TP 2 : Description VHDL

Rappel : Un répertoire TP2 avec 2 sous répertoires
>Additionneur (Partie I)
>AddBCD (Partie II)

Partie I : Logique combinatoire
Additionneur complet 1 bit puis additionneur 4bits

1) Le langage VHDL

Le VHDL est un langage de description matériel qui diffère des langages « software » comme le C ou le Java. Il est destiné à représenter le comportement ainsi que l'architecture d'un système électronique numérique. Il facilite la conception de systèmes en permettant une structuration en fonctions hiérarchiques.

Le but d'un langage de description matérielle tel que le VHDL est de faciliter le développement d'un circuit numérique en fournissant une méthode rigoureuse de description du fonctionnement et de l'architecture du circuit désiré. L'idée est de ne pas avoir à réaliser (fondre) un composant réel, en utilisant à la place des outils de développement permettant de vérifier le fonctionnement attendu. Ce langage permet en effet d'utiliser des simulateurs, dont le rôle est de tester le fonctionnement décrit par le concepteur.

Voici un exemple de code VHDL :

```
entity full_add1bit is
 port(
 a, b, cin: in bit;
 s, cout: out bit --ici pas de virgule
 );
end entity;

Architecture arch of full_add1bit is
begin
 s<=cin xor (a xor b);
 cout<=(cin and a) or (cin and b) or (a and b);
end arch;
```

Identifier les différentes parties intervenant dans la description VHDL de ce circuit simple.

- La partie *entity* permet de définir les ports en entrée/sortie du composant,
- La partie *architecture* permet quant à elle de définir son comportement.

Dans une architecture, toutes les lignes combinatoires ou blocs de process sont exécutés en parallèle. Les instructions sont dites 'Concourantes'

a) Créer un nouveau projet Quartus dans TP2>Additionneur

b) Ajouter au projet un nouveau fichier VHDL

c) Reprendre le code vhdl donné ci-dessus et copier le dans Quartus.

d) Enregistrer le fichier sous le nom suivant : *full_add1bit.vhd*

e) Simuler ce composant en mode fonctionnel

f) En vous inspirant de ce code, concevoir un full add 4 bits en VHDL avec des retenues d'entrée et de sortie. Pour cela, il sera nécessaire d'ajouter des ports (du type Bit) en entrée :
a0, a1, a2, a3 : in bit ;
b0, b1, b2, b3 : in bit ;
s0, s1, s2, s3 : out bit ;

A l'intérieur de l'architecture, il est possible d'utiliser le type 'Signal' afin de tenir compte des carry internes de l'additionneur 4 bits.

signal C0: bit;

g) Simuler l'entité VHDL en mode fonctionnel.

2) La librairie IEEE

La librairie IEEE fournit un ensemble de packages contenant la définition de types élaborés (std_logic, std_logic_vector) et de fonctions arithmétiques et logiques facilitant largement la description des entités. Les librairies IEEE principales sont :

- IEEE.standard ;
- IEEE.std_logic_1164 ;
- IEEE. std_logic_unsigned.all;

Voici le full add 1 bit décrit à l'aide des packages de la librairie IEEE :

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity full_add1bit_lib is
 port(
 a, b, cin: in std_logic;
 s, cout: out std_logic
 );
end entity;

architecture arch of full_add1bit_lib is
 signal resultat: std_logic_vector(1 downto 0);
begin
 resultat <= ('0' & a) + ('0' & b) + ('0' & cin);
 s <= resultat(0);
```

```
 cout <= resultat(1);
end arch;
```

a) Créer un nouveau fichier VHDL au projet

b) Reprendre ce code vhdl et enregistrer le fichier sous le nom suivant : *full_add1bit_lib.vhd*

c) Simuler cette entité avec le simulateur intégré de Quartus

d) Appliquer l'opérateur & sur un des 3 membres dans la ligne de code:
resultat <= ('0' & a) + ('0' & b) + ('0' & cin);
?
resultat <= ('0' & a) + b + cin;

Effectuer à nouveau la simulation et conclure. Donner une définition à cet opérateur &

e) S'inspirer de cette description pour définir un full add 4 bits à l'aide des librairies IEEE.

```
Entity Full_add4bit_lib is
port(
        cin: in std_logic;
        A, B: in std_logic_vector(3 downto 0);

        s: out std_logic_vector(3 downto 0);
        cout: out std_logic
 );
end entity;
```

Il est maintenant possible de faire directement une addition de deux nombres sur 4 bits à l'aide de l'opérateur +.

f) Simuler cette nouvelle entité.

3) Définition de composants (*components*)

Le langage VHDL permet de définir des composants (*components)* qui peuvent être réutilisés par d'autres entités. On parle alors d'instanciation.

Méthode d'instanciation :

- Déclaration du composant

Pour instancier une entité, il faut d'abord indiquer le composant prototype de votre entité dans **l'en-tête de l'architecture** (il faut simplement reprendre la partie *entity* de votre composant à instancier en remplaçant le mot clé *entity* par le mot clé *component*).

Ex :
```
COMPONENT full_add1bit IS
PORT(
        a : IN bit;
        b : IN bit;
```

```
                Cin : IN bit;
                s : OUT bit;
                Cout : OUT bit
        );
        END COMPONENT;
```

- Instanciation du composant :

Les instanciations apparaissent dans le code décrivant le comportement de l'entité (entre le **Begin** et le **End** de l'architecture) sous forme de fonction, avec la syntaxe suivante :

inst_name **:** *name_component* **port map (***signaux connectés à l'instance***) ;**

Ex :

```
UUT : full_add1bit PORT MAP(
                        a => aa,
                        b => bb,
                        Cin => CCin,
                        S => SS,
                        Cout => CCout
                        );
```

Où : a, b, Cin , S, Cout sont les ports du composant et aa, bb, CCin, SS, CCout sont les signaux associés à ces ports

Ou

```
UUT : full_add1bit PORT MAP ( aa, bb, CCin, SS, CCout );
```

Où : aa, bb, Ccin, SS, CCout sont les signaux associés à ces ports dans le même ordre que dans l'entité de définition.

Chaque instance doit avoir un identifiant *inst_name* unique.

a) Créer un nouveau fichier VHDL au projet

b) Enregistrer le fichier sous le nom suivant : *full_add4bit_comp.vhd*

c) Concevoir un full add 4 bits en VHDL (**en incluant les librairies IEEE**) à partir du *component* de l'additionneur 1 bit adéquat qui sera instancié 4fois.

4) Création de symboles

A partir d'une description VHDL, il est également possible sous Quartus de créer de nouveaux objets graphiques, appelés symboles.

Pour ce faire, sélectionnez le fichier VHDL à transformer en symbole *et le menu File -> Create/Update -> Create Symbol File for Current File*.

Ce nouvel objet apparaitra alors dans la bibliothèque des composants lors de la création d'un bloc schématique.

5) Implémentation sur la carte

- Créer un nouveau Bloc diagramme Schématique et utiliser le symbole que vous venez de créer
- Ajouter les entrées et les sorties
- Faire les assignations des pins
- Implémenter sur la carte DE1-SoC

Partie II : Réalisation d'un additionneur décimal

Description du projet à réaliser :

On souhaite additionner deux nombres A et B codés sur 4 bits. (Les valeurs de A ou B sont comprises entre 0 et 15). Ces nombres seront sélectionnés sur la carte DE1-Soc à partir des switch **SW[3..0]** et **SW[7..4]** et seront affichés sur les afficheurs 7 segments de la carte DE1-SoC. Une retenue sera prise en compte et l'addition sera effectuée en **Décimal Codé Binaire**, c'est-à-dire que le résultat et les deux nombres de l'addition seront affichés en nombre décimal (et non hexadécimal) sur 2 afficheurs 7 segments. (1 pour les unités, le deuxième pour les dizaines)

Décodeur BCD :

Le décodeur BCD (Décimal Codé Binaire) est un additionneur 4 bits spécifique.

L'entité VHDL sera nommée *add_bcd.vhd* et sera identique à celle-ci-dessous :

```
entity ADD_BCD is
port(
        A, B : IN std_logic_vector(3 downto 0);
        Cin : IN std_logic;
        S : OUT std_logic_vector(3 downto 0);
        Cout : OUT std_logic_vector(3 downto 0)
);
END ADD_BCD;
```

Vous utiliserez un process pour pouvoir décrire cette entité.

Exemple :

```
Process (A,B,Cin)
begin

end Process;
```

Propriétés des Process :

- ✓ Le code présent à l'intérieur du process s'exécute lorsqu'un changement est détecté sur un des membres de la liste de sensibilité (A ou B ou Cin) .
- ✓ Le test conditionnel n'est utilisable qu'à l'intérieur d'un process

  IF (condition) then
  End if;
- ✓ A l'intérieur d'un process les instructions sont **séquentielles**.
- ✓ Les modifications effectuées sur les signaux à l'intérieur d'un process ne sont effectives qu'à la sortie du process.

➢ Copier le début du code VHDL de ADD_BCD.vhd dans un fichier vhdl dans un **nouveau** projet

- **Compléter le code VHDL** de l'architecture sans l'utilisation de composants.
- Enregistrer le fichier sous le répertoire **>TP2>AddBCD**
- Télécharger le fichier TB_ADD_BCD.vhd et le mettre dans le même répertoire

Quelques remarques sur ce fichier Test Bench : TB_ADD_BCD.vhd

Vous pouvez également ouvrir ce fichier. Ce fichier correspond à du code VHDL dans lequel l'entité n'a pas de port en sortie ou en entrée. Le composant ADD_BCD y est instancié sous le nom UUT (Unit Under Test). Le code de l'architecture permet de générer les stimuli.

Un des avantages d'utiliser des TestBench pour la simulation est que l'on peut utiliser tout le code VHDL (même non synthétisable).

Ex : *Wait for 100 ns* n'est pas synthétisable dans Quartus
Ex : pour générer un signal de type clock en une ligne :
clk <= '0' when done else not Clk after Period / 2;
Remarques :

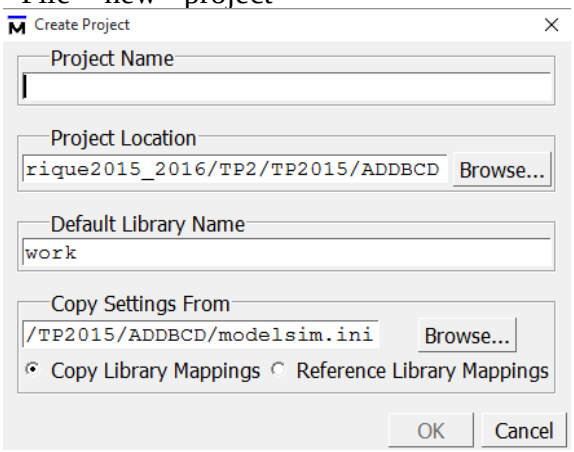
- Le process n'a pas de liste de sensibilité. Dans ce cas, le process exécute le code séquentiellement et recommence indéfiniment.
- L'instruction 'Wait' juste avant 'End Process'. Cette instruction détruit le process car elle le fait attendre indéfiniment. Si cette instruction n'était pas présente, le process de stimuli recommencerait…
- Repérez le code suivant :
  Expected_S<=X"3"; Expected_Cout<=X"1"

  Et la ligne en fin code en dessous du process
  OK_TB <= true when ((ssum = Expected_S) and (ccout =Expected_Cout)) else false;

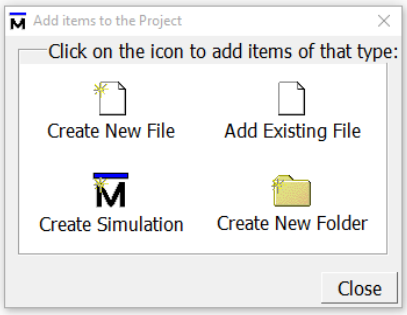
  Ici, les résultats attendus sont comparés avec les ports de sortie S et Cout de l'entité instanciée.

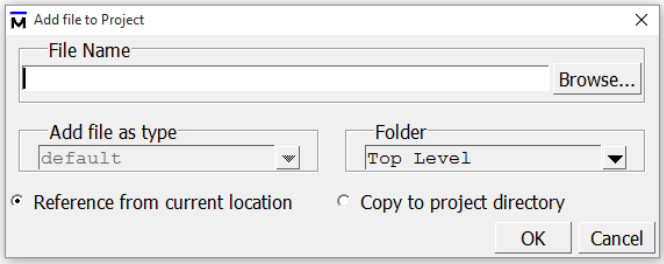
Didacticiel pour la création d'une simulation et compilation avec Modelsim
Création du projet et définition du répertoire Work

- Lancer ModelSim
- >File > new> project



- ***Project Name*** : Indiquer le nom du projet (ex : ADD BCD)
- ***Default Library Name* :** Laisser Work et Cliquer: 'OK'
- Dans la fenêtre choisir 'Add Existing File'



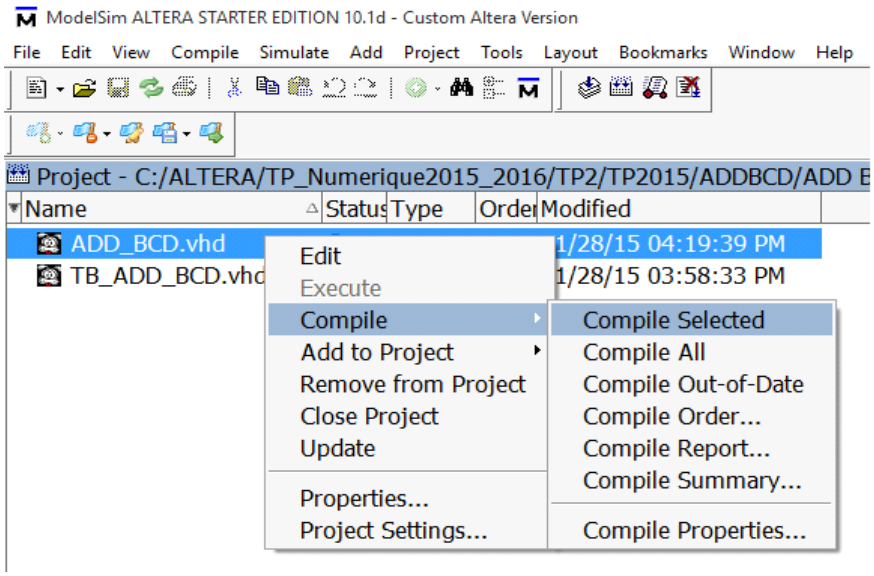


- Puis sélectionner le fichier : ADD_BCD.vhd et cliquer OK pour l'ajouter au projet
- Recommencer l'opération pour le fichier : TB_ADD_BCD.vhd

Compilation de l'entité : ADD_BCD.vhd

Dans l'onglet 'project' sélectionner le fichier ADD_BCD.vhd

- Clic droit



- Choisir :
  >Compile>Compile Selected
  Remarque : Si le fichier est compilé sans erreur, le fichier ADD_BCD.vhd prendra une coche verte… Sinon inspecter l'erreur et modifier le code dans Modelsim.

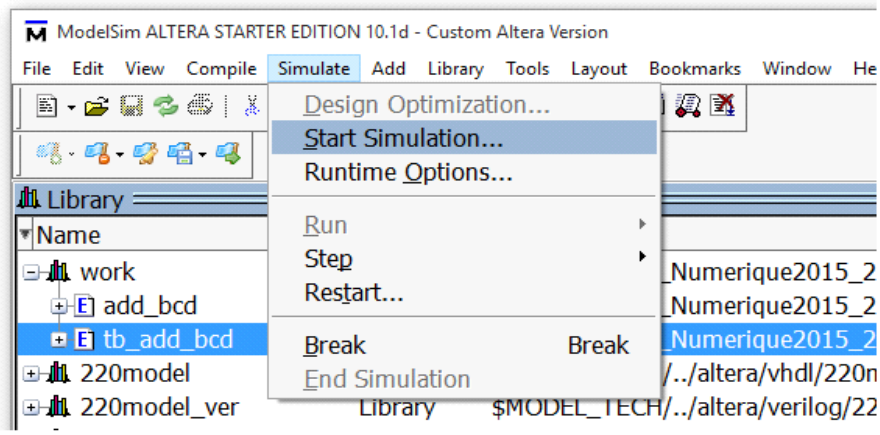
Compilation de l'entité : TB_ADD_BCD.vhd

Procédez de façon identique

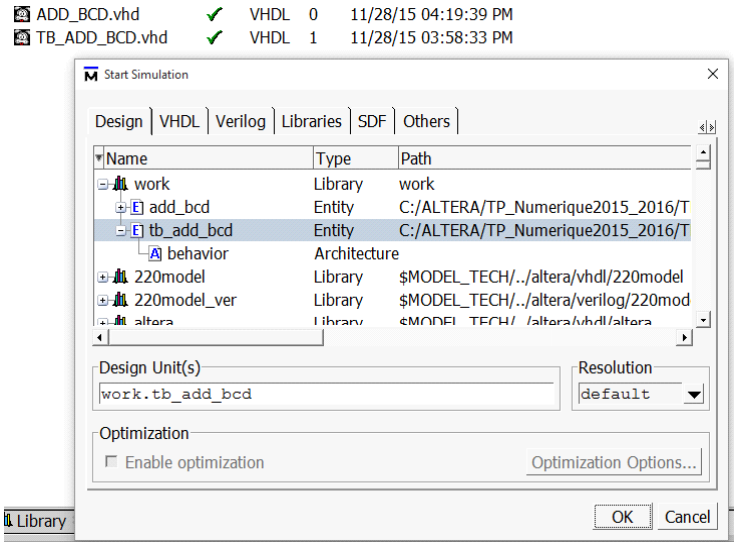
Remarque : Il faut toujours compiler les entités de plus bas niveau en premier et ceux de plus haut niveau en dernier.

Simulation de l'entité de plus haut niveau

- Sélectionnez l'entité de plus haut niveau
- >Simulate>Start Simulation

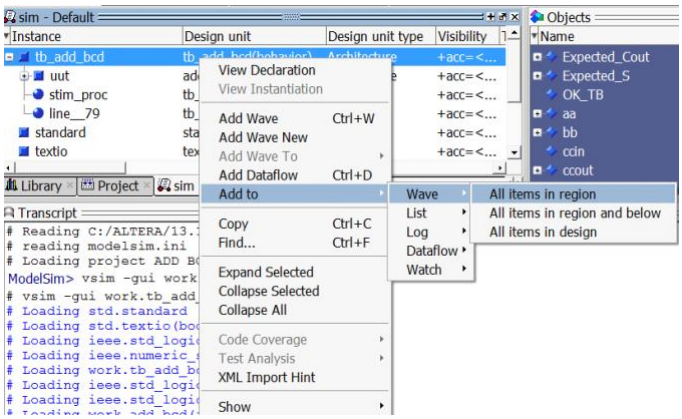


- Dans la fenêtre 'Start simulation', sélectionner l'entité de plus haut niveau. Puis cliquer 'Ok'



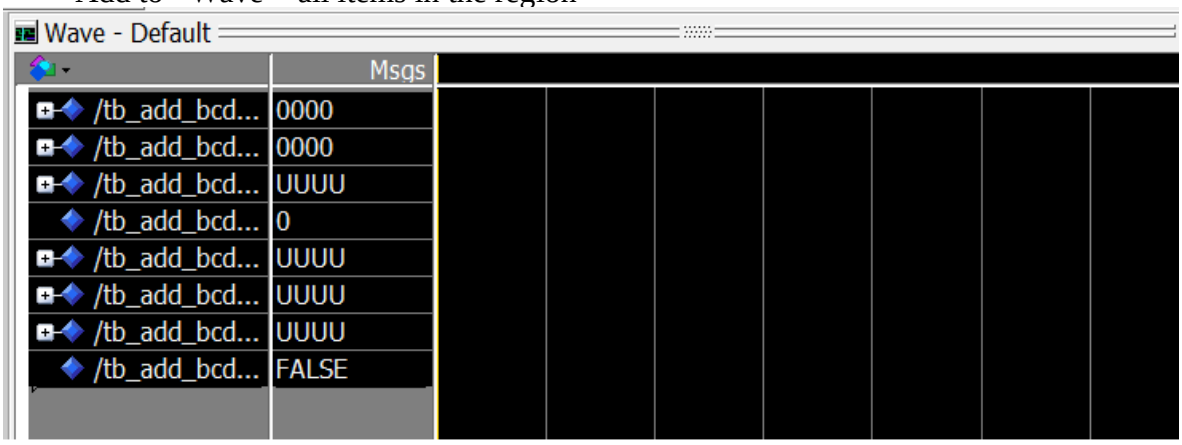
= > Plusieurs Fenêtres vont apparaitre.

- Dans la fenêtre Sim-default sélectionner : Tb_add_bcd



- Puis Clic Droit et sélectionner dans le menu :

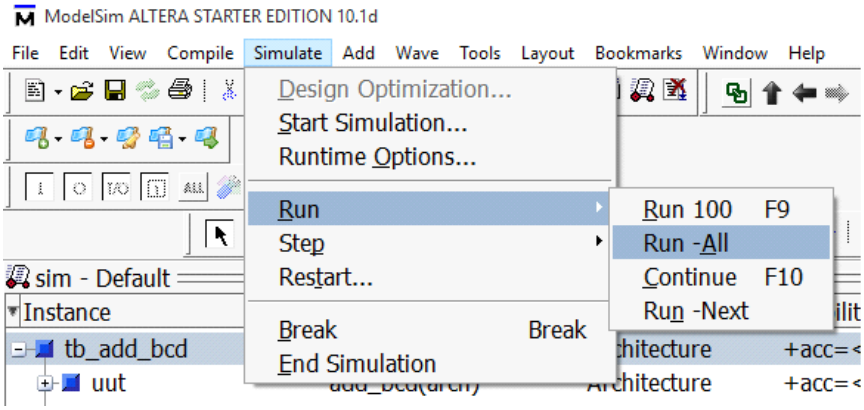
Add to >Wave > all items in the region



Dans la fenêtre 'Wave', l'ensemble des signaux déclarés dans le Test Bench doivent apparaitre.

- Lancement de la simulation :

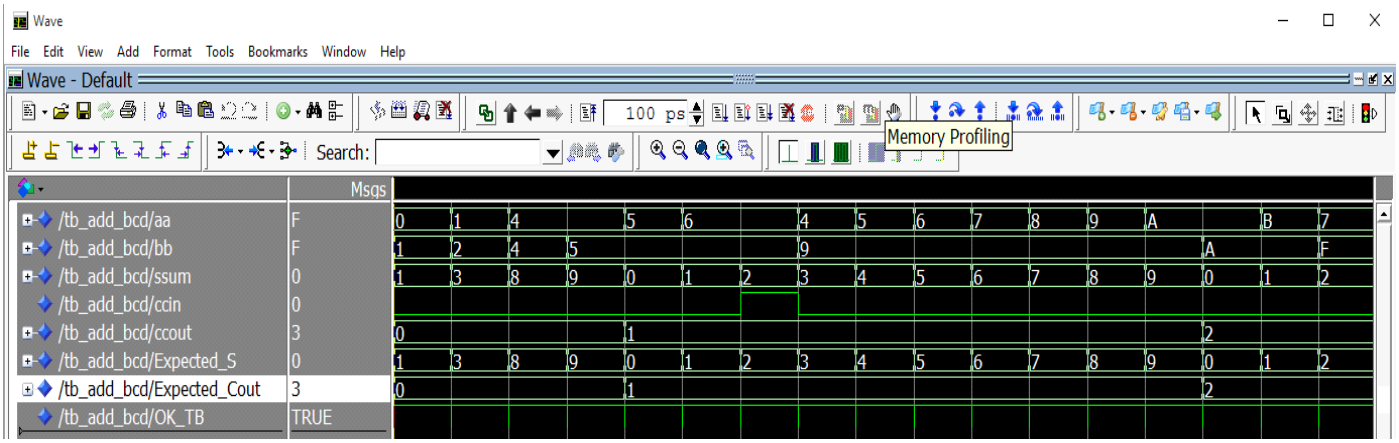
Sélectionner le menu :
>Simulate>Run >Run- All



Les signaux apparaissent dans la fenêtre 'Wave'

- Afin de mieux observer les signaux, dans la fenêtre ‘Wave’, cliquer sur l’icône ‘+’ (Zoom /unzoom), afin d’agrandir cette fenêtre. Puis Dans la zone de tracé des signaux :
  Clic Droit >Zoom Full

En modifiant l’affichage des signaux en Hexadécimal, la fenêtre de visualisation des signaux doit être similaire à celle ci-dessous.



- Modifier le code VHDL ADD_BCD.vhd afin que le signal OK_TB soit à ‘True’ jusqu’à la fin de la simulation.

Fin du TP 2