INF8500 – Vérification et conception des systèmes embarqués

Automne 2019

Laboratoire #1

Modélisation SystemC

Objectifs

L'objectif de ce laboratoire est de comprendre la méthodologie de conception pour systèmes embarqués basée sur la librairie SystemC.

Plus précisément, les objectifs du laboratoire sont :

- S'initier à la librairie SystemC
- Connaître les différents niveaux d'abstraction offert par SystemC
- Mettre en pratique la modélisation des différentes composants d'un système (processeur, bus, mémoire, etc.).
- Se familiariser avec le modèle transactionnel TLM, plus particulièrement simple bus.

Partie 1 Testbench pour 3 copro

La première partie du laboratoire consiste donc à développer un générateur de paquets qui sont des données qui serviront au traitement d'un coprocesseur (on verra l'utilité exact au laboratoire no 2). Il s'agit donc d'aiguiller des paquets en provenance du générateur et à destination de l'un de trois coprocesseurs (copro 1, 2 et 3).

Pour vous aider et tel qu'illustré à la figure 1, des modules déjà codés en SystemC vous sont fournis : le générateur de paquets (*packet generator*), le copro1 ainsi que le module d'affichage (display). La structure d'un paquet est également fournie.

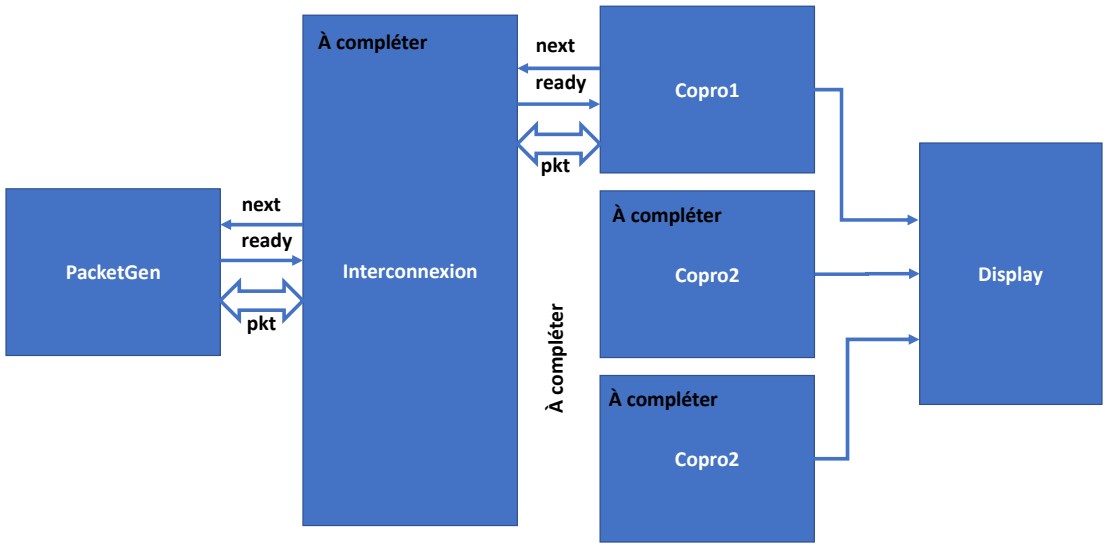

Figure 1

Définitions :

Le paquet :
Le paquet est un objet constitué d'une adresse de destination et d'un message. Le tableau suivant en décrit la structure :

Adresse de destination du paquet Type : *unsigned*	Message du paquet Type : *sc_biguint<64>*

Le tableau suivant établit la liste des méthodes qui vous seront utiles :

Méthode de l'objet paquet	Définition
Packet(*unsigned* **adr**, *sc_biguint<64>* **mesg**)	Cette méthode permet de construire un nouveau paquet constitué d'une adresse de destination **adr** et du message **mesg**
unsigned getAddress()	Cette méthode retourne l'adresse de destination du paquet.

Exemple d'utilisation de l'objet paquet :

```
sc_biguint<64> message = 1234 ;      // construction d'un message
Packet paquet(1,message) ;           // construction d'un paquet ayant
                                     // pour adresse de destination « 1 »
                                     // et comme message « message ».
int adr = Packet.getAddress() ;      // récupération de l'adresse du packet.
                                     // Ici, adr vaudra 1.
```

Pour plus de renseignement, reportez-vous au code source du fichier packet.cpp et packet.h.

Le générateur de paquets :
Ce module (*packet_gen*) va permettre de générer des paquets qui seront envoyés au destinataire en passant l'interconnexion. Il est constitué de 2 ports de sortie (*packet_ready* et *packet_out*) et d'un port d'entrée (*next_packet*). Ce module est sensible au port d'entrée (*next_packet*). Il génère ensuite un paquet qu'il positionne sur le port correspondant (*packet_out*) et signal que celui est prêt à être envoyé (*packet_ready*). Puis, il se remet en attente pour envoyer le prochain paquet (signal *next_packet*).

Pour plus de renseignement, reportez-vous au code source du fichier packet*_gen.cpp* et *packet_gen.h*.

Le coprocesseur 1 (copro1):
Ce module permet de recevoir des paquets émis par le générateur de paquet. *Copro1* se met en attente d'un signal sur le port d'entré (*ready*) indiquant qu'un paquet est disponible. Il récupère ensuite le paquet et transmet un signal sur le port de sortie (*ack*) afin d'avertir que le paquet a été récupéré. Pour l'instant le fonctionnement de copro1 n'est pas défini, ça viendra dans le lab 2.... Ensuite, chaque paquet reçu est retransmis au module d'affichage à l'aide de la même procédure de transfert. Pour plus de renseignement, reportez-vous au code source du fichier station.cpp et *station.h*.

Le module d'affichage :
Ce module reçoit les paquets et des chaînes de caractères des modules station de réception puis les affiche à l'écran. Ce module appelé « display » est constitué de trois (3) ports d'entrée et d'un port de sortie. Un port reçoit le paquet, un autre reçoit une chaîne de caractères et le dernier reçoit un signal (*show_a_message*) pour indiquer qu'un message (la chaîne de caractère et le paquet correspondant) est disponible. Le port de sortie indique aux modules stations que le paquet a bien été reçu. Pour plus de renseignement, reportez-vous au code source du fichier *display.cpp* et *display.h*.

Travail à réaliser pour la partie 1 :

L'interconnexion
Ce module comprend toutes les fonctionnalités nécessaires pour aiguiller un paquet à son destinataire. Il doit être relié au générateur de paquets ainsi qu'aux trois coprocesseurs comme indiqué sur la figure 1. L'interconnexion est averti par le générateur de paquet qu'un paquet est disponible à l'aide d'un port d'entré. Il récupère ensuite le paquet. Ensuite, il lit l'adresse de destination du paquet puis envoie celui-ci vers le coprocesseur correspondant à cette adresse, plus précisément un paquet ayant pour numéro de destination « 1 » sera envoyé vers le module « copro 1 »). Ce module, écrit en SystemC devra respecter les différents signaux et ports des modules auxquels il sera connecté (le

générateur de paquets et les modules de réception qui utiliserons des modes de communication différents).

Le coprocesseur 2 (copro2):
Ce module est identique à copro1, mais il utilise un mode de communication différent. Vous devrez utiliser une FIFO de SystemC (*sc_fifo*). Pour ce faire, reportez-vous aux acétates du cours ainsi qu'aux documents de références du site web du cours portant sur SystemC. Ce module devra être écrit dans le fichier copro2.cpp et copro2.h.

Le coprocesseur 3 (copro3):
Ce module est identique à copro1, mais il utilise un mode de communication différent. Vous devrez utiliser un buffert de SystemC (*sc_buffer*). Pour ce faire, reportez-vous aux acétates du cours ainsi qu'aux documents de références du site web du cours portant sur SystemC. Ce module devra être écrit dans le fichier copro2.cpp et copro2.h.

Partie 2 Raffinement de l'interconnexion en utilisant simple bus

L'objectif de cette partie est de connecter entre eux le générateur de paquets et les 3 coprocesseurs. La figure 2 (page suivante) présente le résultat.

Définitions :

Le bus :
Comme son nom l'indique, *simple bus* modélise un bus c'est-à-dire un canal de communication permettant de relier ensemble différents modules. Ces différents modules se divisent en deux catégories, les modules maîtres (initiateurs) et les modules esclaves (cibles). Les modules maîtres peuvent envoyer des requêtes de lecture et d'écriture tandis que les modules esclaves ne font que répondre à ses requêtes.

Adaptateurs (Wrappers) *Packet_Gen_Adapt*:

Ici le seul maître *PacketGen* (mais on pourrait en avoir plus d'un puisqu'on a un arbitre), possède un adaptateur (*PaketGen_Adapt*) qui devra récupérer les données envoyer en respectant le protocole port à port côté *PacketGen* pour créer la bonne requête côté bus. Afin de pouvoir créer des requêtes sur le bus, le maître doit posséder un port *sc_port<simple_bus_blocking_if>* par lequel il pourra accéder aux méthodes de l'interface bloquante du bus. Ces méthodes sont définies dans le fichier *simple_bus.cpp*. Vous utiliserez la méthode *simple_bus_status burst_write* (2^{e} dans le tableau suivant).

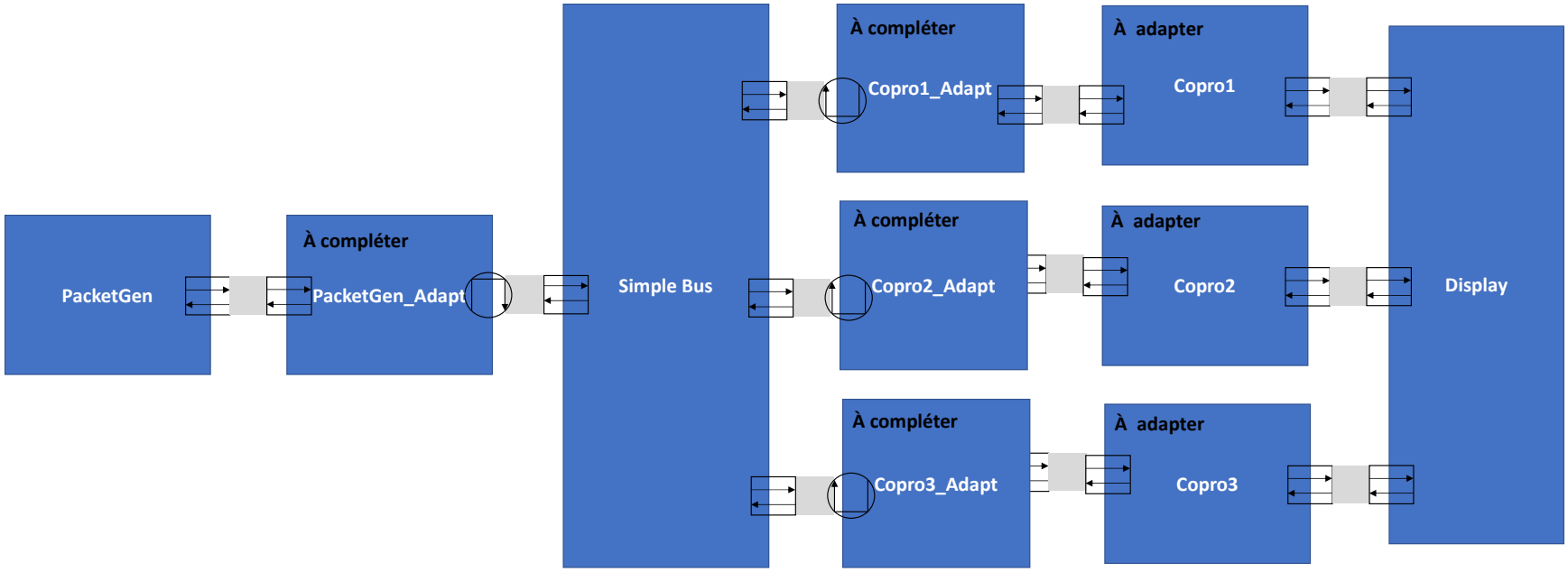
PacketGen
À compléter
PacketGen_Adapt
Simple Bus
À compléter
Copro1_Adapt
À adapter
Copro1
À compléter
Copro2_Adapt
À adapter
Copro2
À compléter
Copro3_Adapt
À adapter
Copro3
Display

Figure 2

Les modules maîtres (module) : `simple_bus_blocking_if.h`	
Interface	Définition

`Virtual simple_bus_status burst_read(`
`  unsigned int unique_priority`
`, int *data`
`, unsigned int start_address`
`, unsigned int length = 1`
`, bool lock = false) = 0;`
	Cette méthode vous permet d'effectuer une lecture bloquante de taille `length*4` octets vers `*data`. Les données sont lues depuis la mémoire d'un esclave à l'adresse de départ `start_adress.` `unique_priority` est la priorité de ce maître et la variable `lock` permet d'empêcher des requêtes de plus hautes priorités de prendre la main

`virtual simple_bus_status burst_write(`
`  unsigned int unique_priority`
`, int *data`
`, unsigned int start_address`
`, unsigned int length = 1`
`, bool lock = false) = 0;`
	Cette méthode permet l'écriture bloquante de la donnée `*data` de taille `length*4` octets à l'adresse `start_adress` d'un esclave.

Copro1_Adapt, Copro2_Adapt et Copro3_Adapt :

D'autre part, les esclaves *copro1*, *copro2* et *copro3* possèdent chacun un adaptateur. Ces derniers héritent de l'interface *simple_bus_slave_if* esclave lors de la déclaration de classe.

Les modules esclaves : simple_bus_slave_if.h	
Interface	Définition

`Virtual simple_bus_status read(`
`int *data`
`, unsigned int address) = 0;`
	Cette méthode permet de spécifier la méthode de lecture sur un périphérique connecté sur le bus. Elle ne sera pas utiliser dans le lab1 et donc pas à implémenter.

`virtual simple_bus_status write(`
`int *data`
`, unsigned int address) = 0;`
	Cette méthode permet de spécifier la méthode d'écriture sur un périphérique connecté sur le bus. Cette méthode est à implémenter. Elle est dépendante du périphérique utilisé. Cette fonction sera à développer lors de l'élaboration de vos modules copros.

`virtual unsigned int start_address()`	Renvoie l'adresse de départ de la mémoire du module (à implémenter)
`virtual unsigned int end_address()`	Renvoie l'adresse de fin e la mémoire du module (à implémenter)

Pour simplifier le laboratoire, nous ne ferons que des *write* au copro (on pourrait imaginer plus tard un maître qui lierait le résultat après un certain processing). Vous aurez donc pour ce lab qu'à concevoir *write*, *start_address* et *end_address*.

Le paquet :
Le paquet est un objet constitué d'une adresse de destination et d'un message. Le tableau suivant en décrit la structure :

ID (32 bits)	Adresse de destination (32 bits)	Payload (128 bits)

Le tableau suivant établit la liste des méthodes qui vous seront utiles :

Méthode de l'objet paquet	Définition
Packet(unsigned **adr**, unsigned int **seed**)	Cette méthode permet de construire un nouveau paquet constitué d'une adresse de destination **adr** et du message généré à l'aide de **seed**
Packet(unsigned int* **packet**)	Cette méthode permet de construire un nouveau paquet à l'aide d'un tableau **packet** de bonne taille
Unsigned getAddress()	Cette méthode retourne l'adresse de destination du paquet.
void setAddress(unsigned addr)	Cette méthode permet de modifier l'adresse d'un paquet

Pour plus de renseignement, reportez-vous au code source du fichier packet.cpp et packet.h.

Travail à réaliser pour la partie 2 :

Adaptateurs (Wrappers) *Packet_Gen_Adapt*:

Vous devrez réaliser la fonction *Packet_Gen_Adapt ::pkt_dispatch* qui prendra un paquet de *packet_gen::generate* et le transférera à l'adapteur du copro à travers simple_bus. Pour ce laboratoire, vous travaillerez avec *burst_write*.

Inspirez-vous de l'exemple donné avec les 2 types de mémoires (*simple_bus_fast_mem.cpp* et *simple_bus_slow_mem.cpp*).

Copro1_Adapt, Copro2_Adapt et Copro3_Adapt :
Tel que discuté plus haut, vous devez donc réaliser la méthode *write* de chaque coprocesseur qui lira le paquet transférer par le bus (initié par *simple_bus_status burst_write*) et le tranférera à son *copro* pour une impression éventuelle dans *display*. Il s'agira d'adapter la méthode au bon protocole. Pour ce faire, ces fonctions doivent s'exécuter en un seul delta-cycle. À l'issue de cette exécution, elles doivent renvoyer une de ces trois valeurs :

• SIMPLE_BUS_OK : La requête s'est terminée avec succès. Le maître peut continuer son exécution

• SIMPLE_BUS_WAIT : La requête se poursuit, la fonction *write* sera réexécutée au cycle suivant. Le maître reste bloqué.

• SIMPLE_BUS_ERROR : La requête s'est soldée par un échec. Le maître peut continuer son exécution s'il le désire.

Il devra posséder une mémoire permettant de contenir un paquet complet (soit un *unsigned int*[6]). Le paquet pourra ainsi être restauré au copro par *packet = new Packet(MEM).*

Inspirez-vous de l'exemple donné avec *simple_bus_master_blocking.cpp*

<u>Rapport, questions et date de remise:</u>

Le rapport et le code sera à rendre avant le laboratoire no 2 (dans 2 semaines). Le rapport sera surtout constitué de questions à répondre. **Je vous envoie ça cette semaine**.

Guy Bois, prof. GIGL
Responsable du INF8500