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 indications 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**

Récupérer le code and\_gate.v et test\_and\_gate.v 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, vérifier que la trace est cohérente

**Exercice**

On veut afficher la trace de manière graphique. Pour cela on peut utiliser le logiciel gtkwave.

1. lancer la commande vvp sur l’executable obtenu dans l’exercice précédent
2. Charger dans gtkwave le fichier vcd obtenu.

**Exercice 2 - Porte logique Ou**

1. Copier le module de la portie logique ET 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 - Additionneur 1 bit**

1. On veut faire un additionneur 1 bit qui prend en entrée 2 nombre a et b, 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 16bit**

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

**Multiplieur 16 bit**

Faire un module multiplier\_16bit qui prend deux vecteurs de bit de taille 16 bit et a en sortie un vecteur de bit de taille 32 effectuant la multiplication des deux entrées

**Régression linéaire simple**

On veut faire un modèle qui prédit le prix d’une maison sur un FPGA à partir de son prix (ce n’est certe pas très utile de faire ça sur fpga mais ce n’est pas grave).

Implémenter un module Verilog qui réutilise les modules précédents afin de calculer la prédiction d’un modèle qui aurait la formule suivante : y = 10 000 + 5 000 \* size

où size représente la taille de la maison dont on veut estimer le prix

**Transpileur**

Faire un script python 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

**Exercice - assign vs always**

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