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

Prof. Yann Thoma

Laboratoire VSN

semestre de printemps 2017 - 2018

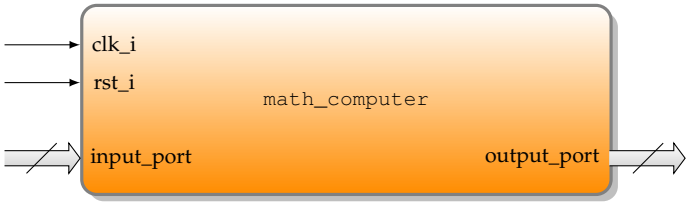
Exercices de vérification SystemVerilog
Vérification d'un bloc mathématique

Composant à tester

Nous souhaitons tester un module capable d'effectuer différentes opérations mathématiques. Les opérations impliquent jusqu'à 3 opérandes et peuvent prendre un nombre de cycle variable pour être exécutés.

Spécifications

Dans sa première version, le système n'effectue qu'une simple addition des deux premières entrées. Un paramètre générique, `DATASIZE`, permet de spécifier la taille des opérandes et du résultat.



Entrées/sorties

Les interfaces d'entrées/sorties sont celles les suivantes :

```
interface math_computer_input_itf;
    logic[`DATASIZE-1:0] a;
    logic[`DATASIZE-1:0] b;
    logic[`DATASIZE-1:0] c;
    logic valid;
    logic ready;

    modport master (
        output a, b, c, valid,
        input  ready
    );

    modport slave (
        input  a, b, c, valid,
        output ready
    );

endinterface
```

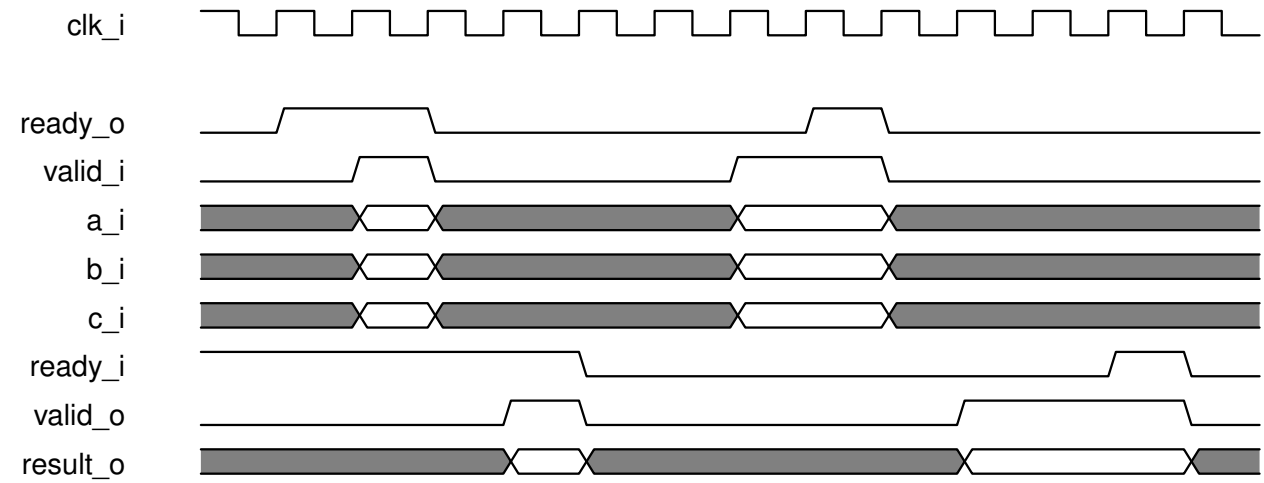
```
interface math_computer_output_itf;
    logic[`DATASIZE-1:0] result;
    logic valid;
    logic ready;

    modport master (
        output result, valid,
        input  ready
    );

    modport slave (
        input  result, valid,
        output ready
    );

endinterface
```

Vous noterez l'utilisation d'interfaces pour les ports d'entrée/sortie. Ceci permet de simplifier un design en ayant moins de connexions à faire. Les calculs sont lancés via l'activation de l'entrée `input_port.valid`, et ce uniquement si la sortie `input_port.ready` est à '1'. En sortie, le résultat doit être présenté en même temps que le signal `output_port.valid` est activé, et l'activation de l'entrée `output_port.ready` permet au composant de présenter une nouvelle valeur en sortie.
Les interfaces sont déclarées dans le paquetage `math_computer_itf.sv`.
Le chronogramme suivant illustre le protocole d'entrée/sortie.



Un paramètre générique `ERRNO` permet d'injecter artificiellement des erreurs dans le design. Il s'agit d'un entier qui offre le comportement suivant :

1. S'il vaut 0, le résultat est valide ;
2. S'il est compris entre 1 et 3 le composant souffre de quelques défauts.

⚠ L'entité à tester ne doit pas être modifiée.

Exercice 1

1. Reprenez le code du math_computer, ainsi que de son banc de test.
2. Observez les différentes constructions liées au langage SystemVerilog
   - Utilisation de macros (cf. fichier `math_computer_macros.sv`)
     - Déclaration de la taille des données ici
   - Utilisation d'interfaces (cf. fichier `math_computer_itf.sv`)
     - Une interface pour le port d'entrée et une pour le port de sortie
   - Utilisation d'un clocking block dans le banc de test
3. Ajoutez un nouveau testcase qui génère des signaux entrées `a` et `b` aléatoires. Faites également ce qu'il faut pour que les signaux `valid` et `ready` soient bien gérés par le banc de test. Il n'est pour l'instant pas nécessaire de séparer le processus en plusieurs tâches parallèles.
   - Ajoutez une tâche pour votre testcase
   - Modifiez le script de lancement pour lancer ce testcase

Exercice 2

1. Créez une classe contenant des variables représentant les entrées du DUT (pas le contrôle, les valeurs)
2. Faites de ces variables des variables aléatoires
3. Modifiez votre banc de test pour utiliser un objet de cette classe qui se verra affecter des valeurs aléatoires et dont ces valeurs seront ensuite utilisées pour forcer les entrées du DUT
4. Créez une sous-classe dans laquelle vous ajoutez les contraintes suivantes :
   — a doit être plus petit que 10 le 50% du temps
   — Si a est plus grand que b, alors c doit être plus petit que 1000
   — a et b ne doivent jamais être paires en même temps
5. Utilisez cette sous-classe à la place de la classe de base

Exercice 3

Nous allons utiliser les fonctionnalités de couverture pour déterminer le moment de fin de la simulation. Pour ce faire :

1. Créez un groupe de couverture permettant d'observer les entrées du système
2. Créez un groupe de couverture permettant d'observer les sorties du système
3. Lancez la simulation et observez le taux de couverture
4. Modifiez le code de manière à ce que la simulation se termine lorsque le taux de couverture du groupe de couverture des entrées est à 100%
5. Comment se passe la simulation ?
6. Les données sont de grande taille, modifiez vos groupes de couverture en y incluant des boites que vous jugez pertinentes
7. Ajoutez des points de couverture croisés que vous jugez pertinents entre a et b et observez la conséquence sur la simulation.
8. Modifiez le banc de test de manière à ce que le groupe de couverture des sorties soit aussi pris en compte pour la fin de la simulation. Pour ce faire, au lancement du testcase utilisez **`fork`** pour lancer une tâche responsable d'observer la couverture à chaque cycle d'horloge et de prendre la décision de terminer la simulation.

Exercice 4

Nous allons remplacer le code vérifiant le bon fonctionnement du système par des assertions. Pour ce faire :

1. Décrivez sous forme littéraire les faits devant être vérifiés
2. Reprenez chacun des points et traduisez-le sous forme d'assertion
3. Placez les assertions dans le fichier `math_computer_assertions.sv`
4. Effectuez une vérification formelle avec QuestaFormal en lançant le script `check.do` depuis le répertoire `sim`, via la commande

   ```
   qformal -do ../scripts/check.do
   ```