HAUTE ÉCOLE D'INGÉNIERIE ET DE GESTION DU CANTON DE VAUD

Prof. Yann Thoma

Laboratoire VSE

semestre d'automne 2024 - 2025

Laboratoire de vérification SystemVerilog
UART

Introduction

Nous désirons vérifier le bon fonctionnement d'un UART disponible sur bus avalon. Depuis le design FPGA il devra donc être accessible via ce bus, pour l'envoi de données, la récupération de données en lecture, et pour atteindre quelques bits de statut.

Spécifications

Soit un UART dont l'entité est la suivante :

Cet UART permet d'envoyer des octets sur une ligne série, et d'en recevoir également.
Pour l'envoi, il suffit d'écrire le mot à envoyer, à la bonne adresse. Celui-ci passe par un FIFO et sera ensuite envoyé en série sur la sortie `tx_o`. La FIFO d'envoi permet de stocker un certain nombre de mots. Si l'on écrit dans cette FIFO alors qu'elle est pleine, le mot est alors perdu.
Pour la réception, les données sérielles reçues sur `rx_i` passent par une FIFO de réception et sont mis à disposition en lecture via le bus avalon. Si la FIFO n'est pas vide, alors une lecture à son adresse renvoie le premier mot de la liste et le supprime de la FIFO. Si la FIFO est pleine et qu'un mot est reçu, alors celui-ci sera perdu.
Afin de gérer correctement le composant, un registre de statut est disponible et permet de connaitre l'état des FIFOS.
L'interface Avalon offre le plan d'adressage suivant :

Adresse	Taille	Direction	Description
0	X	Rd	Registre de statut
1	20	Wr	Envoi d'un mot
2	20	Rd	Récupération d'un mot
3	32	Rd/Wr	Nb de cycles d'horloge par bit

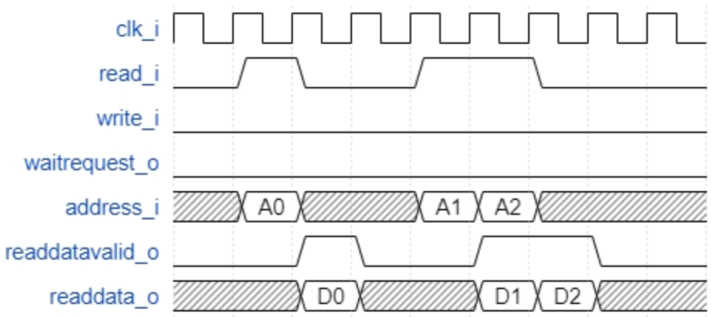
Les bits de status sont les suivants :

Bit	Description
0	Indique si le buffer d'envoi est plein
1	Indique si le buffer de réception est plein
2	Indique s'il y a un élément disponible dans le buffer de réception
3	Indique si le buffer d'envoi est vide

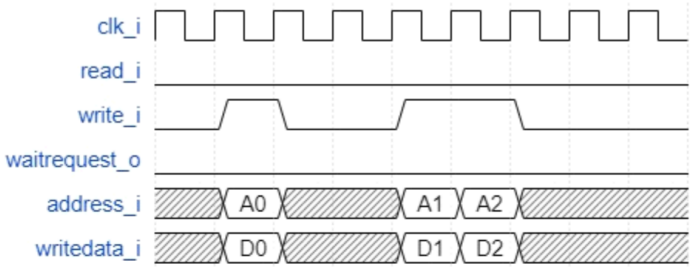
Le bus avalon est un bus légèrement simplifié et différent de celui vu en cours :

- Le signal `byteenable` n'est pas utilisé.
- En lecture, la donnée est prête un cycle après que `avl_read_i` est à `'1'`.
- En lecture, `avl_waitrequest_o` n'est pas utilisé
- En lecture, `avl_readdatavalid_o` s'active lorsque la donnée est disponible.
- En écriture, `avl_waitrequest_o` permet de faire patienter le master, selon le fonctionnement normal du bus avalon.

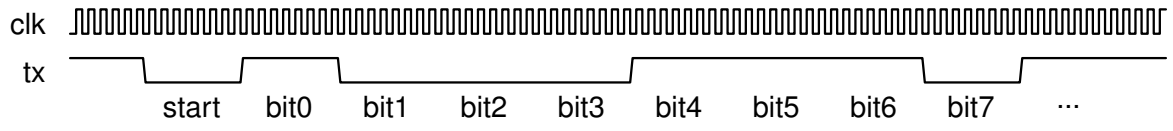
Le chronogramme suivant présente des exemples de lectures :



Le chronogramme suivant présente des exemples d'écritures :



Le chronogramme suivant donne l'exemple d'un envoi de la valeur 0x71 avec `parity_i` à '0', `stops_i` à '0' et `clk_per_bit` à 8.



DUV

Le banc de tests possède trois paramètres génériques :

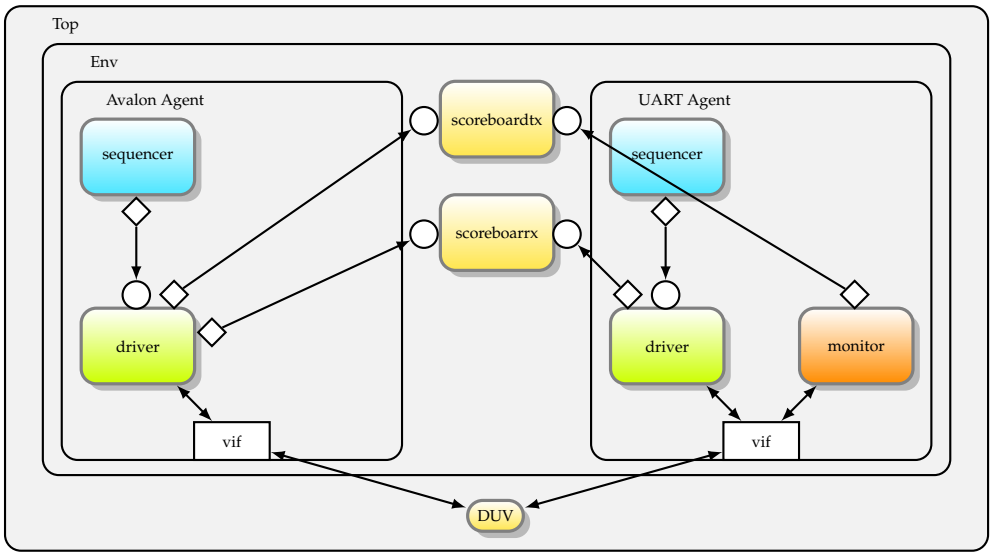
- `FIFOSIZE` : La taille des FIFO d'envoi et de réception
- `DATASIZE` : La taille des mots envoyés et reçus sur la ligne série
- `ERRNO` : Un paramètre exploité pour injecter des comportements différents et des erreurs

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
- 10-16 : 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 du driver, ce qui simplifie le banc de test.

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 `uart_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, tout en les transmettant au scoreboard
- Le moniteur de sortie doit observer les sorties du DUV et reconstruire des transactions pour les transmettre au scoreboard
- Les scoreboards doivent récupérer les informations de l'agent avalon et de l'agent UART 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 `avl_uart_interface_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 petit rapport doit être rendu en même temps que les sources. Celui-ci devra contenir la description de vos choix d'architecture, et devra identifier les tests et scénarios permettant de tester les caractéristiques du DUV que vous avez identifiées.

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 dans l'agent Avalon. Ceci permet de visualiser ensuite une simulation dans le chronogramme.
2. Mettre en place le moniteur de l'agent UART
3. Mettre en place le scoreboardtx
4. Mettre en place un séquenceur basique et un driver dans l'agent UART.
5. Mettre en place le scoreboardrx
6. Complexifier votre banc de test (randomisation/couverture/...)