Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts Western Switzerland

MSE | MASTER OF SCIENCE IN ENGINEERING

Laboratoire 3 :

Emetteur 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 : 28 avril 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 coté 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 sans remplissage de la fifo : Ces tests permettent d'effectuer toutes les transactions possibles de manière unitaire, en envoyant le caractère dès qu'il est chargé dans la fifo.
- Les tests de transactions constant avec remplissage complet de la fifo avant envoi (Burst fifo full) : Ces tests sont les mêmes que pour les tests sans remplissage de la fifo, mais l'envoi est effectué uniquement une fois que tous les caractères testés ont été chargés dans la fifo. Ces tests permettent de s'assurer que l'envoi "burst" fonctionne.
- Les tests de transactions constant avec remplissage de la fifo durant envoi (Burst fifo not full) : Ces tests sont les mêmes que pour les deux catégories précédentes, a l'exception que la commande d'envoi est envoyée dès le chargement du premier caractère et que les prochains caractères sont chargés durant l'envoi afin de tester la fonctionnalité permettant de remplir la fifo en cours d'envoi.
- Les tests aléatoires : Ces tests permettent de tester à l'aide de valeur aléatoire différentes suites de valeurs avec différents paramètres. Les transactions testées sont différentes lors des différents lancements de ces tests.
- Les tests spécifiques : Ces tests peuvent ou pas nécessiter l'utilisation de l'aléatoire. L'idée est de tester un fonctionnement spécifique du système ne touchant pas au traitement général des transactions, par exemple le fait que le changement de la valeur du dot_period en entrée du système durant un

envoi ne modifie pas le dot_period.

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 Reconstruction de transaction (output monitor)

Cette partie m'a pris le plus de temps dans ce projet. Cela vient du fait que la récupération des espaces entre mots en tant que caractères m'a posé des problèmes. le "flow" utilisé par le moniteur de sortie peut se décomposer de la manière suivante :

- Le système a été reset, on attend une transaction et on considère que la transaction ne sera pas un espace au début, ce qui ferait peu de sens.
- Une fois la ligne passé à un, on saute l'étape de reconnaissance des espaces qui est au début de la boucle afin de se concentrer directement sur la reconnaissance d'un caractère. La reconnaissance du caractère se fait à l'aide d'une deuxième boucle interne qui gérera les dot et dash ainsi que les espace entre eux.
- en sortie de cette deuxième boucle, on effectue un check pour s'assurer de la validité du caractère et on génère une erreur si ce n'est pas le cas. Dans le cas contraire, on ajoute la transaction représentant le caractère à la fifo pour la scorboard.
- Une deuxième boucle interne permet de définir si l'on reçoit ensuite un espace entre deux lettres ou entre deux mots. Si c'est entre deux mots, quand la boucle principale recommence, on génère une transaction correspondant à l'espace.

Un point de cette architecture qui pourrait poser problème vient de la gestion d'un espace plus grand que 7 dot. En effet, bien que fonctionnel, la boucle permettant d'attendre la fin d'un tel espace se situe en dehors de l'objection. Je n'ai cependant pas rencontré de problème vis-à-vis de cette observation.

3.2 Comparaison entre output et input char (Scoreboard)

Le scoreboard effectue principalement la comparaison entre les transactions obtenue depuis l'input agent et celles obtenues depuis l'output agent. Les messages d'erreurs générés permettent de récupérer les informations relatives aux deux transactions lorsqu'une erreur survient.

J'ai de plus ajouté une gestion de la récupération d'une transaction de l'output manager dans le cas ou la transaction en provenance de l'input manager n'avait pas son champ "Valid" à true. L'idée était de pouvoir s'assurer que lors de l'envoi d'une transaction non valide, le système ne transmettait rien. Cependant cette implémentation n'a pas pu être menée à bout et ce qu'il en reste ne récupère simplement pas la transaction obtenue depuis l'output agent car l'appel bloquant permettant de récupérer cette transaction bloquait le système, l'output agent monitor n'envoyant pas de transaction étant donnée que la transaction n'était pas envoyée par le système car non valide. C'est un point d'amélioration du projet.

3.3 Testcases (input sequencer)

Tous les testcases implémentés pour ce laboratoire l'ont été dans l'input sequencer. Tous les testcases sont commentés et sont pourvu d'un entête expliquant leur fonctionnement. Je passerais rapidement ici sur ces testcases, parfois en les groupant pour ceux étant sensiblement les mêmes. J'ajouterais ici qu'une procédure permettant de faciliter l'envoi d'une transaction à été implémentée dans le code.

3.4 Hello world, Toutes les lettres et tous les nombres

Ces trois types de testcases sont les trois testcases que l'on retrouve dans les catégories concernant les transactions constantes, je les traiterais donc ici ensemble (9 testcase en tout). Entre ces catégories, ces testcases divergent uniquement par le moment où la commande send est envoyé comme expliqué dans la partie analyse.
Ces trois testcase effectuent chacun l'envoi d'une série de transactions, avec comme valeur de dot de 5 et un waiting time de 0 pour toutes les transactions. "Hello World" permet l'envoi des deux mots "Hello" et "World" avec un espace entre ces deux mots. Un autre de ces testcase permet d'envoyer toutes les lettres de l'alphabet minuscule et majuscule sans espace entre elle, et le dernier testcase permet de faire la même chose avec les nombres.

3.5 Tests "Aléatoire"

Il y a trois tests "Aléatoires" implémentés.

Le premier permet d'envoyer 100 caractères aléatoires à la suite sans espace.
Le second permet d'envoyer 100 mots composés de caractères aléatoires, avec le premier mot ayant une taille de 1, le second de 2, le troisième de trois, etc... jusqu'à un mot de 100 caractères.
Enfin le dernier test permet d'envoyer 1000 mots composés aléatoirement de 1 à 5 caractères aléatoires, avec pour chaque transaction représentant un caractère un dot_period différent et pour chaque espace entre les mots un waiting_time différent.
Si les deux testcases permettent principalement de tester plusieurs cas aléatoires sans forcément toucher au fonctionnement même du système en se concentrant surtout sur le passage de différents caractères avec des espaces entre eux, le troisième

testcase permet aussi de s'assurer de problématique plus spécifiques, tel que la non modification du dot_period par le système ou la bonne gestion des espaces plus longs. J'ai hésité à le placer dans la catégories des tests "spécifiques", mais la forte présence de l'aléatoire me laisse quand même penser que sa place est dans cette catégorie.

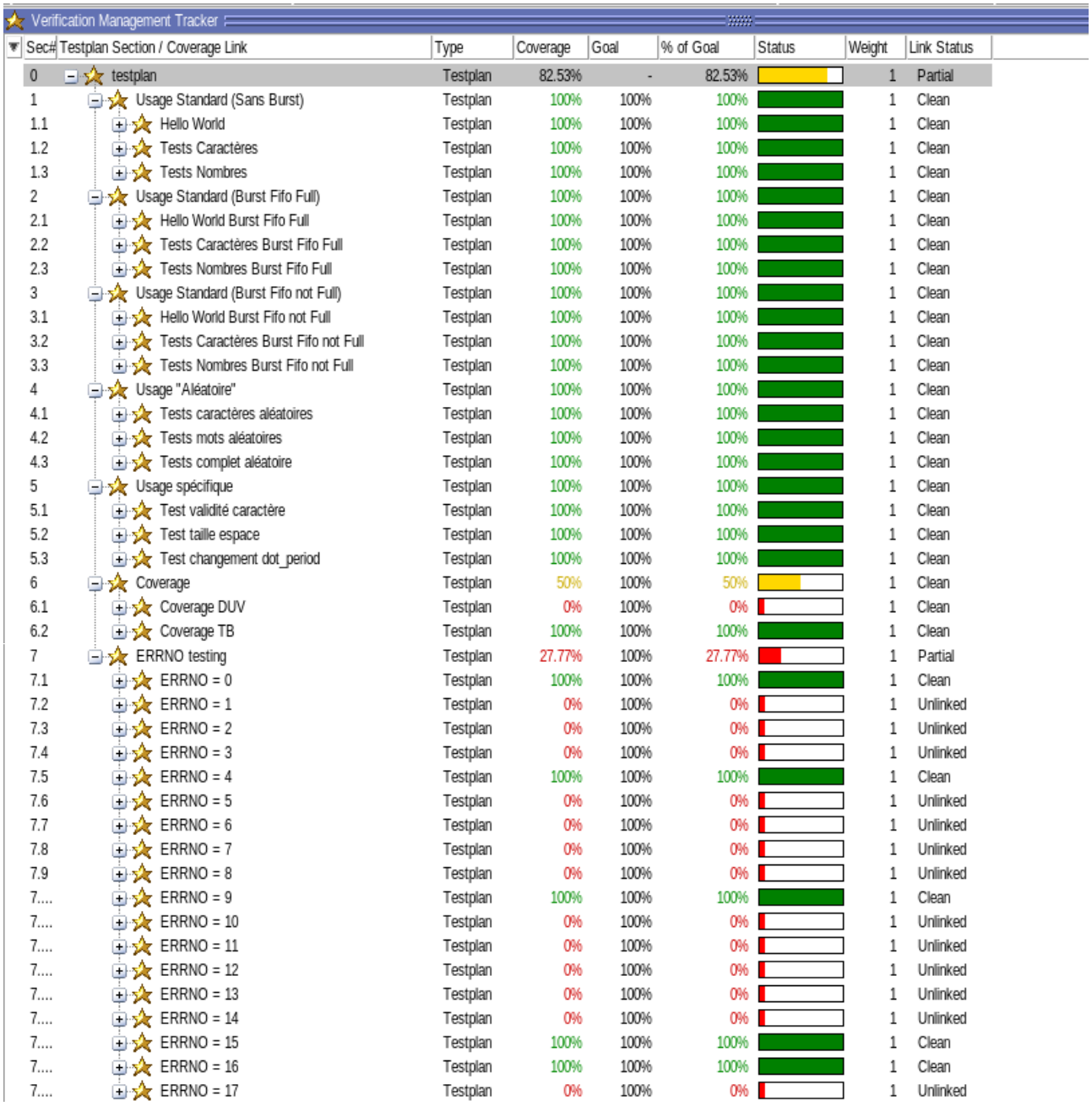
3.6 Tests spécifiques

Trois tests sont présents dans cette catégorie, mais l'un d'eux ne fait pas vraiment ce que j'aurais souhaité qu'il fasse dans l'implémentation. C'est le premier testcase de cette catégorie qui permet de s'assurer que lorsque l'on envoie un caractère non valide, ce dernier n'est pas traité par le système et fournit en sortie. Comme dis dans la partie scoreboard, je n'ai pas pu l'implémenter correctement. Je l'ai laissé tout de même car il ne gêne pas le fonctionnement des autres testcases et car dans une optique future je pense que ce type de test peut être intéressant à implémenter. Le second testcase permet de s'assurer que les espaces entre mots plus longs que la valeur minimale sont quand même considérés comme des espaces. C'est un testcase qui permet d'assurer le bon fonctionnement de l'output manager plus que le bon fonctionnement du système en lui-même. Il met à profit le waiting_time fourni dans la définition des transactions d'entrée. Le dernier testcase permet de s'assurer que le système ne modifie pas le dot_period si il est modifié par une transaction en entrée durant une transmission. En effet, si le système modifie la valeur de dot_period durant la transmission, l'output manager a été écrit de sorte à toujours se baser sur le dot_period de base de la première transaction et fournira donc des résultats ne correspondant pas à ceux fournis en entrée, générant par la même une erreur. La première dot_period vaut 5, tout comme le tout premier testcase lancé si TEST-CASE à la valeur 0, afin de s'assurer que si tous les testcases sont lancé à la suite le fonctionnement attendu reste le même.

4 Simulation

Pour cette partie, j'ai défini un "runnable" supplémentaire dans le fichier default.rmdb appelé "Full52" qui permet de lancer tous les scénarios en précisant pour les scénarios concernant les ERNNO et lançant tous les testcases que la taille de la fifo devait être de 52 afin de pouvoir faire fonctionner correctement tous les testcases. J'ai utilisé ce runnable pour effectuer la simulation

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



Verification Management Tracker

Sec#	Testplan Section / Coverage Link	Type	Coverage	Goal	% of Goal	Status	Weight	Link Status
0	testplan	Testplan	82.53%	-	82.53%		1	Partial
1	Usage Standard (Sans Burst)	Testplan	100%	100%	100%		1	Clean
1.1	Hello World	Testplan	100%	100%	100%		1	Clean
1.2	Tests Caractères	Testplan	100%	100%	100%		1	Clean
1.3	Tests Nombres	Testplan	100%	100%	100%		1	Clean
2	Usage Standard (Burst Fifo Full)	Testplan	100%	100%	100%		1	Clean
2.1	Hello World Burst Fifo Full	Testplan	100%	100%	100%		1	Clean
2.2	Tests Caractères Burst Fifo Full	Testplan	100%	100%	100%		1	Clean
2.3	Tests Nombres Burst Fifo Full	Testplan	100%	100%	100%		1	Clean
3	Usage Standard (Burst Fifo not Full)	Testplan	100%	100%	100%		1	Clean
3.1	Hello World Burst Fifo not Full	Testplan	100%	100%	100%		1	Clean
3.2	Tests Caractères Burst Fifo not Full	Testplan	100%	100%	100%		1	Clean
3.3	Tests Nombres Burst Fifo not Full	Testplan	100%	100%	100%		1	Clean
4	Usage "Aléatoire"	Testplan	100%	100%	100%		1	Clean
4.1	Tests caractères aléatoires	Testplan	100%	100%	100%		1	Clean
4.2	Tests mots aléatoires	Testplan	100%	100%	100%		1	Clean
4.3	Tests complet aléatoire	Testplan	100%	100%	100%		1	Clean
5	Usage spécifique	Testplan	100%	100%	100%		1	Clean
5.1	Test validité caractère	Testplan	100%	100%	100%		1	Clean
5.2	Test taille espace	Testplan	100%	100%	100%		1	Clean
5.3	Test changement dot_period	Testplan	100%	100%	100%		1	Clean
6	Coverage	Testplan	50%	100%	50%		1	Clean
6.1	Coverage DUV	Testplan	0%	100%	0%		1	Clean
6.2	Coverage TB	Testplan	100%	100%	100%		1	Clean
7	ERRNO testing	Testplan	27.77%	100%	27.77%		1	Partial
7.1	ERRNO = 0	Testplan	100%	100%	100%		1	Clean
7.2	ERRNO = 1	Testplan	0%	100%	0%		1	Unlinked
7.3	ERRNO = 2	Testplan	0%	100%	0%		1	Unlinked
7.4	ERRNO = 3	Testplan	0%	100%	0%		1	Unlinked
7.5	ERRNO = 4	Testplan	100%	100%	100%		1	Clean
7.6	ERRNO = 5	Testplan	0%	100%	0%		1	Unlinked
7.7	ERRNO = 6	Testplan	0%	100%	0%		1	Unlinked
7.8	ERRNO = 7	Testplan	0%	100%	0%		1	Unlinked
7.9	ERRNO = 8	Testplan	0%	100%	0%		1	Unlinked
7....	ERRNO = 9	Testplan	100%	100%	100%		1	Clean
7....	ERRNO = 10	Testplan	0%	100%	0%		1	Unlinked
7....	ERRNO = 11	Testplan	0%	100%	0%		1	Unlinked
7....	ERRNO = 12	Testplan	0%	100%	0%		1	Unlinked
7....	ERRNO = 13	Testplan	0%	100%	0%		1	Unlinked
7....	ERRNO = 14	Testplan	0%	100%	0%		1	Unlinked
7....	ERRNO = 15	Testplan	100%	100%	100%		1	Clean
7....	ERRNO = 16	Testplan	100%	100%	100%		1	Clean
7....	ERRNO = 17	Testplan	0%	100%	0%		1	Unlinked

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 4 tests avec un code ERRNO ne devant pas passer passent toujours. Il s'agit des tests ERRNO 4,9,15 et 16.

Un test intéressant à implémenter avec plus de temps aurait été l'assurance que les caractères ajoutés à la FIFO une fois cette dernière pleine sont bien supprimés. Des tests moins portés sur les transactions et plus sur les autres signaux (busy, unknow, full...) aurait aussi été intéressant je pense.

5 Conclusion

Ce laboratoire m'a permis de me renforcer mon utilisation du pseudo-aléatoire et de mettre en pratique les nouveaux concepts abordés en cours.

Concernant les résultats finaux, j'aurais aimé réussir à trouver tous les problèmes pour les différents ERRNO. Le fait que 4 ERRNO sur 17 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. De plus, comme pour le labo précédent, j'aurais surement dû prendre plus de temps pour gérer les logs et la comparaison dès le début en l'organisant mieux à l'aide de packages.

En terme de ressenti personnel, je pense avoir un peu moins bien organisé mon code que lors du laboratoire précédent. L'une des choses que je devrais repenser, c'est les trois testcases des trois premières catégories qui se ressemble énormément et changent peu. Je pense que ce point peut être amélioré.

Je reste tout de même satisfait du laboratoire qui était plus imposant que le laboratoire précédent et je pense avoir investi le temps nécessaire pour obtenir un résultat correct.

Yverdon-les-Bains le 28 avril 2021

Pierrick Muller