Verilog est un Hardware description language comme le VHDL. Il est très utilisé dans l’industrie donc on va voir comment implémenter des modèles simples en verilog.

Nous allons utiliser un logiciel de simulation nommé icarus iverlog qui permet à partir de spécification verilog de générer une simulation exécutable. Cette simulation nous permettra de générer des tests bench éventuellement affichables avec des logiciels de type waveform.

**Exercice 0**

Installer icarus iverilog et gtkwave

Le lien suivant peut donner des indication selon votre système : [https://iverilog.fandom.com/wiki/Installation\_Guide](https://iverilog.fandom.com/wiki/Installation_Guide#Ubuntu_Linux)

Autre lien : <https://bleyer.org/icarus>

En cas de problème persistant demander une VM Linux sur lequel ce sera installé

**Exercice 1**

Récupérer le code accessible à ce [lien](https://drive.google.com/drive/folders/1Vj6FT5TkdBil6kEXeT1GFCewu9C-fnZR?usp=share_link). Il implémente une porte logique ET et le fichier de simulation correspondant.

Pour afficher les waveform d’un test bench il faut faire trois étapes :

1. Compiler les modules verilog avec le test bench
2. lancer le binaire obtenu et afficher la trace textuelle du dump
3. lancer vvp sur l’executable obtenu
4. Charger dans gtkwave le fichier vcd obtenu.

**Exercice**

1. Faire en sorte de compiler le module et le testbench avec iverilog
2. Lancer la simulation avec vvp sur le fichier binaire obtenu
3. Afficher la waveform gtkform et vérifier que la porte logique se comporte de manière cohérente
4. Noter comment le module de la porte ET est défini

**Exercice 2 - Porte logique Ou**

1. Copier le module de la portie logique OU et le modifier pour qu’il implémente une porte logique OU
2. Reprendre le testbench de l’exercice précédent et vérifier que le module se comporte convenablement.

**Exercice Reg vs wire**

En faisant des recherches si nécessaire, expliquer la différence entre reg versus wire. Dans quels cas est-il plus pertinent d’utiliser l’un que l’autre ?

**Exercice - assign vs always**

En faisant des recherches si nécessaire, expliquer la différence entre assign vs XXX. Dans quels cas est-il plus pertinent d’utiliser l’un que l’autre ?

**Exercice - Porte logique ET à trois entrées**

1. Faire un module qui implémente une porte logique ET à trois entrée
2. Reprendre et modifier le fichier de test précédent pour qu’il marche avec trois entrées

**Exercice - Additionneur 1 bit**

1. On veut faire un additionneur 1 bit qui prend en entrée 2 nombre a et b, une retenue, et qui en sortie renvoie l’addition des deux nombres et l’éventuelle retenue de sortie. Implémenter ce module
2. Reprendre et modifier le fichier de test précédent pour faire en sorte de simuler le comportement de l’additionneur 1bit

**Additionneur 8bit**

faire en sorte d’avoir un module qui fait l’addition de deux nombres 8bit et qui renvoie le résultat

**Multiplieur 8 bit**

Faire un module multiplier\_8b qui prend deux vecteurs de bit de taille 8 et a en sortie un vecteur de bit de taille 16 et qui effectue la multiplication des deux entrées

**Régression linéaire simple**

Implémenter une régression linéaire qui utilise les modules précédents qui calcule

y = 10000 + 10000 \* X1 ou le prix est le prix de la maison et size est sa taille

Transpileur

Faire un script python qui serait capable de générer un module regression\_lineare à partir d’un modèle scikit-learn entraîné et qui permettrait de gérer un nombre quelconque de feature.

Bonus : Faire l’équivalent pour générer le code VHDL de la régression linéaire