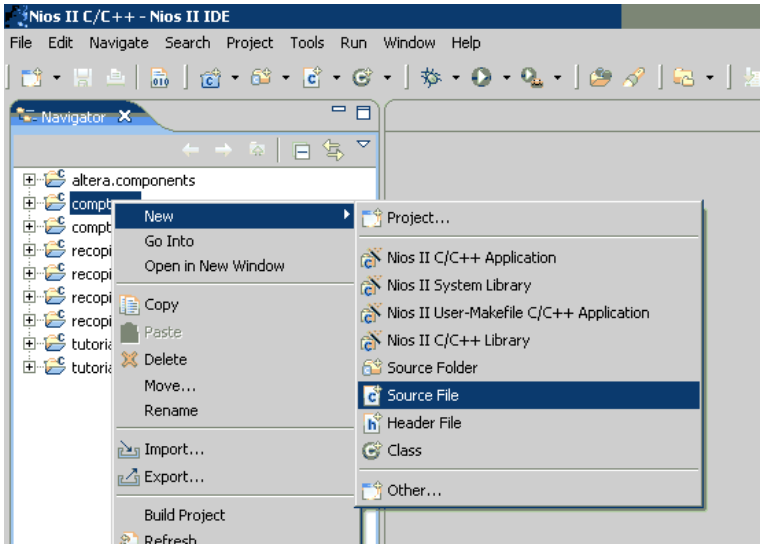


A ce stade le dossier « compteur » apparaît dans le navigateur avec le dossier « compteur_syslib ».

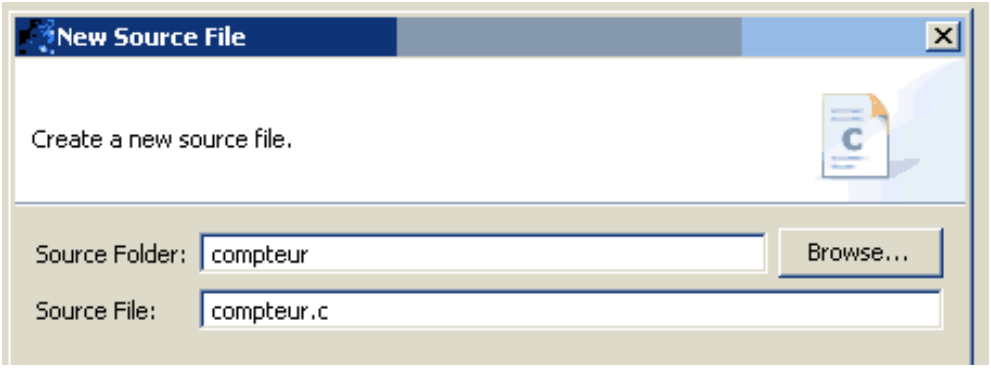


4) Création du programme

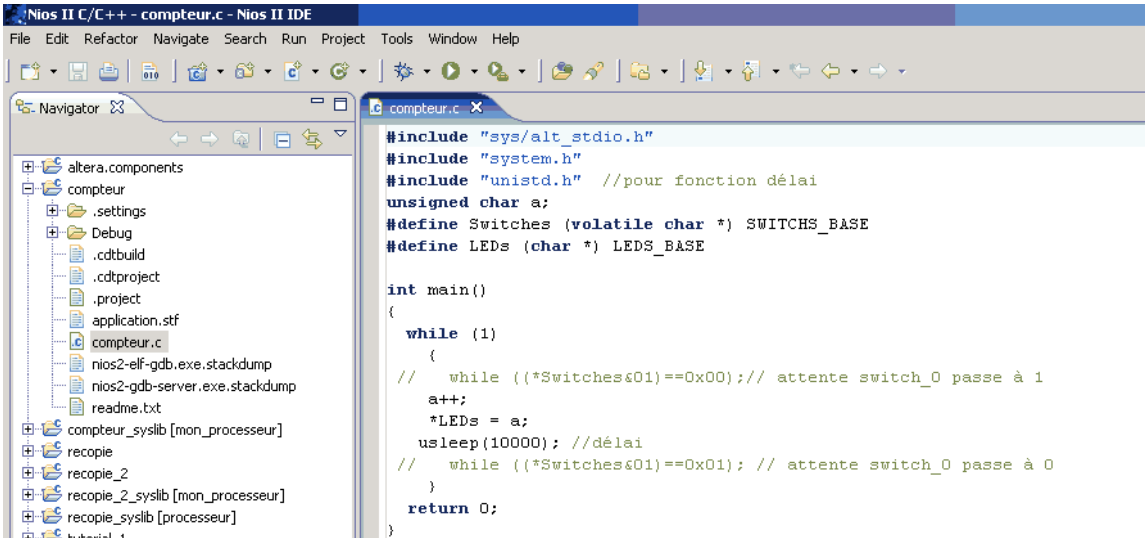
Faire un clic droit sur « compteur » et sélectionner **New=> source file** (qui correspond au programme en c à créer).



Donner un nom au programme avec l'extension **.c (**ici compteur.c).

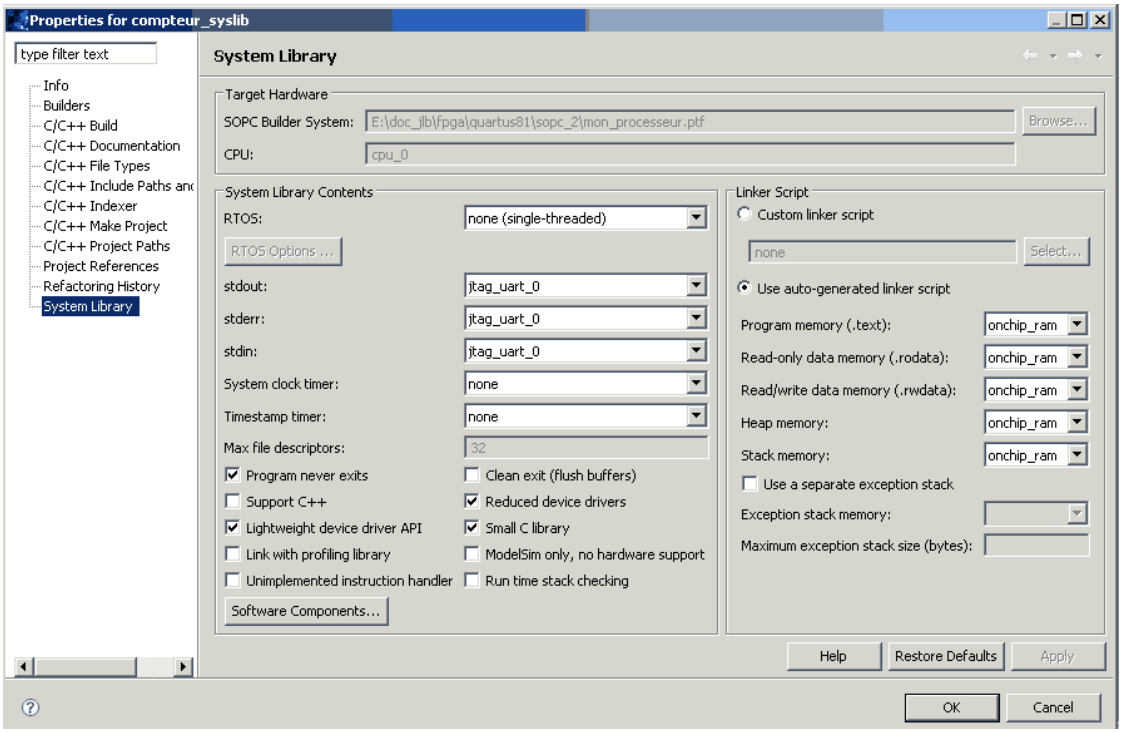


Cliquer sur **finish**. Une fenêtre s'ouvre dans laquelle il n'y a plus qu'à taper le programme d'application.



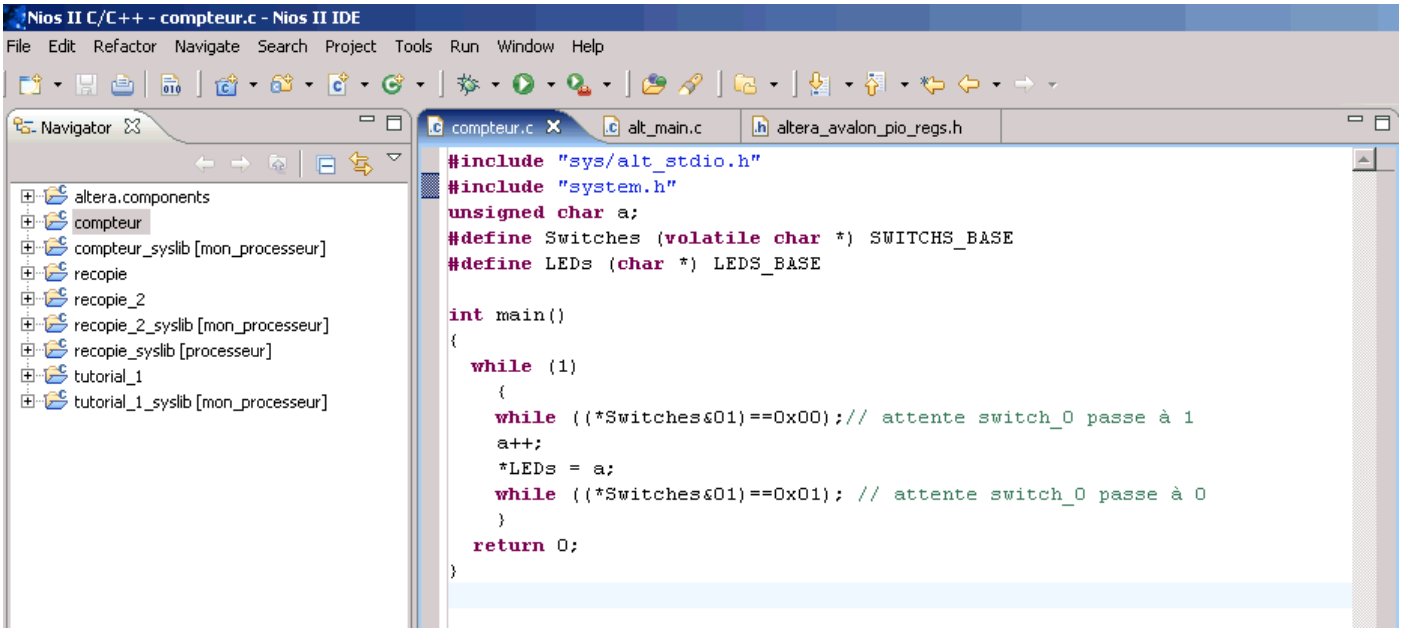
Remarque : l'adjonction de **#include « system.h »** permet d'éviter de renseigner les adresses physiques des périphériques du système qui a été créé.

Configuration de la librairie « compteur » : faire un clic droit sur « compteur_syslib[mon_processeur] » et sélectionner « properties ». Dans la boîte de dialogue sélectionner « system library » et renseigner la boîte comme ci-dessous.



5) Compilation

Faire un clic droit sur le projet « compteur » puis sélectionner « build project ».

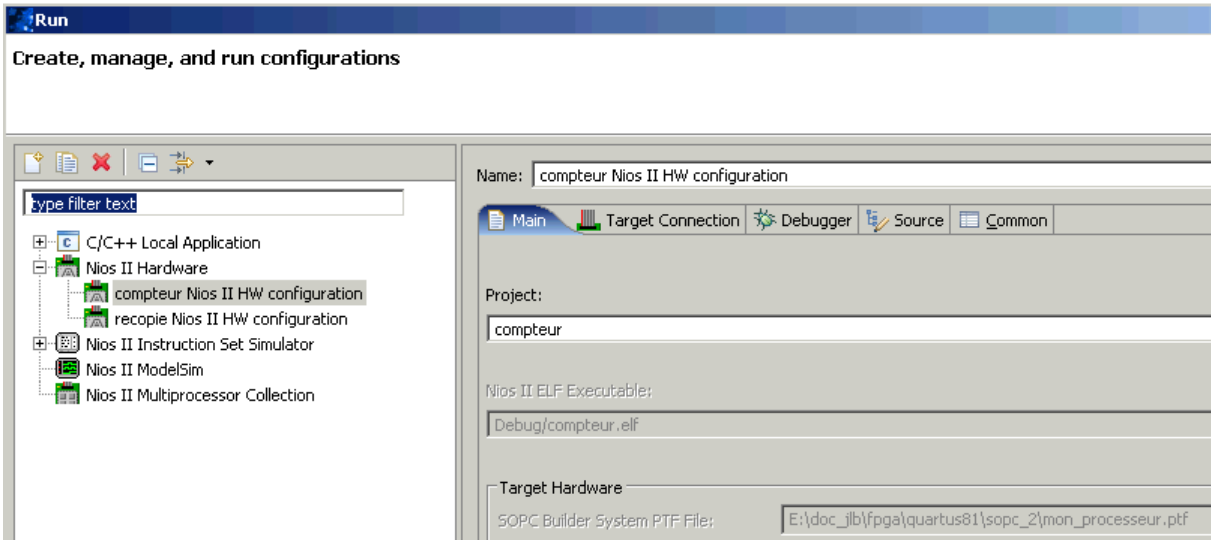


6) Téléchargement du programme et exécution sur la cible (NIOS II HW configuration)

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** » : la boîte de dialogue ci-dessous s'ouvre. Sélectionner **Nios II Hardware**, renseigner les rubriques « Name » et « Project » puis faire « Run ».

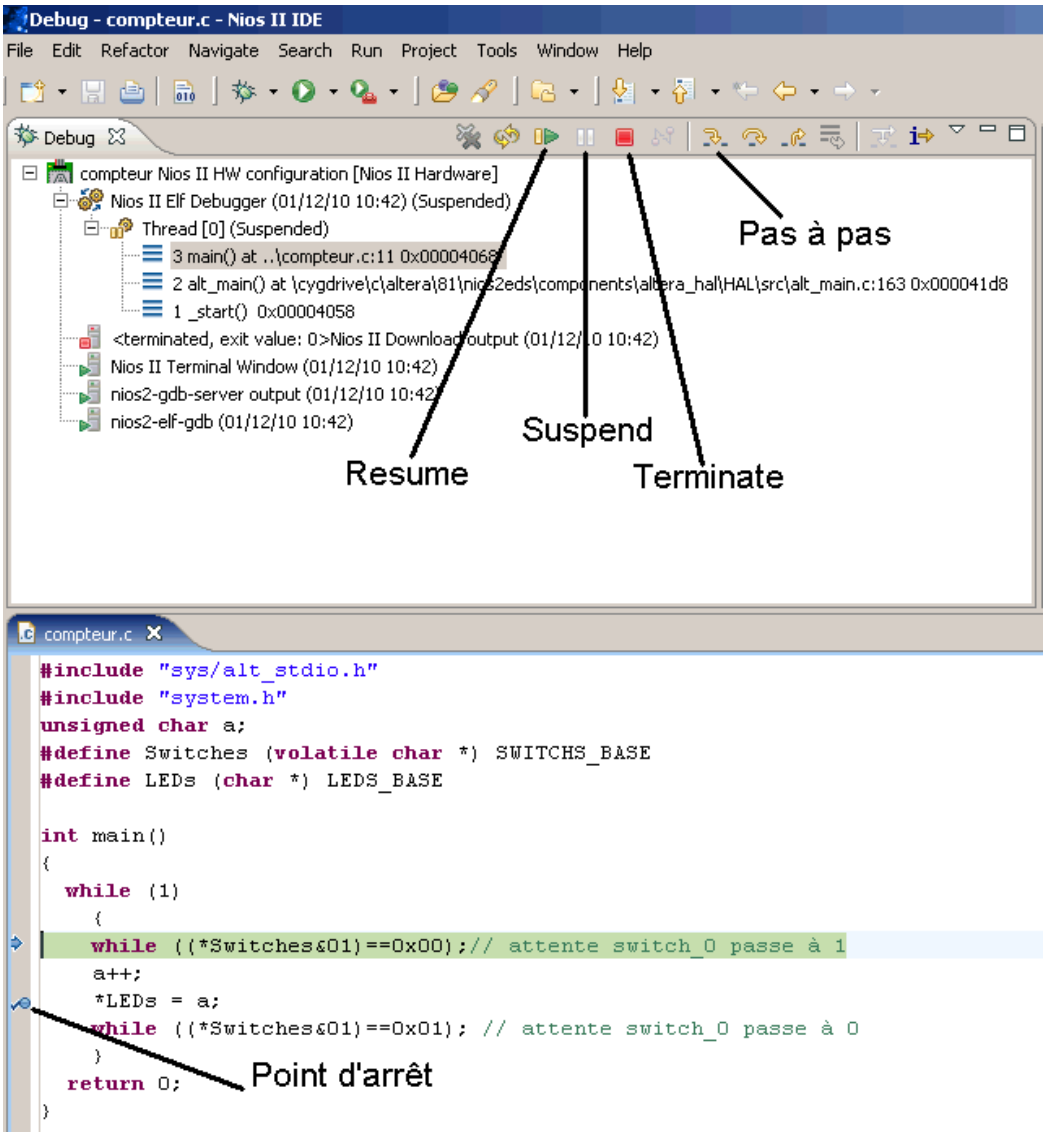


Le programme est téléchargé dans la cible puis est automatiquement exécuté.

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

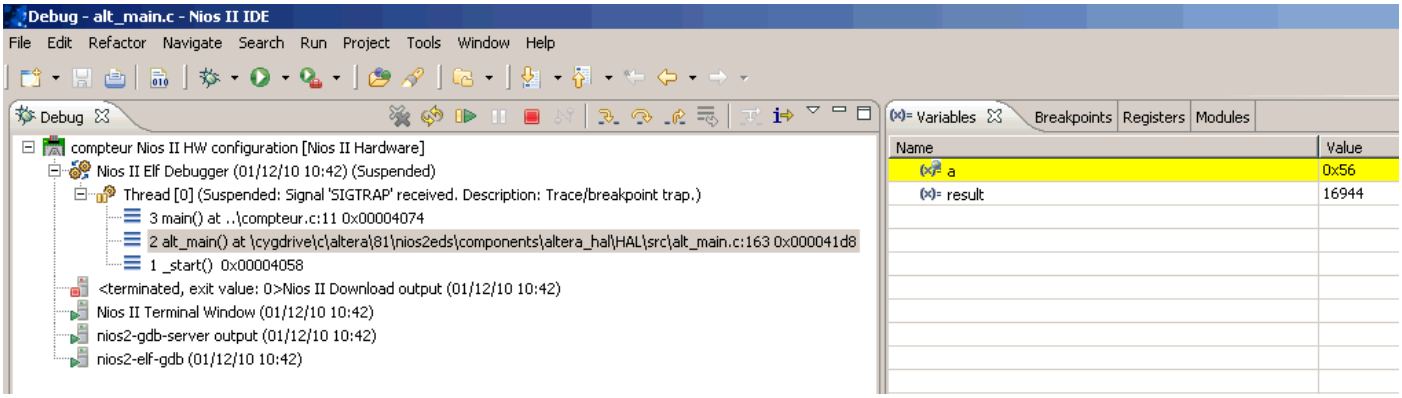
La commande « Resume » lance/relance l'exécution.
La commande « Suspend » suspend l'exécution du programme.
La commande « Terminate » arrête l'exécution et clôt la session.

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



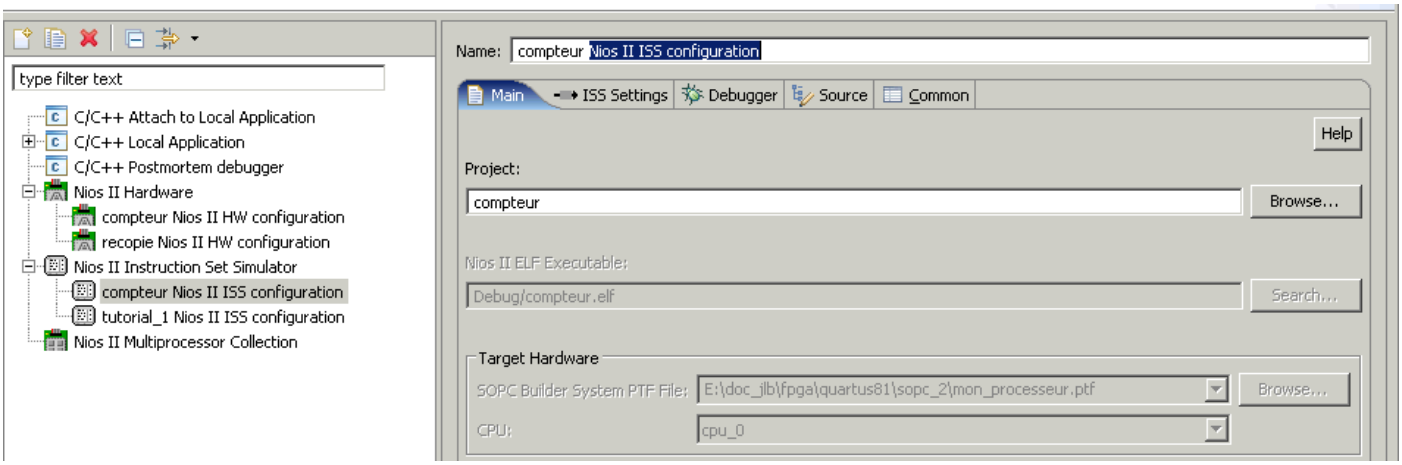
Pour visualiser et/ou modifier une variable :

a) Suspendre l'exécution du programme (touche suspend), ce qui se traduit par **Thread[0] (suspended)**
b) Sélectionner (mise en surbrillance) **alt_main().** 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 »).



7) Téléchargement du programme et exécution sur l'ISS (NIOS II ISS configuration)

Cette possibilité permet de tester le programme lorsqu'on n'a pas de carte cible. C'est l'ISS (Instruction Set Simulator) qui effectue la simulation du programme. La procédure est la même que dans le chapitre 6) sauf qu'on sélectionne **NIOS II ISS configuration**.



Attention cependant car les entrées/sorties physiques ne sont pas simulées !!!!

8) 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 :

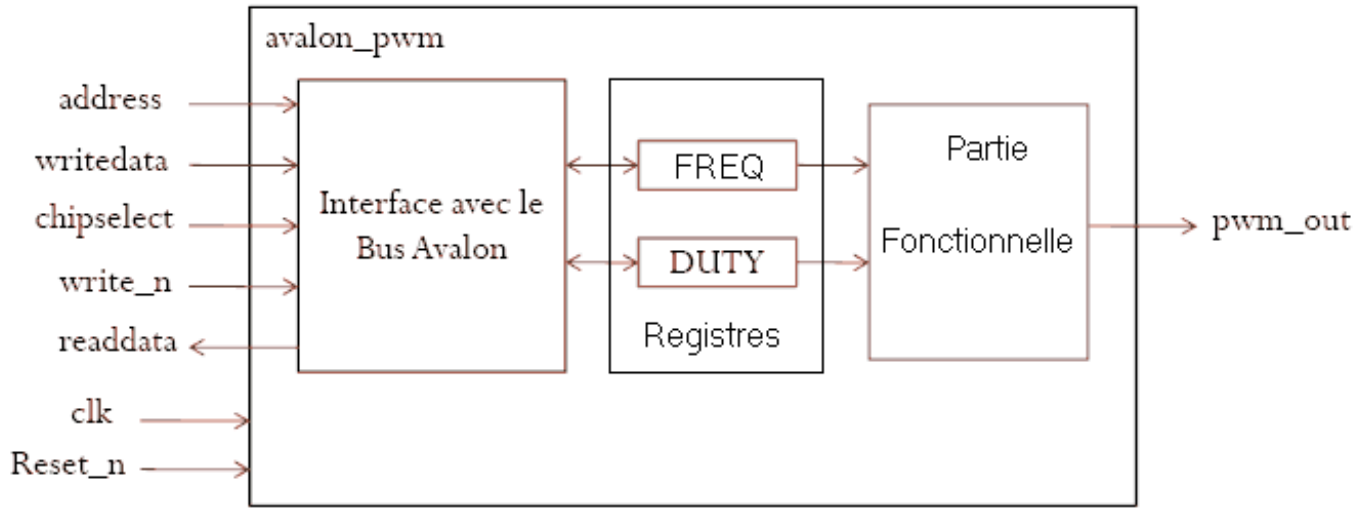
- Intégrer un composant d'un fournisseur tiers
- Créer son propre composant

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)



8.1) 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 is
generic (N : integer :=16);
port (
clk, , reset_n : in std_logic;
freq, duty : in std_logic_vector ((N-1) downto 0);
out_pwm : out std_logic
);
end entity;

ARCHITECTURE arch_pwm of pwm IS
        signal counter : std_logic_vector ((N-1) downto 0);
        signal pwm_on : std_logic;
BEGIN
--Process qui fixe la fréquence de la PWM
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;
--process qui génère le rapport cyclique
compare: process (clk, reset_n)
        begin
        if reset_n = '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;
out_pwm <= pwm_on;
END arch_pwm ;
```

8.2) 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:

```
entity avalon_pwm is
port (
clk, chipselect, write_n, address, reset_n : in std_logic;
writedata : in std_logic_vector (31 downto 0);
readdata : out std_logic_vector (31 downto 0);
out_pwm : out std_logic
);
end entity;
```

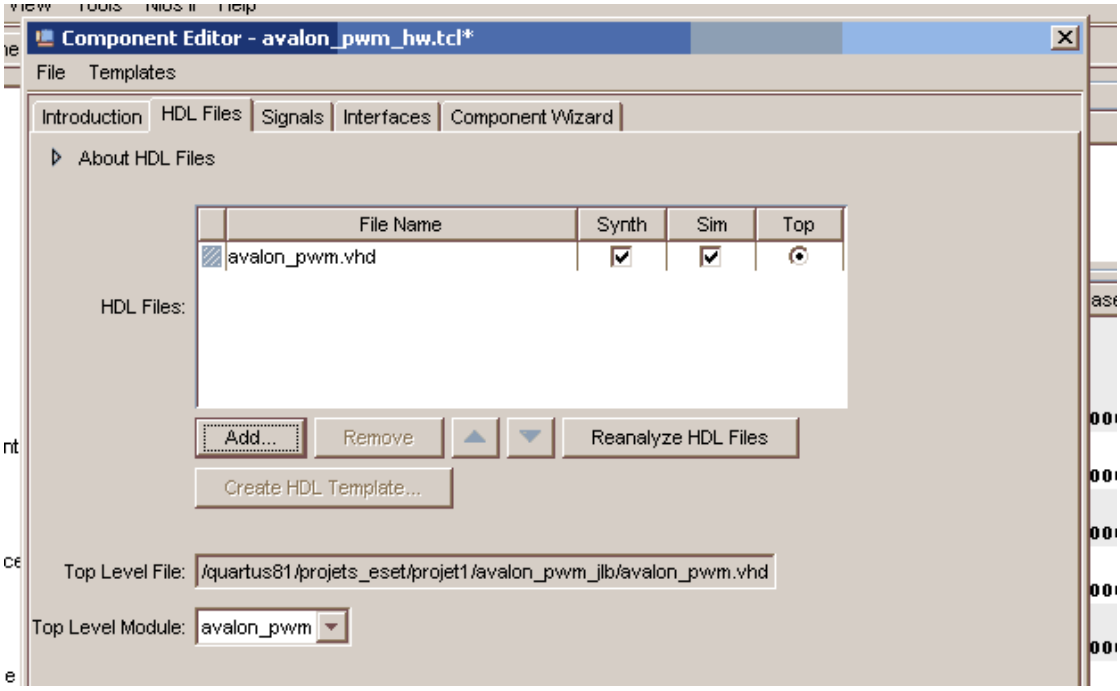
Architecture complète pour un compteur 16 bits:

```
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 pwm_on : std_logic;
BEGIN

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;

compare: process (clk, reset_n)
        begin
        if reset_n = '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;

--interface avalon
registers: process (clk, reset_n)
        begin
        if reset_n = '0' then
        freq <= (others => '0');
        duty <= (others => '0');
        elsif clk'event and clk = '1' then
                if chipselect ='1' and write_n = '0' then
                        if address = '0' then
                        freq (15 downto 0) <= (writedata (15 downto 0));
                        else
                        duty (15 downto 0) <= (writedata (15 downto 0));
                        end if;
                end if;
        end if;
end process;
readdata <= X"0000"&freq when address = '0' else X"0000"&duty;
END arch_avalon_pwm ;
```

8.3) Intégration dans le SOPC Builder

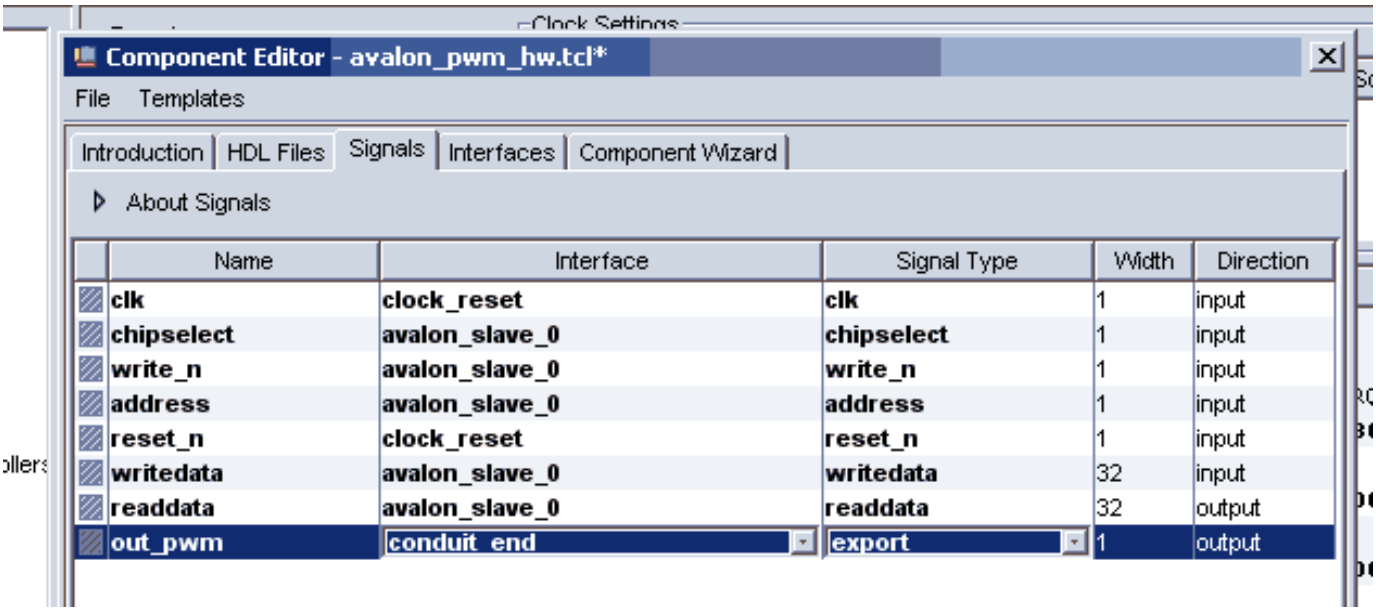
On commence par créer son micro contrôleur comme vu dans le début du document puis dans la fenêtre du SOPC Builder cliquer sur « new » en bas à gauche. Dans la fenêtre qui se présente faire « next » puis « add » pour aller chercher le circuit que l'on vient de décrire (ici avalon_pwm.vhd).



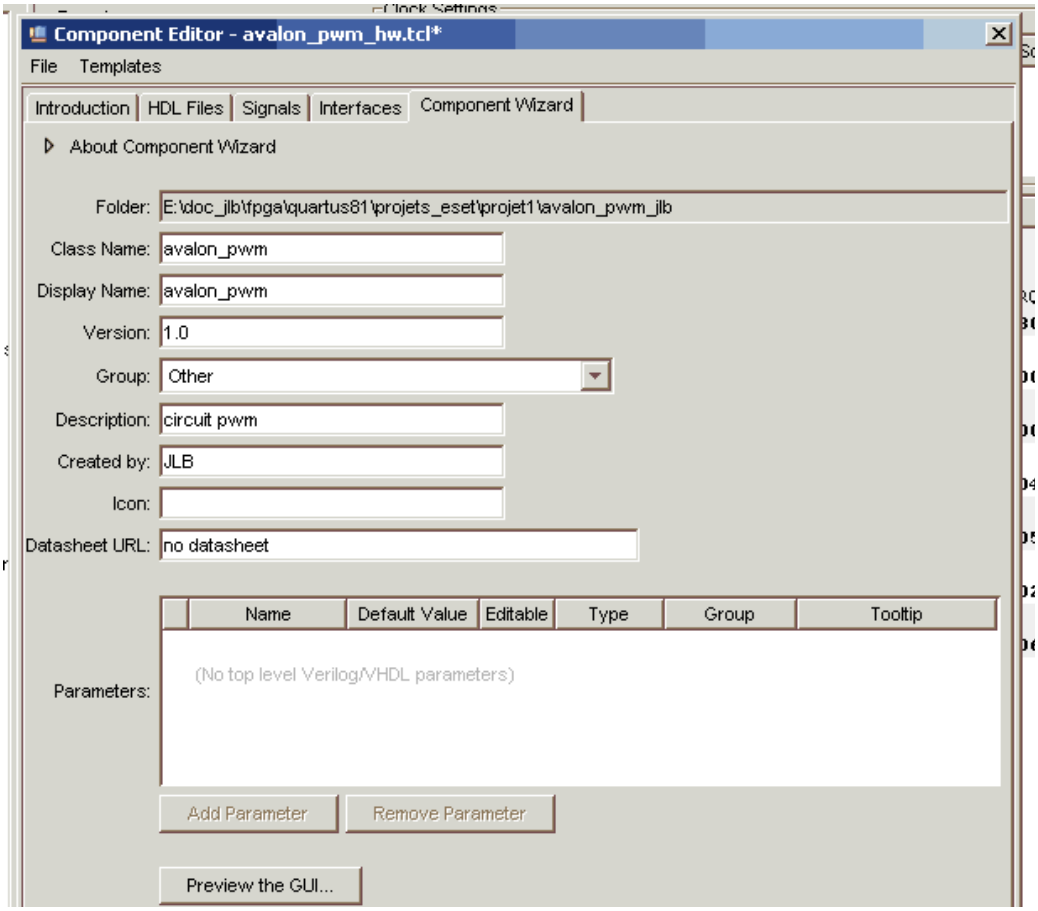
Remarque 1 : il est préférable de copier le dossier projet « avalon_pwm » dans le dossier projet du système que l'on est en train de créer (bug de quartus 8.1 ??) sinon, à chaque fois qu'on viendra ouvrir le projet il faudra recommencer la création du circuit avalon_pwm !!
Remarque 2 : on peut aussi 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 « auto-assign base addresses » .

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 :

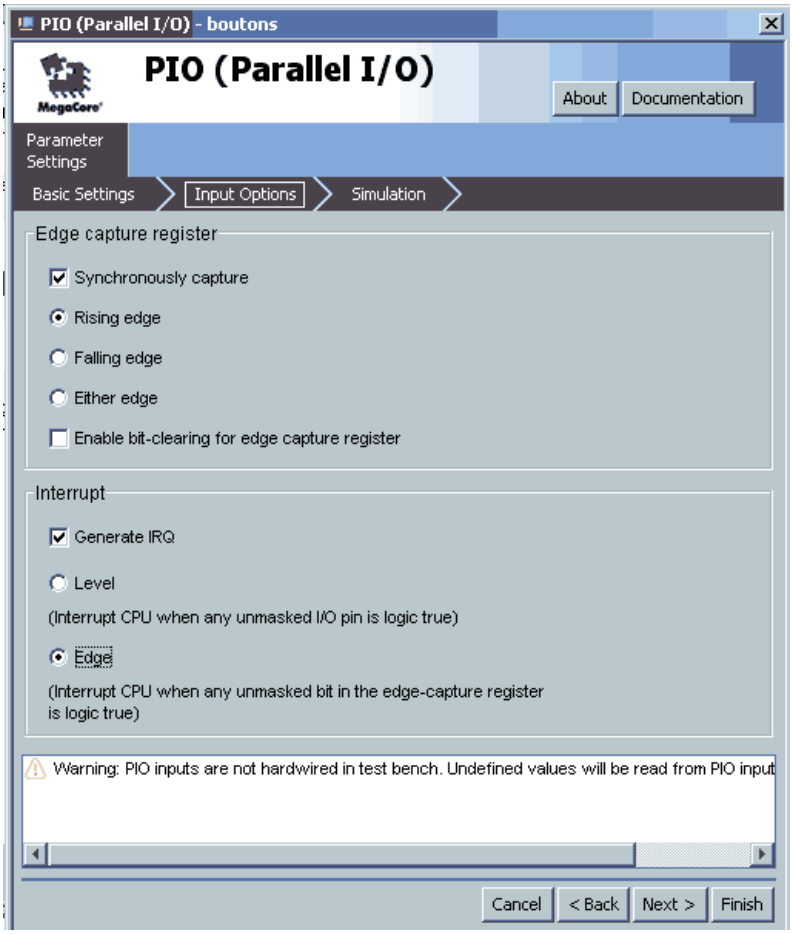
```
#define freq (unsigned int *) AVALON_PWM_0_BASE
#define duty (unsigned int *) (AVALON_PWM_0_BASE + 4)
int main()
{
    *freq = 0x0400;// divise clk par 1024
    *duty = 0x0200;// RC = 50%
```

9) Mise en œuvre des interruptions

La mise en œuvre des interruptions passe par la modification des parties matérielles et logicielles.

9.1) Modification de la partie matérielle

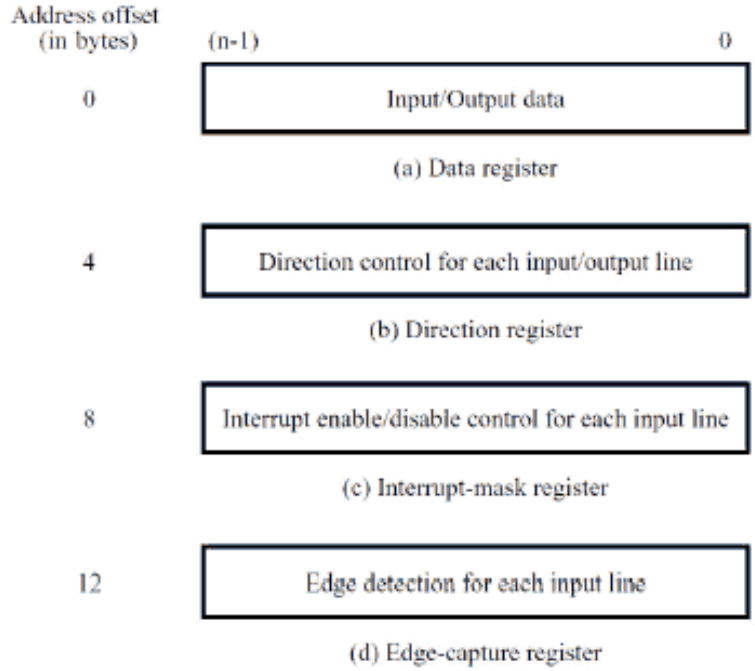
On se propose de générer une interruption lors de l'activation d'un interrupteur de la carte. Pour cela on revient sur le projet et on double-clique sur le système qui a été précédemment généré : ceci a pour effet de lancer SOPC-Builder.
Sélectionner le périphérique concerné par l'interruption (ici ce sera le PIO en entrée nommé « boutons ») et l'éditer (double clic). Renseigner les différents champs (ici l'interruption est déclenchée sur front montant).



Donner un numéro à l'interruption (on prendra 3 par exemple : ce qui correspond à l' IRQ3).

cpu_0	Nios II Processor				
instruction_master	Avalon Memory Mapped Master	**clk_0**			
data_master	Avalon Memory Mapped Master			IRQ 0	IRQ 31
jtag_debug_module	Avalon Memory Mapped Slave		**0x00004800**	0x00004fff	
onchip_sram	On-Chip Memory (RAM or ROM)				
s1	Avalon Memory Mapped Slave	**clk_0**	**0x00002000**	0x00003fff	
sys_clk_timer	Interval Timer				
s1	Avalon Memory Mapped Slave	**clk_0**	**0x00005000**	0x0000501f	0
leds	PIO (Parallel I/O)				
s1	Avalon Memory Mapped Slave	**clk_0**	**0x00005040**	0x0000504f	
boutons	PIO (Parallel I/O)				
s1	Avalon Memory Mapped Slave	**clk_0**	**0x00005050**	0x0000505f	3
timer_led_IR	Interval Timer				
s1	Avalon Memory Mapped Slave	**clk_0**	**0x00005020**	0x0000503f	1
jtag_uart_0	JTAG UART				
avalon_jtag_slave	Avalon Memory Mapped Slave	**clk_0**	**0x00005060**	0x00005067	2
avalon_pwm_0	avalon_pwm				
avalon_slave_0	Avalon Memory Mapped Slave	**clk_0**	**0x00005068**	0x0000506f	

Remarque : Chaque PIO est composé de quatre registres de 32 bits maximum, utilisés ou non selon la configuration retenue. La figure ci-dessous en donne le rôle et la position dans l'espace adressable :

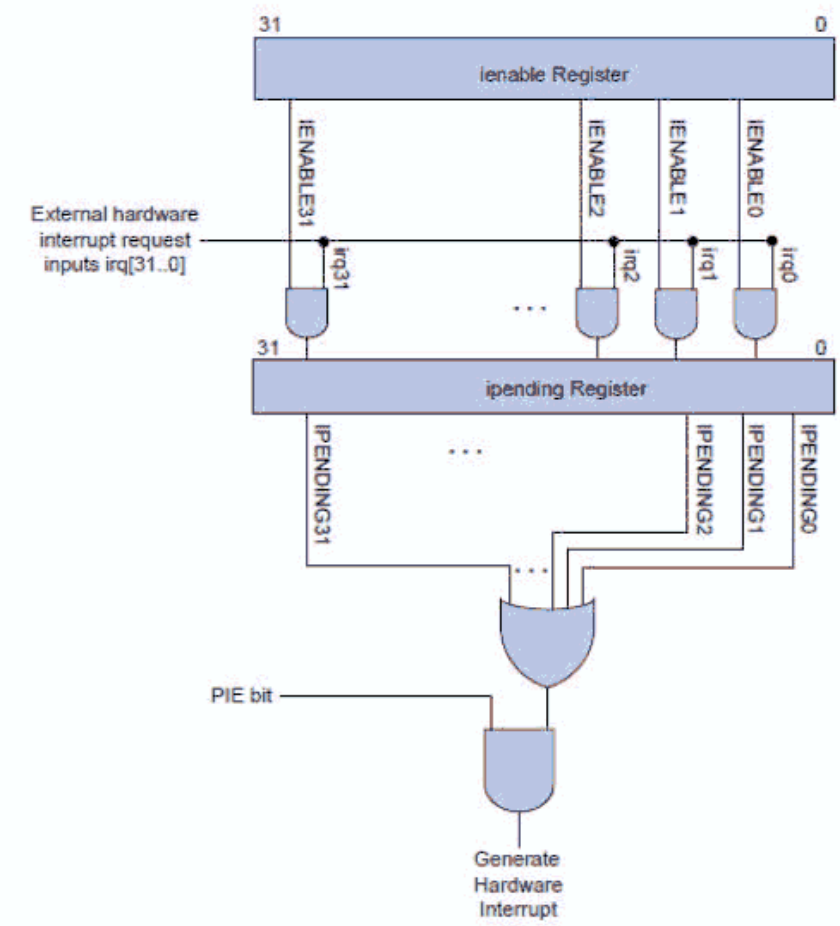


Les registres Interrupt-mask register et Edge-capture register seront exploités pour la mise en œuvre des interruptions.

Relancer la génération du système (clic sur « generate » dans SOPC Builder), puis recompiler le projet pour que les modifications faites sur le système soient prises en compte. A ce stade on peut reconfigurer le FPGA avec la nouvelle version du projet.

9.1) Modification de la partie logicielle

Notes préliminaires pour la compréhension de la mise en œuvre des interruptions.
Le NIOS dispose de 32 interruptions **non vectorisées** notées IRQ0 à IRQ31. Ces interruptions sont câblées comme ci-dessous :



Principe : Lorsqu'un périphérique (le PIO « boutons » par ex.) active sa ligne d'interruption (IRQ3 dans l'exemple), celle-ci se propage au registre suivant (ipending register) si elle y a été autorisée (bit 3 du registre ienable à 1). Dans ce cas, l'interruption se propage vers le processeur NIOS si le bit PIE d'autorisation globale des interruptions (situé dans le registre d'état du processeur NIOS) a été préalablement mis à 1. Si c'est le cas, ceci a pour effet de dérouter le programme en cours d'exécution vers une adresse d'exception prédéfinie. A cette adresse d'exception doit se trouver un programme qui identifie le périphérique demandeur et passe le relai au programme dédié au traitement de l'interruption concernée.

Les 6 registres de contrôle du processeur NIOSII :

Register	**Name**	$b_{31} \cdots b_2$	b_1	b_0
ctl0	status	Reserved	U	PIE
ctl1	estatus	Reserved	EU	EPIE
ctl2	bstatus	Reserved	BU	BPIE
ctl3	ienable	Interrupt-enable bits		
ctl4	ipending	Pending-interrupt bits		
ctl5	cpuid	Unique processor identifier		

En résumé le concepteur doit s'assurer que :

1) Le périphérique concerné est en mesure de basculer la ligne IRQn qui lui a été affectée : ceci est réalisé automatiquement par le SOPC Builder
2) L'adresse du programme d'exception a été définie : ceci est réalisé automatiquement par le SOPC Builder également.

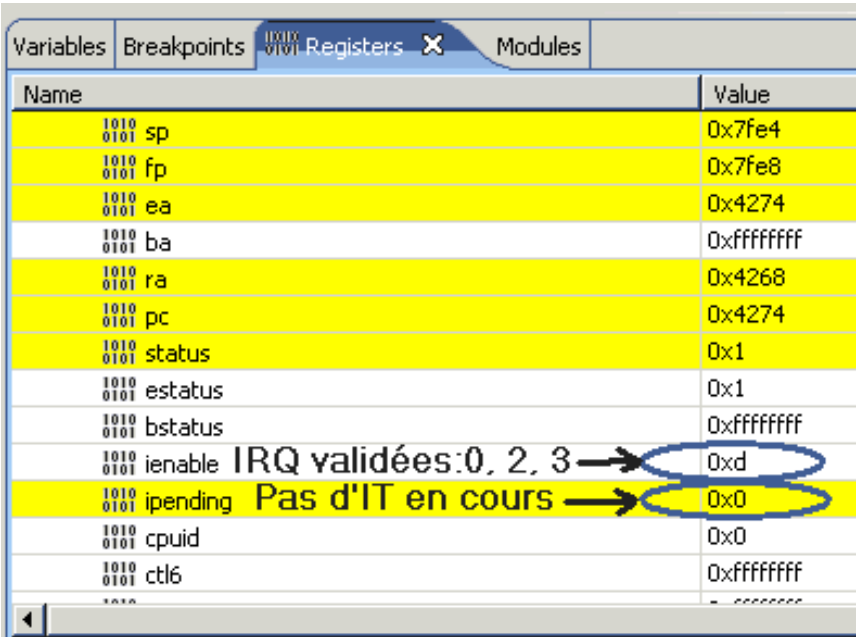
3) Le ou les bits de validation d'interruptions a/ont été positionnés dans le registre de contrôle des interruptions du périphérique concerné (situé à l'adresse base+8): ceci est réalisé par le programme en C par l'instruction (c'est le bouton 2 qui génère l'IRQ dans cet exemple):
   `IOWR_ALTERA_AVALON_PIO_IRQ_MASK(BOUTONS_BASE,4) ;`
   ou par :
   `*(boutons+8)=4;`
4) Le ou les bits correspondant au niveau de l'interruption sont positionnés à 1 dans le registre ctl3 (ienable) du NIOSII : ceci est réalisé par le programme en C par l'instruction :
   `alt_irq_register(BOUTONS_IRQ,cmpt,sp_IT) ;`
   À la compilation « BOUTONS_IRQ » est remplacé par sa valeur effective prise dans le fichier « system.h » (cette valeur est 3 pour IRQ3 dans notre exemple).
   Sp_IT est le nom du sous-programme qui sera appelé lors de l'interruption.
5) L'interruption globale PIE a été validée (mise à 1 du bit 0 du registre ctl0) : ceci est réalisé par le programme en C à partir de l'exécution de l'instruction précédente ou si d'autres interruptions sont autorisées (notamment celle de la liaison JTAG).
6) Le ou les sous-programmes de prise en compte et d'identification des interruptions sont définis : ces sous-programmes sont automatiquement fournis (en librairies).
7) Les sous-programmes d'applications associés aux interruptions sont définis : fournis par le concepteur.
8) A la fin du traitement de l'interruption il faut remettre à 0 le drapeau d'interruption : ceci est réalisé par le programme en C avec l'instruction :
   `IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BOUTONS_BASE,0) ;`
   `Ou :`
   `*(boutons+12)=0;`

Exemple de programme (à chaque action sur le bouton 2, on génère une interruption IRQ3 qui provoque la recopie de l'état des boutons sur les leds).

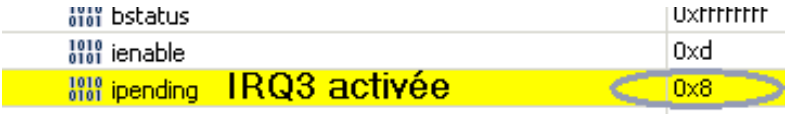
```
#include "sys/alt_stdio.h"
#include "sys/alt_irq.h"
#include "altera_avalon_pio_regs.h"
#include "alt_types.h"
#include "system.h"
#define boutons (volatile char *) BOUTONS_BASE
#define leds (char *) LEDS_BASE

// définition de la fonction  sp_IT de gestion des IT
// « alt_u32 id » sera remplacé par 3 à la compilation

static void sp_IT(void *context,alt_u32 id)
{
  *leds = *boutons;
  // RAZ du drapeau IT
   *(boutons+12)=0;//met à 0 le 4ème registre du PIO
}

int main()
{
  void *cmpt;

  // RAZ EDGE CAPT avant autoriser les IT
*(boutons+12)=0;
// autorise l'IRQ3 et définit le branchement au sp_IT
// L'IRQ3 est définie par la valeur de BOUTONS_IRQ dans system.h
  alt_irq_register(BOUTONS_IRQ,cmpt,sp_IT) ;

  // Autorisation d'une IT générée par SW2 (100)
```

```
  *(boutons+8)=4;
  /* Event loop never exits. */
  while (1)
    {
    }
  return 0;
};
```

Evolution des registres lors de l'exécution du programme en pas à pas :



Puis après apparition de l'IRQ3 :



Fin du document