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**](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.**

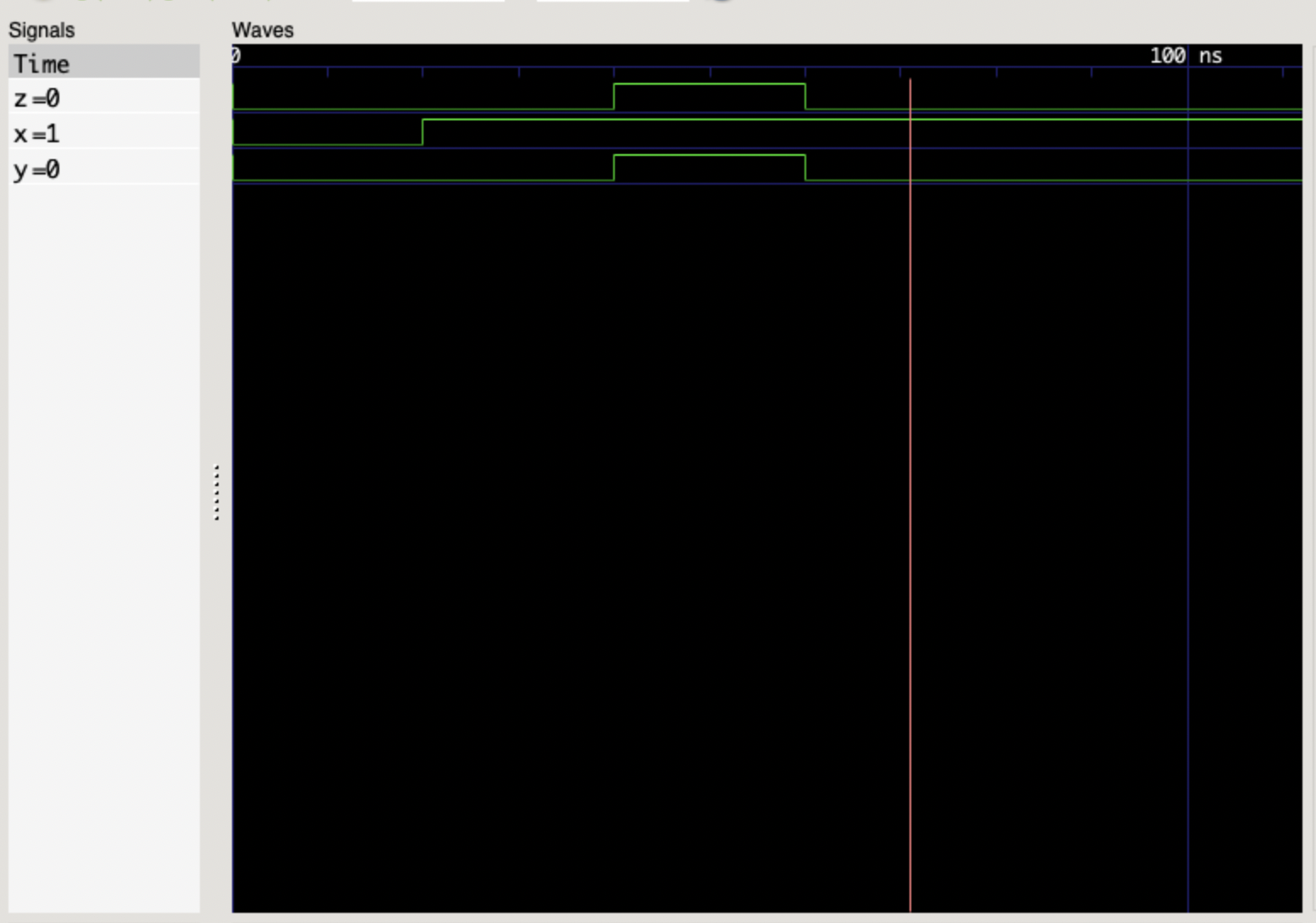
**Pour afficher les waveform d’un test bench il faut faire trois étapes :**

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**

Voici les résultats obtenus :

Une image contenant texte

Description générée automatiquement

1. **Noter comment le module de la porte ET est défini**

Le module de la porte ET est défini de manière à produire une sortie de 1 si et seulement si toutes les entrées sont de 1, sinon la sortie est de 0. Dans le scénario donné, la porte ET est utilisée pour contrôler la sortie z en fonction des entrées x et y.

Au début (t=0), toutes les entrées x, y et z sont à 0. Lorsque t=20, l'entrée x devient 1, mais les entrées y et z restent à 0, donc la sortie de la porte ET reste à 0.

Lorsque t=40, l'entrée y devient 1, donc les entrées x et y sont toutes deux à 1, ce qui entraîne une sortie de la porte ET à 1. Ainsi, la valeur de z devient également 1.

Cependant, lorsqu'on atteint t=60, l'entrée y redevient 0, ce qui signifie que l'une des entrées de la porte ET est 0, donc la sortie de la porte ET est également 0. En conséquence, la valeur de z redevient 0.

**Exercice 2 - Porte logique Ou**

1. **Copier le module de la porte logique OU et le modifier pour qu’il implémente une porte logique OU**

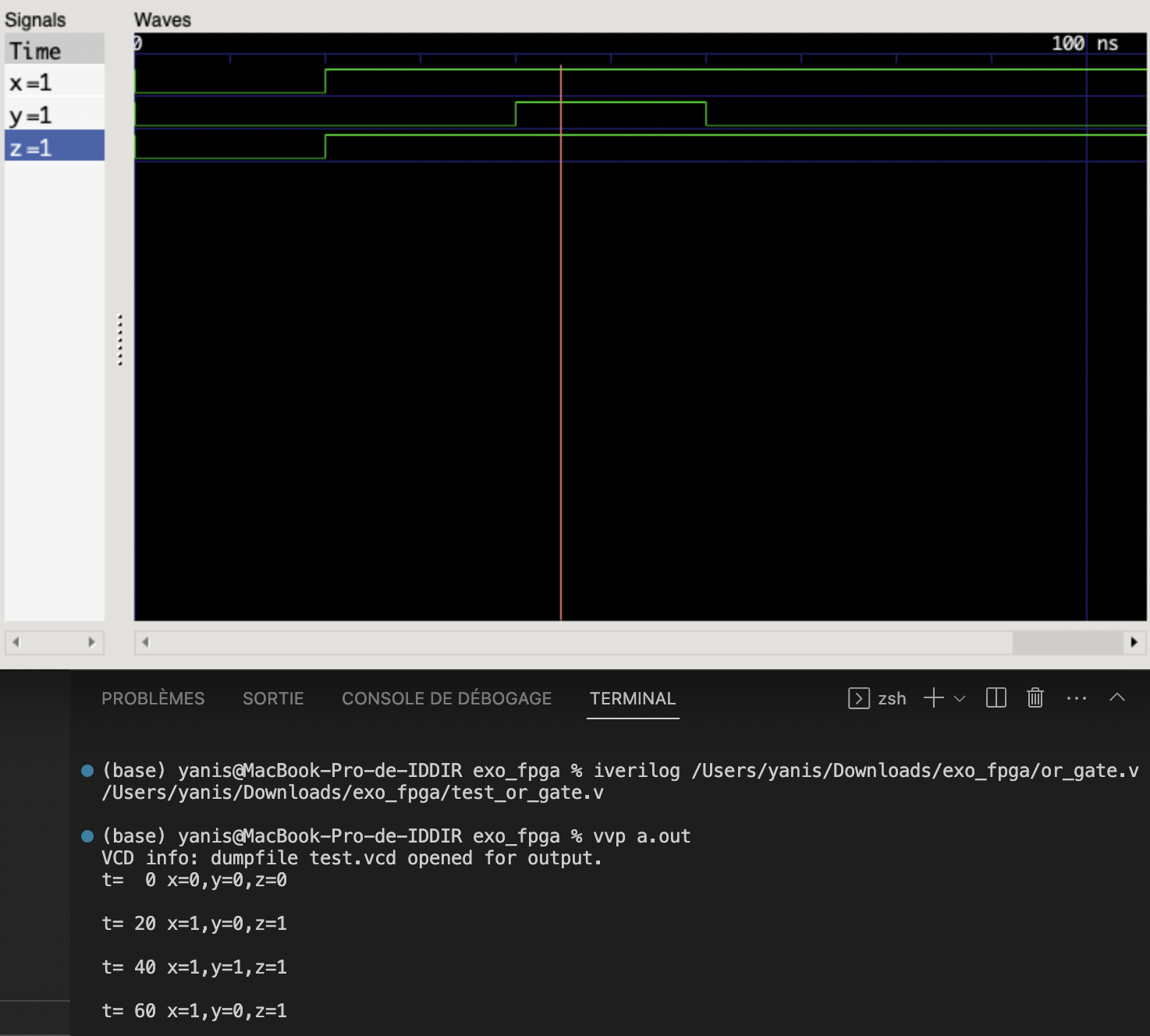
**Une image contenant texte

Description générée automatiquement**

1. **Reprendre le testbench de l’exercice précédent et vérifier que le module se comporte convenablement.**

**Une image contenant texte

Description générée automatiquement**

****

Ici, le module OR est utilisé pour contrôler la sortie z en fonction des entrées x et y. Le module OR est défini de manière à produire une sortie de 1 si au moins l'une des entrées est de 1, sinon la sortie est de 0.

Au début (t=0), toutes les entrées x, y et z sont à 0. Lorsque t=20, l'entrée x devient 1, ce qui entraîne une sortie de la porte OR à 1. En conséquence, la valeur de z devient également 1.

Lorsque t=40, les entrées x et y sont toutes deux à 1, ce qui entraîne une sortie de la porte OR à 1. Ainsi, la valeur de z reste à 1.

Cependant, lorsqu'on atteint t=60, l'entrée y redevient 0, mais la valeur de z reste à 1 car l'entrée x est toujours à 1. Cela signifie que le module OR fonctionne correctement car la sortie ne devient 0 que si les deux entrées sont 0.

**Exercice Reg vs wire**

**En effectuant 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 ?**

La principale différence entre "reg" et "wire" est que "reg" est une variable qui peut stocker des données sous forme d'un registre, tandis que "wire" est une variable qui peut être utilisée pour connecter des composants.

Un "reg" peut stocker une valeur en tant que registre, ce qui signifie qu'il peut être utilisé pour stocker des données sur plusieurs cycles d'horloge. En revanche, un "wire" est simplement un fil qui permet de connecter des composants, et il ne peut pas stocker de données sur plusieurs cycles d'horloge.

En général, il est plus pertinent d'utiliser un "reg" lorsque vous avez besoin de stocker des données sur plusieurs cycles d'horloge, comme dans le cas d'un compteur. D'autre part, un "wire" est plus approprié pour connecter des composants, comme dans le cas d'un bus de données

**Exercice - assign vs always**

**En effectuant 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 ?**

La principale différence entre "assign" et "XXX" dans Verilog est leur utilisation et leur comportement, "assign" est un opérateur qui est utilisé pour assigner une valeur constante à une variable.

D'un autre côté, "XXX" est utilisé pour déclarer une variable et décrire son comportement.

Il est plus pertinent d'utiliser "assign" lorsque vous avez besoin d'assigner une valeur constante à une variable, comme dans le cas d'une constante booléenne ou d'un signal d'horloge. D'autre part, il est plus approprié d'utiliser "XXX" lorsque vous avez besoin de décrire le comportement d'une variable, comme dans le cas d'un registre, d'un fil de connexion ou d'un paramètre de module.

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

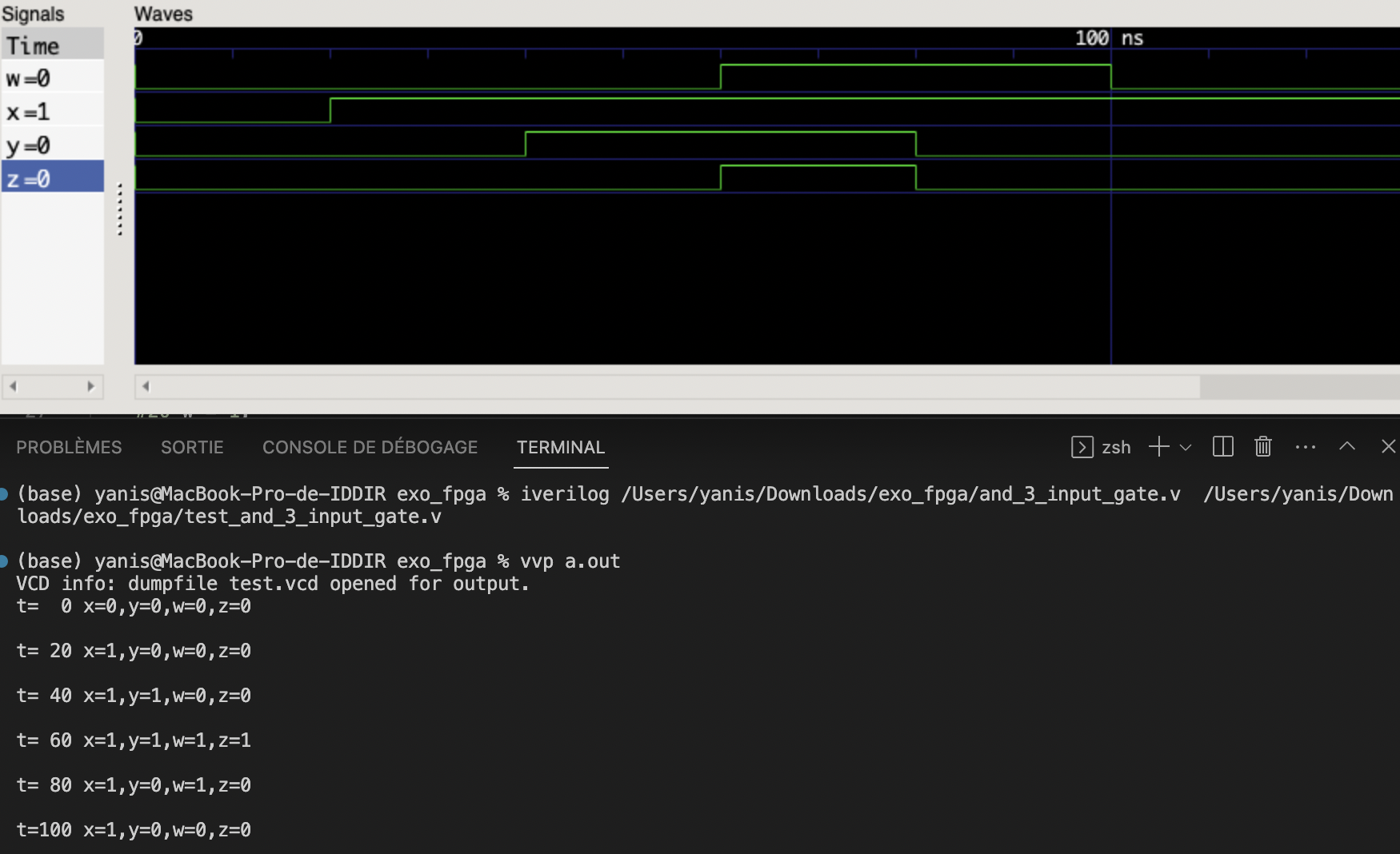
1. **Faire un module qui implémente une porte logique ET à trois entrée**

Une image contenant texte

Description générée automatiquement

1. **Reprendre et modifier le fichier de test précédent pour qu’il marche avec trois entrées**

Une image contenant texte

Description générée automatiquement

**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**

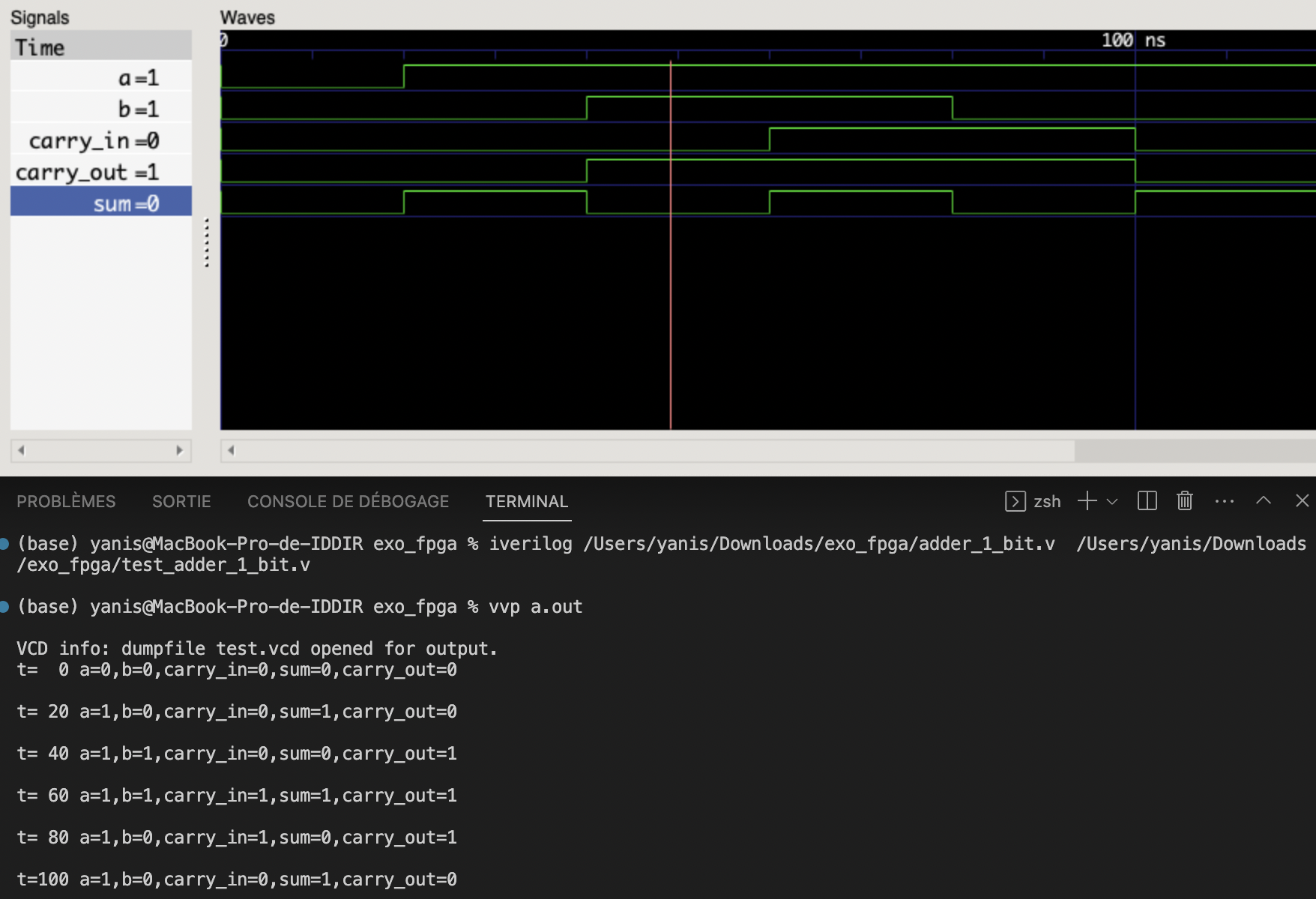
Une image contenant texte, écran, noir, capture d’écran

Description générée automatiquement

1. **Reprendre et modifier le fichier de test précédent pour faire en sorte de simuler le comportement de l’additionneur 1bit**

Une image contenant texte

Description générée automatiquement

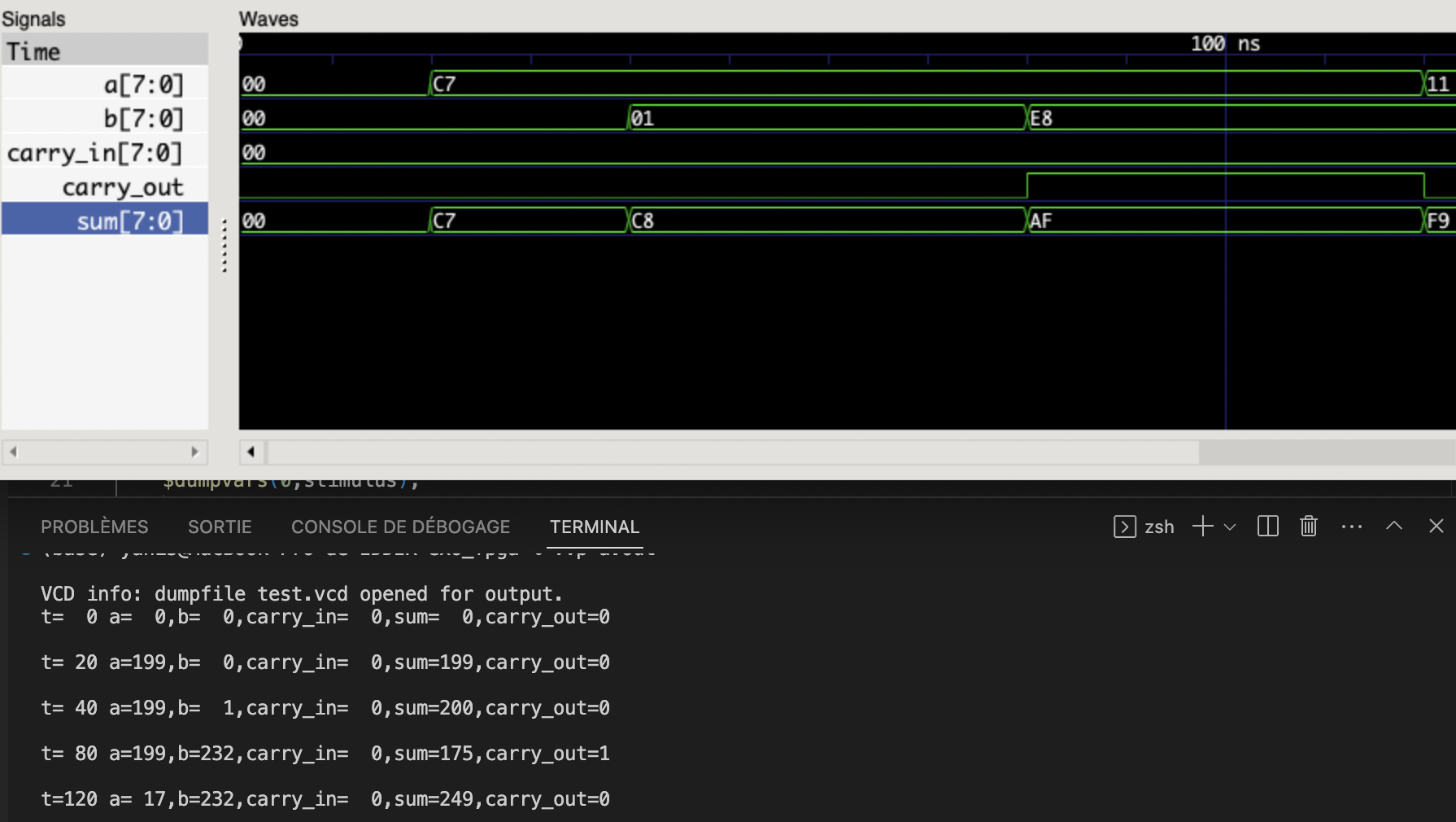


**Additionneur 8bit**

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

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

**Multiplieur 8 bit**

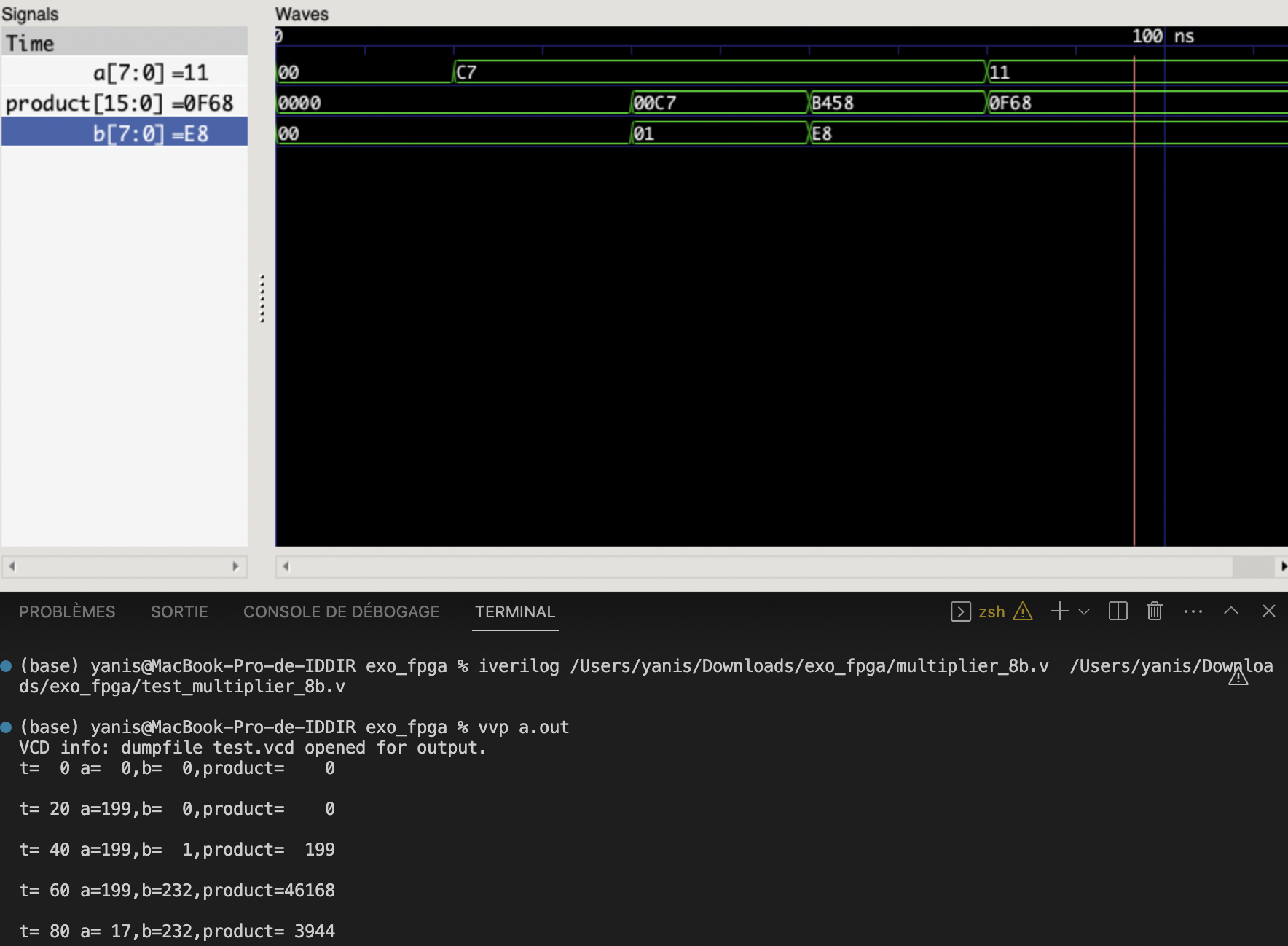
**Faire un module multiplier\_8b qui prend deux vecteurs de bit de taille 8 et a en sortie un vecteur de bit de taille 16 et qui effectue la multiplication des deux entrées**

**Une image contenant texte

Description générée automatiquement**

**Une image contenant texte

Description générée automatiquement**

****

**Régression linéaire simple**

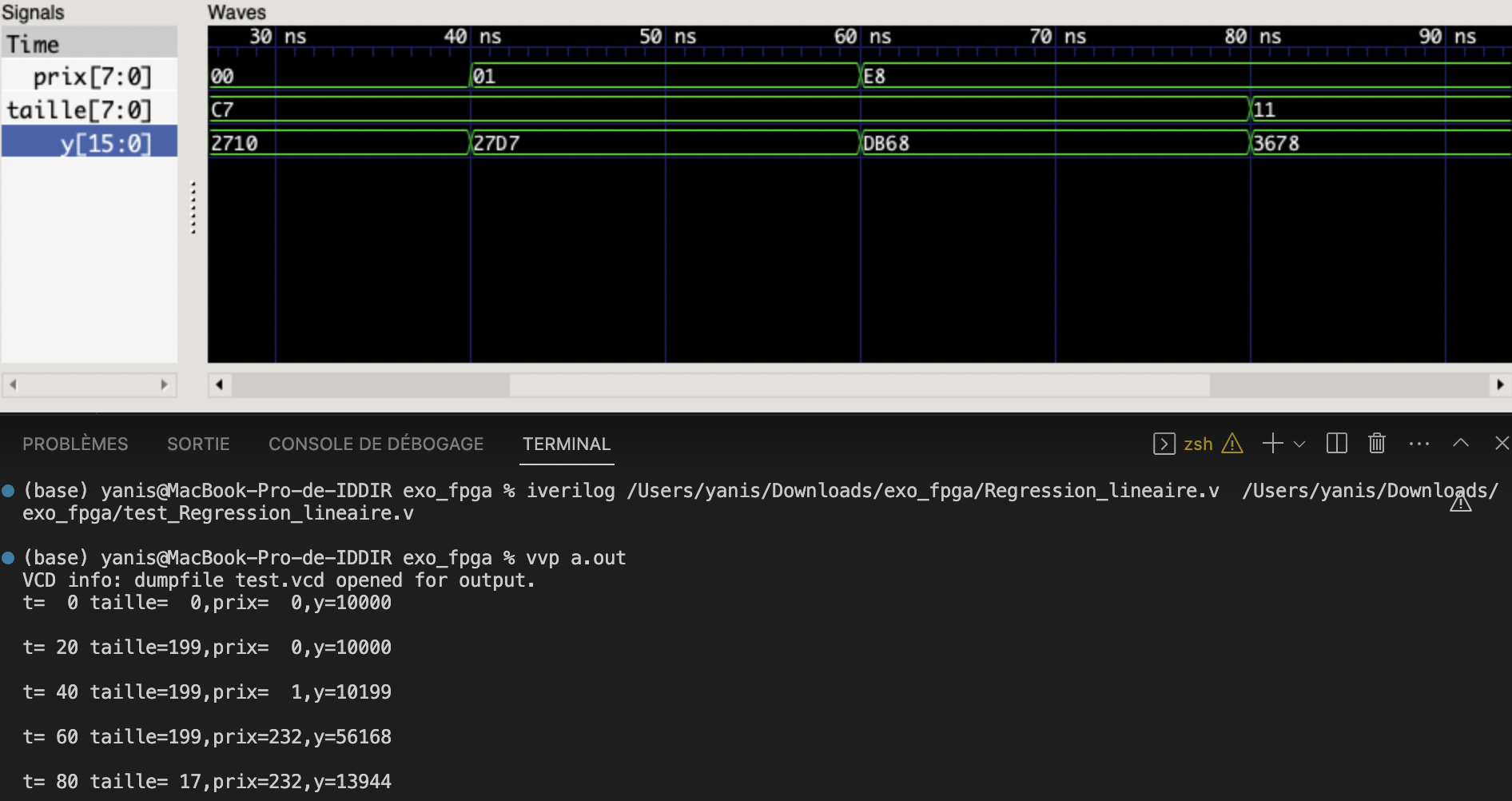
**Implémenter une régression linéaire qui utilise les modules précédents qui calcule**

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

**Une image contenant texte

Description générée automatiquement**

**Une image contenant texte

Description générée automatiquement**

**Transpileur**

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

**Une image contenant texte

Description générée automatiquement**

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

**Une image contenant texte

Description générée automatiquement**