

Département de génie informatique et génie logiciel

INF8500

Conception et vérification des systèmes embarqués

**Rapport de laboratoire #1**

**Modélisation SystemC**

Soumis par

Nicolas Lauzon, 1959682

Timothée Laborde, 1782257

Le 10 octobre 2021

## Question 1 :

Pour la connexion entre le coprocesseur 1 et l’interconnexion, nous utilisons quatre signaux. Le premier signal sert à envoyer le Packet au coprocesseur, le deuxième sert à envoyer un signal booléen True au coprocesseur pour lui indiquer le la valeur dans le signal est la bonne et que celui-ci peut commencer le traitement. Le troisième signal est pour envoyer le Packet traité à l’interconnexion. Le quatrième signal est un valeur booléenne envoyé à l’interconnexion pour lui indiquer que la valeur résultante peut être lu. Les signaux booléens sont utilisés puisque les lectures sur les sc\_signal ne sont pas bloquant. Nous avons donc besoin d’un signal supplémentaire pour confirmer que le signal est prêt et que l’on puisse lire la valeur.

Pour le coprocesseur 2, la seule différence est que nous avons utilisé des sc\_buffer au lieu de sc\_signal pour envoyer les Paquets au coprocesseur puis du coprocesseur à l’interconnexion. La raison que nous utilisons des sc\_signal ici est la même que pour le coprocesseur 1 : les lecture sur les signaux ne sont pas bloquants et nous avons besoin d’un signale supplémentaire pour dire si la valeur à lire est prête.

Finalement, pour le coprocesseur 3, nous avons utilisé deux fifos : une pour envoyer les paquets au coprocesseur puis une pour les retourner à l’interconnexion. Pour le coprocesseur 3, il n’était pas nécessaire de mettre des signaux pour avertir que les valeurs étaient transmises puisque les lectures sur les fifos sont bloquantes, alors nous n’avons pas besoins de faire d’appel à la fonction wait.

## Question 2 :

## Question 3 :

## Question 4 :

Lors d’un appel de burst\_write, une requête d’écriture est déclenchée par simple\_bus. Ceci induit un appel de main\_arction() à la descente de l’horloge qui traitera une nouvelle requête, s’il n’y en a pas de requête courante (m\_current\_request), à l’aide de handle\_request(). Dans handle\_request le bon coprocesseur sera assigné en fonction de l’adresse et effectuera un read ou write. Enfin on récupère l’état de de l’esclave, ce qui pose trois cas : simple\_bus\_ok, simple\_bus\_wait ou simple\_bus\_error. Comme l’indique leur nom, on obtient respectivement, une requête reçue, traitée et le processeur peut passer à la suivante, une requête non-traité qui force le master à l’envoyer de nouveau et enfin un arrêt complet du traitement.

## Question 5 :

Les adaptateurs ne sont effectivement pas indispensables. Nous aurions pu modifier le processeur et les coprocesseurs pour qu’ils interagissent directement entre eux sur le bus. Cependant cela limite l’abstraction, et nous force à « hard-coder » tout changement. Ceci est d’autant plus vrai quand on utilise du matériel avec des interfaces non-modifiables

## Question 6 :

Le simple\_bus et l’adaptateur de la seconde partie nous permettent une communication similaire et plus efficace entre tous les coprocesseurs et le processeur. Cela permet en plus d’avoir un processeur et des coprocesseur synchrones car ils dépendent de l’horloge. Tout ceci vient avec un coût en complexité (difficulté de compréhension et multiplication des intermédiaires) qui peut entraîner des erreurs.

Malgré tout cela reste certainement plus intéressant au manque de maintenabilité de la partie 1, ou le processeur dois s’occuper lui-même de ses coprocesseurs.