HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Prof. Yann Thoma

Laboratoire VSN
semestre de printemps 2020 - 2021

Exercices de vérification SystemVerilog
Assertions pour bus avalon

Concept

Le bus Avalon est un bus synchrone développé par Altera. Il supporte différents modes de communication (simple, pipeline, burst), en fonction de la configuration des maitres/esclaves.
Les figures de ce document sont tirées de *Avalon Interface Specifications, version mai 2013*. Tous les détails du fonctionnement du bus peuvent y être trouvés, notamment dans les pages 22–28.
Attention ! Celle-ci n'est pas la dernière version de la spéc !
Par exemple, le signal `response` (disponible dans la version 09.2018) n'existe pas dans la version 05.2013.
Utilisez svp la version 2013 de la spécification, disponible avec les fichiers de l'exercice.

Exercice 1

Pour tous les exemples suivants, proposez des assertions sous forme textuelle. Vous pouvez les placer dans le fichier `avalon_assertions.sv` sous forme de commentaires.

Exercice 2

A partir des assertions textuelles de l'exercice précédent proposez des assertions SystemVerilog permettant de vérifier que le protocole est bien respecté par le maitre et l'esclave. Pour ce faire, considérez une horloge `clk` et un reset `rst`, et utilisez les signaux tels que spécifiés dans les chronogrammes. Les assertions doivent être placées dans le fichier `avalon_assertions.sv`. Le paramètre générique `AVALONMODE` spécifie le mode de fonctionnement du bus, et est utilisé pour générer les assertions correspondantes grâce à la clause **`generate if`**.

AVALONMODE	Description
0	Simple wait request
1	Attente fixe
2	Pipeline variable
3	Pipeline fixe
4	Burst

Ce paramètre générique est passé au banc de test. Le script de lancement, `sim.do` prend en argument ce paramètre.
Pour pouvoir observer les assertions dans le chronogramme, il faut modifier le fichier `sim.do` en fonction du nom des assertions que vous aurez choisis. Regardez l'exemple dans le fichier et inspirez-vous en.

Simple Wait request (sect. 3.5.1)

Dans le mode simple *wait request*, l'esclave exploite le signal `waitrequest` pour faire patienter le maitre. En le plaçant à 1, les données/contrôle envoyés par le maitre doivent rester stables, et ce jusqu'à ce que `waitrequest` repasse à 0. Ceci est autant valable pour les lectures que pour les écritures, comme illustré par la figure 1. Il n'y a pas de limitation sur le temps durant lequel `waitrequest` reste à 1.

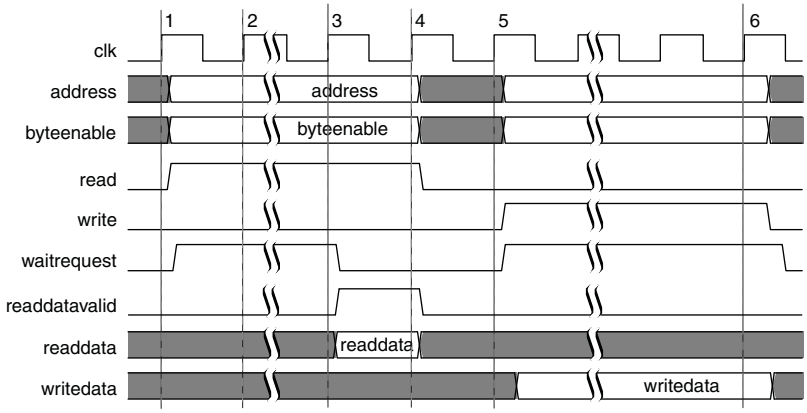

FIGURE 1. *Exemple de lecture/écriture avec utilisation de waitrequest*

Attente fixe (sect. 3.5.2)

Dans le mode à attente fixe, les lectures et écritures se font avec un nombre de cycles d'attente fixes. Le comportement est identique à la situation avec `waitrequest` si l'esclave laisse `wait_request` à 1 pendant n cycles. Le nombre de cycles peut être différent pour l'écriture et la lecture, et le tout est illustré à la figure 2, où $n = 1$ pour la lecture et $n = 2$ pour l'écriture.

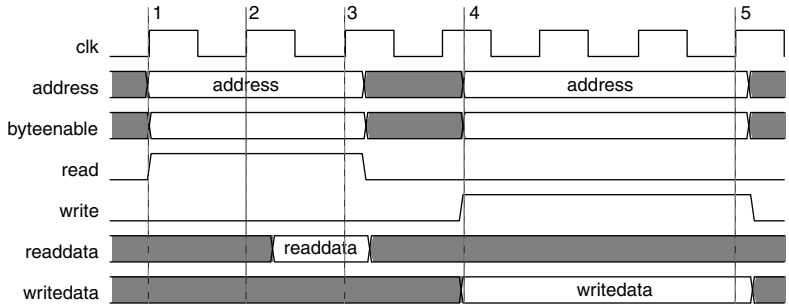

FIGURE 2. *Exemple de lecture/écriture avec attente de durée fixe*

Pipeline (sect. 3.5.3)

Les accès en lecture peuvent être fait de manière pipelinée. Dans la version avec délai variable (cf. Figure 3), le signal `waitrequest` permet à l'esclave de faire patienter le maitre jusqu'à ce qu'il puisse mettre les données valides sur le bus. Le signal `readdatavalid` lui permet d'indiquer que des données valides sont présentes sur le bus.
Dans le mode avec délai fixe, les données sont toujours présentes sur le bus un délai fixe après le passage de `waitrequest` à 0. La figure 4 illustre ce mode de fonctionnement, avec un délai fixe de 2 cycles.

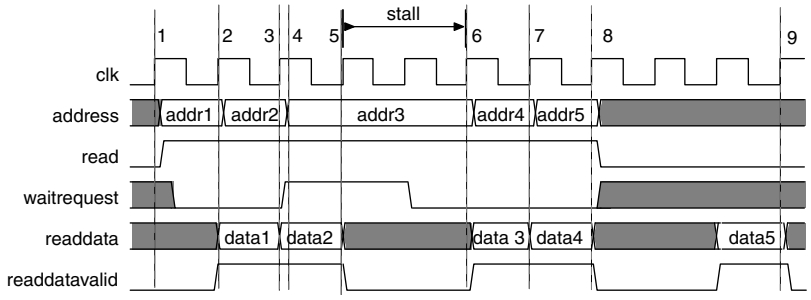

FIGURE 3. *Exemple de lecture en mode pipeline avec délai variable*

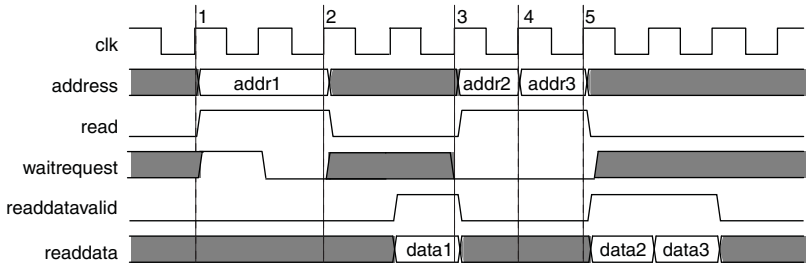

FIGURE 4. *Exemple de lecture en mode pipeline avec délai fixe*

Burst (sect. 3.5.4)

En mode Burst, le maitre ne fournit qu'une adresse de base, ainsi qu'un nombre de mots à transférer. Il asserte le signal `beginbursttransfer` et attend que `waitrequest` passe à 0. Il peut alors demander un nouveau transfert de type burst. Les données sont ensuite livrées par l'esclave après une attente quelconque. Il le fait en assertant le signal `readdatavalid`, comme illustré à la figure 5.

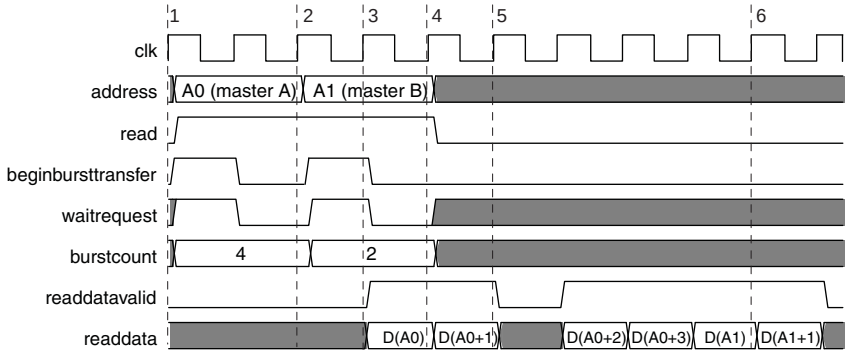

FIGURE 5. *Exemple de lecture en mode burst*

En écriture, un transfert de type burst doit attendre que le transfert précédent soit terminé. Là aussi, une adresse de base et le nombre de mots à transmettre sont indiqués en même temps que le signal `beginbursttransfer` est asserté. Le signal **`write`** indique que le maitre envoie un mot, et `waitrequest` permet à l'esclave de faire patienter le maitre jusqu'à ce qu'il soit prêt. La figure 6 illustre ce type de transfert.

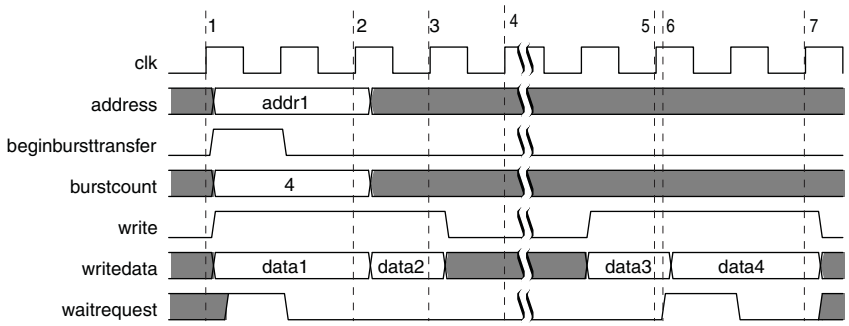

FIGURE 6. *Exemple d'écriture en mode burst*