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 2 :

Commande de velux et store

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 :	31 mars 2021

1 Introduction

1.1 Composant à tester

Nous souhaitons régler la température de plusieurs pièces d'une maison par une commande électronique. Nous disposons de capteurs qui nous fournissent des informations sur la température, l'ensoleillement et s'il pleut ou vente. D'autre part nous disposons de deux actuateurs, soit l'entraînement électrique du velux et du store.

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 intéressant afin de tester ce système correctement. le système prenant deux entrées pour la température et l'ensoleillement pouvant être représenter sur N bits, les scénarios mis en place ne pouvait pas tester l'intégralité des cas. Je me suis demandé si il était pertinent de tester séparément les résultats obtenus pour le store et ceux obtenu pour le velux, mais après réflexion, le contrôle des résultats est toujours exécuté pour les deux résultats obtenus et ce même pour un test spécifique à l'une ou l'autre des sorties, simplement car les résultats sont disponibles et que la référence est calculé en fonction des stimulis peu importe le test effectué.

2.1 Scénarios

J'ai pu définir plusieurs "catégories" de test afin de m'assurer du bon fonctionnement du système. Les tests contenus dans ces catégories diffèrent principalement entre eux en fonction du nombre de bit choisi pour les entrées de température et d'ensoleillement :

- Les tests complets : Tests de toutes les entrées possible pour le système. Ces tests sont seulement possibles pour une valeur de N relativement faible, le nombre de cas à tester étant de $2^N * 2^N * 2^1 * 2^1$, ce qui peut engendrer un nombre de cas à tester trop grand et donc prendre trop de temps.
- Les tests "aléatoires" : Tests de différents cas "aléatoires", par tranche. L'idée ici est de tester un cas "aléatoire" par tranche de différents cas du système. Ainsi, on obtient le même nombre de cas tester que pour un système utilisant $N = 3$, mais ce pour n'importe quelle valeur de N, et testant donc un cas correspondant pour chaque cas possible du système.
- Les tests limites : Tests permettant principalement de tester le bon fonctionnement de la conversion d'une valeur de température ou d'ensoleillement sur N bits vers une valeurs sur 3 bits afin de pouvoir calculer le résultat. L'idée est de prendre pour chaque tranche de valeurs sur N bits correspondant à

une valeur sur 3 bits la valeur maximum et minimum de cette tranche afin de pouvoir tester tous les cas limites.

La granularité des tests n'est pas spécialement précise, et c'est voulu. La réflexion est la suivante : Le système qui nous intéresse est le système avec $N = 3$. Le fonctionnement du système pour toutes les autre valeur de N dépend du bon fonctionnement de la conversion. Ainsi, le test complet du système de base ainsi que le test de la conversion, couplé avec des tests aléatoires permettent d'obtenir une bonne couverture des problèmes. L'idée des cas limites vient du fait que si l'on test la conversion de la valeur max et min pour une tranche, les valeurs interne à la tranche devrait être converties correctement aussi. Les test aléatoires viennent rajouter la possibilité de tester de multiples valeurs interne à ces tranches afin d'assurer un contrôle encore plus accrue sur le fonctionnement du système.

L'ensemble des scénarios envisagés pour ce testbench sont disponible 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. Je vais ici détailler les points importants qui ont guidé cette implémentation.

3.1 Fonction de génération des références et de check des résultats

Comme dis dans la partie analyse, les résultats du velux et du store sont toujours comparé ensemble.

La fonction de check des résultats contrôle que les résultats obtenus par le TB et le DUV correspondent. De plus, cette fonction prend en compte le fait que le comportement du système lors de présence de pluie et d'un ensoleillement élevé (6 ou 7) n'est pas définis. En cas d'erreur, l'erreur est signalée et les stimulis, les résultats observés et les résultats de référence sont logués. En cas de réussite, les stimulis sont logué uniquement dans l'optique de pouvoir reproduire le test.

La fonction de génération des références effectue trois opérations :

— Elle convertit la valeur de température et d'ensoleillement sur N bits en une valeur sur 3 bits
— Elle calcule le résultat du velux et du store en fonction de la valeur convertit sur 3 bit

— Elle met à jour un signal permettant d'ignorer le résultat du check si la valeur du store doit être "indéfinie" en fonction des entrées.

3.2 Testcase complet

Le testcase complet génère tous les ensemble de stimulis possible en fonction de N, et teste donc tous les cas du systèmes. Ce testcase n'est pas prévu pour être utilisé avec une valeur de N engendrant un nombre de test trop élevé. La limite de N dépend des performance de la machine hôte effectuant les tests, dans mon cas le temps pour effectuer le testcase complet est long à partir de $N = 9$. Dans le cadre de ce laboratoire, 4 scénarios utilisent ce testcase, avec un N de 3 à 6.

3.3 Testcase "aléatoires"

Le testcase "aléatoire" fonctionne de sorte à ce que chaque cas de figure envisagé dans le système avec $N = 3$ soit couvert. Pour se faire, 4 boucles imbriquées sont utilisé dans le code. La première itère de 0 à $2^3 - 1$ et représente la température. La seconde itère de la même manière et représente l'ensoleillement. La troisième et la quatrième itère de 0 à 1 et permettent de représenter la présence ou l'absence de pluie/vent.

A l'intérieur de ces 4 boucles, le calcul de valeurs aléatoire pour la température et l'ensoleillement est effectué de la manière suivante :

— On génère un entier aléatoire contenu entre 0 et $2^{N-3} - 1$, ce sera la valeur ajouté à la valeur obtenu pour la tranche que l'on traite en ce moment
— On récupère la valeur minimale possible pour la conversion dans la tranche que l'on souhaite en fonction de N. Dans le testcase limite, cette valeur correspond à la partie minimale testée. Afin de calculer cette valeur, on additionne la valeur d'un pas de conversion à lui même x fois, pour x = i ou y en fonction du calcul de l'ensoleillement ou de la température. Par exemple, si $N = 5$, et que l'on veut la tranche de conversion pour la température correspondant à 1, on à $i = 1$ et on commence par calculer la valeur d'un pas de conversion : $step = 1 << (N - 3) = 1 << (5 - 3) = 4$. Un fois ce pas calculer, on obtient la valeur min avec le calcul $i * step$, ce qui nous donne 4 dans ce cas.
— On ajoute l'entier aléatoire à la valeur minimale afin d'obtenir une nouvelle valeur contenue dans la tranche que l'on est en train de tester.

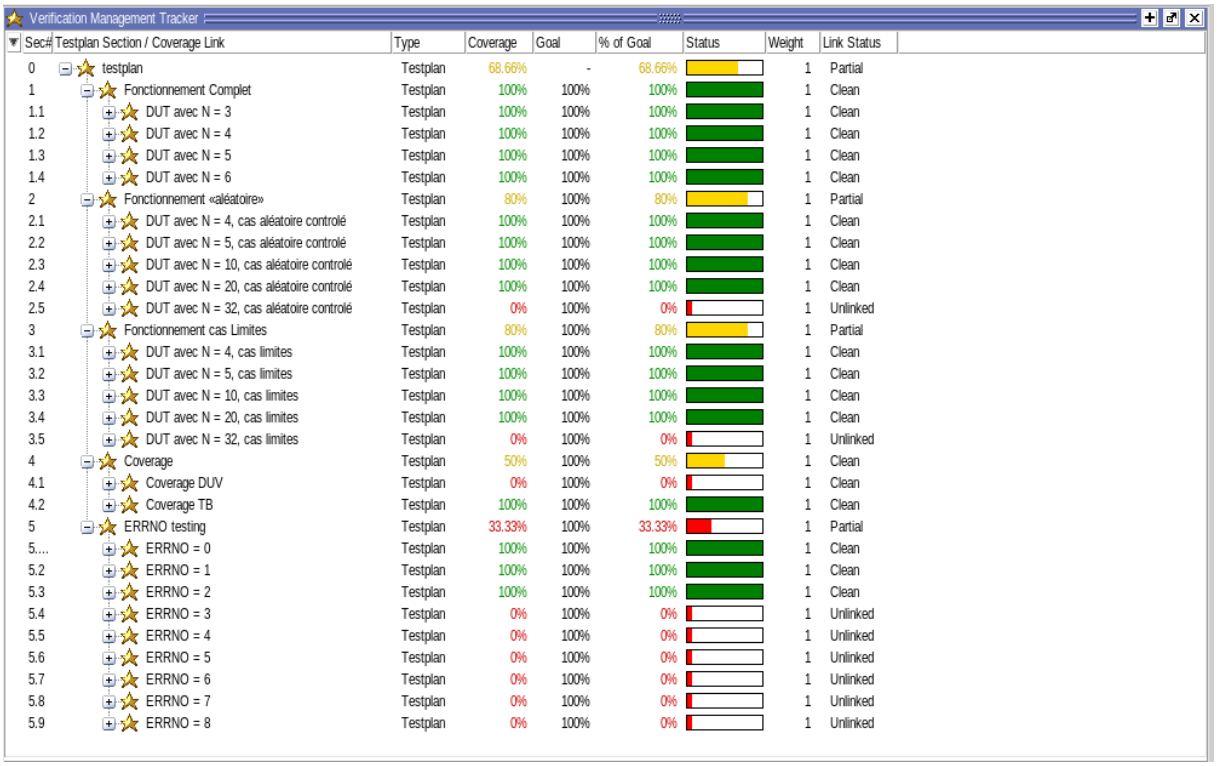
3.4 Testcase limite

Le testcase limite reprend le même principe que le testcase aléatoire mais en contrôlant les valeurs minimale et maximale pour une même tranche de conversion. Ainsi, deux boucle s'ajoutent permettant de s'assurer que 4 cas de figure supplémentaire seront tester pour chaque scénarios, soit "temp min, sun min", "temp min, sun max", "temp max, sun min" et "temp max, sun max". Le calcul de la valeur minimale de chaque tranche est le même que pour le testcase aléatoire, et pour obtenir la

valeur maximale on rajoute la valeur d'un pas de conversion à la valeur minimale et on soustrait 1. On peut ainsi facilement obtenir les valeurs max et min pour chaque tranche de conversion.

4 Simulation

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



Sec#	Testplan Section / Coverage Link	Type	Coverage	Goal	% of Goal	Status	Weight	Link Status
0	testplan	Testplan	68.66%	-	68.66%		1	Partial
1	Fonctionnement Complet	Testplan	100%	100%	100%		1	Clean
1.1	DUT avec N = 3	Testplan	100%	100%	100%		1	Clean
1.2	DUT avec N = 4	Testplan	100%	100%	100%		1	Clean
1.3	DUT avec N = 5	Testplan	100%	100%	100%		1	Clean
1.4	DUT avec N = 6	Testplan	100%	100%	100%		1	Clean
2	Fonctionnement «aléatoire»	Testplan	80%	100%	80%		1	Partial
2.1	DUT avec N = 4, cas aléatoire controlé	Testplan	100%	100%	100%		1	Clean
2.2	DUT avec N = 5, cas aléatoire controlé	Testplan	100%	100%	100%		1	Clean
2.3	DUT avec N = 10, cas aléatoire controlé	Testplan	100%	100%	100%		1	Clean
2.4	DUT avec N = 20, cas aléatoire controlé	Testplan	100%	100%	100%		1	Clean
2.5	DUT avec N = 32, cas aléatoire controlé	Testplan	0%	100%	0%		1	Unlinked
3	Fonctionnement cas Limites	Testplan	80%	100%	80%		1	Partial
3.1	DUT avec N = 4, cas limites	Testplan	100%	100%	100%		1	Clean
3.2	DUT avec N = 5, cas limites	Testplan	100%	100%	100%		1	Clean
3.3	DUT avec N = 10, cas limites	Testplan	100%	100%	100%		1	Clean
3.4	DUT avec N = 20, cas limites	Testplan	100%	100%	100%		1	Clean
3.5	DUT avec N = 32, cas limites	Testplan	0%	100%	0%		1	Unlinked
4	Coverage	Testplan	50%	100%	50%		1	Clean
4.1	Coverage DUV	Testplan	0%	100%	0%		1	Clean
4.2	Coverage TB	Testplan	100%	100%	100%		1	Clean
5	ERRNO testing	Testplan	33.33%	100%	33.33%		1	Partial
5....	ERRNO = 0	Testplan	100%	100%	100%		1	Clean
5.2	ERRNO = 1	Testplan	100%	100%	100%		1	Clean
5.3	ERRNO = 2	Testplan	100%	100%	100%		1	Clean
5.4	ERRNO = 3	Testplan	0%	100%	0%		1	Unlinked
5.5	ERRNO = 4	Testplan	0%	100%	0%		1	Unlinked
5.6	ERRNO = 5	Testplan	0%	100%	0%		1	Unlinked
5.7	ERRNO = 6	Testplan	0%	100%	0%		1	Unlinked
5.8	ERRNO = 7	Testplan	0%	100%	0%		1	Unlinked
5.9	ERRNO = 8	Testplan	0%	100%	0%		1	Unlinked

Ce qui est ressorti de la simulation, c'est que les tests utilisant un N inférieur à 32 fonctionnait correctement. J'ai effectuer une simulation manuelle du testcase avec $N = 31$, et je n'ai pas eu d'erreur.

En regardant plus précisément les résultats obtenus pour les testcase avec $N = 32$, J'ai vu des warnings me signifiant qu'il y avait des problèmes lors de la conversion sur 3 bits pour le calcul de la référence. Il semblerait en effet qu'avec le type integer en VHDL allant de $-(2^{31} - 1)$ à $+(2^{31} - 1)$, certaines conversion échouent. Pour résoudre ce problème, j'ai tenter de travailler avec le type unsigned, qui est d'ailleurs le type utilisé pour les testcase "aléatoire" et limites. Cependant, la fonction de conversion pose toujours problème et un travail supplémentaire devrait être apporté à l'avenir afin de résoudre ce problème.

5 Conclusion

Ce laboratoire m'a permis de me familiarisée avec plusieurs points du VHDL qui ne m'avais jamais sauté aux yeux auparavant. Le premier vient des limitations engendrée par le type INTEGER, et donc des problèmes qui peuvent en découler. Bien que je ne puisse pas être sure à 100% du fait que les problèmes que j'ai rencontrée avec $N >= 32$ viennent de ces limitations, il y a de bonnes chance que ce soit le cas. Le deuxième point, c'est l'utilisation du pseudo-aléatoire que je n'avais jamais eu l'occasion d'utiliser auparavant dans mes autres projets.

Concernant les résultats finaux, je pense avoir accomplit ce qui m'étais demandé, à l'exception de la gestion des cas avec $N >= 32$ qui pose encore problème. Je pense que si j'avais plus de temps à disposition, j'améliorerais deux points de ce travail :

— La gestion des cas avec $N >= 32$. J'avais commencer à travailler sur ce problème en fin de laboratoire, en testant des solutions utilisant des shift pour effectuer les opérations sur les unsigned dans le calcul de la référence directement, mais je n'ai pas pu arriver au bout et j'ai donc choisi de laisser ça de coté.
— La gestion des logs et de la comparaison, qui pourrait être mieux organisé avec l'aide de package, comme vu dans le laboratoire sur le logger. J'ai tenter de garder une structure propre pour les logs, mais je pense que ce point pourrait être amélioré.

Une dernière chose dont je voudrais parler, c'est le nombre de test effectué. Théoriquement, les tests aléatoire et limites pourraient être effectués sur l'entièreté des valeurs de N possible en entrée du DUV. J'ai choisi par soucis de clarté de me consacrer à certains cas. Cependant, la structure du code permet de rajouter aisément des scénarios pour les tests aléatoire et limites sans devoir modifier le code, uniquement en les ajoutant au plan de test et en les linkant dans le dossier rmdb.

Yverdon-les-Bains le 31 mars 2021

Pierrick Muller