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

Prof. Yann Thoma

Laboratoire VSE

semestre d'automne 2025 - 2026

Analyseur de paquets

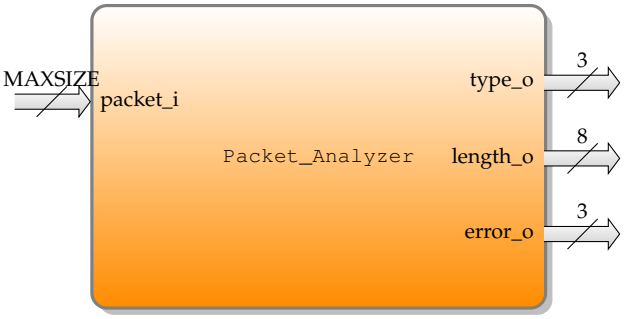
Composant à tester

Nous souhaitons tester un système combinatoire responsable d'analyser des paquets de type Ethernet (ou ressemblant).

Entrées/sorties

Les entrées/sorties sont :

Nom	Taille	Dir	Description
`packet_i`	`MAXSIZE`	in	Paquet à analyser
`type_o`	3	out	Type du paquet
`length_o`	8	out	Taille du paquet
`error_o`	3	out	Vecteur d'erreurs



Fonctionnement

L'analyseur de paquets est un composant purement combinatoire, et a pour but de fournir différentes informations sur les données présentes.
Chaque paquet est décomposé en trois parties principales :

1. Le header
2. Les données
3. Le CRC

Le header a une taille de 6×8 bits. Il des information sur le type de paquet, sa longueur, une adresse source et une adresse destination.
Les trois premiers bits (0 à 2) du header définissent le type. Celui-ci peut valoir 1, 2 ou 5, les autres valeurs n'étant pas valides. En cas de type non valide, le bit 1 de `error_o` doit être à 1.
Les données correspondent aux données bruts transmises.
Pour un paquet de type 1, la longueur des données est définie par les bits 8 à 15 du header, en byte. Pour un paquet de type 2, la longueur des données est définie par les bits 10 à 15 du header, en byte. Pour un paquet de type 1, la longueur des données est définie par les bits 12 à 15 du header, en byte.

Le nombre de bits de la longueur définit donc la longueur maximale des données du paquets (toujours en byte).
Les bits 16 à 31 définissent une source, et les bits 32 à 47 définissent une destination. Il existe certaines règles pour ces deux adresses, liées à deux *range* d'addresses, appelés *Group0* et *Group1* :

1. Si l'adresse source est hors de ces deux ranges, le bit 2 de `error_o` doit être à 1.
2. Si l'adresse destination est hors de ces deux ranges, le bit 3 de `error_o` doit être à 1.
3. Si les deux adresses sont valides mais pas dans le même groupe, alors le bit 4 de `error_o` doit être à 1.

Le CRC est calculé sur le header ET les données. Pour le type de paquet 1, le CRC est sur 8 bits, et pour les paquets de type 2 ou 5, le CRC est sur 16 bits. Si le composant constate que le CRC n'est pas valide, le bit 0 de `error_o` doit être mis à 1.
La version du DUV qui vous est fournie est parfaitement fonctionnelle, toutefois un paramètre générique `ERRNO` permettra d'injecter articifiellement des erreurs dans le design. Il s'agira d'un entier qui offre le comportement suivant :

1. S'il est compris entre 0 et 1, le résultat est valide ;
2. S'il est compris entre 10 et 23, le résultat n'est pas valide.

Ce paramètre générique vous permettra de valider votre banc de test, en le simulant avec toutes les valeurs de `ERRNO`.

1 Vérification

Un squelette de banc de test vous est fourni. Il ne fait évidemment pas grand chose, mais offre déjà une décomposition des responsabilités. Le script `sim.do` vous permet de lancer une simulation. Via celui-ci il vous sera possible de fournir deux paramètres génériques au testbench : un numéro de testcase ainsi que la valeur de `ERRNO`.
Vous pouvez donc définir plusieurs scénarios à jouer. Gardez le numéro 0 pour lancer tous les scénarios à la suite. Ceci permettra de disposer de `vrun` pour lancer toutes les validations en une seule fois. Développez donc votre banc de test en faisant des simulations, puis au final lancez un `vrun directed` pour voir si les erreurs sont bien détectées.
Dans le banc de test vous pouvez constater l'usage d'un logger. Allez voir dans le code du logger pour découvrir quelques macros vous permettant de passer des paramètres aux fonctions de log.
Il vous est demandé d'exploiter de l'aléatoire et de la couverture pour mettre en place votre vérification.

A rendre

Votre code, ainsi qu'un petit rapport expliquant vos choix sont à rendre sur cyberlearn. Un script `vse_rendu.sh` permet de générer une archive de manière propre. Il vérifie que le banc de test est bien présent, de même qu'un fichier `rapport.pdf`.

Barème de correction

Documentation	20%
Pertinence des testcases	10%
Exploitation d'aléatoire	20%
Exploitation de la couverture	10%
Fonction de vérification	10%
Détection / non détection correcte	10%
Codage	10%
Commentaires au niveau du code	10%