

Laboratoire 5 : Récepteur Morse

Département : Technologies de l'Information et de la Communication (TIC)
Unité d'enseignement : Vérification des systèmes numériques (VSN)

Auteurs : Pierrick Muller
Professeur : Yann Thoma
Date : 14 juin 2021

1 Introduction

1.1 Composant à tester

Nous souhaitons tester un système permettant l'envoi de code Morse pour les caractères alphanu-mériques standards. Le composant accepte des codes ASCII en entrée et est capable de les envoyer sur un lien série qui peut ensuite agir sur la génération d'un signal auditif ou lumineux. Le composant possède un FIFO interne permettant de stocker plusieurs caractères et de les envoyer ensuite en rafale.

La description du système est précisée dans le document du laboratoire.

2 Analyse et conception

Dans cette partie, j'ai défini les scénarios qui étaient intéressants afin de tester ce système correctement. J'ai laissé de côté les tests concernant le Reset du système, car le fonctionnement de ce dernier n'était pas explicitement donné dans le document de description du laboratoire, et que je ne voulais pas faire de supposition sur son fonctionnement.

2.1 Scénarios

J'ai pu définir plusieurs "catégories" de tests afin de m'assurer du bon fonctionnement du système.

- Les tests de transactions constant : Ces tests ne profitent ni du coverage ni de la randomisation, et servent à s'assurer que l'envoi de chaque caractère (excepté espace et CR) fonctionne correctement.
- Les tests de transactions aléatoires : Ces tests permettent de stresser le système en testant aléatoirement certaines valeurs avec certaines caractéristiques du système (dot_period aléatoire, caractère aléatoire). Ces tests profitent de l'aléatoire et du coverage afin d'observer un nombre de cas suffisant.
- Les tests d'erreurs : Ces tests permettent de s'assurer qu'un comportement erroné en entrée du système engendre bien le comportement recherché en sortie, c'est à dire la détection de l'erreur.

L'ensemble des scénarios envisagés pour ce testbench sont disponibles et décrit dans le document xml permettant de linker les scénarios avec différents testcases dans le code.

3 Réalisation et implémentation

L'entièreté du code est disponible dans le rendu du laboratoire, commenté. Je vais ici détailler les points importants qui ont guidé cette implémentation.

3.1 Modification du driver

Le "flow" suivi par le driver lors de la traduction d'une transaction en signaux d'entrée peut se décomposer de la manière suivante :

- Le driver teste si la valeur du morse de la transaction correspond à un caractère ou à un espace ou CR.
- Si il s'agit d'un caractère, le driver commence par envoyer sur le port d'entrée morse la valeur 0 pendant 3 dot_period si ce n'est pas la première transaction du cycle de vie du système et que la précédente transaction n'était pas un espace ou un CR afin de représenter l'espace entre deux caractères.
- Par la suite, toujours si il s'agit d'un caractère, une boucle parcourt tous les bits de la valeur en morse et envoie les valeurs nécessaires (dot ou dash avec comme durée dot_period) sur l'entrée morse_i jusqu'à tomber sur un X signifiant la fin de la valeur. Une attente de 1 dot_period est effectuée entre chaque dot/dash, sauf si le nombre de bits est de 5, auxquels cas la dernière attente n'est pas effectuée afin de ne pas poser de problèmes dans le calcul des cycles de morse_i à 0 par la suite
- Dans le cas où la transaction représente un espace ou un CR, le driver passe l'entrée morse_i à 0 et attend 7 coups de dot_period. si il s'agit d'un CR, la sortie morse_i est laissé pendant trois fois la valeur de log_marge_relative dot_period afin de s'assurer que la marge soit respectée et qu'aucun CR ne soit considéré comme un espace.
- Une fois l'un ou l'autre des scénarios vus plus haut effectués, la transaction est stockée comme ancienne transaction pour que la prochaine transaction puisse effectuer les tests nécessaires lors de l'attente entre char/dot.

Cette architecture permet de traiter chaque transaction avec un temps d'attente correcte afin d'éviter de devoir effectuer des calculs supplémentaires comme j'avais pu le faire lors du laboratoire 3 (Attente de 4 dot_period lors d'un espace au lieu de 7 car j'avais déjà attendu auparavant les 3 dot_period nécessaires entre caractères).

3.2 Modification du Scoreboard

Le scoreboard s'occupe de contrôler les résultats obtenus en sortie du système par rapport aux transactions envoyées. Le flow de contrôle est le suivant :

- Le scoreboard récupère une transaction envoyée par le sequencer. Si la transaction reçue n'est pas valide, il ne fait rien.
- Il récupère par la suite la transaction obtenue en sortie par le moniteur et commence par contrôler qu'il n'y a pas d'erreur de dot_period. Si une erreur est présente, il affiche un message d'erreur.
- La valeur ascii de la transaction en entrée du système est convertie en uppercase si il s'agit d'un caractère lowercase afin de comparer le résultat avec la valeur de sortie qui est toujours en uppercase.
- Le scoreboard affiche une erreur si les résultats ne correspondent pas ou si la transaction en entrée était une série de dot/dash inconnu qui n'a pas été reconnue comme inconnue en sortie du système

3.3 Modification du Sequencer (Testcases)

Tous les testcases implémentés pour ce laboratoire l'ont été dans le sequencer. Tous les testcases sont commentés et sont pourvu d'un entête expliquant leur fonctionnement. Je passerais rapidement ici sur ces testcases.

3.3.1 Tests de transactions constant : Caractères et nombres, sans espace

Les deux premiers testcases permettent de tester l'envoi de tous les caractères et de tous les nombres sans espaces. la valeur de `dot_period` pour chacune des transactions est fixé à 5 et une boucle avec à chaque itération une contrainte fixant la valeur d'ascii permet ces envois. Ces deux testcases ne font rien de plus.

3.3.2 Tests de transactions aléatoires : Caractères et `dot_period`

Les tests de transactions aléatoires profitent de la randomisation et du coverage afin de pouvoir tester un grand nombre de série de caractères et d'espace différents. De plus, la randomisation du `dot_period` permet aussi de s'assurer du bon fonctionnement du mécanisme de `dot_period`.

Deux tests se concentrant chacun sur une facette spécifique (Caractères et `dot_period`) ont été implémenté. Je passerais ici rapidement sur le fonctionnement et les règles mises en place permettant l'implémentation correcte de ces tests.

Concernant la première facette, une classe héritant de la classe de base a été implémentée afin de pouvoir définir un groupe de coverage et des contraintes permettant de s'assurer de la validité du caractères ASCII généré. Le groupe de coverage permet d'exclure de l'analyse de coverage tous les caractères non valides et de tester le fait que chacun des caractères valides est observé un certain nombre de fois, 250 dans l'implémentation actuelle. Une dernière contrainte permet de randomiser la valeur de `dot_period` entre 1 et 100. Finalement, le test en lui même est une boucle ne s'arrêtant que quand le coverage est à 100. Un contrôle dans le testcase permet de s'assurer que deux espaces/CR ne peuvent pas se suivre dans le test afin d'éviter que deux espaces se suivent et soient considéré comme un CR.

Le deuxième test permet de tester des valeurs de `dot_period` plus grande. une classe héritant de la classe de base a aussi été implémentée et reprend la même contrainte concernant les caractères ASCII que le test se concentrant sur les caractères. Deux contraintes permettent de définir la valeur de `dot_period`. L'une d'elles définit une variable multiple entre 0 et 7, et la seconde utilise cette contrainte afin de pouvoir choisir une valeur parmi celles correspondant à la partie des valeurs possibles représentées par le multiple. Dans notre cas, afin de ne pas perdre trop de temps avec des valeurs de `dot_period` trop grande, c'est une plage de 2 puissance 14 valeurs séparées en 8 bloc de valeur qui a été définie. Le groupe de coverage s'assure qu'au moins une valeur des 8 boites a été observée durant la durée du test.

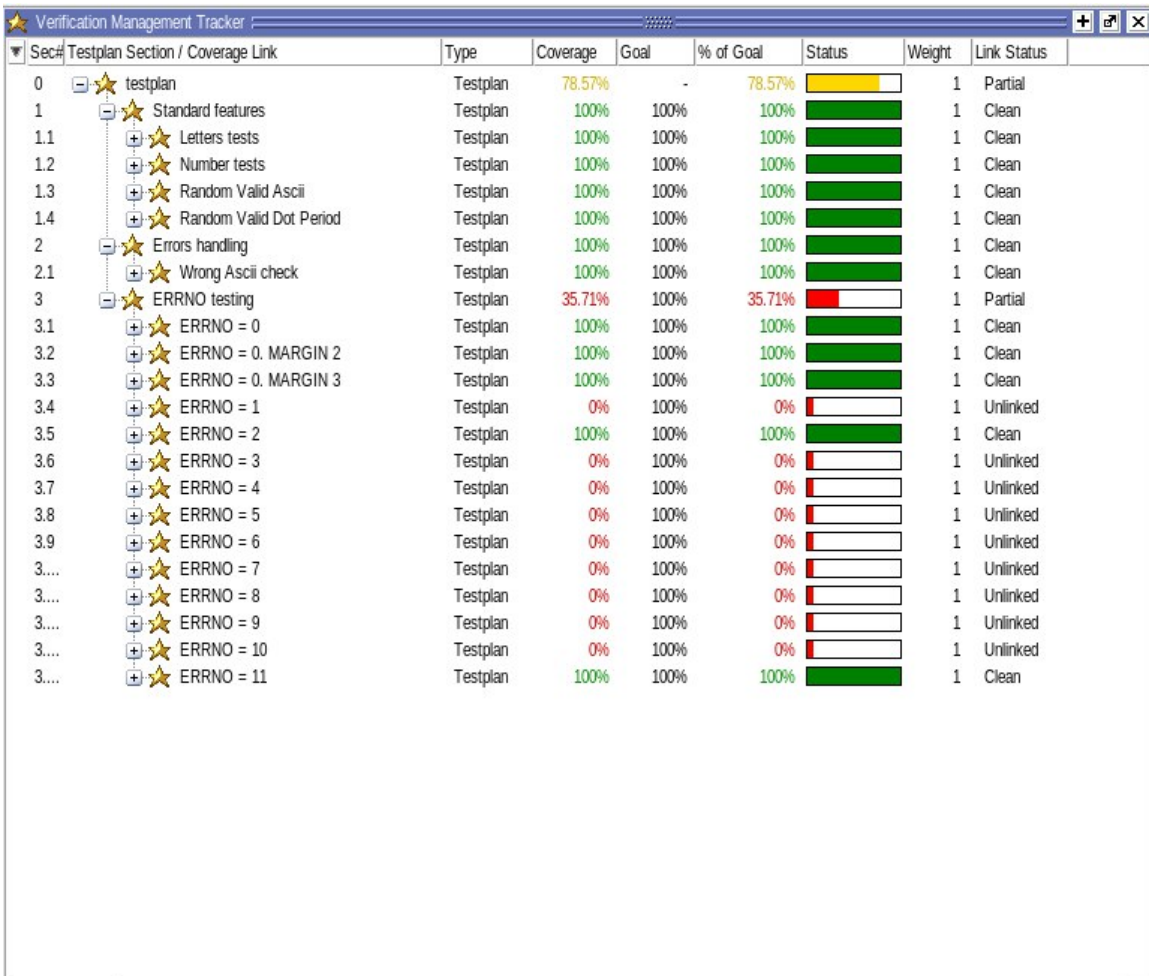
3.3.3 Tests d'erreurs : Suite de dot/dash non valide

Un test a été implémenté afin de s'assurer de la bonne détection des erreurs. 22 valeur Morse n'étant pas valide ont été stocké, puis sont envoyées les unes à la suite

des autres. Pour tester le bon fonctionnement de cette détection d'erreur, un champ permettant de préciser qu'il s'agit d'une transaction a été ajoutée. Ce champ est testé dans le scoreboard en s'assurant que la sortie "Unknown" était à 1 lors de la création de la transaction de sortie. Ce test permet donc de tester que si une suite de dots et de dash non valide a été envoyée, elle est bien repérée en sortie du système.

4 Simulation

Après avoir défini les scénarios et les avoir linké avec les testcases dans le fichier rmdb, j'ai pu obtenir une visualisation des résultats :



Sec#	Testplan Section / Coverage Link	Type	Coverage	Goal	% of Goal	Status	Weight	Link Status
0	testplan	Testplan	78.57%	-	78.57%	<div><div></div></div>	1	Partial
1	Standard features	Testplan	100%	100%	100%	<div><div></div></div>	1	Clean
1.1	Letters tests	Testplan	100%	100%	100%	<div><div></div></div>	1	Clean
1.2	Number tests	Testplan	100%	100%	100%	<div><div></div></div>	1	Clean
1.3	Random Valid Ascii	Testplan	100%	100%	100%	<div><div></div></div>	1	Clean
1.4	Random Valid Dot Period	Testplan	100%	100%	100%	<div><div></div></div>	1	Clean
2	Errors handling	Testplan	100%	100%	100%	<div><div></div></div>	1	Clean
2.1	Wrong Ascii check	Testplan	100%	100%	100%	<div><div></div></div>	1	Clean
3	ERRNO testing	Testplan	35.71%	100%	35.71%	<div><div></div></div>	1	Partial
3.1	ERRNO = 0	Testplan	100%	100%	100%	<div><div></div></div>	1	Clean
3.2	ERRNO = 0. MARGIN 2	Testplan	100%	100%	100%	<div><div></div></div>	1	Clean
3.3	ERRNO = 0. MARGIN 3	Testplan	100%	100%	100%	<div><div></div></div>	1	Clean
3.4	ERRNO = 1	Testplan	0%	100%	0%	<div><div></div></div>	1	Unlinked
3.5	ERRNO = 2	Testplan	100%	100%	100%	<div><div></div></div>	1	Clean
3.6	ERRNO = 3	Testplan	0%	100%	0%	<div><div></div></div>	1	Unlinked
3.7	ERRNO = 4	Testplan	0%	100%	0%	<div><div></div></div>	1	Unlinked
3.8	ERRNO = 5	Testplan	0%	100%	0%	<div><div></div></div>	1	Unlinked
3.9	ERRNO = 6	Testplan	0%	100%	0%	<div><div></div></div>	1	Unlinked
3....	ERRNO = 7	Testplan	0%	100%	0%	<div><div></div></div>	1	Unlinked
3....	ERRNO = 8	Testplan	0%	100%	0%	<div><div></div></div>	1	Unlinked
3....	ERRNO = 9	Testplan	0%	100%	0%	<div><div></div></div>	1	Unlinked
3....	ERRNO = 10	Testplan	0%	100%	0%	<div><div></div></div>	1	Unlinked
3....	ERRNO = 11	Testplan	100%	100%	100%	<div><div></div></div>	1	Clean

Ce qui ressort de cette simulation, c'est que tous les testcases unitaires semblent fonctionner correctement, et certaines erreurs ont pu être repérées grâce aux tests effectués. Cependant, on remarque quand même que 2 tests avec un code ERRNO ne devant pas passer passent toujours. Il s'agit des tests ERRNO 2 et 11

Un test intéressant à implémenter avec plus de temps aurait été l'implémentation de testcases supplémentaires mettant au défi la valeur de marge qui a été mise en place dans ce laboratoire.

5 Conclusion

Ce laboratoire m'a permis de mettre en pratique les principes de randomisation et de coverage vu en cours. J'ai pris beaucoup de plaisir à travailler avec System Verilog qui me donne l'impression de pouvoir établir mes tests de manière plus propre et facile qu'avec VHDL.

Concernant les résultats finaux, j'aurais aimé réussir à trouver tous les problèmes pour les différents ERRNO. Le fait que 2 ERRNO sur 11 faux passe encore me laisse penser que j'ai peut-être mal compris un aspect de la vérification, en me concentrant trop sur les transactions et pas assez sur les mécanismes interne au système. J'avais déjà effectué cette erreur lors du laboratoire précédent.

En terme de ressenti personnel, je suis un peu déçu de ne pas avoir réussi à nouveau à trouver la cause de toutes les erreurs. J'aurais aimé avoir plus de temps pour pouvoir continuer à chercher un peu.

Je reste tout de même satisfait du laboratoire je pense avoir investi le temps nécessaire pour obtenir un résultat correct.

Yverdon-les-Bains le 14 juin 2021

Pierrick Muller