Modélisation Transactionnelle des Systèmes sur Puces

Ensimag 3A, filière SLE

Janvier 2010

Consignes :

- Durée : 2h.
- Tous documents autorisés.
- Le barème est donné à titre indicatif.
- On attend des réponses courtes et pertinentes, inutile de recopier le cours.
- Les schémas brouillons seront pénalisés.

1 Questions de cours

Pour chacune des question de cours, la réponse ne devra pas dépasser 4 lignes.

Question 1 (1 point) *Dans la déclaration suivante, quelle est l'utilité du mot clé « `virtual` » à la deuxième ligne (avant `public`) ?*

```
template<typename T>
class rendezvous_in_if : virtual public sc_core::sc_interface {
   public:
      virtual T get() = 0;
};
```

Il est nécessaire puisque sc_interface risque d'être au sommet d'une hiérarchie de classe en losange, i.e. d'être dérivée plusieurs fois par la même classe fille.

Question 2 (1 point) *A quoi sert un `sc_port` ? un `sc_export` ? Quelle est la différence entre les deux ?*

Un `sc_port` permet de faire des appels de fonctions vers l'extérieur d'un module, un `sc_export` permet d'en recevoir (d'exposer une interface au reste du programme).

Question 3 (1 point) *Pour exécuter sur une machine de type Pentium un logiciel embarqué sur une plate-forme TLM avec ISS MIPS, doit-on compiler avec un compilateur croisé ? Pourquoi ?*

Oui, car l'ISS reproduit fidèlement le comportement du processeur, il va interpréter les instructions MIPS.

Question 4 (1 point) *La bibliothèque TLM-2 apporte un certain nombre de choses qui ne sont pas présentes dans la bibliothèque SystemC. Citez deux de ces apports (leur nom, et une description en quelques mots).*

Les sockets : un `sc_port` et un `sc_export` dans un même objet.

Les TLM generic payloads : standardisation du contenu des transactions.

Les interfaces (forward, backward, non-blocking, ...) pour savoir quelles fonctions on peut appeler.

...

Question 5 (1 point) *Le code*

```
#include <systemc>
class m : public sc_module { };
```

lève l'erreur suivante :

```
test.h:2: error: expected class-name before '{' token
```

Pourquoi ? Proposez une version correcte.

Il manque `sc_core` devant `sc_module`.

5

2 Questions de compréhension

Le niveau d'abstraction TLM peut être décliné en plusieurs variantes. Dans cette question, nous ne considérerons que deux de ces variantes : PV (modèle sans temps, ou « non-timés ») et PVT (modèles avec temps précis). En général, un modèle PVT est plus difficile à écrire qu'un modèle PV, mais les modèles PVT ont des applications impossibles au niveau PV (par exemple, l'analyse de performance).

Question 6 (4 points) *On entend parfois dire qu'il est préférable de développer le logiciel embarqué sur des modèles PV pour obtenir du code plus robuste.*

Justifiez cette affirmation. Appuyez votre argumentation sur au moins un exemple.

3 Implémentation d'un timer

On s'intéresse maintenant à un composant appelé « timer ». La documentation technique du composant est donnée en annexe. Ce composant sera intégré dans une plateforme contenant également un CPU, une RAM, et un bus. Il n'y a pas de contrôleur d'interruption, le timer est connecté directement au CPU.

Un squelette de code pour le timer est donné :

Fichier timer.h :

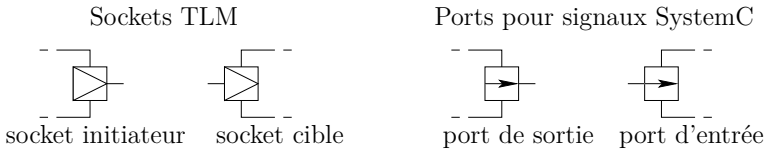
```
1  #ifndef TIMER_H
2  #define TIMER_H
3
4  #include "basic.h"
5
6  #define TIMER_LOAD  0x0
7  #define TIMER_START 0x4
8  #define TIMER_VALUE 0x8
9
10 SC_MODULE(TIMER) {
11         SC_HAS_PROCESS(TIMER);
12
13         basic::target_socket<TIMER> target;
14         sc_core::sc_out<bool> irq;
15
16         TIMER(sc_core::sc_module_name name, sc_core::sc_time period);
17
18         tlm::tlm_response_status
19                 read(basic::addr_t a, basic::data_t& d);
20         tlm::tlm_response_status
21                 write(basic::addr_t a, basic::data_t d);
22
23 private:
24         void count(void);
25         void interrupt(void);
26
27         sc_core::sc_event irq_event;
28         sc_core::sc_time m_period;
29
30         // Timer internal registers
31         basic::data_t m_timer_value;
32         basic::data_t m_timer_load;
33         basic::data_t m_timer_control;
34 };
35
36 #endif // TIMER_H
```

Fichier timer.cpp :

```
1   #include "basic.h"
2   #include "timer.h"
3
4   TIMER::TIMER(sc_core::sc_module_name name, sc_core::sc_time period) :
5           sc_core::sc_module(name), m_period(period) {
6           SC_THREAD(count);
7           SC_THREAD(interrupt);
8   }
9
10  void TIMER::interrupt() {
11          while(true) {
12                  // lancer une interruption sur le port irq à chaque fois que
13                  // l'événement irq_event est notifié.
14                  // non implémenté
15          }
16  }
17
18  void TIMER::count() {
19          while (true) {
20                  if ((m_timer_value > 0) && (m_timer_control == 1)) {
21                          m_timer_value--;
22                          if (m_timer_value == 0) {
23                                  irq_event.notify();
24                                  m_timer_value = m_timer_load;
25                          }
26                  }
27                  wait(m_period);
28          }
29  }
30
31  tlm::tlm_response_status
32  TIMER::read(basic::addr_t a, basic::data_t& d) {
33          switch(a) {
34          case TIMER_LOAD:
35                  // non implemente
36                  break;
37          case TIMER_START:
38                  // non implémenté
39                  break;
40          case TIMER_VALUE:
41                  d = m_timer_value;
42                  break;
43          default:
44                  return tlm::TLM_ADDRESS_ERROR_RESPONSE;
45          }
46          return tlm::TLM_OK_RESPONSE;
47  }
48
49  tlm::tlm_response_status
50  TIMER::write(basic::addr_t a, basic::data_t d) {
51          int timer = 0;
52          switch(a) {
53          case TIMER_LOAD:
54                  m_timer_load = a;
```

```
55                  m_timer_value = a;
56                  break;
57          case TIMER_START:
58                  // non implémenté
59                  break;
60          case TIMER_VALUE:
61                  return tlm::TLM_COMMAND_ERROR_RESPONSE;
62                  break;
63          default:
64                  return tlm::TLM_ADDRESS_ERROR_RESPONSE;
65          }
66          return tlm::TLM_OK_RESPONSE;
67  }
```

Question 7 (1 point) *En utilisant les conventions vues en cours et rappelées partiellement ci-dessous, donnez une vue graphique de la plate-forme.*



Question 8 (1.5 points) *Écrire le corps de la fonction `sc_main` qui construit la plateforme.*

Question 9 (1 point) *Dans le corps de la fonction `TIMER::write`, on trouve une instruction `return tlm::TLM_COMMAND_ERROR_RESPONSE`. Expliquez son rôle.*

Question 10 (3 points) *Donnez le code pour les trois cas non-implémentés dans les méthodes `read` et `write`.*

Question 11 (2.5 points) *Donnez le corps du `SC_METHOD` « `TIMER::interrupt` ». Attention, les interruptions sont en mode "pulse", donc le signal d'interruption doit passer à 1 pendant une période (`m_period`), puis repasser à 0. On suppose qu'une interruption est toujours terminée quand l'interruption suivante commence.*

4 Logiciel embarqué

On souhaite maintenant écrire en langage C le logiciel embarqué sur le processeur (portable sur la vraie puce et la plate-forme TLM, en simulation native ou non). On suppose l'existence des fonctions vues dans le cours :

```
extern void write_mem(uint32_t addr, uint32_t data);
extern uint32_t read_mem(uint32_t addr);
extern void cpu_relax();
extern void wait_for_irq();
```

et la constante suivante définie :

```
// Adresse de base du composant cible timer :
const int timer_start_addr = /* ... */;
```

Au démarrage du processeur, le vecteur d'interruption est correctement initialisé et fait en sorte que la routine d'interruption suivante soit appelée pour chaque interruption reçue en provenance du timer :

```
int irq_from_timer_received = 0;
void __interrupt_from_timer() {
        irq_from_timer_received = 1;
}
```

Question 12 (2 points) *Écrire une fonction* `pause(int n);`.

Cette fonction doit prendre en argument un nombre de périodes de timer, et faire une attente de n fois cette durée (par exemple, si la période est 1 ns, alors `pause(5)` *fait une attente de 5 ns). Il faudra programmer le timer, et attendre que le timer expire et envoie une interruption. On peut supposer qu'il n'y a qu'un utilisateur du timer, mais le code devra être robuste au cas où le processeur a plusieurs sources d'interruption (dans ce cas, les autres interruptions auraient une routine différente de* `__interrupt_from_timer`*).*

```
void pause(int n) {
   // on programme le timer
   write_mem(timer_start_addr + TIMER_LOAD,    n);
   write_mem(timer_start_addr + TIMER_START, 0x1);
   // on attend une irq
   while(!irq_from_timer_received) {
      wait_for_irq();
      // ou cpu_relax(); mais c'est moins efficace.
   }
   // on arrete le timer
   write_mem(timer_start_addr + TIMER_START, 0x0);
   // et on acquite le traitement (doit etre apres l'arret du timer)
   irq_from_timer_received = 0;
}
```

11
20

Annexe

Timer Documentation

Le composant Timer est un module esclave *Advanced Microcontroller Bus Architecture* (AMBA) destiné à être connecté à un bus haute-performance *Advanced High-performance Bus* (AHB).

Fonctionnalités

Le composant Timer permet de programmer une interruption qui arrivera dans le futur, après une attente d'une durée fixée à l'avance.

Entrées/Sorties

Le composant Timer possède les entrées/sorties suivantes :

- Interface esclave compatible AMBA
- Une sortie d'interruption `int_out`

Les interruptions sont ici en mode "pulse"

Fonctionnement interne

Initialement la sortie d'interruption du composant est à 0 (ou `false`). Le timer contient un compteur qui doit d'abord être initialisé via le registre `TIMER_LOAD`. Le timer est ensuite démarré en écrivant la valeur 1 dans le registre `TIMER_START`, et son compteur interne va être décrémenté de 1 toutes les nanosecondes. Quand le compteur atteint la valeur 0, une interruption est lancée, et le compteur est rechargé à sa valeur initiale et le décompte recommence.

Récapitulatif des registres

Adresse relative	Type	Taille	Valeur initiale	Nom	Description
0x00	Lecture/Écriture	32 bits	0	TIMER_LOAD	Registre Chargement du timer
0x04	Lecture/Écriture	32 bits	0	TIMER_START	Registre Démarrage du timer
0x08	Lecture seule	32 bits	0	TIMER_VALUE	Registre Valeur courante du compteur

Registre TIMER_LOAD

Ce registre contient la valeur initiale du compteur interne du timer. Cette valeur est utilisé quand le décompte commence (soit suite à une expiration du timer, soit quand le timer est démarré via le registre TIMER_START.

Registre TIMER_START

Écrire 1 dans ce registre démarre ou redémarre le timer. Écrire 0 dans ce registre stoppe le décompte. L'effet d'une écriture d'une autre valeur est indéterminée.

Une lecture renvoit 1 si le timer est en train de décompter, et 0 s'il est arrêté. La valeur initiale est 0.

Registre TIMER_VALUE

La valeur courante du compteur interne. En d'autres termes, c'est le nombre de milisecondes restantes avant la prochaine interruption.