

# Informatique Industrielle

## Synthèse logique



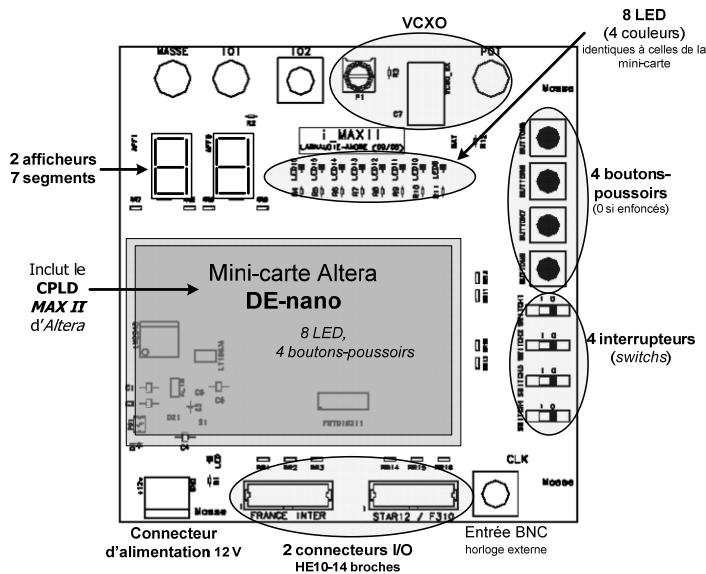
salle de TP

Manuel d'utilisation

## Chaîne de synthèse *Quartus II*

simulateur *Modelsim*

### Kit CPLD *iMaxII* /carte *DE2*



Carte *DE-nano* d'Altera

**Kit *iMaxII*** = Carte *DE-nano* enfichée sur sa carte d'extension *X-MaxII*

Agnès Priou

IUT de Cachan - Geii1

# Où se procurer les logiciels ?

Le logiciel Quartus II et le simulateur Modelsim sont téléchargeables gratuitement sur le site d'Altera ([www.altera.com](http://www.altera.com)). Ils ne nécessitent pas de licence, à condition de prendre la bonne version :

- Quartus II **Web Edition** (en février 2011, c'est la version 10.1)
- Modelsim-**Altera Starter Edition**

Notez, pour les anciens utilisateurs de Quartus II, que la version 10 n'inclut plus de simulateur : il faut utiliser un simulateur extérieur comme Modelsim (logiciel indépendant à installer séparément).

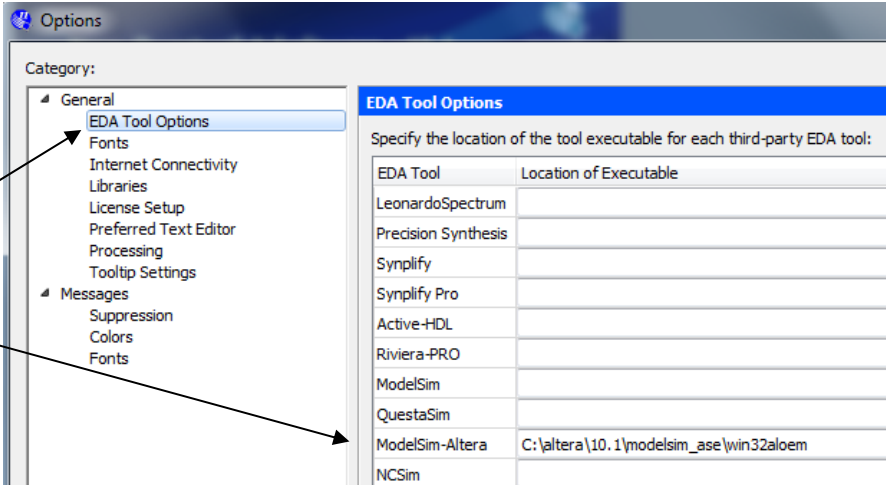
Les deux versions de ce manuel d'utilisation (QII 9 avec son simulateur intégré ou QII 10) sont disponibles en ligne sur <http://formation.u-psud.fr/courses/IUTCGEIS1TN> (tapez « agnes priou » sous Google).

## Paramétrage initial de Quartus II

Lors de la première utilisation de Quartus II, réalisez les changements suivants sur les options par défaut :

- menu **Assignments** → sous-menu **Settings** → dans la catégorie **Analysis & Synthesis Settings** → clic sur le bouton **Moore Settings** → dans l'option **Block Design Naming**, il faut choisir QuartusII (et non *Auto*).  
⇒ Vous éviterez ainsi que Quartus II renomme (mal) les bits de bus isolés.
- Si vous souhaitez utiliser le simulateur Modelsim (logiciel indépendant, à installer séparément) : il faut que le chemin d'accès de Modelsim soit connu :

menu **Tools**  
→ sous-menu **Options**  
→ dans la catégorie *General*  
    > *EDA Tools Options*  
→ compléter le *path* de la ligne ModelSim-Altera



EDA Tool	Location of Executable
LeonardoSpectrum	
Precision Synthesis	
Synplify	
Synplify Pro	
Active-HDL	
Riviera-PRO	
ModelSim	
QuestaSim	
ModelSim-Altera	C:\altera\10.1\modelsim_ase\win32aloem
NCSim	

# Table des matières

<b>A.</b>	<b>Le kit CPLD iMaxII de l'IUT .....</b>	<b>5</b>
I.	La micro-carte <i>DE-nano</i> d'Altera .....	6
II.	La carte d'extension X-maxII (IUT) .....	7
<b>B.</b>	<b>La chaîne de synthèse Quartus II .....</b>	<b>8</b>
	<b>Pour commencer : créer le projet .....</b>	<b>10</b>
	<b>Obtenir la description complète du système logique.....</b>	<b>10</b>
	Présentation .....	10
	Ecrire une description VHDL : .....	10
	Dessiner un schéma logique : .....	11
	Créer le symbole associé à une description : .....	11
	Compiler le projet : .....	12
	Exporter un schéma sous forme VHDL : .....	12
	<b>Simuler.....</b>	<b>12</b>
	<b>Effectuer la synthèse pour le kit iMaxII ou DE2 .....</b>	<b>12</b>
	Consulter le fichier csv pour connaître les ressources de la carte .....	12
	Fabriquer le schéma de synthèse et attribuer une broche à chaque port .....	13
	Compiler le projet pour fabriquer le fichier de programmation .....	13
	<b>Programmer le CPLD du kit <i>iMaxII</i> ou le FPGA de la carte DE2.....</b>	<b>14</b>
<b>C.</b>	<b>Des p'tits trucs utiles dans les schémas QII.....</b>	<b>15</b>
<b>D.</b>	<b>Le simulateur Modelsim .....</b>	<b>16</b>
I.	Préparer Quartus II pour travailler avec Modelsim .....	16
II.	Lancer Modelsim à partir de Quartus II.....	17
	Compiler sous Quartus II (pour vérification).....	17
	Lancer Modelsim à partir de Quartus II .....	17
III.	<b>Effectuer une simulation avec Modelsim.....</b>	<b>17</b>
	Charger le simulateur avec le fichier à simuler : .....	17
	Choisir les signaux à visualiser dans la fenêtre Wave.....	18
	Appliquer les vecteurs de test (= les valeurs des entrées) / Simuler .....	18
	Effectuer les calculs de la simulation (Run) : .....	19
	Exemple de script Modelsim (fichier .do).....	19
	En savoir plus sur les scripts et les test benches .....	20
IV.	<b>Quand Modelsim est utilisé seul.....</b>	<b>20</b>
	Créer un projet : .....	20
	Compiler : .....	20

<b>E.</b>	<b>Création d'un automate sous forme graphique .....</b>	<b>21</b>
	Un exemple d'automate.....	21
	Saisie du diagramme d'état (State Machine) .....	21
	Une fois l'automate sous forme VHDL.....	23
<b>F.</b>	<b>Check list en cas de problème .....</b>	<b>24</b>

## A. Le kit CPLD iMaxII de l'IUT

Le kit CPLD *iMaxII* est construit autour de la **micro-carte DE-nano d'Altera** (à base de CPLD), enfichée sur une **carte d'extension X-MaxII** développée à l'IUT en 2008 par B. Larnaudie et C. André.

L'ensemble *DE-nano* + carte d'extension *X-MaxII*, avec son câble de programmation USB et son alimentation 12 V, constitue le **kit iMaxII**.

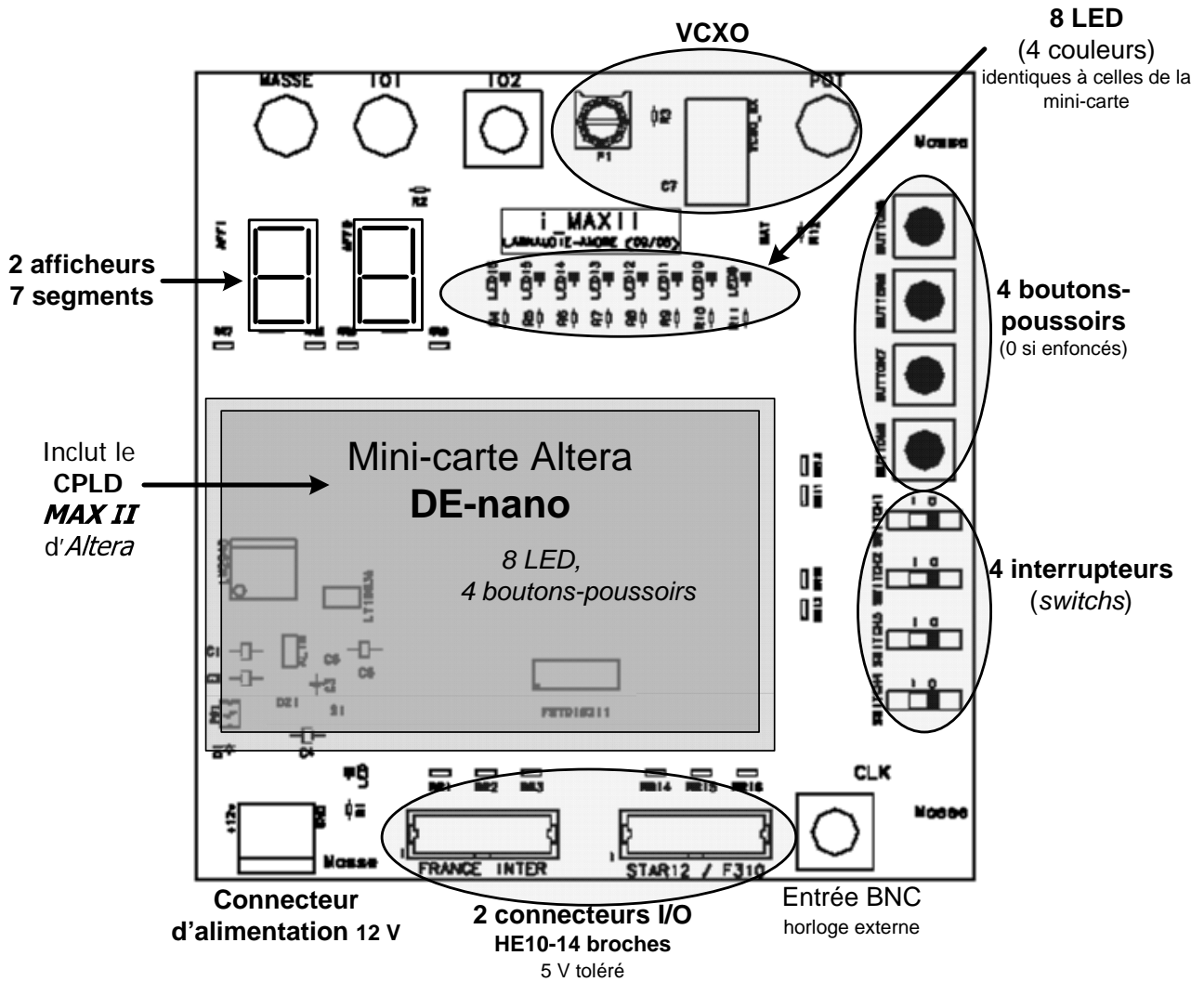


Fig. 1. Le kit *iMaxII* : carte *DE-nano* d'Altera enfichée sur la carte d'extension IUT

## I. LA MICRO-CARTE *DE-NANO* D'ALTERA

Pour plus de détails, voir sa documentation *Max\_II\_Micro\_UserManual\_v1.31.pdf* (en ligne).

La carte *DE-nano* peut être **utilisée seule, alimentée par le seul port USB**. Elle contient :

- un gros CPLD de la famille *MaxII*, le EPM2210F324 (2210 *Logic Elements*, 324 *pins*, 272 *I/O pins*) ;
- **4 boutons-poussoirs** avec anti-rebond ; ils fournissent un **'0' logique quand ils sont enfoncés** ;
- **8 LED** (2 rouges, 2 jaunes, 2 vertes, 2 bleues) ; elles **s'allument** quand on y applique un **'0'** ;
- une horloge **50 MHz** pour le CPLD ;
- deux zones de prototypage, utilisées pour communiquer avec la carte d'extension *X-MaxII*.

La carte *DE-nano* est **alimentée en 3,3 V**. L'alimentation est fournie par le câble USB quand la carte *DE-nano* est utilisée seule, ou par la carte d'extension quand elle y est enfichée : cela rend le kit *iMaxII* indépendant du PC, donc utilisable en robotique.

Attention ! **Les entrées/sorties se font en 3,3 V** sur la carte *DE-nano*. Mais la carte d'extension, que nous utiliserons pour toutes les I/O (via les connecteurs d'extension HE10), est protégée et **accepte en entrée 5 V ou 3,3 V**.

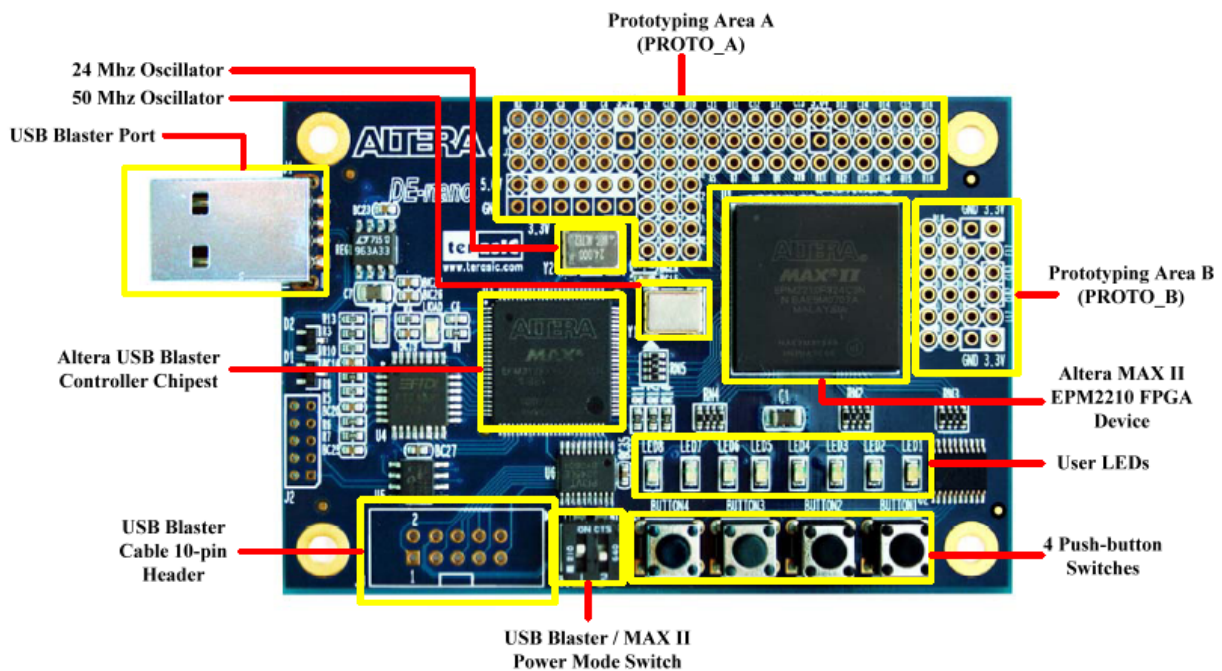


Fig. 2. La micro-carte *DE-nano* d'Altera (source Altera)

Note : la carte *DE-nano* du kit *iMaxII* a été très légèrement modifiée pour s'adapter à la carte d'extension (suppression des deux inductances de l'alimentation via le port USB) : elle ne peut plus servir seule une fois enfichée, car elle n'est plus alimentée par le port USB.

Ne touchez pas aux *switchs* de programmation à côté des boutons-poussoirs (*switch 1* = Up, *switch 2* = Down). Ils permettent de travailler en mode normal (programmation CPLD par le câble USB, mode JTAG).

## II. LA CARTE D'EXTENSION X-MAXII (IUT)

La carte d'extension *X-MaxII* a été développée à l'IUT en 2008 par B. Larnaudie et C. André pour répondre à nos besoins dans tous les semestres.

Elle fournit principalement des moyens de dialogue supplémentaires (afficheurs 7 segments, LED, interrupteurs et boutons-poussoirs, connecteurs d'extension,...), ainsi qu'un connecteur d'alimentation 12 V qui permet de l'alimenter par une batterie et donc de la rendre autonome. Elle dispose aussi d'un VCXO pour la maquette de Semestre 3.

Voici la liste des ressources offertes par la carte d'extension *X-MaxII* (schéma disponible en ligne) :

- 2 **afficheurs 7 segments** (sans décodeur) ; une LED de segment **s'allume** avec un **'0'**.
- 8 **LED** identiques à celles de la carte *DE-nano* : 2 rouges, 2 jaunes, 2 vertes, 2 bleues ; il faut appliquer un **'0'** pour allumer une LED.
- 4 **boutons-poussoirs** ; ils fournissent '1' au repos, **'0' si enfoncés**.
- 4 **interrupteurs** (*switchs*).
- 1 **VCXO** (*Voltage Control Crystal Oscillator*), utilisé en E&R de S3.
- 1 entrée d'horloge (BNC).
- 2 **connecteurs d'extension** HE10 14 broches : l'un est prévu pour dialoguer avec les cartes microcontrôleur *Star12* ou *F310-XP* (l'alimentation 5 V y est disponible en option), l'autre est destiné à l'E&R de S3 (entrée VCXO). **Toutes les entrées sont protégées pour supporter 5 V**, mais le CPLD travaille en 3,3 V.

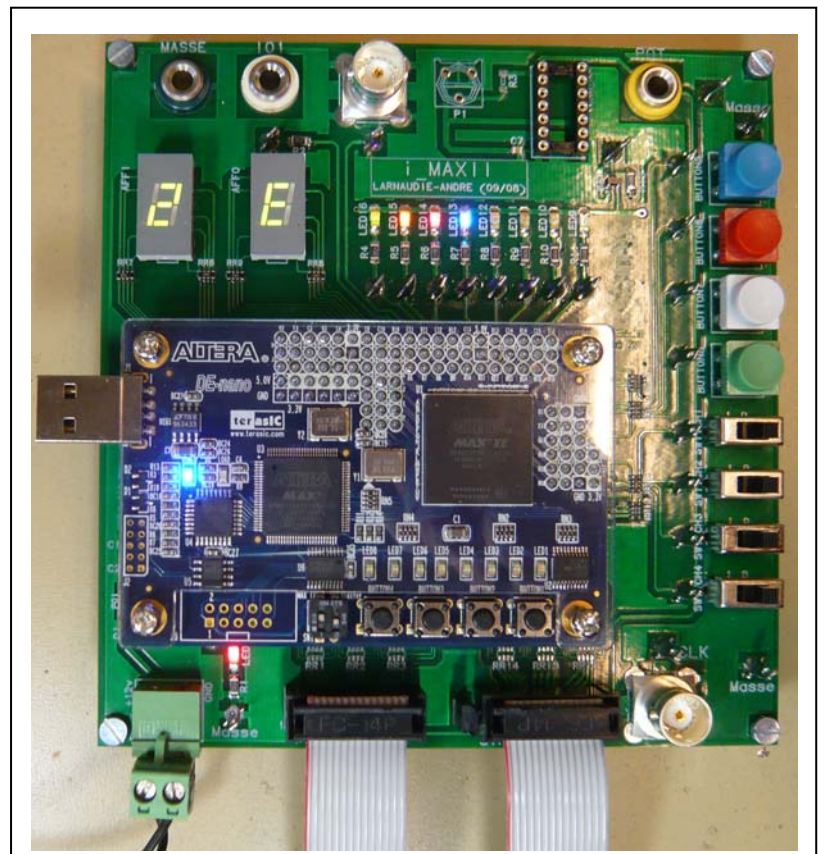
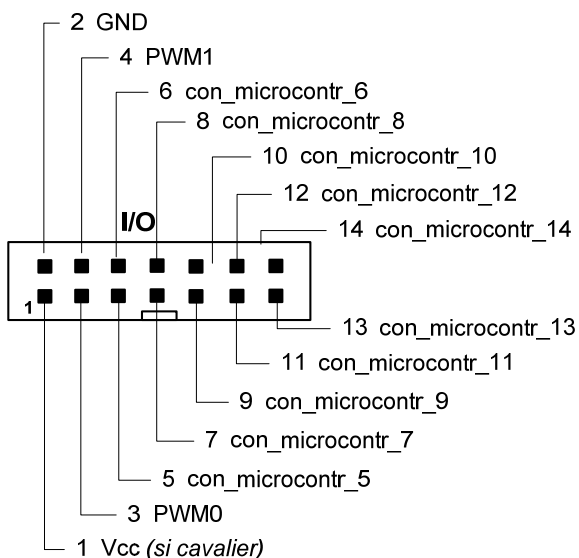


Attention, les **sorties du CPLD se font en 3,3 V**. Il faut vérifier que c'est suffisant pour les entrées de circuit commandées.

La carte d'extension est alimentée par une tension de **12 V** et dispose de **deux régulateurs 5 V et 3,3 V**.

### Connecteur « microcontrôleur » *Star12/F310*

(cf *iMaxII\_pin\_assign\_V7.csv*)

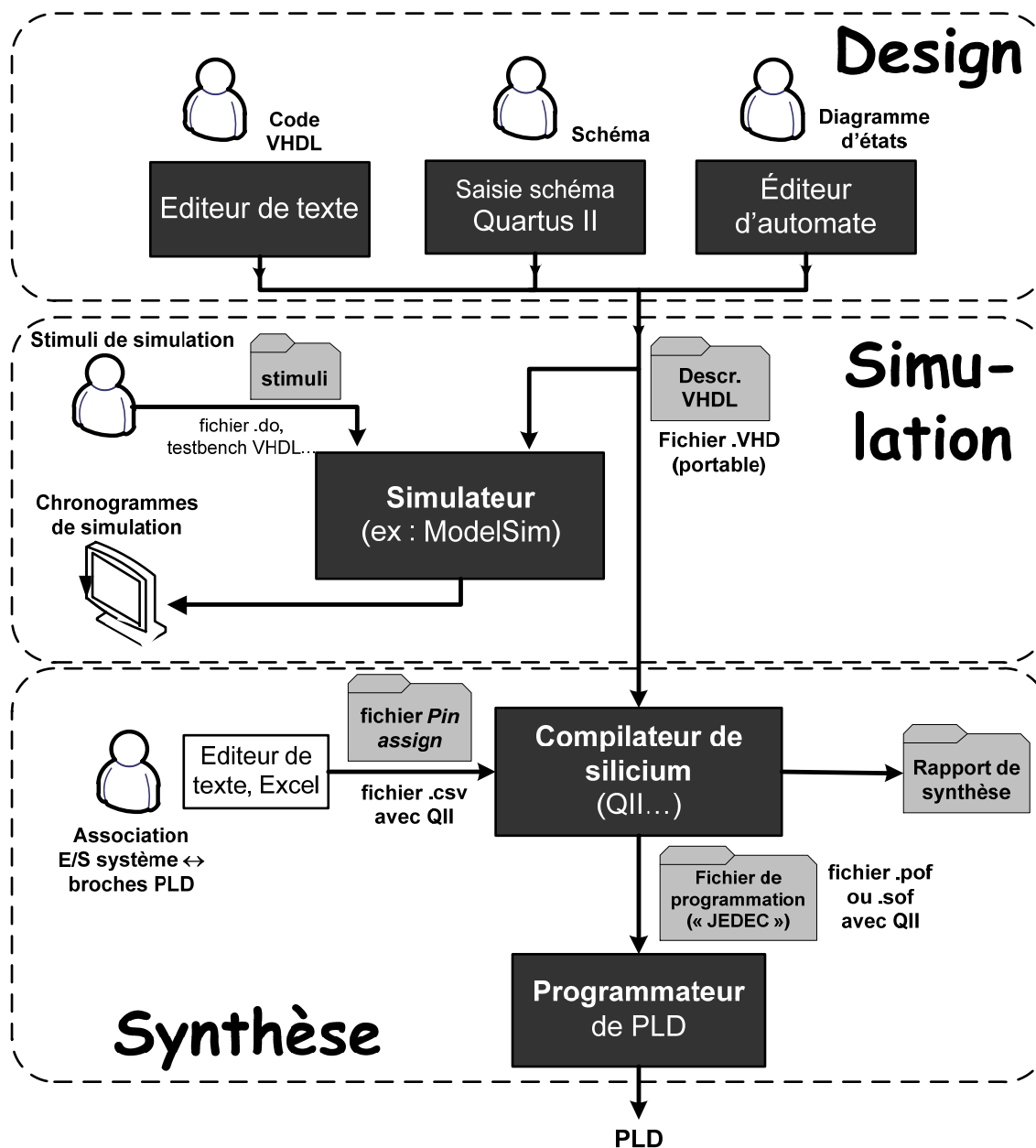




## B. La chaîne de synthèse Quartus II

Le logiciel *Quartus II* est téléchargeable sur le site d'Altera (*Web Edition, gratuite sans licence*), ainsi que le simulateur *Modelsim* (-Altera Starter Edition).

La succession des trois grandes étapes à réaliser pour implanter (réaliser) un système logique dans un PLD est représentée sur le flot de synthèse suivant, où *QII* est une abréviation pour *Quartus II* :



Voici la liste des étapes à réaliser, quelle que soit la carte cible :

- créer le **projet** ;
- faire la description complète du système : fichiers **VHDL** (\*.vhd) et **schémas** (\*.bdf) ;
- **simuler** (chaque bloc, puis l'ensemble) ;
- effectuer la **synthèse** sur la carte choisie, en associant les broches du système aux ressources disponibles sur la carte (utilisation du fichier *csv* de la carte).





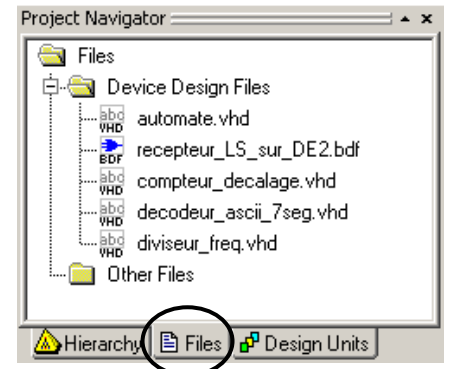
## Pour commencer : créer le projet

Lancer le logiciel **Quartus II** d'Altera.

Créer un projet (**File > New Project Wizard**) avec les informations suivantes :

- Dossier de travail **c:\tmp\xxx** sans espaces ni accents (lettres, chiffres et caractère souligné)
- Choisir un circuit cible (celui de votre carte !). Par exemple :
  - ▶ Carte **iMaxII** : famille **Max II** (il faut préciser *All* pour la voir), circuit **EPM2210F324C3** (c'est un gros CPLD)
  - ▶ carte **DE2** : choisir la famille **Cyclone II** et le circuit **EP2C35F672C6** (FPGA).

On peut ajouter au projet tous les fichiers *VHDL* ou *bdf* existants. Le **Navigateur de Projet** se présente de la façon ci-contre (on visualise ici l'onglet *Files*).



### Pour ajouter un fichier au projet à tout moment :

menu *Project* ou clic droit sur le dossier *Files* dans l'onglet *Files* du navigateur de projet, option « *Add/Remove Files in Project* ».



**Sur la carte iMaxII** : vérifier que toutes les broches non utilisées seront considérées comme des entrées trois-états: menu **Assignments > Device**, puis clic sur le bouton **Device and Pin Options**, puis dans l'onglet **Unused Pins**, choisir l'option **As Input tri-stated** et non **As Output driving ground** (utilisé par défaut ! ce qui crée un conflit sur la carte d'extension).

## Obtenir la description complète du système logique

### Présentation

Chaque bloc élémentaire du schéma à réaliser (registres, compteurs, automates...) doit être codé en VHDL ou dessiné sous forme d'un schéma, puis simulé **seul** : c'est l'étape de **développement et de test unitaire**. L'étape de simulation peut être réalisée dans un simulateur comme *Modelsim*.

On **rassemble** ensuite les blocs élémentaires pour fabriquer un schéma plus complexe : cet assemblage peut être effectué en VHDL (instructions *port map*) dans un éditeur quelconque, ou sous la forme d'un **schéma** dans le logiciel *Quartus II*. Avantage de cette deuxième solution : on dispose d'un schéma propre et complet, plus explicite qu'une suite d'instanciations. Et Quartus peut le traduire en VHDL pour la simulation !

A partir d'un schéma, on peut **obtenir un code VHDL qui est la traduction du schéma**. Ce code VHDL est portable (contrairement au schéma) et peut être proposé au simulateur de votre choix.



Attention : la succession des trois étapes *Codage VHDL / Dessin d'un schéma / Création de symboles* dépend de votre niveau d'apprentissage et du travail souhaité.

### Ecrire une description VHDL :

(non utilisé en DUT-S1)

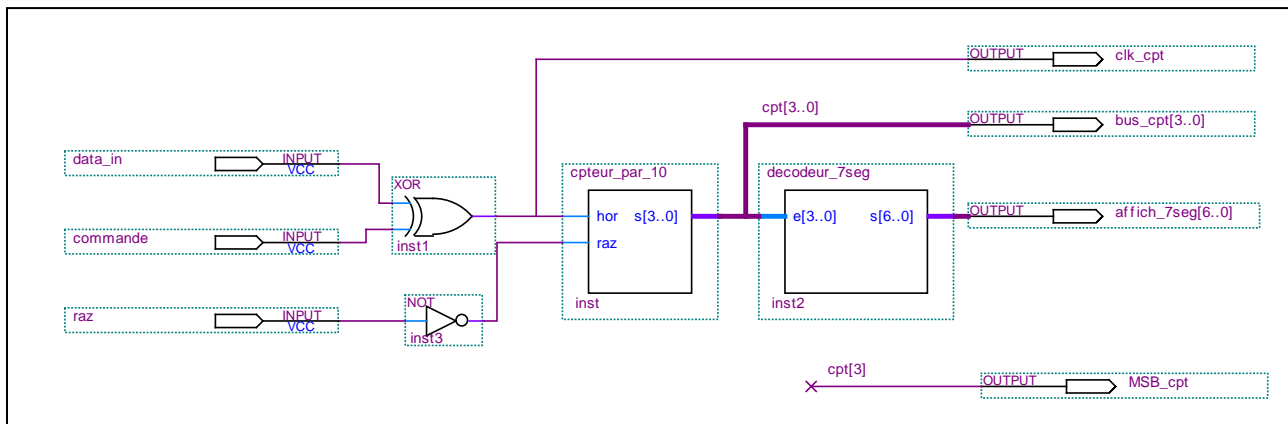
**File > New** option VHDL File puis **ajouter le fichier au projet**

Une fois le code VHDL écrit, il faut **fabriquer son symbole** (cf page suivante « Créer le symbole associé à une description »).


## Dessiner un schéma logique :



(fig. 3 et 4)

- Créer un schéma vierge (extension . bdf) : **File > New** option **Block Diagram/Schematic File**.






**Fig. 3. Exemple de schéma (mélangeant portes et symboles), dans l'étape Design.** Voir aussi Partie C.

Les composants du projet et les bibliothèques de *Quartus* sont accessibles en cliquant sur l'outil . Les différentes sections permettent d'accéder aux éléments disponibles : les symboles que vous avez déjà créés (à partir d'une description VHDL ou d'un schéma) sont dans *Projet*, les portes logiques et les ports d'E/S sont dans *QII>Primitives* ou *Pins*.

- Insérer les blocs désirés, les relier avec des connexions simples ou des bus<sup>1</sup>, en utilisant un des deux outils de liaison (choisir le bon !)  pour un fil ou  pour un bus. Les bus apparaissent en gras.
- **Nommer chaque équipotentielle utile** (clic droit, *Propriétés*) avec un nom bien choisi : par exemple, *monFil* pour un fil  
*monBus[3..0]* pour un bus de 4 fils (attention aux crochets et à l'ordre : *s[3..0] ≠ s[0..3]*)  
*monBus[2]* pour le bit 2 du bus *MonBus*

**Inutile de connecter physiquement un fil de bus isolé (ou un sous-bus) à son bus, car le nom suffit à faire l'association : sig[5] est forcément connecté au bus sig (bit 5), sig[3..1] aussi.**

- Terminer le schéma en disposant les **ports d'entrée et de sortie**  (outil  ou section *Pins* de l'outil ). Si vous êtes à l'étape *Design*, choisissez des noms évocateurs, ils apparaîtront à l'intérieur du symbole que vous allez créer ensuite. Si vous dessinez un schéma « de synthèse », les noms de ces ports doivent être ceux des ressources physiques de votre carte (cf paragraphe « Attribuer une broche à un port »).

 les noms de port suivent les mêmes règles que les noms de fils (attention aux bus !)


- Sauvegarder et ajouter le fichier au projet.

## **Créer le symbole associé à une description pour l'insérer dans un schéma de niveau supérieur ou un schéma de synthèse :**

Cette étape est indispensable si on veut fabriquer un schéma pour assembler des blocs élémentaires (comme sur la Figure 3), ou si on veut créer le schéma pour la synthèse (exemple Figure 4).

Sélectionner chaque fichier VHDL ou schéma (sa fenêtre d'édition doit être active → clic dedans), puis


**File > Create/Update > Create Symbol Files for Current File**

Chaque symbole créé devient accessible dans l'éditeur de schéma : *Symbol Tool* , section **Project**.

<sup>1</sup> Un bus est un ensemble de *n* fils. *Équipotentielle* est le mot savant pour désigner un fil.


## Compiler le projet :

- ▶ Vérifier que toutes les descriptions (VHDL, schémas) sont dans le projet : menu *Project* ou clic droit sur *Files* dans l'onglet *Files* du navigateur de projet, « *Add/Remove Files in Project* ».

 Quartus n'apprécie pas (« *duplicate entity* ») que le projet contienne simultanément un schéma et sa traduction VHDL. Dans ce cas (obtenu en préparant la simulation), gardez le fichier VHDL, seul exportable pour la simulation.

- ▶ Si ce n'est pas déjà fait, **choisir le schéma ou le fichier VHDL au plus haut niveau** (« au top ») : clic droit sur le nom de ce fichier dans l'onglet *Files* > **Set at Top-Level Entity**

⇒ l'entité correspondante apparaît dans l'onglet *Hierarchy* du Navigateur de Projet (vérifiez).

 Le fichier au top dépend de l'étape en cours de réalisation : c'est un fichier **VHDL** (celui associé au schéma de principe) pour la simulation, mais c'est le **schéma de synthèse** pour la réalisation sur la carte.


- ▶ Compiler : outil **Analysis & Synthesis** (*simulation*) ou **Start Compilation** (*synthèse*)

Note : le rapport complet de la compilation est disponible avec l'icône **Compilation Report**.


## Exporter un schéma sous forme VHDL :

On peut exporter un schéma de principe (jamais le schéma de synthèse) vers un autre logiciel (comme le simulateur *ModelSim*) sous la forme d'une description VHDL. Il suffit de demander sa **traduction en code VHDL** (attention, la fenêtre d'édition du schéma doit être active) :

### File > Create/Update > Create HDL Design File

 Le fichier VHD généré est rajouté automatiquement au projet. Eliminez le schéma \*.bdf du projet (en le gardant ouvert dans la fenêtre d'édition pour consultation). QII n'apprécie pas (« *duplicate entity* ») que le projet contienne simultanément un schéma et sa traduction VHDL. Il faut garder le fichier VHDL, seul exportable pour la simulation.

Notez que lors de la traduction, les bus du port apparaissent toujours sous la forme de signaux **std\_logic\_vector**. Il sera parfois nécessaire, si *Quartus II* signale des types incompatibles, de transformer en **std\_logic\_vector** (le type le plus portable) les bus *unsigned* du port des entités décrites en VHDL.

 Une bonne habitude à prendre : ne sortir d'une entité qu'avec des bus **std\_logic\_vector**. Comme les bus internes sont souvent en *unsigned*, il faut ajouter des conversions si nécessaire.

## Simuler

Cette étape doit être réalisée avant de passer à la synthèse proprement dite. A partir de la version 10 de Quartus II, c'est le simulateur *Modelsim* (logiciel indépendant) qui est le meilleur choix.

Voir la partie spécifique à la simulation.

## Effectuer la synthèse pour le kit iMaxII ou DE2

### Consulter le fichier csv pour connaître les ressources de la carte

De nombreuses **broches**<sup>2</sup> du circuit PLD de la carte *iMaxII* ou *DE2* sont câblées à une LED, un interrupteur (*switch*), un bouton-poussoir, un afficheur 7 segments, une horloge quartz, un connecteur d'extension, etc. Pour la synthèse, chaque **port** d'entrée/sortie (□) de notre système logique doit être connecté à une ressource bien choisie de la carte (**broche** du PLD).

Cette association **port ↔ ressource** (*pin*) se fait simplement, en choisissant le « bon » nom pour chaque port (□) et en important un fichier (fichier *csv* de *pin assignment*) qui liste les noms des ressources de la carte et les broches de PLD associées. Les étapes suivantes vous détaillent comment faire cette association.

---

<sup>2</sup> Broche = *pin* en anglais. *Pinout* = brochage = liste des associations *pin-port*

Pour associer un **port** du **système logique** à une **broche** du **circuit**, vous allez utiliser le fichier « compatible tableur » (.csv) fourni. Ce fichier est spécifique à chaque carte. Celui du kit *iMaxII*, appelé *iMaxII\_pin\_assignments.csv*, est disponible en ligne<sup>3</sup>. Il se présente comme suit, avec le **nom** de chaque ressource qui apparaît en début de ligne (le *csv* de la carte *DE2* est différent, mais suit le même principe) :

# Copyright (C) 1991-2008 Altera Corporation	
# Quartus II Version 8.0 Build 215 05/29/2008 SJ Web Edition	
...	
<b>aff0[0]</b> ,Output,PIN_G3,1,3.3-V LVTTTL,,aff0[7..0],16mA	← Afficheur 7 seg n°0
...	
<b>aff0[7]</b> ,Output,PIN_C3,1,3.3-V LVTTTL,,aff0[7..0],16mA	
<b>aff1[0]</b> ,Output,PIN_B5,2,3.3-V LVTTTL,,aff1[7..0],16mA	
...	
<b>aff1[7]</b> ,Output,PIN_C10,2,3.3-V LVTTTL,,aff1[7..0],16mA	← Afficheur 7 seg n°1
<b>button[8]</b> ,Input,PIN_F18,3,3.3-V LVTTTL,,button[8..1],16mA	← Bouton-poussoir n°8
...	
<b>button[1]</b> ,Input,PIN_U15,4,3.3-V LVTTTL,,button[8..1],16mA	
<b>clock_50</b> ,Input,PIN_J6,1,3.3-V LVTTTL,,16mA	
<b>Led[16]</b> ,Output,PIN_C12,2,3.3-V LVTTTL,,Led[16..1],16mA	← LED n°16
...	
<b>Led[1]</b> ,Output,PIN_U13,4,3.3-V LVTTTL,,Led[16..1],16mA	
<b>switch[1]</b> ,Input,PIN_L18,3,3.3-V LVTTTL,,switch[4..1],16mA	Etc.
...	
<b>switch[4]</b> ,Input,PIN_M18,3,3.3-V LVTTTL,,switch[4..1],16mA	
<b>vcxo</b> ,Input,PIN_J13,3,3.3-V LVTTTL,,16mA	

### Fabriquer le schéma de synthèse et attribuer une broche à chaque port

1. Créez un **NOUVEAU** schéma vierge, nommé *toto\_synthese.bdf* si votre système s'appelle *toto* et ajoutez-le au projet.
2. Dessinez le schéma **de synthèse** (exemple figure 4), obtenu en important le **symbole** de votre système (à fabriquer si pas encore fait : **File>Create/Update>Create Symbol Files for Current File**) et et lui ajoutant des ports d'E/S.

*Des portes optionnelles peuvent permettre d'adapter votre schéma aux spécificités de la carte (NOT...).*

3. **Nommez chaque port** (□) de votre schéma avec un **nom lu dans le fichier csv** (celui de la ressource que vous voulez connecter sur le port). Par exemple, *LED[j]* pour connecter un port de sortie à la LED *j*, *CLOCK\_50* pour relier un port d'horloge au quartz 50 MHz de la carte, etc.



(déconseillé) on peut aussi modifier le fichier *.csv* pour y imposer le nom du prt utilisé sur le schéma :

**MonNomDePort**,Output,PIN\_U13,4,3.3-V LVTTTL,,Led[16..1],16mA permet de renommer la Led[1]

4. Quand tous les noms de port (□) du schéma sont ceux du fichier *csv*, **importez le fichier csv** (qui peut se trouver dans un autre dossier) :

**Assignements > Import Assignments**

### Compiler le projet pour fabriquer le fichier de programmation

5. **Placez le schéma de synthèse au plus haut niveau** dans le projet (clic droit sur son nom dans l'onglet *Files* > Set at Top-Level Entity, vérifiez dans l'onglet *Hierarchy* du Navigateur de Projet).

Pour la carte *iMaxII* : avez-vous configuré (lors de la création du projet) les broches non utilisées comme des **entrées trois-états** ? menu **Assignments > Device**, puis clic sur le bouton **Device and Pin Options**, puis dans l'onglet **Unused Pins**, choisir l'option *As Input tri-stated* (et non *As Output driving ground*).

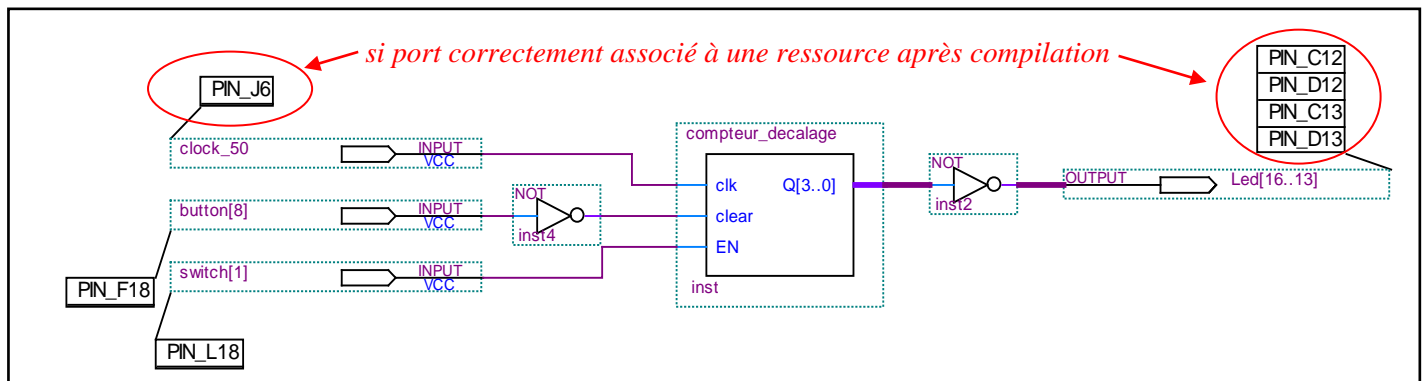
Lors de la compilation, **le message suivant ne doit pas apparaître** : "Warning: The Reserve All Unused Pins setting has not been specified, and will default to 'As output driving ground'." → sinon, après la programmation, toutes les LED vont s'allumer et toutes les entrées non utilisées, de type boutons-poussoirs et interrupteurs, peuvent provoquer un court-circuit *Vcc-masse* qui coupera l'alimentation de la carte.



<sup>3</sup> Tapez *Agnes Priou* sous Google pour accéder au cours *Techniques numériques – Semestre 1*.

6. Compilez : outil **Start Compilation** ou menu **Processing > Start Compilation**

7. **Vérifiez** que chaque port a bien été associé à une broche de la carte : regardez le schéma *bdf* (il faut avoir coché View > Show Location Assignments). Voici par exemple ce que vous devez obtenir<sup>4</sup> sur le schéma *bdf* (parfois la mise à jour de l'affichage se fait attendre : fermez, puis rouvrez le *bdf*) :



**Fig. 4. Schéma de synthèse.** Inclut un seul symbole : celui du système à synthétiser. Grâce à l'importation du fichier *csv*, chaque port est associé à une broche du circuit (port *clock\_50* → broche *PIN\_J6*). Notez qu'un port peut être **un fil ou un bus**. On peut rajouter des inverseurs (ici, sur un fil et sur un bus) pour adapter le schéma à la carte.

Remarque : le « **schéma de synthèse** » de la figure 4 comporte uniquement le **symbole à synthétiser**, avec ses ports. Il est idéal pour la lisibilité et le test. Par exemple, on y voit clairement que le signal *clear* est activé par le bouton-poussoir n°8. On peut inverser des signaux pour tenir compte des spécificités de la carte (sur la Fig. 4, le niveau '0' du bouton-poussoir enfoncé est compensé par l'inverseur pour obtenir un *clear* actif à 1).

## Programmer le CPLD du kit *iMaxII* ou le FPGA de la carte DE2

Alimenter la carte cible choisie. Connecter la carte au PC par le câble USB. Sur le kit *iMaxII*, vérifier que les petits interrupteurs de programmation sont bien positionnés (*switch 1 Up, switch 2 Down*).

Lancer l'utilitaire de programmation avec le bouton **Programmer** .

Vérifications : sélectionner si nécessaire, avec le bouton "*Hardware Setup*", le matériel *USB\_Blaster[USB-0]*, qui est visible si la carte branchée et reconnue et si le pilote (*driver*) de programmation *USB Blaster* est installé. Le mode de programmation est *JTAG*.

Normalement, le dernier fichier « JEDEC » obtenu est déjà affiché (fichier *.pof* sur la carte *iMaxII* ou *.sof* sur la carte *DE2*), ainsi que le circuit cible de votre carte. **Vérifiez le nom du fichier de programmation et le circuit cible.**

*Si le circuit cible est incorrect : menu **Assignments > Device** et recompilation.*

Cocher la case **Program/Configure**, puis cliquer sur le bouton **Start** pour démarrer la programmation.

C'est tout ! Notez que la programmation du CPLD du kit *iMaxII* restera effective même hors tension, mais pas celle du FPGA<sup>5</sup> de la carte *DE2*.

<sup>4</sup> Cela suppose la **compilation réussie** après importation du fichier *csv* + schéma de synthèse *au top* du projet.

<sup>5</sup> Un FPGA s'efface quand on coupe son alimentation. Il est toujours associé à un circuit (*Config Device*, EPCS16 sur la carte *DE2*) qui mémorise une programmation et reprogramme automatiquement le FPGA à la mise sous tension. C'est ce circuit qu'il faut programmer si on veut retrouver la même programmation à chaque mise sous tension (ne pas le faire à l'IUT).

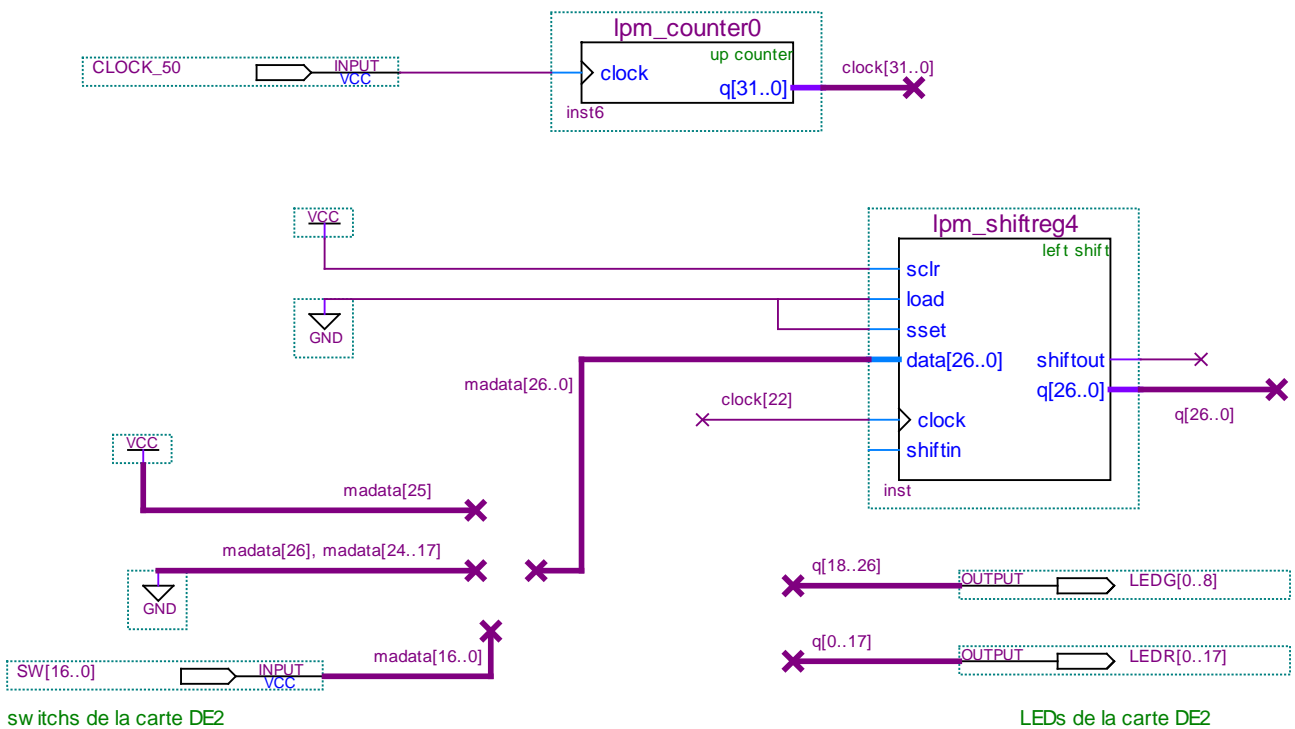
## C. Des p'tits trucs utiles dans les schémas QII

Dans un fichier *bdf*, on peut connecter n'importe quel bus à une **combinaison** quelconque de

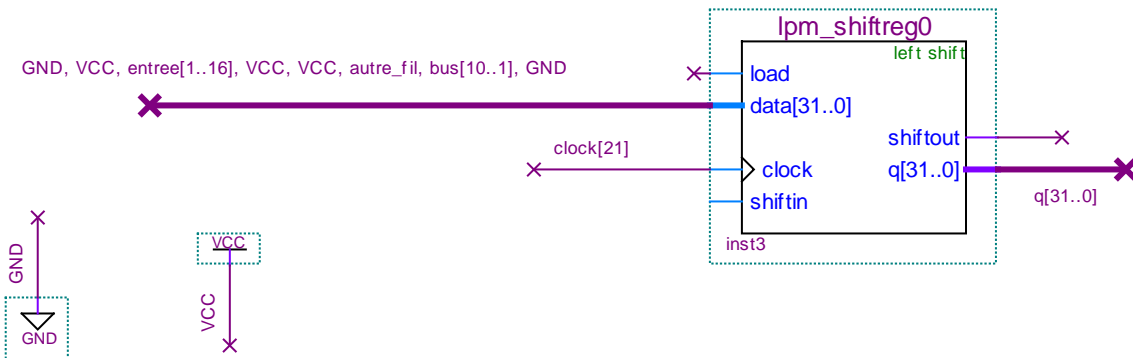
- fils ou bus,
- masse (GND)
- alimentation (Vcc, soit 3,3V sur la carte *iMaxII*).

C'est très utile pour connecter les entrées non utilisées ou pour créer des bus plus grands en rassemblant d'autres bus, des fils et des potentiels constants.

Voici des exemples de ce que vous pouvez faire sur un schéma :



On peut aussi connecter le bus d'entrée de la façon suivante (notez la création des fils GND et VCC) :





## D. Le simulateur Modelsim

La simulation est l'étape qui suit une compilation réussie et doit normalement précéder la programmation du circuit. Elle consiste à **fabriquer des vecteurs de test** (= les entrées à appliquer au système), puis à lancer la simulation proprement dite. A la fin de celle-ci, le simulateur affiche la réponse du circuit aux vecteurs de test sous la forme de chronogrammes.

Modelsim est un logiciel **indépendant** qu'on peut aussi mettre en œuvre à partir de Quartus II.

Il nécessite un projet listant les fichiers **VHDL** à simuler : ce projet est automatiquement **celui de Quartus II** si Modelsim est lancé à partir de QII.

### I. PREPARER QUARTUS II POUR TRAVAILLER AVEC MODELSIM

1. Vous devez disposer d'une version de *Modelsim*, à télécharger sur le site Altera en fonction de votre version de QII.

Par exemple, pour *QII Web Edition 10.1* (à vérifier dans *Tools > License Setup*), prendre le *ModelSim Altera Starter Edition* pour QII 10.1 (qui ne nécessite pas de licence, ainsi que *QII Web Edition*).

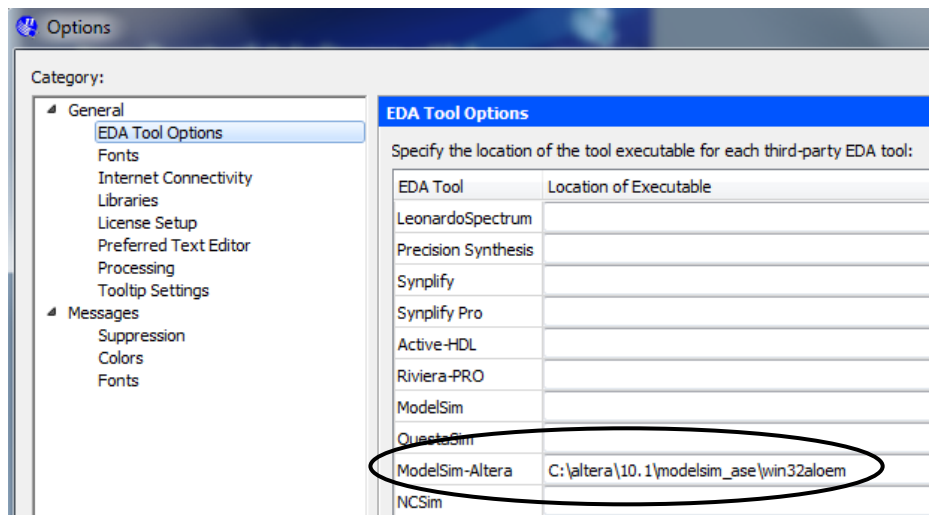
2. Il faut évidemment que le chemin d'accès de Modelsim soit connu :

menu **Tools > Options**

> dossier *General*

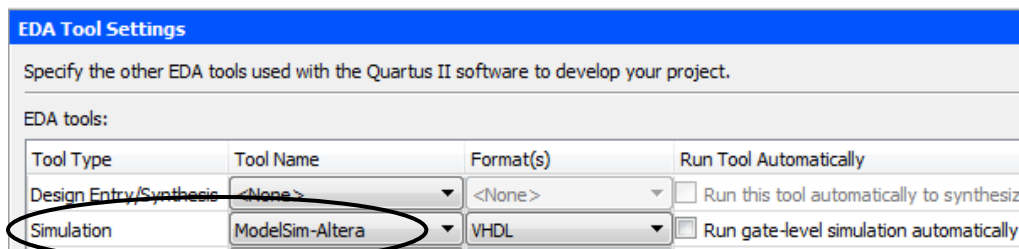
> *EDA Tools Options*

→ compléter le *path* de la ligne *ModelSim-Altera*.



3. Lors de la création du projet Quartus II, il faut demander à **utiliser le Modelsim d'Altera**.

Sinon, on peut le faire plus tard dans QuartusII : **Assignments > Settings > paragraphe EDA Tools settings** :



Le dossier de simulation sera un sous-dossier du dossier de travail (celui du projet), nommé *simulation | Modelsim*.

---

## II. LANCER MODELSIM A PARTIR DE QUARTUS II

Le projet *QuartusII* doit contenir les fichiers **VHDL** à simuler (les seuls reconnus par Modelsim).

💣 Attention, les schémas *\*.bdf* sont ignorés par Modelsim, il faut les **remplacer** dans le projet par leur traduction en VHDL (obtenue avec *File > Create/Update > Create HDL Design File*).

### Compiler sous Quartus II (pour vérification)

Mettre « au Top » le fichier VHDL à simuler, souvent celui associé au schéma de principe (ne jamais utiliser en simulation le schéma de synthèse s'il existe !). Compiler :

outil **Start Analysis & Synthesis** (suffisant en simulation) ou outil **Start Compilation** (plus long)

#### 💣 Les sources de problèmes possibles :

1. Quartus II n'apprécie pas de compiler successivement la description VHDL (.vhd) et la description schématique (.bdf) d'une même entité → éliminez le fichier bdf (non portable).
2. Problème « Pas d'entité au Top », même en présence d'un seul fichier à compiler → clic droit sur le fichier dans l'onglet *Files* du gestionnaire de projet, puis « Set at Top-Level Entity ». Vérifiez dans l'onglet *Hierarchy*.
3. Si un Test bench (fichier VHDL) est présent, il sera refusé par le compilateur QII (non synthétisable). Il n'est compilable que sous Modelsim -> créer un projet dans Modelsim.

### Lancer Modelsim à partir de Quartus II

Vérifiez que le projet ne contient que des fichiers VHD. Faites une dernière compilation (vérification) sous Quartus II (outil **Analysis & Synthesis**). Puis lancer le simulateur :

**Tools > Run EDA Simulation Tools > EDA RTL Simulation**

Attention ! Si on utilise une ROM, il faut que le fichier d'initialisation de la ROM soit **dans le sous-dossier de simulation** du dossier de travail (*\simulation\Modelsim*). On doit aussi trouver dans ce sous-dossier de simulation un script comme *simu\_IUT.do*.

→ le logiciel ModelSim d'Altera se lance **dans une nouvelle fenêtre**.

La (re)compilation par Modelsim des codes VHDL (et des bibliothèques) a lieu automatiquement lors du lancement de Modelsim<sup>6</sup>. On peut refaire une compilation à tout moment avec l'outil **Compile** de Modelsim.

Notez qu'il n'y a pas de projet visible dans Modelsim, c'est celui de QII qui est utilisé.

---

## III. EFFECTUER UNE SIMULATION AVEC MODELSIM

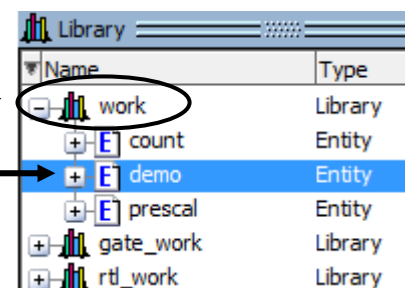
### Charger le simulateur avec le fichier à simuler :

Dans l'onglet **Library**, dérouler le dossier work, qui affiche toutes les entités compilées avec succès (celles des fichiers VHD du projet).

Charger l'entité à simuler dans le simulateur : **double-cliquer** sur son nom<sup>7</sup>

⇒ Une fenêtre *Objects* apparaît, qui affiche la liste de **tous les signaux de l'entité simulée** (si pas visible : **View > Objects**)

⇒ un nouvel onglet **Sim** apparaît à côté de l'onglet *Library*. La ligne de commande (dans la fenêtre *Transcript* en bas) est maintenant celle de *VSIM*.



---

<sup>6</sup> grâce à un script généré par QII exécuté au lancement (si *truc* est le fichier *au top*, ce script s'appelle *truc\_run\_msim\_rtl\_vhdl.do* dans le sous-dossier *simulation\Modelsim*)

<sup>7</sup> ce chargement du simulateur équivaut à la commande *vsim* dans un script de simulation *.do*

## Choisir les signaux à visualiser dans la fenêtre Wave

si elle n'est pas déjà ouverte, ouvrir la fenêtre des chronogrammes : **View > Wave**

Glisser-déposer les signaux souhaités **de la fenêtre Objects vers la fenêtre Wave** (ou clic droit > *Add*).

Si le bloc à simuler comporte des sous-blocs : on peut changer les signaux affichés dans *Objects* à partir de l'onglet **Sim** en déroulant les listes pour y choisir le sous-bloc voulu (utile pour afficher l'état interne d'un automate par exemple).

Remarque : cette étape n'est pas indispensable si on utilise la commande « **add wave -r /\*** » dans un script *.do*. Mais cette commande affiche tous les signaux, il faudra donc effacer ceux non désirés.

## Appliquer les vecteurs de test (= les valeurs des entrées) / Simuler

Plusieurs méthodes :

- utiliser un *Test Bench* (fichier *.vhd*) qui décrit les vecteurs de test et les résultats attendus. Portable hors *ModelSim*, durable, mais long à écrire (à réserver pour les grosses simulations).
- utiliser un script de simulation *.do* (voir plus bas) : simple, durable, facile à modifier (mais non portable hors *ModelSim*). **Conseillé**, surtout pour les petites simulations.
- la méthode manuelle (clic droit sur un signal de la fenêtre *Wave*, puis *Force* ou *Clock*) : fastidieux et surtout éphémère (aucune trace sous la forme d'un fichier, sauf si on récupère la ligne de commande générée pour la copier dans un fichier *.do*).

La méthode manuelle peut aussi consister à taper des commandes *force* sur la ligne de commande au lieu de clics droits (commande *force* voir plus bas). Ce sont ces commandes qu'on va plutôt mémoriser dans un script de simulation *.do*.

## Utilisation d'un script de simulation pour imposer les entrées et lancer la simulation :

Un script de simulation (fichier *.do*) est un fichier texte qui contient des commandes *Modelsim* pour imposer des stimuli sur les **entrées** (*force*), effectuer une simulation sur une certaine durée (*run*), ou même, au préalable, charger le simulateur (*vsim*) et préparer la fenêtre *Wave* (*add wave*). Le script est mémorisé sur le disque dur (fichier texte d'extension *.do*) et peut être exécuté à tout moment d'un simple clic. Il ne reste ensuite à qu'à analyser les chronogrammes. Cette méthode est idéale pour le **test unitaire de petits blocs**.

1. **Créer un script** (outil **New File** ou *File > New > Source > Do*) et le sauvegarder immédiatement **avec l'extension .do**, par exemple dans le sous-dossier *Simulation\Modelsim* (choisir un nom évocateur comme *truc\_simu\_1.do*). On peut le rajouter au projet *Modelsim* (s'il existe) pour l'ouvrir ou le lancer facilement.

2. **Remplir** le script (**cf exemple encadré** page suivante) en se limitant aux commandes **force** et **run**

La commande *force* permet d'imposer les valeurs d'un signal (voir *exemple .do*). Par exemple :

pour une remise à zéro qui dure 300 ns : **force raz 1 0, 0 300ns**

pour une horloge périodique 1 kHz : **force hor 0 0, 1 500us -r 1ms**

La commande *run* lance les calculs pendant la durée indiquée : **run 1ms**


3. **Exécuter le script .do** : menu **Tools > Tcl > Execute Macro**

ou clic droit > *Execute* sur le fichier *.do* dans le projet s'il existe

ou taper *do nom\_script.do* sur la ligne de commande de *Modelsim* ou *Vsim*.

Si le script n'inclut pas la commande **Run** (mais sa présence est conseillée), il faut ensuite utiliser l'outil *Run* de la barre d'outils, avec une durée correcte. C'est la commande *Run* qui effectue les **calculs** de la simulation à partir des stimuli, tout le reste n'est que de la préparation.

☛ Si la fenêtre *Wave* reste vide malgré *Run*, **vérifier dans la fenêtre Transcript** (en bas) qu'il n'y a pas de message d'erreur en rouge. Voir la « Check Liste en cas de problème ».

**Important** : les chronogrammes obtenus à chaque lancement du script **s'ajoutent** sur la fenêtre *Wave* tant qu'on n'utilise pas le bouton **Restart**. L'instant 0 est toujours celui du début de la prochaine simulation *Run* (temps **relatifs**). Utiliser l'outil **Restart**  pour effacer la fenêtre *Wave*.

4. **Analyser** les résultats de la fenêtre *Wave*, modifier si nécessaire le script (il faut en général plusieurs modifications pour aboutir au script idéal) et le re-exécuter, avec ou sans *Restart*.

Remarque : pour un **bus** de la fenêtre *Wave*, *clic droit* > *Radix* pour afficher le bus en base 10 ou 16.

👍 Le script peut être complété (par Copier/Coller) avec le texte obtenu sur la ligne de commande *VSIM* quand on exécute une commande à la souris.

Mieux encore, on peut ajouter au script le **paramétrage détaillé de la fenêtre *Wave*** effectué à la souris : ajout des signaux souhaités, choix d'une base particulière pour les bus, unité par défaut pour l'axe des temps, présence de curseurs, etc. Ce paramétrage est mémorisé dans un fichier texte (nommé *wave.do* par défaut) quand vous utilisez l'outil **Save** à partir de la fenêtre *Wave* : il suffit de le recopier dans votre script.

### Exemple de script *Modelsim* (fichier *.do*)

# script pour la simulation d'un circuit à 4 entrées : *clk*, *cmd* (bit), *bus\_data* (bus de 4 bits) et *raz*

# à exécuter avec le menu **Tools > Tcl > Execute Macro**

# ou en tapant **do test.do** (si le script s'appelle *test.do*) sur la ligne de commande *Modelsim*

# ou *clic droit* > *Execute* sur le *.do* dans le projet *Modelsim* (s'il existe)

# La compilation est supposée déjà faite : ce script s'occupe seulement de la **simulation**.

# compilation à faire **AVANT** l'exécution du script (recompiler à chaque modification d'un code source).

# (optionnel dans le script) **Lancement du simulateur** (mais pas encore de la simulation = « run ») :

# (équivalent à un double-clic sur le nom de l'entité dans le dossier *work* de l'onglet *Library*)

**vsim work.nom\_entite\_TOP**

# on donne le nom de **L'ENTITE** (pas du fichier source *.vhd*) : celle qui est **au TOP !!**

# (option) ouvre la fenêtre *wave* avec tous les signaux visibles

**add wave -r /\***

# **Définition des vecteurs de test (stimuli) associés aux entrées :**

# on peut utiliser l'option **-r** (*repeat*) pour répéter un stimulus avec la période voulue (horloge...).

# Les unités sont à préciser (*ps* par défaut en version 10 !) : *10ns*, *25us*, *11ms*, *2sec*...

# Pas de commentaire (#) sur une ligne de commande !!

# **exemple de signal périodique** (ici une horloge de 500 ns / 2 MHz) avec **-r** (*-repeat*) :

**force** Clk 0 0, 1 250ns **-r** 500ns

# **exemples de signaux non périodiques** (ici, chaque *force* décrit un chronogramme complet) :

**force** cmd 0 0, 1 3us, 0 6us, 1 2ms

**force** raz 1 0, 0 315ns, 1 2us, 0 3157us

# exemple de valeurs pour un **bus** (ici périodique, avec des valeurs données en binaire) :

**force** bus\_data 0000 0, 0011 2us, **1110** 5us **-r** 8us

# on peut aussi utiliser les bases 10 ou 16 : **10#14** ou **16#E** au lieu du binaire **1110**

# visualisation (*Wave*) : afficher les bus en base décimale ou hexa pour mieux vérifier les résultats.

# **Lancement de la simulation** proprement dite :


**run** 3ms

### Effectuer les calculs de la simulation (*Run*) :

Nécessaire seulement si le script n'inclut pas la commande *Run* (*run* est conseillé dans le script) :

outil **Run** de la barre d'outils (ajuster la durée affichée ou répéter la commande plusieurs fois)

ou commande **run** sur la ligne de commande *VSIM* (exemple : **run 2us**).

Les chronogrammes obtenus à chaque **Run s'ajoutent** sur la fenêtre *Wave* tant qu'on utilise pas la commande **Restart** (outil ).

Si on décide d'ajuster les stimuli à partir de la ligne de commande de la fenêtre *Transcript* :  
avant chaque *Run*, on peut modifier les vecteurs de test (commande `force...` ou clic droit sur un signal), sachant que l'instant 0 est toujours celui du début de la **prochaine** simulation *Run*. Par exemple, la commande `force raz 1 0` (ou `force raz 1`) permet de mettre le signal *raz* à 1 pour la prochaine simulation *Run*.

### **En savoir plus sur les scripts et les test benches**

L'utilisation de « script » permet de piloter facilement la simulation sous *Modelsim* en l'absence de *testbench* : l'exécution du script applique les stimuli (vecteurs de test) au circuit à tester et lance la simulation, puis l'affichage des chronogrammes. Le script est idéal pour le **test unitaire de petits blocs**.

Inconvénients : un script n'est pas portable sous un autre compilateur (ce sont des commandes spécifiques à *ModelSim*) et ne permet pas de vérification automatique des résultats.

Une autre méthode de test, plus lourde à mettre en œuvre, est l'écriture en VHDL de **Test Bench** (banc de test) : les stimuli sont définis dans le code VHDL du Test bench (code non synthétisable !), qui analyse aussi la réponse en la comparant à la réponse attendue. Un Test Bench est **portable** et permet la **vérification automatique** des résultats. A réserver pour le test d'ensembles plus gros : test de cartes par exemple.

Notez qu'il existe des outils en ligne d'écriture du « squelette » d'un testbench.

---

## **IV. QUAND MODELSIM EST UTILISE SEUL**

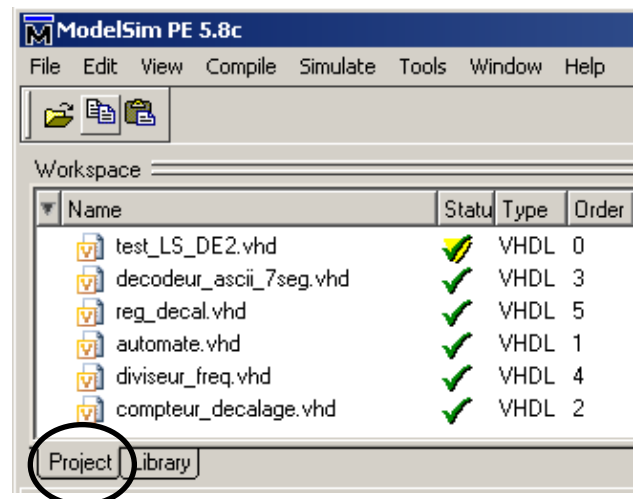
Il faut rajouter au préalable la création de projet et la compilation volontaire, c'est tout.

### **Créer un projet :**


**File > New**

Y ajouter tous les fichiers sources **.vhd** à simuler

Tous les sources **.vhd** apparaissent dans l'onglet **Project** du gestionnaire de projet.



### **Compiler :**

**Compile > Compile All (ou Selected)** ou **outil** 

*Pour le reste, voir paragraphe III et suivants*

## E. Création d'un automate sous forme graphique

Pour développer une application à base d'automates sur une carte PLD, on apprécie de pouvoir décrire l'automate sous la forme d'un dessin, celui du **diagramme de transition**. C'est possible avec Quartus II, dans un format propriétaire (fichier *smf*). L'intérêt est qu'on peut ensuite demander à Quartus une traduction en code **VHDL**, qui est, elle, portable.

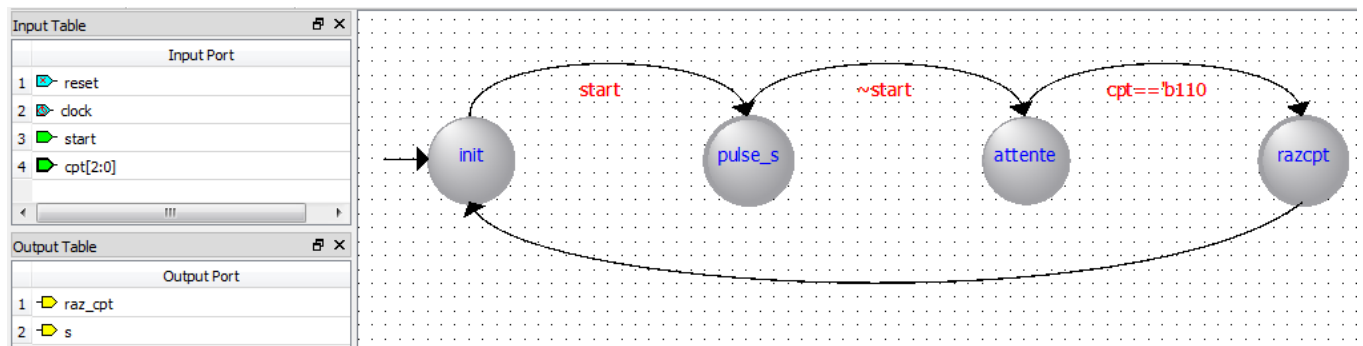
L'insertion dans un schéma de niveau supérieur se fait de façon habituelle, sous la forme d'un symbole.

Comme d'habitude, il faut être dans le cadre d'un projet bien paramétré.

### Un exemple d'automate

Nous allons créer à titre d'exemple un automate dont les entrées sont les 4 signaux *clock*, *reset* (synchrone), *start* et un bus *cpt*, et dont les sorties sont les 2 signaux *raz\_cpt* et *s*.

Voici son diagramme d'état après dessin sous *Quartus v.10* (notez que les sorties n'y sont pas apparentes !):

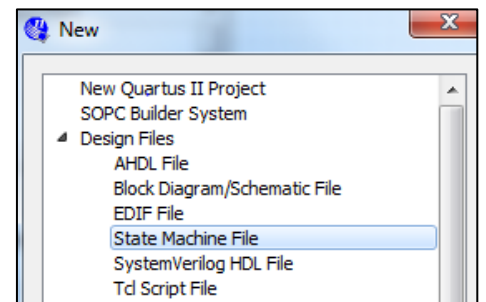


### Saisie du diagramme d'état (State Machine)

Quartus propose un mode de saisie assisté de diagrammes d'état, dans un **fichier State Machine File** (fichier *.smf*).

La création d'un fichier *.smf* s'effectue par :

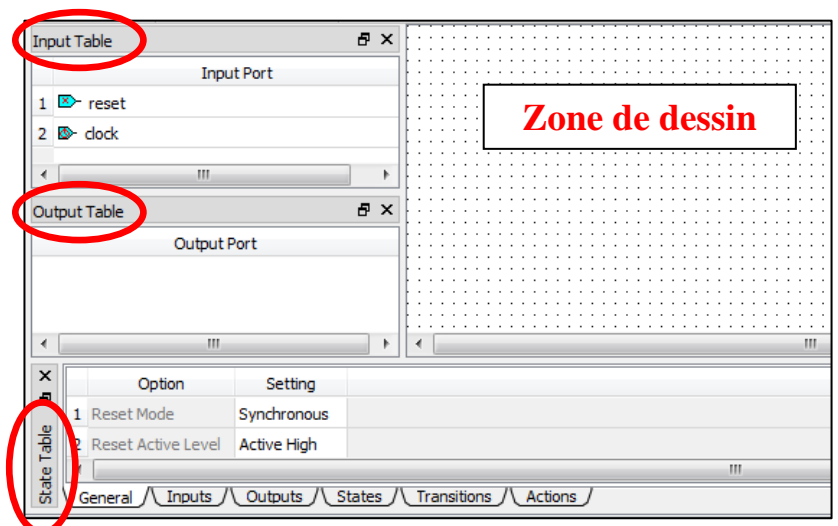
**Menu File > New** Choisir **State Machine File**



Une feuille de dessin (fichier d'extension *.smf*), s'ouvre dans la fenêtre d'édition (comme ci-contre).

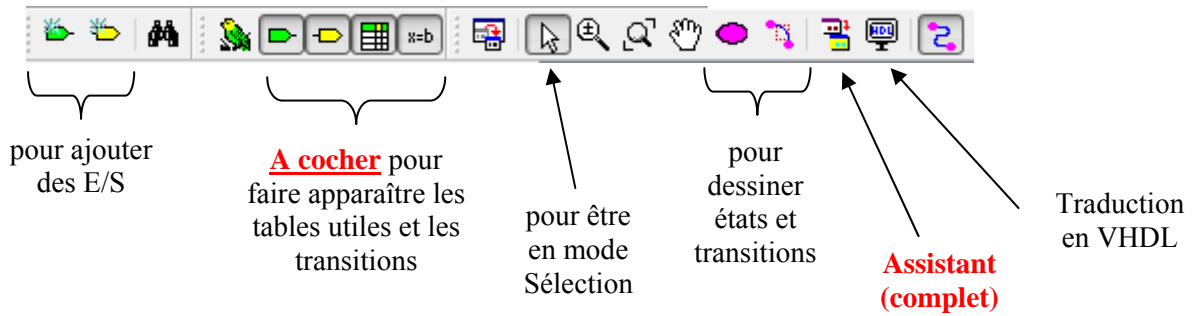
On va y dessiner le diagramme de transition voulu.

Sauvegardez dès que possible la feuille de dessin avec un nom bien choisi, par exemple *nom\_projet.smf*.








Vous disposez d'une panoplie **d'outils** très utiles :



Nous allons dessiner le diagramme de transition dans la feuille de dessin à l'aide de la barre d'outils ci-dessus. Suivez les étapes :

1. Placer les états : **outil State Tool**  de la boîte à outils (commencer par l'état initial).
2. Renommer les états si souhaité : double-clic sur l'état ou utiliser l'onglet *States* de la *State Table* (visible si l'outil **State Table** est enfoncé).
3. Ajouter les **entrées et les sorties** (dans *Input Table / Output Table*) : utiliser les deux outils **Add New Input/Output Port** . **Attention** : la syntaxe (*Verilog*) pour un **bus** est `mon_bus[7:0]`
4. Ajouter les **transitions** : **outil Transition Tool**  dans la boîte à outils (attention à faire partir et arriver les flèches à l'intérieur d'une « bulle » d'état **et non du bord**, sinon Quartus « plante »).
5. Ajouter les **conditions** des transitions (voir exemples ci-dessous) : le plus simple est **d'utiliser la State Table** et son onglet **Transitions** :

La table de transition est visible si l'outil **State Table** est enfoncé

	Source State	Destination State	
1	init	pulse_s	start
2	pulse_s	attente	~start
3	attente	razcpt	cpt=='b110
4	razcpt	init	

State Table

General Inputs Outputs States **Transitions** Actions

Exemples de syntaxe (*Verilog* !) de condition :

**a   ~a   a & b   a | ~b   mon\_bus=='b1010   mon\_bus>='b0110**

Note : on peut, en mode *Selection tool*, déplacer ou **double-cliquer sur la transition** pour la modifier.

6. Ajouter les **actions** sur les sorties dans chacun des états : **double-clic sur l'état** puis onglet **Actions** puis clic dans **Output Port** (attention, **les sorties non spécifiées seront à 0 par défaut**). Ou (plus rapide) : outil **State Machine Wizard** (voir note encadrée plus loin).

A la fin, l'onglet *Actions* de la *State Table* peut par exemple ressembler à ceci :

	Output Port	Output Value	In State	Additional Conditions
1	raz_cpt	1	init	
2	raz_cpt	1	pulse_s	
3	raz_cpt	1	razcpt	
4	s	1	pulse_s	
5	s	1	razcpt	

State Table

General Inputs Outputs States Transitions **Actions**





L'outil **State Machine Wizard** permet d'avoir une vue d'ensemble (moins graphique, mais plus complète) et de faire toutes les modifications / ajouts voulus, en particulier les affectations de sorties.

Il permet aussi de gérer les options cachées : remise à zéro, sorties combinatoires ou *registered*...

Remarque : quand les conditions de transition au départ d'un état sont fausses, le maintien dans l'état courant est assuré de façon explicite dans le code VHDL à l'aide de *else* (il faut décocher l'option "*transition to source state*" pour voir disparaître cette instruction *else*).

7. Lorsque le dessin de la machine d'état est fini, il faut le **transformer en une description VHDL** en utilisant l'outil **Generate HDL file** de la barre d'outil (et non le menu *File > Create...*)

N'hésitez jamais à vérifier le code VHDL généré en cas de doute (écriture de condition, affectation de sorties...).



**Il est recommandé de sauvegarder la feuille de dessin assez souvent.**

## Une fois l'automate sous forme VHDL

On peut analyser (et non compiler !) le fichier **VHDL** (ouvert dans la fenêtre active) à l'aide de la commande **menu Processing > Analyze Current File**

Il ne reste plus qu'à vérifier en simulation que l'automate fonctionne...

### Ensuite, la démarche est classique :

- ▶ création d'un **symbole** à partir du fichier VHDL ouvert  
**Fichier > Create/Update > Create Symbol Files**
- ▶ Simulation à partir du code source VHDL
- ▶ Dessin d'un schéma de synthèse avec des noms de port bien choisis (attention, **le schéma bdf ne doit pas avoir le même nom que le fichier smf !**)
- ▶ Etc.

## F. Check list en cas de problème

### ON VOIT DES WARNINGS ET ERRORS LORS DE LA COMPILATION

*Certains warnings sont normaux, d'autres pas. L'expérience vous permettra de savoir lesquels.*

1. Tous vos fichiers et dossiers sont-ils des noms dépourvus d'espaces et d'accents ?
2. Avez-vous bien indiqué le fichier *au top* ?
3. **Avez-vous associé chaque port de votre système à une broche du PLD ?** (= sur votre schéma, les ports sont-ils associés avec une broche physique par *Quartus II* ?). Il faut utiliser les bons noms (ceux du fichier *csv*), importer le fichier *csv*, avoir compilé au moins une fois avec le schéma à synthétiser « au top ». Eventuellement, fermez le fichier *bdf* et rouvrez-le pour mettre à jour l'affichage du brochage dans le schéma. Une source d'erreur fréquente est que le fichier « au top » n'est pas le bon (vérifiez dans l'onglet *Hierarchy* du projet).
4. ⚠ *Warning* à la compilation « *Converted elements in bus name xxx using legacy naming rules* » (+ apparition dans le *Pin Planner* de nouveaux noms + rien ne marche sur la carte) : il indique que QII va **renommer des fils de bus** (en général, ce sont des signaux du fichier *csv*, comme *LED[i]*). Ce **warning ne doit pas** apparaître.

**Solution** : une option de compilation doit être changée (une fois pour toute) :

menu *Assignments* puis *Settings > Analysis & Synthesis Settings* → bouton *Moore Settings*  
→ dans l'option *Block Design Naming*, il faut choisir *QuartusII* (et non *Auto*). Recompilez.

5. *Warning* à la compilation : “*Warning: The Reserve All Unused Pins setting has not been specified, and will default to 'As output driving ground'.*” Ce **warning ne doit pas** apparaître.

**Solution** : vous avez raté l'étape encadrée et signalée par ☠ lors de la création du projet !  
Rectifiez et recompilez.

6. (rare) si la barre d'outils du fichier *bdf* disparaît (c'est le « Block Editor ») : clic droit sur l'onglet *toto.bdf* → *Detach Window*. Choisir le menu *Tools* → *Customize* et cocher *Block Editor*.

### ÇA NE FONCTIONNE PAS EN SIMULATION SOUS MODELSIM

7. Vérifier dans la fenêtre *Transcript* (en bas) qu'il n'y a aucune erreur quand vous exécutez le script.
8. *Modelsim* ne voit pas certaines de vos entités dans *work* ? Vérifiez qu'elles figurent sous forme de fichiers VHD dans le projet sous *Quartus II*.
9. *Modelsim* affiche une erreur de **type** pour des **bus** ? elle est souvent due à l'incompatibilité entre le type *unsigned* d'un port d'entité et le type *std\_logic\_vector* utilisé par QII pour ses traductions schéma→VHDL. Solution : utilisez le type *std\_logic\_vector* pour les bus de vos ports d'entités (gardez *unsigned* pour les bus internes et ajoutez une conversion) ou (déconseillé) modifiez le code généré par QII.
10. Avez-vous bien choisi la durée de simulation ? la période de vos signaux périodiques ? Un signal qui apparaît comme une barre indique souvent une durée de simulation ou une période mal choisie.

### LA PROGRAMMATION DE LA CARTE SE PASSE MAL (OUTIL PROGRAMMER)

11. Vérifiez votre câblage USB.  
Des problèmes liés à l'USB ont été constatés sur certains PC : branchez le câble USB à l'arrière du PC plutôt qu'à l'avant. Si ça continue, rebootez le PC.
12. Avez-vous utilisé le bon circuit (*device*) ? Vérifiez dans l'outil *Programmer*.

13. Avez-vous utilisé le bon fichier de programmation ? (fichier *.pof*, dans l'outil *Programmer*, qui apparaît normalement sans chemin d'accès). Eventuellement : effacez le fichier *pof* du disque dur et recompilez. Dans l'outil *Programmer* : sélectionnez le fichier *.pof*, effacez-le (bouton *Delete*), puis rajoutez-le (bouton *Add File*).
14. Les mini-switchs de la carte *DE-nano* sont-ils toujours bien positionnés ? (*switch 1 = Up, switch 2 = Down*) ?

---

### ÇA NE FONCTIONNE PAS SUR LA CARTE

15. Avez-vous recompile depuis votre dernière modification ? (d'un schéma ou d'un code VHDL, du projet...)
16. Avez-vous vérifié, avant de programmer, que tous les ports d'entrée/de sortie sont bien associés à une broche du composant ? (partie *Synthèse*). Cf erreur 3.
17. Si Quartus II met un *warning* indiquant qu'il renomme des fils de bus, attention ! Ce warning apparaît si vous n'avez pas fait à l'installation le paramétrage suivant et si vous utilisez des bits isolés.  
Solution : réalisez (une fois) le changement suivant sur les options par défaut de Quartus II :  
    menu **Assignments** → sous-menu **Settings** → dans la catégorie **Analysis & Synthesis Settings** → clic sur le bouton **Moore Settings** → dans l'option **Block Design Naming**, il faut choisir QuartusII (et non *Auto*).  
    ⇒ Vous éviterez ainsi que Quartus II renomme (mal) les bits de bus isolés.
18. Vérifiez le PLD cible choisi : dans l'outil *Programmer* ou le menu *Assignments > Device*.
19. Avez-vous utilisé le bon fichier de programmation ? (fichier *.pof*, dans l'outil *Programmer*, qui apparaît normalement sans chemin d'accès). Eventuellement : effacez le fichier *pof* du disque dur et recompilez. Dans l'outil *Programmer* : sélectionnez le fichier *.pof*, effacez-le (bouton *Delete*), puis rajoutez-le (bouton *Add File*).
20. (*Rare*) Déconnectez l'alimentation de la carte et rebranchez-la.