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

Affichage linéaire d'une valeur entre deux bornes

TIC

unité d'enseignement VSE

Auteurs: **Alexandre Iorio**

Professeur: **Yann Thoma**

Assistant: **Clément Dieperink**

Salle de laboratoire **A07**

Date: **26.10.2024**

Table des matières

0. Conditions de réalisation

Ce laboratoire a été réalisé en utilisant le logiciel `QuestaSim` pour la simulation et la vérification de composants.

Le logiciel est exécuté dans l'environnement virtuel `Ubuntu 64-bit reds-2024-02-daring-duck` fourni par la HEIG-VD.

L'attribution des ressources à la machine virtuelle est la suivante

- 4 CPU
- 32 GB de RAM

Détail de la machine hôte:

```
OS:         Ubuntu 24.04.1 LTS x86_64
Host:       Latitude 5520
Kernel:     6.8.0-47-generic
CPU:        11th Gen Intel i7-1185G7 (8) @
4.800GHz
Memory:     64032MiB
```

La durée d'exécution des processus est donc relative aux performances de la machine hôte et de l'environnement virtuel.

1. Introduction

Dans le cadre de ce laboratoire, nous allons tester un composant `min_max_top` permettant l'affichage linéaire d'une valeur comprise entre deux bornes `Min` et `Max`. Le but est de simuler et de valider le bon fonctionnement de ce composant en utilisant différentes configurations de tests et en intégrant des paramètres génériques tels que la taille des valeurs `VALSIZE` et le niveau d'erreur `ERRNO`. Les étapes incluent la définition de scénarios de tests, l'automatisation des vérifications, et l'analyse de la couverture pour garantir la fiabilité du système testé.

2. Commandes

Afin de pouvoir utiliser le `testbench` il est nécessaire de disposer de `QuestaSim`.

Le script `sim.do` à disposition permet de lancer le `testbench` avec différentes configurations, à savoir :

- `TESTCASE` : le numéro du scénario de test à exécuter
- `VALSIZE` : la taille des valeurs à afficher
- `ERRNO` : le niveau d'erreur à simuler

```
vsim do ../scripts/sim.do all <TESTCASE>
<VALSIZE> <ERRNO>
```

Ci-dessous des exemples de commandes permettant la compilation ainsi que l'exécution du `testbench` avec `TESTCASE` = 0, `VALSIZE` = 5 et `ERRNO` = 0.

Pour exécuter le testbench dans l'interface graphique de `QuestaSim`, exécuter la commande suivante :

```
vsim do ../scripts/sim.do all 0 5 0
```

pour exécuter le testbench dans le terminal, exécuter la commande suivante :

```
vsim -c -do "do ../scripts/sim.do all 0 5 0"
```

Si aucun argument n'est passé, le script `sim.do` exécutera le testbench avec les paramètres suivants : `TESTCASE` = 0, `VALSIZE` = 4 et `ERRNO` = 0.

Pour exécuter dans le terminal, se placer dans le dossier `sim` et exécuter la commande suivante :

```
vsim -c -do ../scripts/sim.do
```

Pour exécuter dans l'interface graphique de `Questasim` se placer dans le dossier `sim` et exécuter la commande suivante :

```
vsim do ../scripts/sim.do
```

Afin de lancer une serie de tests, il est possible d'utiliser la commande suivante :

```
vrun directed
```

Pour utiliser l'interface graphique on peut utiliser la commande suivante :

```
vrun -gui directed
```

2.1 Ordre des paramètres du script `sim.do`

l'exécution du script `sim.do` avec la configuration mentionnée ci-dessus exécutera la commande `do_all` dont la l'implémentation est la suivante :

```
proc do_all {TESTCASE VALSIZE ERRNO} {
          compile_duv
          compile_tb
          sim_start $TESTCASE $VALSIZE $ERRNO
        }
```

2.2 Explications des paramètres

2.1.1 TESTCASE

TESTCASE est un entier qui définit le scénario de test à exécuter. Il est compris entre 0 et 3. Chaque scénario de test est défini au chapitre 4 et permet de tester une configuration spécifique du composant. Le testcase 0 est un test de base qui exécutera la totalité des scénarios de test.

2.1.2 VALSIZE

VALSIZE est un entier qui définit la taille en `bit` des valeurs à afficher. Cette valeur débute à 2.

Une valeur supérieure à 26 bits n'est pas valide et créera une erreur de type `Troube with Simulation Kernel`. Quant à une valeur inférieur à 2 bits, elle créera des erreurs d'index au lors de la compilation du DUV.

```
** Error: ../src_vhdl/min_max_top.vhd(17):
(vopt-1152) Index value 2 is out of index range 1
downto 0 of ieee.std_logic_1164.STD_LOGIC_VECTOR.
** Error: (vopt-2064) Compiler back-end code
generation process terminated with code 2.
```

Note, le module `min_max_top` étant obfuscé, il est impossible de déterminer la taille des valeurs à afficher. Il est donc recommandé de ne pas dépasser la valeur de 24 bits.

Nous traiterons des limitations de la taille de `VALSIZE` directement lors de la création de la class de génération des valeurs.Celle-ci arrêtera, le testbench en cas de dépassement de la taille de `VALSIZE`.

note: Après utilisation du testbench, on s'apercoit que le temps de compilation du DUV ainsi que tout les actions qui précèdent l'instanciation de la class `value_generatot`, par l'appel d'une classe enfant, est realtivemen long. Il serait donc préférable de limiter la taille de `VALSIZE` hors de la class. Cela n'a pas été fait afin de maintenir un code compréhensible et modulaire.

2.1.3 ERRNO

Pour une valeur de ERRNO comprise entre 0 et 3, le résultat est valide. Pour une valeur de ERRNO comprise entre 16 et 21, le résultat n'est pas valide.

3. Structure du banc de test

Le banc de test met à disposition deux interfaces.

- `interface min_max_in_itf` : interface permettant de définir des valeur en entrée du composant `min_max_top`
- `interface min_max_out_itf` : interface permettant de récupérer la valeur de sortie du composant `min_max_top`

Voici la structure des interfaces :

```
interface min_max_in_itf#(int VALSIZE);
            logic[1:0] com;
            logic[VALSIZE-1:0] max;
            logic[VALSIZE-1:0] min;
            logic osci;
            logic[VALSIZE-1:0] value;
        endinterface
```

```
interface min_max_out_itf#(int VALSIZE);
            logic[2**VALSIZE-1:0] leds;
        endinterface
```

Nous avons décidé de structurer le testbench avec des classes.

Dans notre cas nous avons 3 classes.

- `value_generator` : Une classe parente permettant d'éviter la redondance de code pour la génération de valeurs des scénarios, elle contient aussi les contraintes relatives à l'entier des scénarios ainsi que des *utils* permettant l'affichage d'informations pouvant être utiles pour le debug.
- `random_value_generator` : Une classe fille de `value_generator` permettant de générer des valeurs aléatoires pour toutes les propriétés de l'interface `min_max_in_itf`.
- `boudary_value_generator` : Une classe fille de `value_generator` permettant de générer des valeurs aux limites pour toutes les propriétés de l'interface `min_max_in_itf`. Cette classe contient ses propres informations de coverage.
- `full_random_generator` : Une classe fille de `random_values_generator` qui utilise les fonctions de randomisation de la class parent. Cette classe contient ses propres informations de coverage.
- `mode_generator` : Une classe fille de `random_values_generator` qui utilise les fonctions de randomisation de la class parent mais applique un mode spécifique. Cette classe contient ses propres informations de coverage.

3.1 Ordonnancements des actions

De manière à ce que le banc de test possède les valeurs nécessaires aux bons instants, nous créons les valeurs sur les flancs d'horloge montant et nous les vérifions sur les flancs d'horloge descendants. De cette manière.

4. Scénarios de test

Afin de couvrir un maximum de comportements du module `min_max_top`, plusieurs scénarios de test ont été définis.

4.1 Scénario de test : Full Random

Ce scénario de test est le plus complet et permet de tester le module `min_max_top` avec des valeurs aléatoires générées dans le range des valeurs possibles de l'interface.

4.1.1 Coverage de Full Random

Afin de garantir un coverage des valeurs de 100%, nous vérifions que chaque valeur de l'interface ait été testée sur la plage [0, 2**VALSIZE-1], max allant de la valeur max / 2 a la valeur max, min allant de 0 à la valeur (max / 2) - 1. C'est trois valeurs sont réparties dans 4 boites.

4.1.2 Limites de Full Random

Le scenarion Full Random couvre des tailles de valeurs jusqu'à `VALSIZE = 18`, cela de manière à avoir un temps raisonnable situé en dessous de `5 - 6 [m]`. Au delà de cette valeur, le temps d'exécution du testbench est trop long et le système devient instable.

Par l'absence de valeur explicitement indiquée, nous admettons que la couverture de `VALSIZE = 18` est suffisante pour garantir le bon fonctionnement du module `min_max_top`. L'affichage de `256K` leds semble suffisant pour un `binlin`.

Étant donné l'obfuscation du `DUV` nous ne nous intéresserons pas aux instabilités du système dans ce laboratoire.

4.2 Scénario de test : Boundary

Ce scénario de test permet de tester le module `min_max_top` avec des valeurs aux limites des bornes possibles. Les valeurs de `min` et `max` sont définies à `0` et `2**VALSIZE-1`.

Quant à la valeur `value`, elle va osciller entre `0` et `2**VALSIZE-1` jusqu'à que toutes les combinaisons de `osci`, `com` et `value` avec `0` et `2**VALSIZE-1` aient été testées.

4.2.1 Coverage de Boundary

Afin de garantir un coverage des valeurs de 100%, nous vérifions que toute les combinaisons de `osci`, `com` et `value` avec `0` et `2**VALSIZE-1` aient été testées.

4.2.2 Limites de Boundary

Ce scenario peut couvrir les tailles de `VALSIZE` jusqu'à la limite du systeme etant donné que taille de `VALSIZE` n'est pas un facteur limitant pour ce scénario de test, cependant afin d'être raisonnable, nous avons limité la taille de `VALSIZE` à 21.

4.3 Scénario de test : Mode

Un scénarion de test spécifique pour tester un mode précis du module `min_max_top` à été mis en place sur le mode `00`. Cela de manière à demontrer de la possibilité de tester des modes spécifiques avec l'architecture mise en place. Ce test est quasiment similaire au Full Random, mais sur un mode spécifique.

Ce test à exactement les mêmes limites que le Full Random.

4.4 Scénarion de test complet

En executant le scénario de test avec un `TESTCASE = 0`, le testbench exécutera les trois scénarios de test précédents. Bien evidemment, il y a de la redondance, à savoir que le scénario de test Full Random est executé deux fois pour un même mode ainsi que pour le scénario testant les limites pour un `VALSIZE` inferieur ou égal à 6.

5. Génération des réferences

Le fonctionnement du DUV etant connu, il nous est possible de générer des références pour les différents scénarios de test.

Pour ce faire, nous avons besoin de quatres taches :

Commande	Fonction	Description
00	Marche normale	Si Valeur est compris entre Max et Min (bornes incluses)
		— les leds de Min à Val sont allumées avec intensité forte
		— les leds de (Val+1) à Max sont allumées avec intensité faible
		— toutes les autres leds sont éteintes
		Si Valeur est hors de l'intervalle [Max Min] toutes les leds sont éteintes
01	Mode linéaire	Affichage de Val en linéaire, Leds de 0 à Val sont allumées avec une intensité forte
10	Test éteint	Toutes les leds de l'afficheur sont éteintes (état '0')
11	Test allumé fort	Toutes les leds de l'afficheur sont allumées avec une intensité forte (état '1')

En fonction du mode de fonctionnement déterminé dans l'interface `min_max_in_itf`, nous allons générer les références relative au tableau ci-dessus et stocker les valeurs dans une variable `leds_ref`.

6 Vérification des résultats

Afin de vérifier les résultats, nous allons comparer les valeurs de `leds_ref` obtenues lors de la génération des références avec les valeurs de `leds` présentes dans l'interface de sortie `min_max_out_itf`.

En cas d'erreur, un signal `error_signal` sera activé jusqu'au prochain front d'horloge descendant afin de signaler le traitement necessaire d'une erreur.

De plus, cela affichera un message avec les éléments qui ont généré l'erreur.

Le système n'est pas arrêté en cas d'erreur et le testbench continue de s'exécuter.

7. Gestions des erreurs

Lors de l'exécution du testbench avec injection d'erreur, le système va les detecter et la tache de vérification va activer le signa`error_signal`. Dès lors, la tache `wait_for_error` traitera l'erreur au prochain front montant de l'horloge.

Dans le cadre de ce laboratoire, le traitement de l'erreur consiste uniquement à incrémenter le compteur d'erreur `error_count`.

Il serait idéal de créer un snapshot de l'interface `min_max_in_itf` et `min_max_out_itf` pour analyser les valeurs en entrée et en sortie du composant `min_max_top` lors de l'erreur. Cela en complément du chronogramme généré par le testbench.

8. gestion de la couverture

La couverture est gérée par la tache `wait_for_coverage` qui va analyser la couverture relative au scénario/s joué/s et se terminé une fois le 100% atteint.

C'est cette tache qui mettra fin au testbench.

9. Test dirigé

Afin de lancer une série de tests, nous avons rajouté au fichier `default.rmdb` les scénarios de test à exécuter. Ils exécutent tous le `TESTCASE` 0, avec un `VALSIZE` de 10 et les `ERRNO` suivants: [0, 1, 2, 3 ,16, 17, 18, 19, 20, 21].

9. Conclusion

Dans ce laboratoire, nous avons appliqué les concepts abordés en cours de *Vérification des systèmes embarqués (VSE)* en testant un `binlin` possédant différents modes, le module `min_max_top`. Nous avons mis en place un banc de test définissant des scénarios prenant en compte l'aléatoire aisni qu'un scénario de valeurs limites, permettant de couvrir un maximum de configurations et de détecter d'éventuels dysfonctionnements.

De plus, l'utilisation de classes modulaires et la séparation des tâches d'exécution et de vérification ont facilité l'automatisation et la maintenance du testbench. La gestion des erreurs et de la couverture a permis une vérification exhaustive du composant, avec une couverture de 100 % assurée avant la fin des simulations.

Cependant, après analyse des de l'architecture ainsi que des durées total d'exécution des tests, il serait judicieux de revoir l'architecture du banc de test en ayant des tests plus ciblés et moins redondants.

Nous avons beaucoup appris sur le langage `SystemVerilog` et ses limites. Nous sommes dès lors capables d'écrire des bancs de test simple permettant la vérification de modules tout en garantissant une durée d'exécution des tests raisonnable.

10. Références

- **Cours** de Vérification des systèmes embarqués, HEIG-VD, 2024
    - VSE_01_intro_verification_systeme_embarque.pdf
    - VSE_09_sv_randomization.pdf
    - VSE_10_sv_coverage.pdf
- **Chat GPT** pour l'aide à la compréhension du langage SystemVerilog