HEIG-VD HAUTE ÉCOLE D'INGÉNIERIE ET DE GESTION DU CANTON DE VAUD

Prof. Yann Thoma

Laboratoire VSE

semestre d'automne 2023 - 2024

Laboratoire de vérification SystemVerilog
Datastream Analyzer

Introduction

Dans le cadre d'un système visant à récupérer des données d'électrodes enregistrant l'activité de neurones, nous sommes intéressés à vérifer un analyseur de données. Les données lui sont transmis sous forme de flux, et son rôle est d'extraire des statistiques sur des fenêtres d'observations. Les deux interfaces fonctionnent selon un protocole différent, qui seront détaillés plus loin.

Spécifications

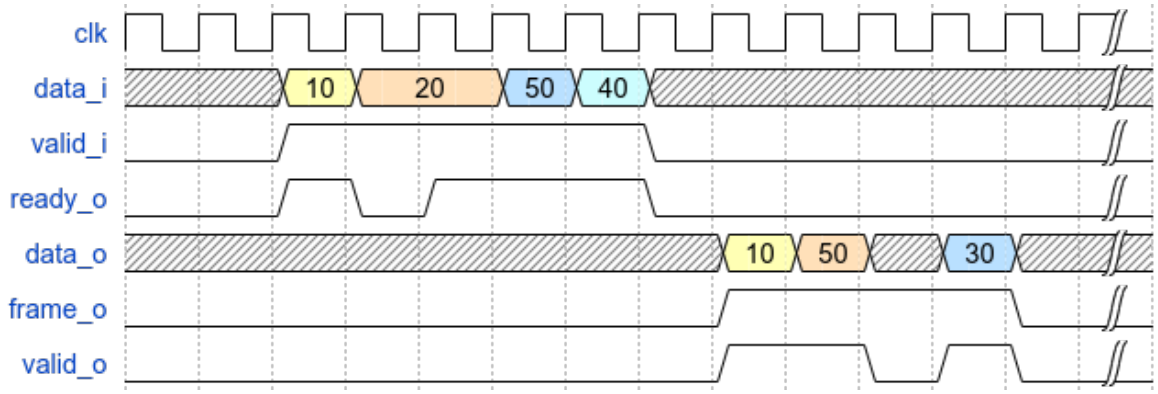
Le système que nous vérifions vise donc à analyser des *fenêtres* de données, et de donner des statistiques sur ces fenêtres. Une fenêtre, dont la taille est un paramètre générique du composant est donc analysée, et les résultats de l'analyse sont transmis immédiatement en sortie. Les données en entrée peuvent continuer à arriver pendant l'envoi des statistiques de la fenêtre précédente.
Il ne s'agit PAS de fenêtre glissante, c'est-à-dire qu'une fenêtre est analysée, puis une autre, sans *overlap*.
Le composant possède les ports suivants :

Il possède en outre trois paramètres génériques :

- `DATASIZE` : La taille des données, autant en entrée qu'en sortie
- `WINDOWSIZE` : La taille des fenêtres d'observation
- `ERRNO` : Un paramètre exploité pour injecter des comportements différents et des erreurs

Entrées/Sorties

Le chronogramme suivant illustre les protocole d'entrée et de sortie, avec une taille de fenêtre de 4 :



En entrée, les données sont transmises en parallèle de l'activation du port `valid_i`. Le composant accepte les données en activant le signal `ready_o`. Une donnée est donc considérée transmise lorsque ces deux signaux de contrôle sont actifs en même temps.
Lorsqu'une fenêtre a été analysée, le composant fourni en sortie les trois statistiques suivantes, dans cet ordre :

1. La valeur minimale observée dans la fenêtre
2. La valeur maximale observée dans la fenêtre
3. La moyenne des valeurs observées

La transmission se fait en activant le port `frame_o`, qui doit ensuite rester actif jusqu'à ce que la fenêtre ait été transmise. Les trois données sont transmises sur trois cycles d'horloge distincts, en parallèle de l'activation du signal `valid_o`. Le composant a donc possibilité de les envoyer les trois en trois cycles, ou en insérant des cycles *vides* entre les données valides.

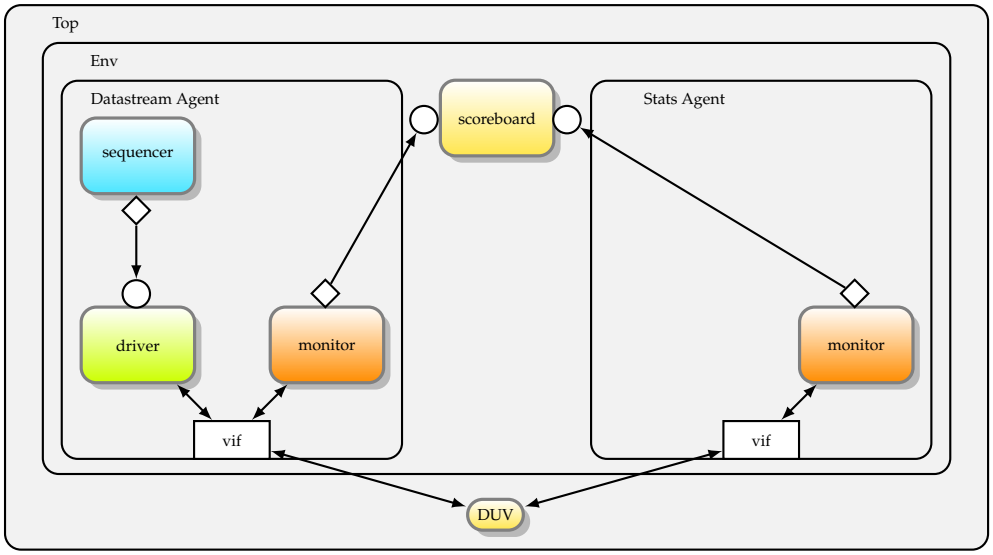
DUV

Un paramètre générique, `ERRNO`, permet d'injecter des erreurs dans le DUV. Il est également un paramètre du banc de test, et peut donc être forcé via le script de lancement de la simulation.
Le DUV fonctionne ainsi, en fonction de `ERRNO` :

— 0-2 : Fonctionnement normal
— 10-15 : Fonctionnement erroné

Architecture du banc de test

Le banc de test a été décomposé selon la méthodologie vue en cours, afin de bien différencier les responsabilités (séquenceur, driver, moniteur, scoreboard). La structure globale vous est déjà fournie, avec les instanciations de composants et les connexions via des FIFOs. Passez un peu de temps à bien comprendre cette architecture en regardant les fichiers SystemVerilog.



Vous pouvez noter les points suivants :

1. Les transactions envoyées par l'agent Input au scoreboard viennent d'un moniteur, ce qui permet de bien décorréler l'application des stimuli de l'observation de ce qui se passe sur le bus.
2. Il n'y a pas de driver de sortie, car rien n'est en entrée du DUV du côté sortie.

Le banc de test fourni offre donc déjà une structure qui se construit automatiquement.

— Le banc de test instancie le DUV et crée un objet `datastream_analyzer_env` qui contient toutes les fonctionnalités de test
— Le séquenceur a pour responsabilité de générer des séquences de test, et les envoyer au driver
— Le driver est responsable de jouer les séquences en intéragissant avec les entrées du DUV
— Le moniteur d'entrée doit observer ce qui se passe sur le bus pour reconstruire des transactions et les transmettre au scoreboard
— Le moniteur de sortie doit observer les sorties du DUV et reconstruire des transactions pour les transmettre au scoreboard
— Le scoreboard doit récupérer les informations du moniteur d'entrée et du moniteur de sortie afin de déterminer si tout se passe bien ou non
— Les interfaces du DUV sont déclarées dans des fichiers à part
— Les transactions, ainsi que deux types mailbox associés sont déclarés dans des fichiers à part
— L'utilisation de mailbox pour la communication entre les éléments permet de disposer de FIFOs. La taille des mailbox est définie à leur création, et les méthodes `put` et `get` sont bloquantes
— ⚠ Pour l'utilisation des mailbox il y a un point auquel il faut faire attention. Il faut être sûr que l'objet mis dans la mailbox ne sera plus réutilisé par l'envoyeur. Il faut donc avoir une nouvelle instance de l'objet à chaque fois.
— Une variable `testcase` est passée à chaque composant, via le script de lancement de la simulation.
— La décomposition en plusieurs fichiers devrait faciliter la collaboration dans le groupe

Outre la possibilité de simuler le système, vous avez à disposition la possibilité d'exploiter de la vérification formelle. Pour ce faire le script `check.do` permet de la lancer depuis le répertoire `comp`, via la commande suivante :

```
qverify -do ../scripts/check.do
```

Les assertions doivent se trouver dans le fichier `datastream_analyzer_assertions.sv`. Notez également que ces assertions sont également exploitées lors de la simulation. Si vous voulez les observer dans le chronogramme, vous pouvez modifier le fichier `sim.do` afin de les y ajouter manuellement (décommentez l'exemple et modifiez le nom de l'assertion en conséquence).

Travail

Votre travail consiste à développer un banc de test en SystemVerilog afin de pouvoir tester le comportement du composant fourni. Outre le développement de ce banc de test, un plan de vérification devra être réalisé et rendu en même temps que les sources. Celui-ci devra identifier les tests et scénarios permettant de tester les caractéristiques identifiées. Le plan de vérification est à intégrer dans le rapport, et celui-ci doit présenter les modifications que vous avez faites sur le code fourni.

Suggestions

Pour commencer nous suggérons de passer au travers du code fourni pour bien en comprendre la structure. Vous y trouverez plusieurs `TODO` qui correspondent à des endroits où du code doit

être modifié/ajouté. Vous avez évidemment le droit de modifier d'autres éléments, mais ces `TODO` sont le minimum vital.
Afin de vous faciliter le développement, nous suggérons de suivre les étapes suivantes :

1. Mettre en place un séquenceur basique et un driver. Ceci permet de visualiser ensuite une simulation dans le chronogramme.
2. Mettre en place le moniteur d'entrée
3. Mettre en place le moniteur de sortie
4. Mettre en place le scoreboard
5. Complexifier votre banc de test (randomisation/couverture/randomisation/...)