

# TP 1 – Prise en main

## Le logiciel utilisé : Quartus

Quartus est un logiciel distribué par le constructeur du FPGA que nous utiliserons (Intel – ex. Altera). Il permet de développer et charger sur le FPGA la configuration matérielle souhaitée.

Pour cette première séance de prise en main, nous ne vous demanderons pas de créer le projet Quartus à utiliser : pour chaque exercice, télécharger le projet correspondant depuis Moodle, et le décompresser dans votre dossier personnel sur le réseau.

## Exercice 1 : Multiplexeur 4 vers 1

Télécharger le projet **TP\_1\_Ex\_1** sur Moodle et le décompresser dans votre dossier personnel.

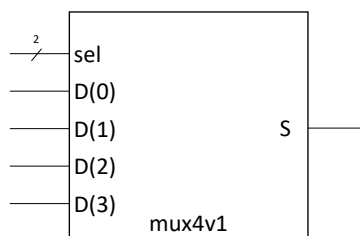
Ouvrir **Quartus prime 18.1** (attention à bien prendre la bonne version) depuis le dossier « Applications locales GE2I » sur le bureau. Choisir « File ➔ Open project » et ouvrir le fichier **DE2\_115.qpf** dans le dossier que vous avez décompressé.

Dans « Project Navigator » à gauche, choisir « Files » Dans la liste déroulante, et ouvrir le fichier **mux4v1.vhd**.



### 1) Entité

Remplir les ports de l'entité de manière à ce qu'ils correspondent à la vue externe ci-dessous :



Attention à bien choisir les types des signaux d'entrée/sortie de manière intelligente. Pour cela, posez-vous la question suivante : « le choix que j'ai fait serait-il pertinent si j'avais un mux 8 vers 1 ? Ou même un mux 32 vers 1 ? »

### 2) Architecture

Coder l'architecture. Vous devrez utiliser l'instruction la plus appropriée.

### 3) Synthèse

Faire un clic-droit sur le fichier **mux4v1.vhd**, et sélectionner « Set as Top-Level Entity ».

Cliquez ensuite sur le menu « Processing ➔ Start ➔ Start Analysis and Synthesis » (attention à bien sélectionner le bon outil !)

Si des erreurs apparaissent, vérifier la syntaxe de votre fichier mux4v1 et du package.

#### 4) Simulation

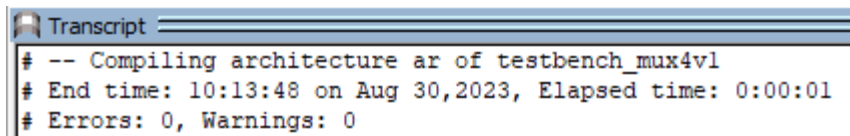
Accéder au menu « Tools → Options », puis dans la catégorie « EDA Tool Options », coller le texte suivant sur la ligne « ModelSim-Altera » (**PAS sur la ligne « ModelSim » tout court**) :

```
C:\intelFPGA\18.1\modelsim_ase\win32aloem
```

NOTE : Il sera nécessaire de configurer le simulateur à chaque séance, car le paramètre n'est malheureusement pas mémorisé.

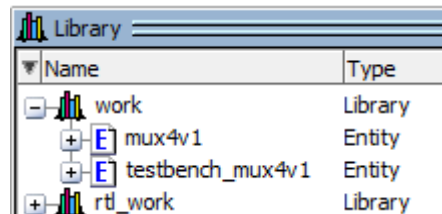
Afin de simuler le composant, nous utiliserons un testbench. La notion de testbench sera détaillée dans un cours ultérieur, pour l'instant celui-ci sera fourni. L'outil ModelSim permet de simuler un composant à l'aide d'un testbench qui stimulera les entrées du composant. Pour le lancer, accéder au menu « Tools → Run simulation Tool → RTL simulation ».

Pour l'instant, Quartus n'a compilé que le composant mux4v1 : nous devons également compiler le testbench. Dans ModelSim, accéder au menu « Compile → Compile... ». Par défaut, la fenêtre qui s'ouvre dans un sous-dossier de votre projet : remonter dans le dossier racine du projet, puis ouvrir « src ». Sélectionner le fichier de testbench (**testbench\_mux4v1.vhd**). Enfin, cliquer sur le bouton « Compile ». Vérifiez que vous n'avez pas d'erreur dans la fenêtre « Transcript » en bas, puis fermez la fenêtre de compilation en cliquant sur « Done ».



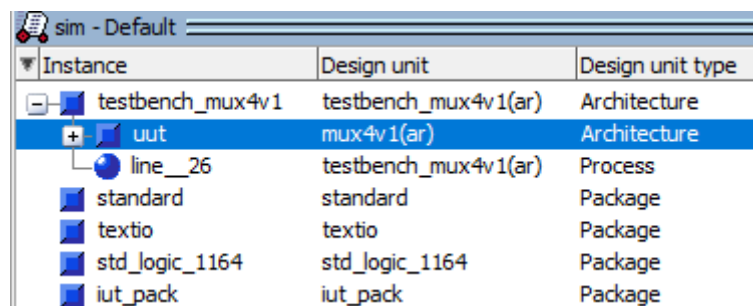
```
Transcript
# -- Compiling architecture ar of testbench_mux4v1
# End time: 10:13:48 on Aug 30,2023, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
```

Les deux fichiers doivent maintenant apparaître dans la bibliothèque « work » : dérouler celle-ci dans la fenêtre « Library » pour vérifier :

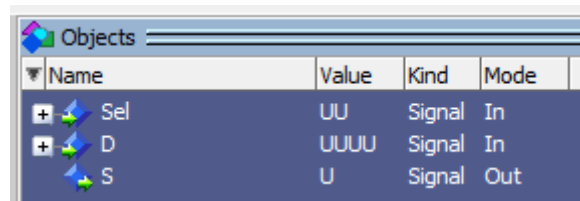


Nous pouvons maintenant démarrer la simulation en faisant un clic-droit sur le testbench, et en cliquant sur « Simulate ».

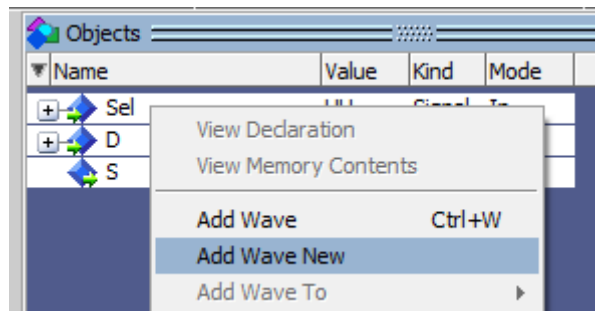
L'interface de simulation qui s'ouvre est découpée en plusieurs fenêtres. Dans la fenêtre « sim - Default », sélectionner le composant « testbench\_mux4v1 → uut ».



Lorsque le composant est sélectionné, toutes ses entrées-sorties apparaissent dans la fenêtre « Objects ».

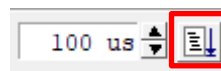


Sélectionnez les 3 signaux. Pour cela, maintenir la touche « contrôle » au clavier, et cliquer sur les signaux successivement. Faites un clic-droit sur la sélection et choisissez « Add Wave New ».



Cela a pour effet de créer une « Wave » (un chronogramme) contenant les trois signaux.

Dans la barre d'outils, choisissez une durée de simulation de 100  $\mu$ s (« 100 us »), et cliquez sur le bouton « Run » (bouton immédiatement à droite de la durée) :



Afin de visualiser la totalité du chronogramme sur l'écran, cliquer sur le bouton « Zoom full » :



Votre composant est maintenant simulé. Vérifiez que le fonctionnement est bien celui attendu pour un multiplexeur, et faites valider. Vous devrez être capable d'expliquer chaque changement de valeur de la sortie à l'enseignant.

Une fois la simulation validée, fermez ModelSim.

## 5) Test sur site

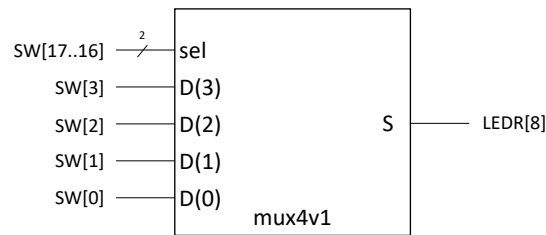
Il faut maintenant synthétiser pour une utilisation sur carte réelle. Accédez au menu « Processing → Start Compilation ».

Pendant la compilation, branchez la carte à l'alimentation et au PC. Attention, le port USB doit être branché sur le port « BLASTER » de la carte, tout en haut à gauche.

Une fois la compilation terminée, allumez la carte en appuyant sur le bouton rouge, puis accédez au menu « Tools → Programmer ». Cliquez sur le bouton « Hardware setup » et, dans la liste déroulante, sélectionnez « USB-Blaster ». Fermez la petite fenêtre, puis cliquez sur « Start ».

NOTE : Il est possible que la première fois, le processus échoue (« Failed » en haut à droite). Si c'est le cas, cliquez sur « Start » une seconde fois.

Les connexions avec les boutons et LEDs de la carte sont les suivantes :



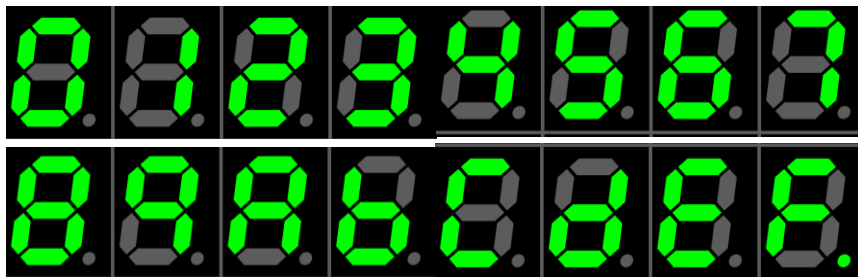
Vérifiez le bon fonctionnement du multiplexeur, et faites valider. Vous devrez être capable d'expliquer le comportement à l'enseignant.

## Exercice 2 : Décodeur 7 segments

Téléchargez le projet **TP\_1\_Ex\_2\_3** sur Moodle, décompressez-le dans votre dossier personnel et ouvrez-le avec Quartus 18. Ouvrez le fichier **deco7seg.vhd**.

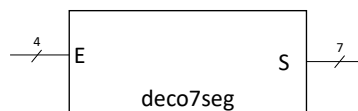
### 1) Table de vérité

Dresser la table de vérité complète (pour les 16 chiffres hexadécimaux) d'un décodeur 7 segments.



### 2) Entité

Remplissez le code de l'entité de manière à ce qu'elle corresponde à la vue externe ci-dessous :



### 3) Architecture

Codez l'architecture du composant à l'aide de l'instruction adaptée.

### 4) Synthèse

Placez le composant **deco7seg** en « top level » et synthétisez-le avec la commande vue dans l'exercice 1.

### 5) Simulation

Effectuez toutes les opérations vues à l'exercice 1 jusqu'à la création du chronogramme. Dans ModelSim, il vous faudra penser à compiler le testbench (**testbench\_deco7seg.vhd**).

Lancez la simulation pour une durée de 200  $\mu$ s.

Mettez votre chronogramme en plein écran à l'aide du bouton « + » en haut à droite. Nous allons maintenant modifier le rendu du signal E : faites un clic-droit sur le nom du signal dans la fenêtre du chronogramme, et dans le menu « Radix », choisissez « Hexadécimal ».

Vérifiez que les valeurs obtenues sur **S** correspondent bien à la valeur attendue, puis faites valider. Vous devrez expliquer ce que modifie le paramètre « Radix » à l'enseignant. Est-ce pertinent de modifier également le Radix de la sortie du composant ?

## 6) Test sur site

Chargez votre composant sur la carte. Les connexions sont les suivantes :



Testez toutes les combinaisons d'entrée, et vérifiez que le comportement est bien celui attendu.

Faites valider.

## Exercice 3 – Encodeur de priorité

Un encodeur de priorité est un composant qui détermine, parmi plusieurs entrées, l'entrée active ayant la plus forte priorité. Si aucune entrée n'est active, le composant fournit la valeur zéro. Si une ou plusieurs entrées sont actives, le composant fournit en sortie le numéro de l'entrée active ayant le poids le plus fort en binaire naturel. Pour éviter toute confusion sur la valeur zéro,  $n$  entrées sont numérotées de 1 à  $n$ .

### 1) Réflexion préliminaire

Sachant que l'on utilisera les interrupteurs de la carte pour stimuler les entrées de l'encodeur, répondre aux questions ci-dessous :

- Connaissant la valeur maximale qui peut être affichée sur un afficheur 7 segments, indiquez l'indice de l'interrupteur ayant le poids le plus fort pouvant être utilisé.
- Sachant que l'on n'utilise pas l'interrupteur 0, indiquer le range du vecteur à utiliser en entrée du composant : **inter[ ??? .. ??? ]**
- Dessinez la vue externe d'un composant nommé **encodeur\_prio** qui prend en entrée un vecteur nommé **inter** et fournit en sortie un nombre en binaire naturel nommé **numero**.

### 2) Table de vérité

Dresser la table de vérité pour un encodeur ayant 3 entrées **inter[3..1]**. Attention à bien choisir un type de table de vérité adapté !

En déduire l'idée générale d'une table de vérité d'un encodeur de priorités. Quel type de table de vérité avez-vous choisi pour ce composant ? Quelle instruction faudra-t-il utiliser en VHDL ?

Pour la suite, on utilisera le nombre d'interrupteurs déterminé dans les questions préliminaires.

### 3) Entité – architecture

Accédez au menu « Project ➔ Add/remove Files in project ». Cliquez sur le bouton « ... » à droite du champ « File name », puis ouvrez le fichier **encodeur\_prio.vhd** situé dans le dossier **src**. Validez.

Dans le fichier **encodeur\_prio.vhd**, remplissez l'entité puis l'architecture en utilisant l'instruction adaptée.

### 4) Synthèse - simulation

Placez le composant **encodeur\_prio** en « top level », puis lancez la synthèse, puis la simulation. Dans ModelSim, compilez le testbench (**testbench\_encodeur\_prio.vhd**).

Simulez pour une durée de 300 µs, et vérifiez que les cas présentés sont bien corrects.

### 5) Test sur site

Nous avons maintenant deux composants, qu'il va falloir raccorder. Nous n'avons pas encore vu cette notion, donc le composant réalisant le raccordement est fourni.

Ajoutez au projet le fichier **exo3\_complet.vhd** fourni dans le dossier **src** et mettez-le en « top level ». Compilez et chargez sur la maquette.

Vérifiez que le comportement est bien correct. Faites valider.

## Exercice bonus – Comptage des interrupteurs

### 1) Questions préliminaires

Dresser la table de vérité d'un composant réalisant le cahier des charges suivant :

Le composant prend en entrée un vecteur de 4 valeurs représentant 4 interrupteurs. Le composant doit compter le nombre d'interrupteurs étant à 1, et renvoyer cette valeur en binaire naturel sur sa sortie.

### 2) Codage

On utilisera comme base le projet déjà réalisé dans l'exercice précédent : copier l'intégralité du dossier **TP\_1\_Ex\_2\_3** sous le nom **TP\_1\_Ex\_bonus**.

Modifier l'architecture du composant **encodeur\_prio** pour réaliser le nouveau cahier des charges. Attention, on ne modifiera pas l'entité, et on dispose donc d'un vecteur d'entrée trop grand : il faudra donc accéder uniquement aux bits qui nous intéressent dans le vecteur **inter**.

### 3) Test sur site

Compiler le projet, et le charger sur la carte. Vérifier le bon fonctionnement et faire valider.