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

Objets protégés, logging et comparaison

Etape 1 : logger

La première partie de ce laboratoire vise à réaliser un type protégé qui sera utilisé lors des laboratoires suivants pour le logging et le reporting. Pour ce faire nous allons partir d'un logger basique dont vous avez les sources (fichier `src_tb/common_lib/logger_pkg.vhd`). Afin de disposer d'un élément intéressant pour les prochains laboratoires, nous aimerions y apporter les fonctionnalités suivantes :

1. Possibilité d'écrire les logs dans un fichier dont le nom sera défini grâce à une fonction d'initialisation
   — Il doit être possible de ne pas les écrire dans un fichier, mais seulement dans la console
2. Une procédure par gravité doit être offerte
   (a) Note
   (b) Warning
   (c) Error
   (d) Failure
3. Pour chaque procédure, proposez une version prenant en paramètre un message devant être affiché
4. Deux compteurs doivent permettre de savoir combien d'erreurs et de warnings ont été générés
5. Le niveau de verbosité doit pouvoir être défini grâce à une méthode. Pour ce faire, le développeur doit pouvoir définir ce niveau comme étant Note, Warning, Error ou Failure. Une fois défini, seuls les messages de niveau défini ou supérieur seront affichés/loggés. Par défaut le niveau devra être Note.
6. Une fonction de reporting doit pouvoir être appelée en fin de simulation et doit résumer ce qui s'est déroulé pendant la simulation (nombre d'erreurs et de warnings)

Attention à ne pas dupliquer du code. Si tel est le cas, alors il doit y avoir moyen de le modulariser. Pour lancer la simulation, allez dans le répertoire `comp` présent dans les sources fournies, et lancez la commande suivante :

```
vsim
```

Ceci lance QuestaSim. Ensuite, dans la console, tapez :

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

Etape 2 : comparateur générique

Nous allons maintenant réaliser un comparateur générique. Il va s'agir d'une fonction offerte par un paquetage générique, qui devra comparer deux éléments et appeler la méthode d'erreur du logger si la comparaison n'est pas correcte. Nous aimerions que notre procédure soit générique par rapport au type des éléments à comparer. Pour ce faire, nous allons utiliser la généricité des paquetages, pour d'une part spécifier le type des éléments, et d'autre part spécifier une fonction de traduction en chaine de caractères. Cette dernière sera utile pour envoyer un message pertinent au

logger. Le fichier `src_tb/common_lib/comparator_pkg.vhd` contient la déclaration du comparateur. A vous de créer le corps de la procédure **`compare`**. Le code est à ajouter là où vous trouvez le commentaire `TODO`. Vous pouvez observer deux spécialisations de ce paquetage, une pour des vecteurs, et une pour le type `transaction_t`, dans les fichiers `vector_comparator_pkg.vhd` et `transaction_comparator_pkg.vhd`.
Finalement, il peut être intéressant de pouvoir spécifier la fonction de comparaison en elle-même. Dans le cas d'une transaction, par exemple, certains champs ne sont pas forcément pertinents à comparer, et dès lors il faut pouvoir utiliser une fonction spécifique.
Le fichier `src_tb/common_lib/complex_comparator_pkg.vhd` contient la déclaration d'une procédure **`compare`** un peu plus générique. A vous de créer son corps. Observez ensuite la spécialisation du paquetage, depuis le fichier `transaction_lazy_comparator_pkg.vhd`.

Etape 3 : quelques éléments intéressants

Votre code sera terminé une fois que vous aurez effectué les deux étapes précédentes et qu'il n'y aura plus un seul commentaire `TODO` présent. N'hésitez pas à modifier le banc de test et les deux testers pour valider votre code.
Le code que vous avez à disposition illustre quelques éléments intéressants du langage VHDL-2008 :

1. Il y a plusieurs paquetages génériques présents. Observez la manière dont ils sont déclarés, et la manière dont ils sont spécialisés ensuite.
2. Deux contextes sont déclarés : le contexte `common_ctx` qui exporte les paquetages de la librarie commune, et le contexte `project_ctx` qui exporte les divers éléments pertinent pour le projet particulier. Observez ces constructions et la manière dont elles sont utilisées au début de chaque fichier. Le testbench en lui-même est composé de 3 fichiers (deux testeurs et un top), qui utilisent tous le même contexte.
3. Le logger est un type protégé pouvant être partagé par plusieurs process et plusieurs composants.
4. Le paquetage `project_logger_pkg` ne contient que la déclaration d'un logger. Ceci permet de le partager au travers de plusieurs composants. Ceci est illustré par le compteur d'erreurs dont vous pouvez observer qu'il est bien partagé.