1 - Canal Primitif (Primitive Channel)

1.1 Présentation

Nous voulons réaliser un bus de niveau transactionnel entre un maitre (Master) et un esclave (Slave). Pour cela, nous allons définir un canal primitif. Ce type de canal permet de connecter des modules entre eux et de fournir un ensemble de services pour communiquer.

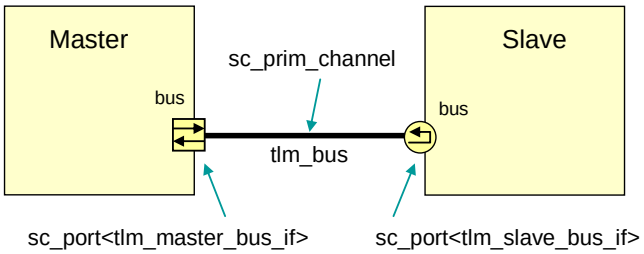

Figure 1 : Présentation du bus TLM.

1.2 Les interfaces

L'interface est la partie visible du canal qui fourni des fonctions ou méthodes qui peuvent être différentes du coté maitre et du coté esclave. Les méthodes, coté maitre, (*tlm_master_bus_if*) sont :

- Service 1 : Lecture d'une donnée à une adresse,
```
int read2slave(unsigned int addr_)
```
- Service 2 : Ecriture d'une donnée à une adresse,
```
void write2slave(unsigned int addr_, int data_)
```
- Service 3 : 2 lignes d'interruptions (int0 et int1)
```
const sc_event& int0_event() const
const sc_event& int1_event() const
```

Du coté esclave (tlm_slave_bus_if), nous avons besoin des services suivants :

- Service 1 & 2 : Un événement qui permet d'être prévenu d'une lecture ou d'une écriture
```
const sc_event& rw_event() const
```
- Service 1 & 2 : Des fonctions qui permettent de récupérer le type d'accès (lecture ou écriture), l'adresse
```
const bool is_read() const
const unsigned int getAddress() const
const int getData() const
```
- Service 1 : Une fonction qui permet d'envoyer la donnée au maitre dans le cas d'une lecture.
```
void senddata2master(int data_)
```
- Service 2 : la donnée dans le cas d'une écriture par le maitre
```
const int getData() const
```
- Service 3 : Deux fonctions qui envoient respectivement l'interruption int0 et int1
```
void int0_notify()
void int1_notify()
```

Travail demandé

1) Ecrire l'interface tlm_master_bus_if (*fichier tlm_master_bus_if.h*)
2) Ecrire l'interface tlm_slave_bus_if (*fichier tlm_slave_bus_if.h*)

1.3 L'implémentation de la communication primitive tlm_bus

La lecture et l'écriture utilise un événement *rw2slave_ev* (renvoyé par la méthode *rw_event()*) pour prévenir l'esclave qu'une demande d'accès en lecture ou écriture est en cours. Le bus ne peut effectuer qu'un seul accès à la fois.

L'esclave est prévenu qu'un accès en lecture/écriture a lieu grâce à un événement (*rw2slave_ev*). Lors de la réception de cet événement, 2 cas peuvent se produire :

- C'est une écriture. La méthode *is_read()* renvoi false. Des méthodes permettent de récupérer l'adresse (*getAddress()*) et la donnée (*getData()*).
- C'est une lecture. La méthode *is_read()* renvoi true. Seule la méthode de récupération de l'adresse est utile. L'esclave ira chercher la donnée correspondant à l'adresse et enverra la donnée en utilisant le méthode *senddata2master()*.

De plus, l'esclave peut envoyer à travers le bus 2 interruptions (*int0_notify()* et *int1_notify()*) qui correspondent à 2 événements.

En résumé, il faut :

- Un événement *rw2slave_ev* (le maitre effectue un accès en lecture/écriture),
- Un événement *w2master_ev* (l'esclave envoi la donnée vers le maitre),
- Deux événements pour les interruptions *int0_interrupt*, *int1_interrupt*,
- Des champs pour stocker la donnée, l'adresse et le type d'accès,
- Un mutex noté *mutex* qui permet de contraindre un seul accès au bus à la fois.

Travail demandé

1) Ecrire la communication primitive (*tlm_bus.h* et *tlm_bus.cpp*) qui hérite de *sc_prim_channel* et des deux interfaces *tlm_master_bus_if* et *tlm_slave_bus_if*.

```
struct TLMBus : public sc_prim_channel,
                public tlm_master_bus_if, tlm_slave_bus_if
 { ...
```

1.4 Le test

Pour tester le bus, nous allons écrire le maitre (Master) et un module de test (Slave) comme le montre la Figure 1.

Travail demandé

1) Ecrire le maitre qui comprend 3 processus :
   - Un processus qui effectue des lectures/écritures,
   - Un processus qui réagit sur l'interruption int0 et affiche un message,
   - Un processus qui réagit sur l'interruption int1 et affiche un message.
2) Ecrire l'esclave qui modélise une mémoire. Cet esclave comprend 2 processus :
   - Un processus qui réagit aux lectures/écritures.
   - Un processus qui envoie des interruptions sur les lignes int0 et int1.