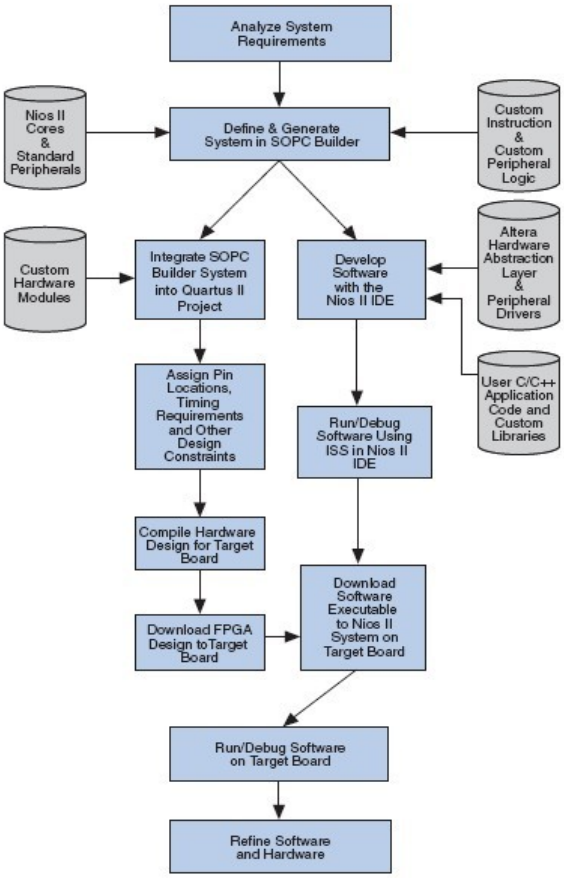
Découverte du système NIOS II Altera

Note: Les illustrations correspondent à la version logicielle Quartus 8.1

1) Objectif pédagogique

Cette première séance, incontournable, offre la possibilité de découvrir les outils de développement du matériel et du logiciel. On construira un premier système simple, on en fera la synthèse complète, on y associera une application logicielle. L'ensemble sera chargé et testé sur la carte de développement. On aura de ce fait une première approche du **SOPC Builder** pour la génération du système, de **Quartus** pour la synthèse et le placement/routage, de l'**IDE** pour le développement logiciel, compilation et debug.

Figure 1–2. Nios II System Development Flow



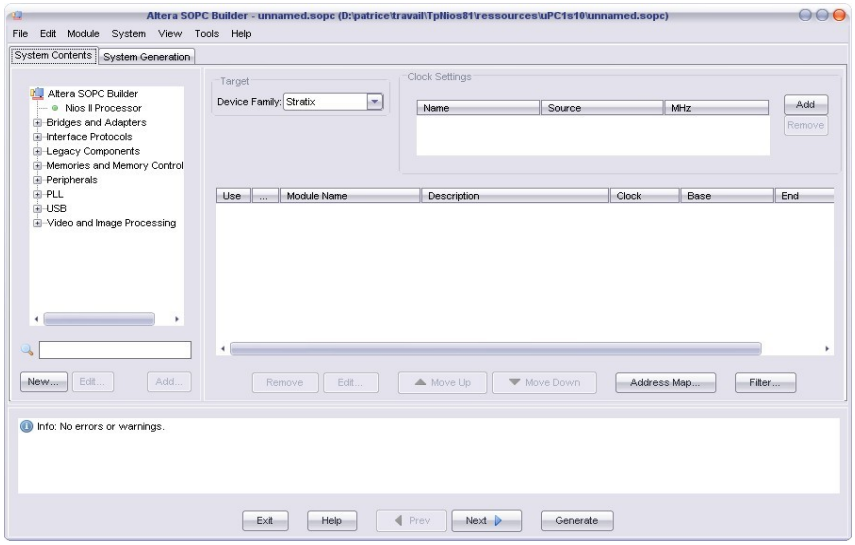
2) Élaboration du système

➔ **Recopier** dans votre espace de travail le répertoire *Tp_sopc* et ouvrir avec **Quartus** le projet **uPC1S10.qpf** qu'il contient.
➔ Ouvrir la feuille de schéma de niveau supérieur: **file>open>** *uPC1S10.bdf* .Elle contient les entrées-sorties non encore affectées du système que l'on va construire.

2.1) Démarrage du SOPC Builder

➔ La commande **Tools>SOPC Builder** propose de créer un système que l'on appellera *nios2_system* et on fera le choix de VHDL comme langage de génération.

➔ Après ce choix (OK) , l'interface graphique du SOPC Builder apparaît. On vérifiera que la fréquence du système est bien de 50 Mhz.



On va pouvoir maintenant construire le système voulu. Il comportera les composants suivants:

1. Le CPU NIOS
2. Un timer nécessaire pour les routines systèmes basées sur le temps;
3. Le contrôleur de mémoire externe flash
4. Le contrôleur de mémoire externe RAM
5. Le contrôleur de mémoire externe SDRAM
6. L'interface JTAG UART
7. Le BUS externe permettant de connecter ces mémoires(Avalon tri-state bridge)
8. L'interface pour l'afficheur LCD
9. Le PIO interface parallèle pour les LEDs
10. Le PIO interface parallèle pour les boutons poussoirs
11. Le PIO interface parallèle pour les afficheurs 7 segments
12. Le composant générateur d'identification du matériel
13. La pll génératrice d' horloge pour la mémoire SDRAM

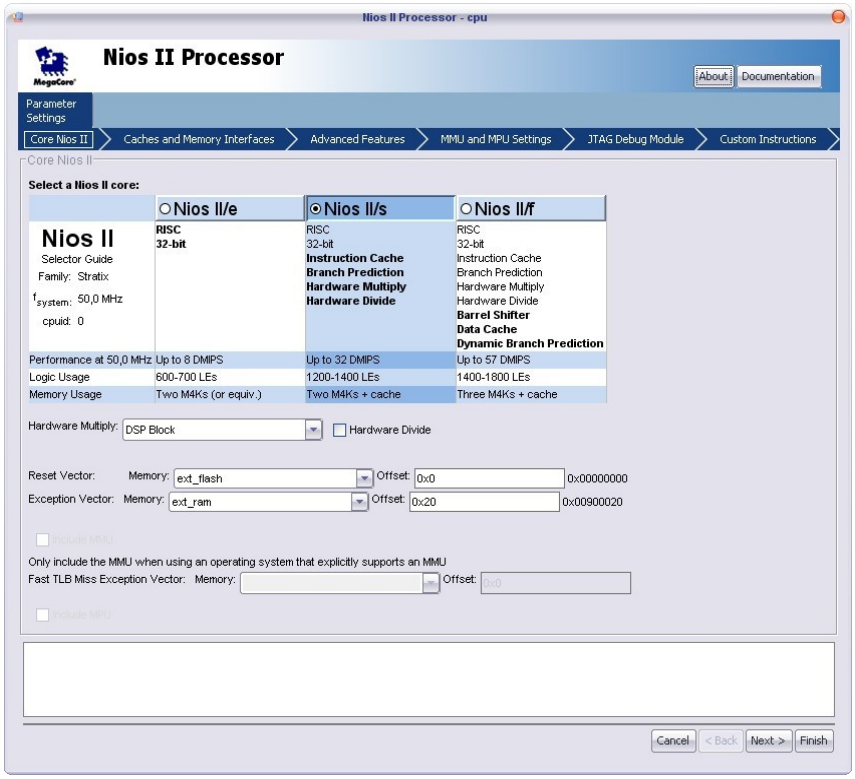
Tous le composants seront extraits du menu du SOPC Builder: **System contents**

*Ne pas s'inquiéter des différents messages (**warning** ou error) apparaissant pendant la construction du système. Ils sont temporaires et dûs à des éléments manquants. Cependant, avant de générer le système , ces messages devront avoir disparus en fin de construction.*

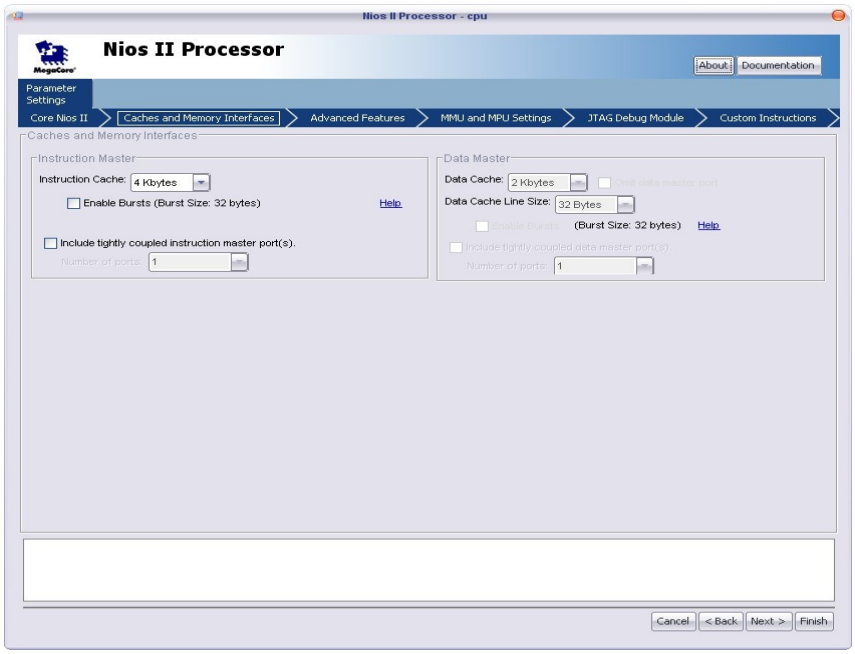
2.2) Le CPU NIOS

➔ Effectuer un double-click sur la ligne **Nios II Processor** . On va implanter une version moyenne du processeur.

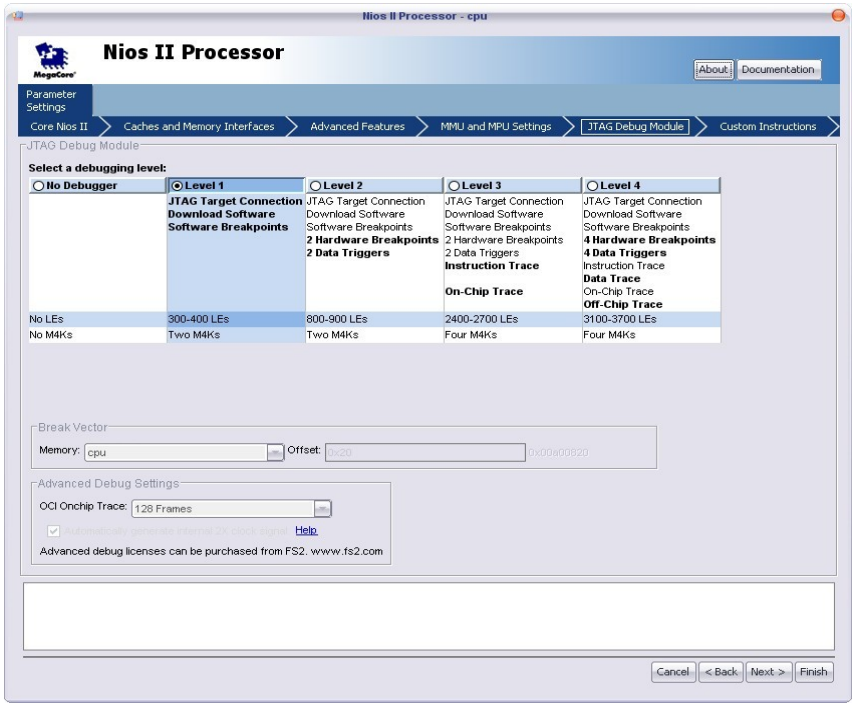
Comme le montre l'image ci-après, la première fenêtre « **Core NIOS II** » permet de faire ce choix mais ne pourra pas être entièrement configurée en ce qui concerne les vecteurs d'exceptions tant qu'on n'a pas de mémoire disponible, cela sera fait un peu plus tard.



➔ Faire **Next**



➔ Dans la fenêtre « caches and memory interfaces », on fixera **4K Octets** de mémoire cache puis **Next 3 fois**



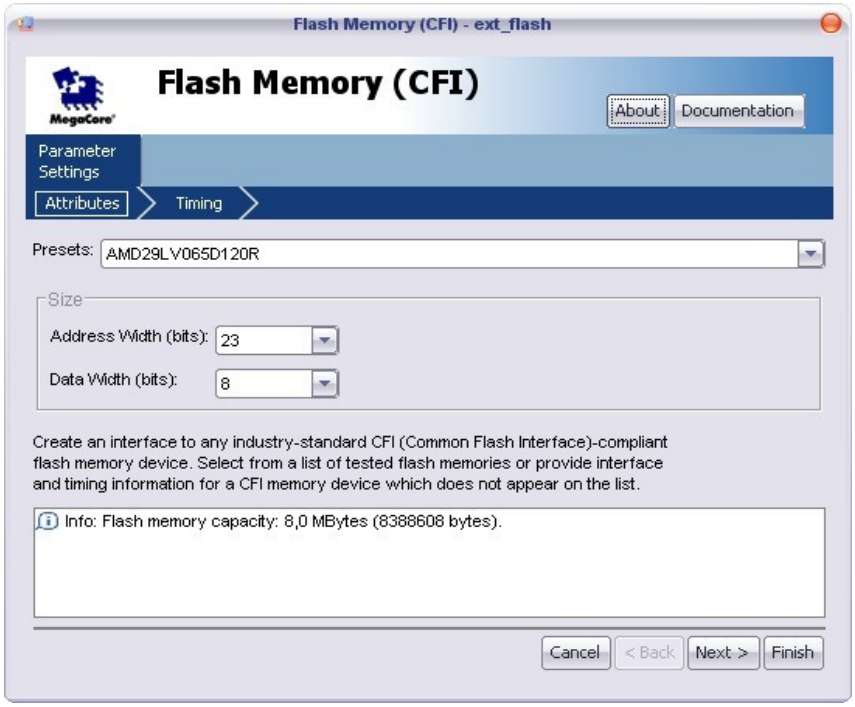
➔ **Level 1** sera choisi pour le « JTAG Debug module »
➔ On peut valider le tout par **Finish**
➔ Dans le champ Modul name, renommer *cpu_o* (sélection puis clic droit > **rename**) en *cpu*

2.3) Le Timer

➔ Ajouter un composant **Peripherals>Microcontroller peripherals>interval Timer**
➔ Garder les valeurs par défaut, puis **Finish**
➔ Renommer le composant *timer_0* en *system_timer*

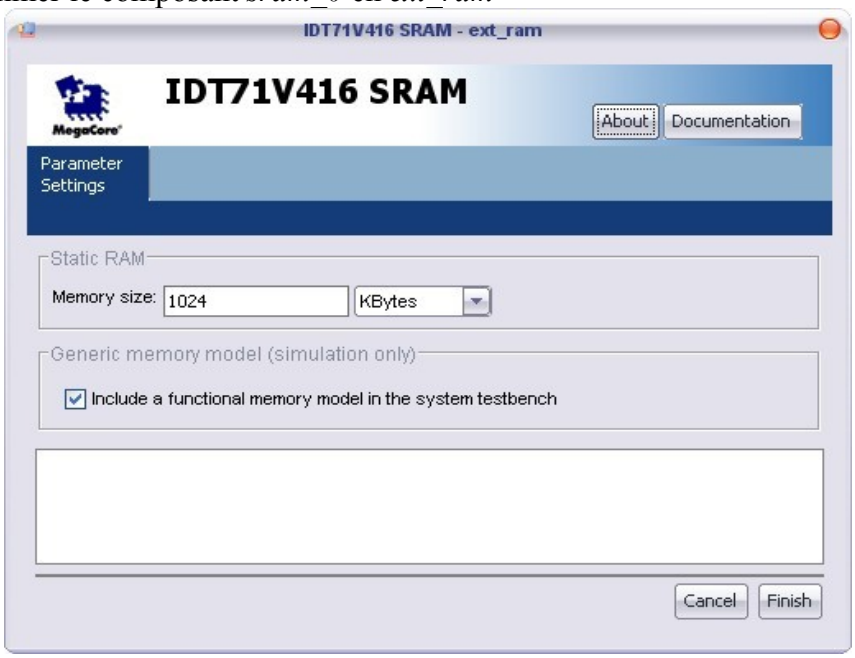
2.4) Le contrôleur de mémoire externe Flash

- ➔ Ajouter le composant **Memories and memory controller> Flash Memory(CFI)**
- ➔ Fixer le type de mémoire à: AMD29LV065120R puis garder les valeurs par défaut et **Finish**
- ➔ Renommer le composant *cfi_flash_0* en *ext_flash*



2.5) Le contrôleur de mémoire externe SRAM

- ➔ Ajouter le composant **Memories and memory controller>SRAM >IDT71V416**
- ➔ Garder toutes les valeurs par défaut et **Finish**
- ➔ Renommer le composant *sram_0* en *ext_ram*



2.6) Le contrôleur de mémoire externe SDRAM

➔ Ajouter le composant **Memories and memory controller>SDRAM>Sdram Controller**
➔ Choisir le type (presets) : **single Micron MT48LC4M32B2-7 chip**
➔ Garder toutes les valeurs par défaut et **Finish**
➔ Renommer le composant *sdram_0* en sd*ram*

2.7) L'interface JTAG UART

➔ Ajouter le composant : **Interface Protocols> Serial> JtagUart**
➔ Garder les valeurs par défaut et **Finish**

➔ Renommer le composant *jtag_uart_0* en *jtag_uart*

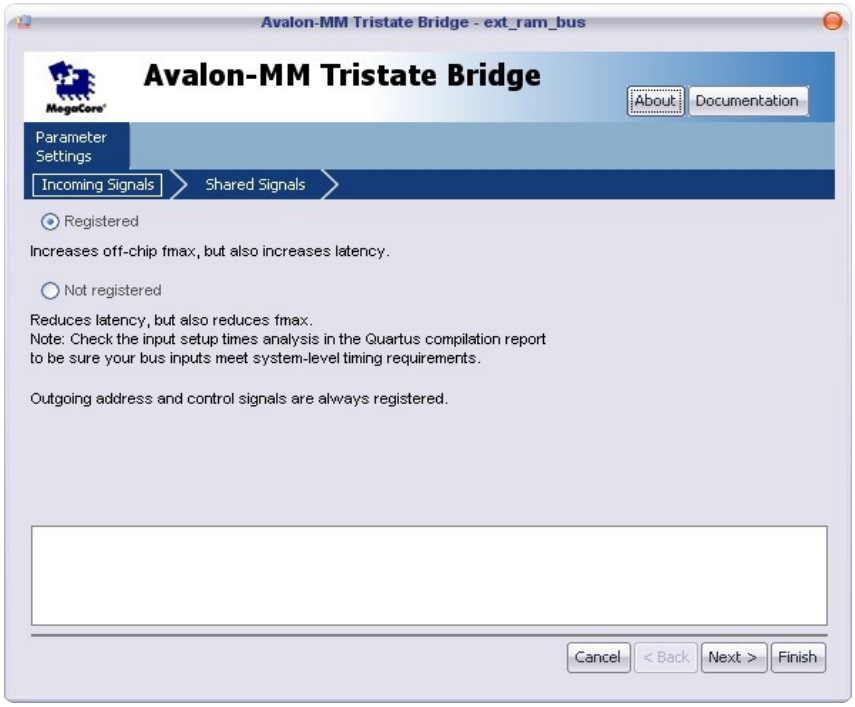
2.8) L' interface de Bus externe

➔ Ajouter le composant : **Briges and Adapters>Memory Mapped>Avalon-MM tri-state bridge**

➔ Garder les valeurs par défaut et **Finish**

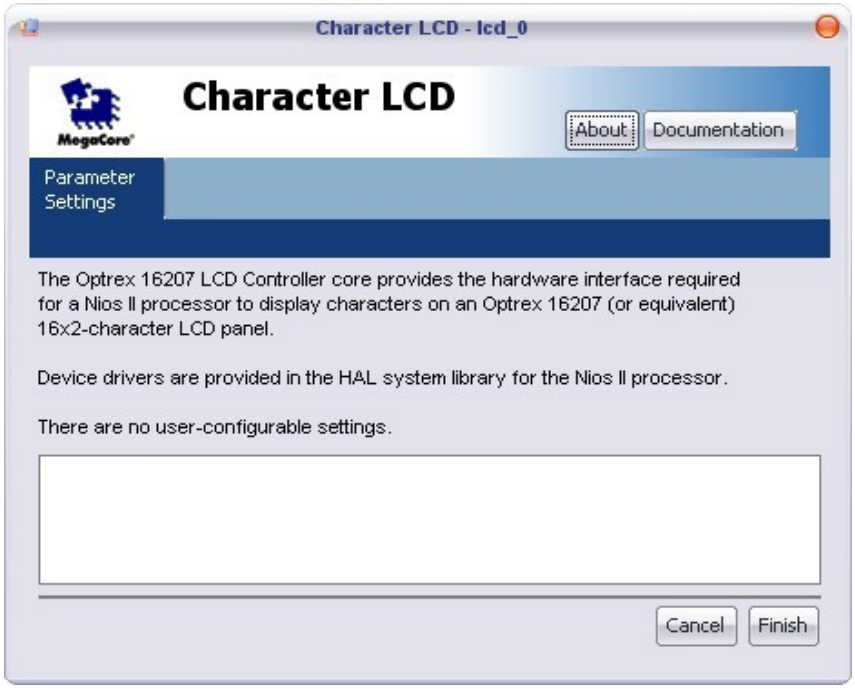
➔ Renommer le composant *tri_state_bridge_0* en *ext_ram_bus*

➔ Il faut alors connecter au bus les deux composants *ext_ram* et *ext_flash*. Pour réaliser cela, dans la partie « connection » du SOPC Builder, on déplace la souris sur le fil *ext_ram_bus.tristate_master*. Une connexion étant repérée par un point noir, on y connecte les deux esclaves (s1) *ext_ram* et *ext_flash* en cliquant les points d'intersection.



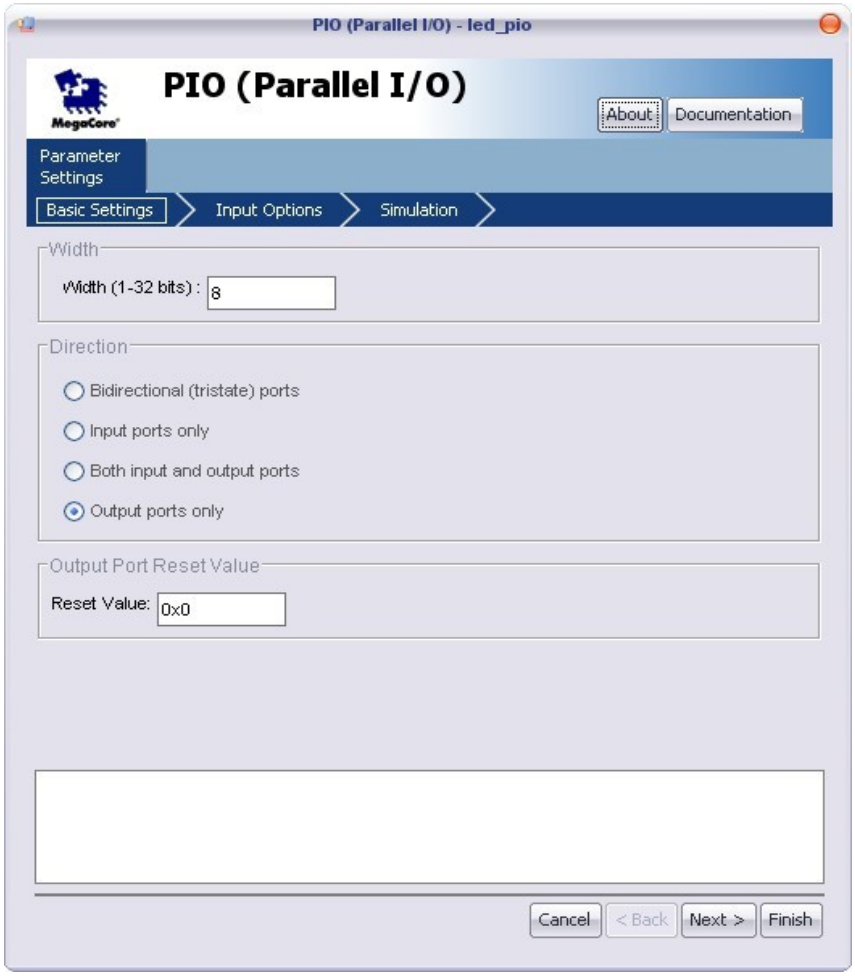
2.9) L' interface pour l'afficheur LCD

- ➔ Ajouter le composant : **Peripherals>Display>character LCD**
- ➔ Garder les valeurs par défaut et **Finish**
- ➔ Renommer le composant lcd_*0* en *lcd_display*



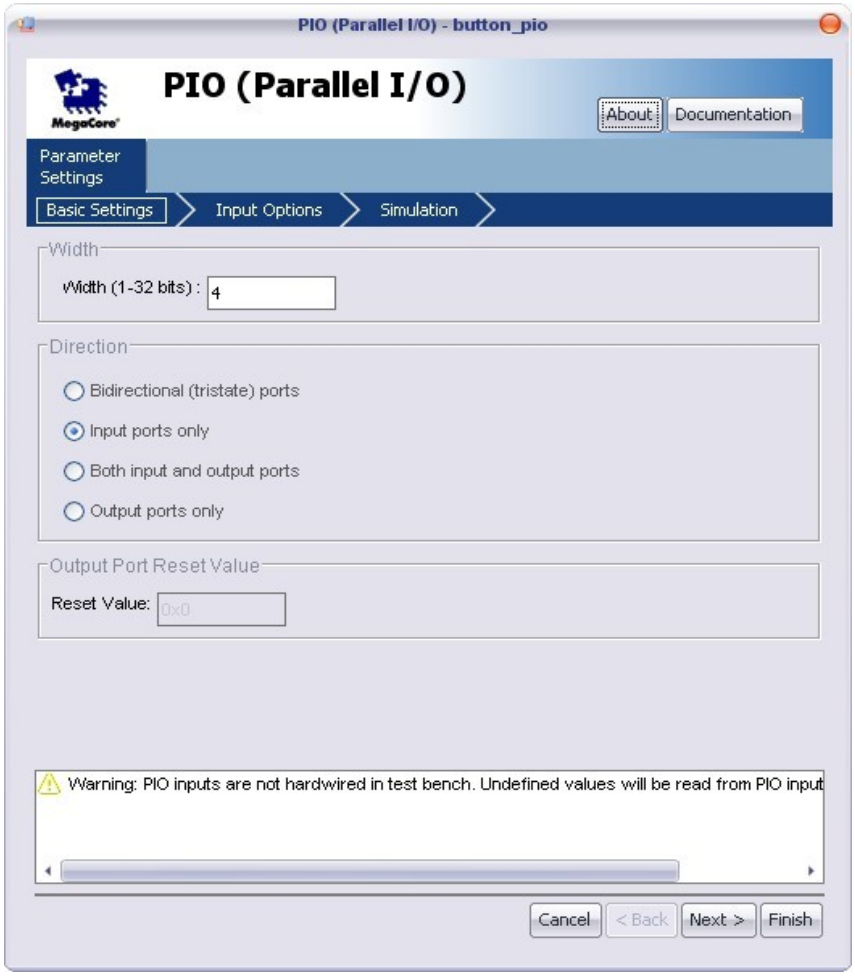
2.10) Le PIO interface parallèle pour les LEDs

- ➔ Ajouter un composant **Peripherals>Microcontroller peripherals>PIO (Parallel I/O)**
- ➔ Garder les valeurs par défaut (8bits en sorties) puis **Finish**
- ➔ Renommer le composant *pio_0* en *led_pio*



2.11) Le PIO interface parallèle pour les boutons poussoirs

- ➔ Ajouter un composant **Peripherals>Microcontroller peripherals>PIO (Parallel I/O)**
- ➔ Dans Basic Settings, choisir **4 bits** et **Inputs Ports Only**
- ➔ Dans Inputs Options , faire les choix: **Synchronously capture / Rising edge** et de **Generate IRQ / Edge**, puis **Finish**
- ➔ Renommer le composant *pio_0* en *button_pio*

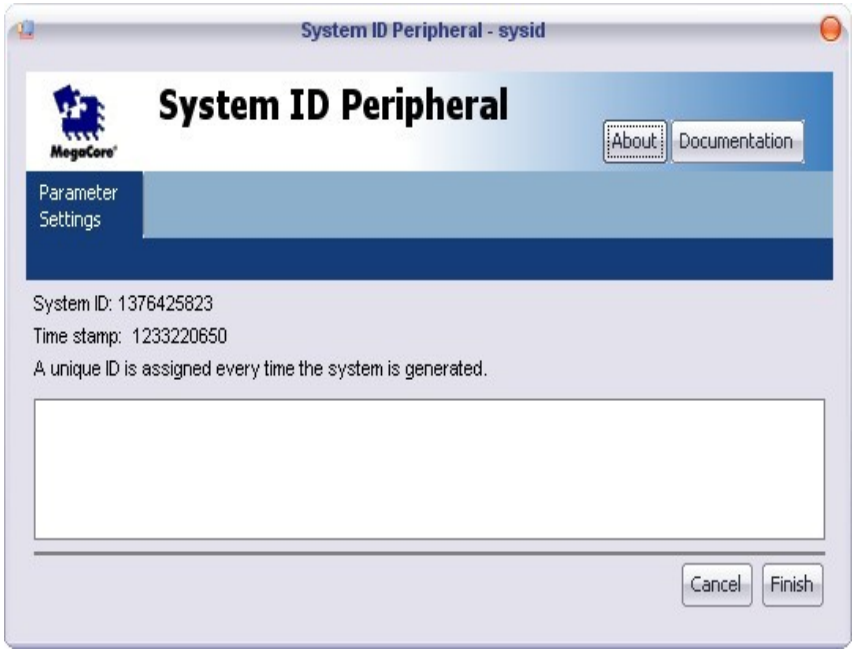


2.12) Le PIO interface parallèle pour les afficheurs 7 segments

- ➔ Ajouter un composant **Peripherals>Microcontroller peripherals>PIO (Parallel I/O)**
- ➔ Dans Basic Settings, choisir **16 bits** et **Outputs Ports Only**
- ➔ **Finish**
- ➔ Renommer le composant *pio_0* en *seven_seg_pio*

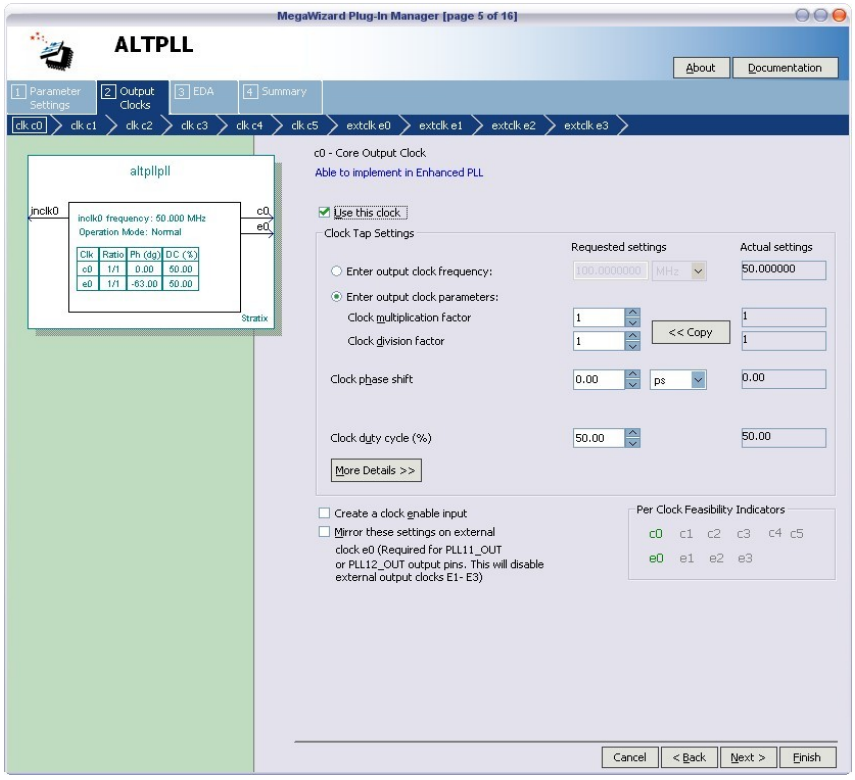
2.13) Le composant générateur d'identification du matériel

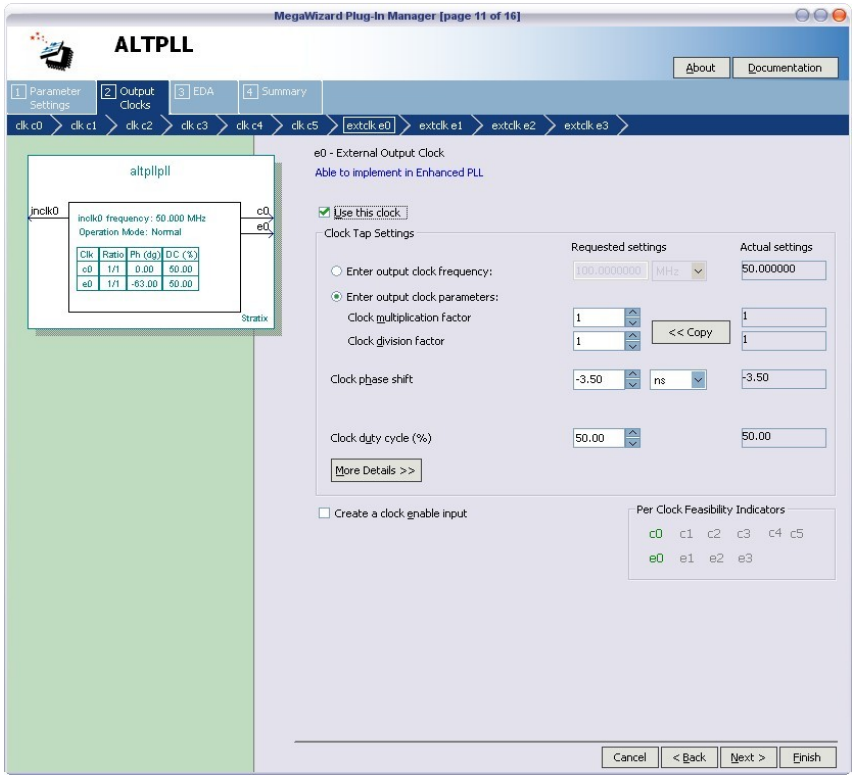
- ➔ Ajouter un composant **Peripherals>Debug and performance>System ID Peripheral**
- ➔ **Finish**
- ➔ Renommer le composant sysid_*0* en *sysid*



2.14) La PLL génératrice d'horloge pour la SDRAM

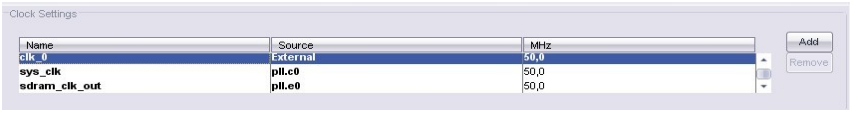
- ➔ Ajouter un composant **PLL>pll**
- ➔ Dans la fenêtre ouverte (PLL settings) cliquer sur **Launch Altera's ALTPLL MegaWizard**
- ➔ Se déplacer dans l'onglet **Output clocks > clock c0**. On fera une copie de l'horloge système (50 Mhz) en cochant la case « **use this clock** », en fixant à 1 les paramètres « Clock multiplication factor » et « Clock division factor »
- ➔ Se déplacer dans l'onglet **Output clocks > extclk e0**. On crée une horloge de 50 Mhz pour alimenter la SDRAM en cochant la case « **use this clock** », en fixant à 1 les paramètres « Clock Multiplication factor » et « Clock division factor ». La phase sera fixée à -3,50 ns.
- ➔ **Finish**
- ➔ La pll s'appelle *pll*





Après avoir créé le composant pll dans le système, les horloges apparraissent dans la partie « clock settings » du SOPC Builder. On peut les renommer en double-cliquant sur leur nom.

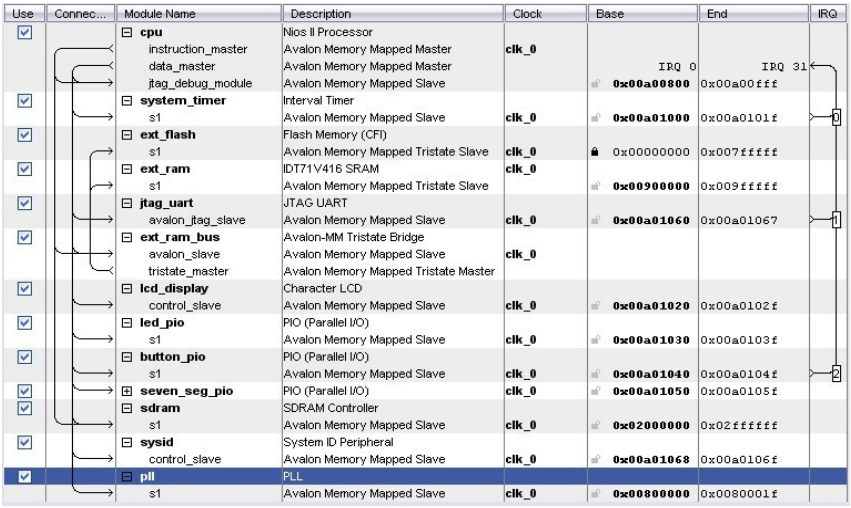
➔ Affecter la source pll.c0 du nom (name) *sys_clk* et pll.e0 du nom (name) *sdram_clk_out*



2.15) Adresses et numéros d'interruptions

➔ Selectionner le composant *ext_flash* dans le champ « Modul Name »
➔ Dans le champ « base » fixer 0x0 comme valeur après avoir double-cliquer.
➔ **Module > Lock Base Address** permet de verouiller cette adresse pour la flash
➔ **System > Auto-Assign Base Addresses** va faire de l'ordre dans la carte mémoire
➔ **System > Auto-Assign IRQs** va organiser les numéros d'interruptions

Le système NIOS est entièrement construit,il n'y a plus d'erreurs ou de conflits signalés, presque plus de warning, on va pouvoir le générer.



➔ Faire **Next** puis **Generate** (sans simulation) . La génération est assez rapide.
➔ En fin de génération réussie , on peut quitter le SOPC Builder par **Exit**

3) Synthèse et placement-routage

3.1) Compilation du projet Quartus

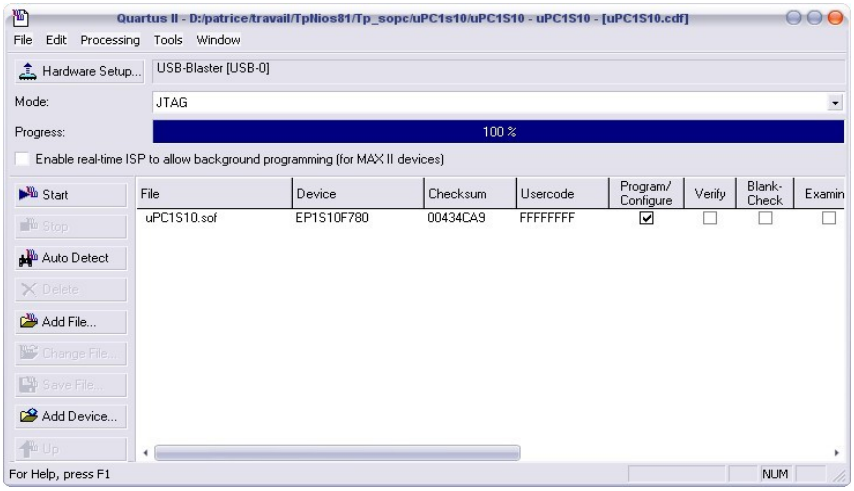
Dans Quartus , le projet au plus haut niveau est représenté par la feuille de schéma uPC1S10.bdf qui doit être ouverte. On y a déjà placé les différentes entrées-sorties avec leur noms conformes au fichier d'affectation des pins. Il ne manque que le système représenté pas son symbole.

- ➔ Dans la barre d'outil présente à gauche de la feuille de schéma, cliquer sur le symbole représentant une porte « ET » et marqué « symbol Tool »
- ➔ Dans **Project** sélectionner *Nios2_system* et **OK**
- ➔ On dépose le composant sur la feuille de schéma, terminant ainsi la partie construction du matériel
- ➔ Ouvrir la fenêtre console TCL **View>Utility windows>Tcl** Console
- ➔ Dans la console, vérifier par la commande **pwd** que vous êtes dans le bon répertoire de travail. Si c'est le cas , la commande **dir** vous montre la présence du fichier *uPC1s10.tcl*
- ➔ Réaliser l'affectation des pins par la commande : **source uPC1S10.tcl**
- ➔ Dans la barre d'outil du haut, choisir l'icône marqué « **Start Compilation** » (un triangle violet). Accepter de **sauvegarder** le projet . L'ensemble des opérations de synthèse VHDL, placement des fonctions sur des primitives Stratix1S10, et routage va s'enchainer donnant lieu à des rapports détaillés pour chacune des phases.

3.2) Configuration du circuit Stratix1S10

Cette opération de configuration du FPGA peut être réalisée en dehors du contexte **Quartus** , ce qui est expliqué ici est donc tout à fait optionnel. En effet, l'outil de programmation peut aussi être invoqué depuis l'**ide** outil de développement logiciel. Cette deuxième méthode sera, en pratique la plus utilisée mais la façon d'opérer est la même dans les deux cas.

- ➔ Dans les icônes de la barre du haut, cliquer sur le troisième en partant de la droite marqué « **Programme**r ». L'interface de programmation apparaît.
- ➔ Vérifier la présence de l'indication **USB-Blaster** à coté de Hardware Setup sinon réparer à l'aide de cette dernière commande.
- ➔ **Add File...** permet de choisir le fichier de programmation uPC1S10.sof
- ➔ **Start** lance le chargement de ce fichier dans le circuit Stratix1S10



3.3) Analyse et conclusions sur la partie matérielle du projet

A ce stade, il est important d'exercer notre esprit critique et de se poser quelques questions sur les caractéristiques, qualités et défauts du matériel créé. Des réponses nous sont données dans Quartus, soit dans la fenêtre hierarchie du projet , dans le navigateur mais surtout dans la fenêtre « compilation report ».

Questionnaire

- ✔ Que contient le fichier Nios2_system.ptf ?
- ✔ Quel est le rôle exact de ce fichier ?
- ✔ La séquence « Compile design » (à gauche dans le navigateur) contient 5 phases. A quoi correspond chacune de ces phases ?
- ✔ Quelles sont les ressources Stratix utilisées en terme de :
  - ◆ Nombre de pins ?
  - ◆ Nombre total d'éléments logiques ?
  - ◆ Nombre total d'éléments mémoires ?
  - ◆ Classer par ordre décroissant en termes de surface les différentes parties du NIOS en estimant un pourcentage pour chacune d'entre elles
- ✔ Quelle est la fréquence maximale théorique de fonctionnement ?
- ✔ Quel est l'élément qui limite cette fréquence ?
- ✔ Pourrait-on améliorer cette fréquence sur cette même plateforme ? Si oui, par quel moyen ?

3.4) Option

A ce stade, le matériel est au point, on peut parfaitement quitter Quartus.

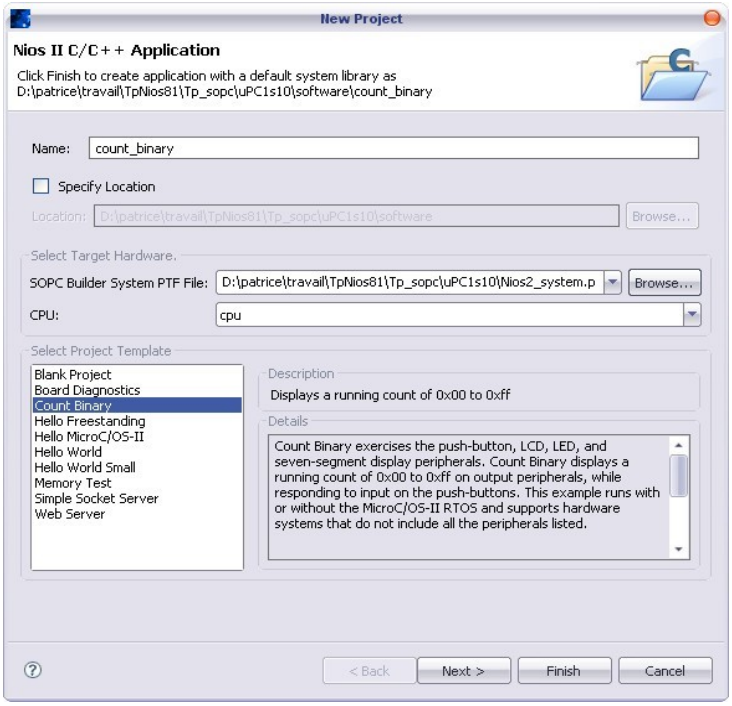
4) Logiciel associé au NIOS

Tous les programmes sont écrits en langage C ou C++. L'outil de développement est une suite Eclipse adaptée par Altera et nommée ici **NIOS II IDE.** On va créer un premier projet basé sur un exemple déjà écrit par Altera.

Attention ! Toujours vérifier que votre espace de travail logiciel est le répertoire **software** présent à l'intérieur de votre projet uPC1S10

4.1) Construction de l'application logicielle

- ➔ **Altera >NIOS II EDS 8.1> NIOS II IDE** invoque l'environnement de travail logiciel.
- ➔ **File > Switch workspace** permet de vérifier que le répertoire de travail est celui souhaité (*Tp_sopc/software*)
- ➔ **File>New Nios II C/C++ Application** ouvre une fenêtre de construction du projet logiciel
- ➔ Dans select Project template , choisir **count_binary**
- ➔ Renommer count_binary_0 en *count_binar*y et rechercher le fichier Nios2_system.ptf comme cible pour le SOPC Buider Ptf File.
- ➔ **Next**. Dans la fenêtre suivante, choisir: **Select or create a system library** puis cliquer sur New **System Library Project**
- ➔ Dans la fenêtre Nios II System library fixer **name** à : **uPC1S10_syslib . C**e sera le nom de la bibliothèque commune à tous les applications logicielles associées à ce même matériel
- ➔ **Finish**. On revient sur la fenêtre précédente. **Finish.**



4.2) Compilation

➔ Dans la fenêtre navigateur, sélectionner *count_binary* puis clique-droit et **Build project** . Cette action lance tout le processus compilation / édition de lien / création de l'éxecutable.

Remarque : Si le FPGA est déjà configuré , on peut directement se servir de la commande **Run As Nios II Hardware** qui compile et procéde au démarrage du programme.

4.3) Configuration du FPGA

Cette partie est optionnelle car la configuration a pu être faite à partir de Quartus . Si ce n'est le cas, ou si la configuration a été perdue (carte éteinte par exemple), il faut bien entendu que le FPGA soit configuré avec un système NIOS avant de penser pouvoir y faire exécuter un programme.

➔ **Tools > Quartus II Programmer...** invoque l'outil de programmation. La marche à suivre dés lors est la même que celle expliquée précédemment.

4.4) Lancement du programme

➔ Dans la fenêtre navigateur, sélectionner *count_binary* puis clique-droit et **Run As> Nios II Hardware** . Cette action lance le processus d'exécution du programme sur la cible NIOS.

Questions

➔ Dans quelle mémoire se trouve le programme exécutable ? Fonctionne t-il toujours si l'on le charge dans une autre mémoire ? Comment faites-vous ?
➔ Quelle est la taille mémoire du programme exécutable ? Où voit-on cette information ?
➔ Que contient le fichier *system.h* se trouvant sous uPC1S10_syslib>Debug>Sysyem description ?
➔ Quand et comment a été créé ce fichier ?

4.5) Lancement du programme avec débugger

➔ Dans la fenêtre navigateur, sélectionner *count_binary* puis clique-droit et Debug **As> Nios II Hardware** . Cette action lance le processus d'exécution du programme sur la cible NIOS avec débugger. Un point d'arrêt est positionné en début de programme. On peut ensuite faire du pas à pas, visualiser le code assembleur, l'état des variables , poser d'autres point d'arrêt.

Pour les détails du débugger, on se réfèrera à la documentation de l'ide se trouvant à l'adresse suivante : http://127.0.0.1:60955/help/index.jsp et on explorera la rubrique **Tutorials > Debugging the Project**

4.6) Simulation de l'exécution du programme (ISS)

➔ Dans la fenêtre navigateur, sélectionner *count_binary* puis clique-droit et **Run As> Nios II Instruction Set Simulator** .