Modélisation Transactionnelle des Systèmes sur Puces

Ensimag 3A, filière SLE

Matthieu Moy — Février 2016

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 trop brouillons seront pénalisés.

1 Question de cours sur C++

1.1 Héritage et constructeurs

On considère le programme suivant :

```
#include <iostream>
using namespace std;

struct Composant {
        string m_name;
        Composant(const char *name) {
                m_name = name;
        }
};

struct A {
        const int a;
        Composant c;

        A () {
                a = 42;
        }

        void printA() {
                cout << a << endl;
        }
};
```

```
int main() {
        A a;
        a.printA();
}
```

Question 1 (1.5 points) *Le programme est-il compilable ? Si oui, qu'affiche-t-il ? Si non, proposez une correction.*

Il n'est pas compilable, il faut chainer sur les constructeurs de a (constant) et c (pas de constructeur par défaut).

1.2 Templates

On considère la classe template suivante, qui implémente une liste chaînée en utilisant un élément fictif comme premier élément. Certains types ont été remplacés par `D1`, ... `D6` :

```
template<typename T1>
struct cell {
        D1 elem; // <--
        cell *next;
};

template<typename D2> // <--
struct list {
        D3 first; // <--
        D4 last; // <--

        list() : last(&first) {
                first.next = NULL;
        }

        D5 add(T2 elem) { // <--
                cell<T2> *c = new cell<T2>();
                c->elem = elem;
                c->next = NULL;
                this->last->next = c;
                this->last = c;
        }

        void print() {
                cell<T2> *i = first.next;
                while (i != NULL) {
                        cout << i->elem << endl;
                        i = i->next;
                }
        }
```

```
};

int main() {
        list<D6> t; // <--
        t.add(42.0);
        t.add(43.0);
        t.add(41.0);
        t.print(); // Affiche 42.0, 43.0, 41.0
}
```

Question 2 (1.5 points) *Par quoi faut-il remplacer `D1`, `D2`, `D3`, `D4`, `D5` et `D6` pour que la déclaration soit proprement typée ?*

0,5 point par définition.

```
#define D1 T1
#define D2 T2
#define D3 cell<T2>
#define D4 cell<T2> *
#define D5 void
#define D6 float
```

2 Questions de cours sur TLM et les SoCs

Question 3 (1.5 points) *Quelle est la différence entre un `sc_port`, un `sc_export` et un `socket` ? Quelles sont leurs utilités dans le cadre de TLM-2 ? Quelles sont leurs utilités hors du cadre de TLM-2 ?*

`sc_port` permet d'envoyer un appel de méthode vers l'extérieur d'un module. `sc_export` d'en recevoir un de l'extérieur. Un `socket` contient un port et un export, ce qui permet d'envoyer des requêtes et de recevoir des réponses en TLM-2.

Question 4 (1.5 points) *En conclusion de son exposé, Laurent Maillet-Contoz a cité plusieurs nouvelles tendances et challenges à traiter en modélisation TLM. Citez-en 3 (ou éventuellement, citez des nouveaux challenges non-mentionnés pendant la conférence).*

- Virtualisation
- Sécurité
- Injection de fautes
- Maintenance des modèles (faire en sorte que les vieux modèles restent à jour)
- Simulation hybride et multi-physique.
- Gestion de l'énergie et de la température

Question 5 (1 point) *Le "post-silicon debug" consiste à identifier et résoudre des problèmes qui se produisent sur une puce physique. Que peut apporter un modèle TLM pour le "post-silicon debug" ? (de préférence, donner un exemple)*

Un cas délicat est celui d'une puce sur laquelle le logiciel ne démarre pas : pas d'entrée-sortie possible car l'exécution plante avant de les avoir mises en place. En simulation, on peut observer le comportement de la puce même si le logiciel n'est pas fonctionnel.

3 Modélisation du temps et ordonnancement (scheduling) en SystemC

On considère le programme suivant :

```
#include <systemc>
#include <iostream>

using namespace std;
using namespace sc_core;

SC_MODULE(A) {
        void f() {
                // name() affiche le nom SystemC du composant.
                cout << name() << ".f1" << endl;
                wait(3, SC_SEC);
                cout << name() << ".f2" << endl;
                sleep(2);
                cout << name() << ".f3" << endl;
        }
        SC_CTOR(A) {
                SC_THREAD(f);
        }
};

int sc_main(int, char**) {
        A a("a"), b("b");
        cout << "Debut" << endl;
        sleep(1);    // <--- Attention
        sc_start();
        sleep(1);    // <--- Attention
        cout << "Fin" << endl;
        return 0;
}
```

On rappelle que la fonction `sleep(N)` est une fonction POSIX (pas une fonction SystemC) qui provoque une attente de `N` secondes du processus courant.

La norme SystemC autorise plusieurs exécutions possibles de ce programme (i.e. le scheduler a la liberté de choisir entre ces exécutions). Certaines exécutions produisent les

mêmes affichages.

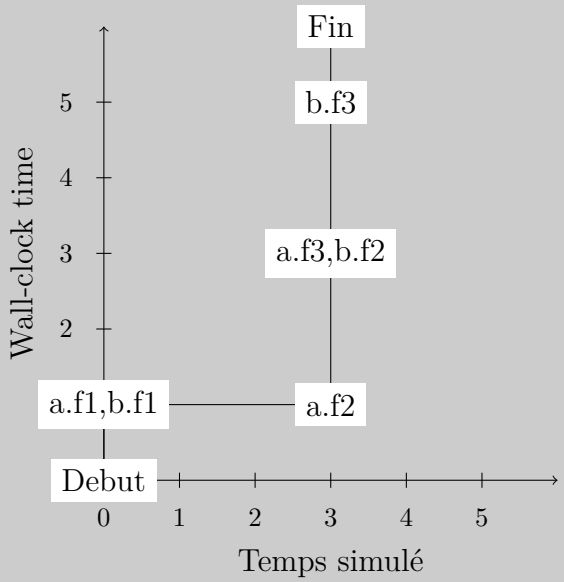
Question 6 (1.5 points) *Combien y a-t-il d'exécution différentes (i.e., produisant des affichages différents) autorisées par SystemC ? Donnez cette ou ces traces d'exécution.*

Initialement, les deux processus sont éligibles, et peuvent donc s'exécuter dans n'importe quel ordre. À l'instant simulé t=3, même situation. Il y a donc 4 exécutions :

```
Debut
a.f1
b.f1
a.f2
a.f3
b.f2
b.f3
Fin

Debut
b.f1
a.f1
a.f2
a.f3
b.f2
b.f3
Fin

Debut
a.f1
b.f1
b.f2
b.f3
a.f2
a.f3
Fin

Debut
b.f1
a.f1
b.f2
b.f3
a.f2
a.f3
Fin
```

Question 7 (2 points) *Représentez une des exécutions sur un graphique à deux dimensions. On mettra le "wall-clock time" sur l'axe des ordonnées et le temps simulé*

sur les abscisses. Le schéma fera apparaître `Debut`, `a.f1`, `b.f1`, ..., `a.f3`, `b.f3`, `Fin`. *Si on néglige le temps de calcul, combien de temps prendra une exécution en "wall-clock time" ? Combien de temps en temps simulé ?*



On a donc 3 secondes de temps simulé pour une exécution de 6 secondes en wall-clock time.

4 Implémentation d'un accélérateur matériel

Dans cette partie, nous allons ajouter un accélérateur matériel à la plate-forme du TP3 (jeu de la vie sur processeur MicroBlaze). Une opération nécessaire au fonctionnement du jeu de la vie est l'initialisation de la mémoire à 0, ce qui est pour l'instant fait en logiciel. Une manière de faire est d'utiliser la fonction `memset` de la libc :

```
void clr_screen(uint32_t base_addr) {
        int nbytes = (VGA_LINE * VGA_HEIGHT) / CHAR_BIT;
        memset(base_addr, 0, nbytes);
}
```

Le logiciel tourne sans libc, mais on peut donner une implémentation logicielle de `memset` :

```
#define memset soft_memset

static inline uint32_t copy_last_byte(uint32_t value) {
        value &= 0xFF;
        return value |
                (value << 8) |
                (value << 16) |
                (value << 24);
}
```

```
/**
 * Recopie 'num' fois la valeur 'value' prise sur un octet,
 * à partir de l'adresse 'ptr'.
 * Précondition : value et num doivent être multiples de 4.
 */
void soft_memset(uint32_t ptr, int value, size_t num) {
        int count;
        value = copy_last_byte(value);
        for (count = 0; count < num / 4; count++) {
                write_mem(ptr + count * 4, value);
        }
}
```

Question 8 (1 point) *Expliquez la raison d'être de la fonction* `copy_last_byte` *ainsi que celle des opérations* `* 4` *et* `/ 4`*. On rappelle que* `write_mem` *fait une écriture d'un mot mémoire, à une adresse bien alignée.*

`memset` recopie les bits de poids faibles de `value` sur chacun des octets de la zone concernée. Comme on ne peut ici faire que des accès mot par mot (32 bits), qui sont de toutes façons plus efficaces que d'accéder aux octets un par un, on fait 4 fois moins d'accès, mais on copie les octets 4 par 4.

Nous allons maintenant modéliser un composant matériel permettant une implémentation matérielle de `memset`. cette fonction est un sous-ensemble de ce qu'un DMA programmable est capable de faire, nous appellerons donc le composant `DMA`, même s'il est ici plus que limité.

Le gain sur le TP3 sera bien entendu très faible, puisqu'on n'utilise cette fonction que deux fois, au démarrage du programme.

La fonction `memset` du logiciel utilisant le composant matériel est définie comme suit. Elle aura exactement le même comportement que `soft_memset`, mais utilisera le composant matériel. Pour plus de clarté, on la nomme `hard_memset` (mais c'est une fonction du logiciel embarqué) :

```
void hard_memset(uint32_t ptr, int value, size_t num);
```

Une spécification partielle du composant `DMA` est fournie en annexe.

4.1 Version simplifiée sans interruptions

Dans cette partie, on fait l'hypothèse simplificatrice que l'initialisation mémoire est faite en temps nul par le composant `DMA` (l'initialisation mémoire est terminée quand l'écriture sur le registre `START` termine). Cette hypothèse peut être utilisée dans le modèle TLM du composant et dans le logiciel embarqué.

Question 9 (1 point) *Expliquez les rôles respectifs des sockets initiateur et cible du composant* `DMA`*.*

Initiateur : pour pouvoir accéder à la RAM. Cible : pour recevoir les ordres depuis le logiciel embarqué.

Le modèle TLM du composant `DMA` est donné par la classe ci-dessous :

```
class DMA : public sc_core::sc_module {
public:
        ensitlm::target_socket<DMA> target;
        ensitlm::initiator_socket<DMA> initiator;

        tlm::tlm_response_status
                read(ensitlm::addr_t a, ensitlm::data_t& d);

        tlm::tlm_response_status
                write(ensitlm::addr_t a, ensitlm::data_t d);

        SC_CTOR(DMA) {/* */};
private:
        uint32_t m_addr, m_num, m_value;

        /**
         * Écrit 'num' fois la valeur 'value' (sur un octet) à partir
         * de l'adresse 'ptr'.
         */
        void run(uint32_t ptr, int value, size_t num);
};
```

Question 10 (1 point) *Donnez une implémentation de la fonction* ***`run`*** *du composant matériel* ***`DMA`****. On peut réutiliser* ***`copy_last_byte`*** *si besoin.*

```
void DMA::run(uint32_t addr, int value, size_t num) {
        uint count;
        assert(addr % 4 == 0);
        assert(num % 4 == 0);
        value = copy_last_byte(value);
        for (count = 0; count < num / 4; count++) {
                tlm::tlm_response_status status =
                        initiator.write(addr + count * 4, value);
                assert(status == tlm::TLM_OK_RESPONSE);
        }
}
```

On donne un squelette d'implémentation des fonctions `read` et `write` du modèle de composant `DMA` :

```
tlm::tlm_response_status
DMA::write(ensitlm::addr_t a, ensitlm::data_t d)
{
        switch (a) {
        case DMA_ADDR:
                // ...
                break;
        case DMA_VALUE:
                // ...
                break;
        case DMA_NUM:
                // ...
                break;
        case DMA_START:
                // ...
                break;
        default:
                SC_REPORT_ERROR(name(), "<some error message>");
                return tlm::TLM_ADDRESS_ERROR_RESPONSE;
        }
        return tlm::TLM_OK_RESPONSE;
}

tlm::tlm_response_status
DMA::read(ensitlm::addr_t a, ensitlm::data_t& d)
{
        switch (a) {
        case DMA_ADDR:
                d = m_addr;
                break;
        case DMA_VALUE:
                d = m_value;
                break;
        case DMA_NUM:
                d = m_num;
                break;
        case DMA_START:
                // ...
                break;
        default:
                SC_REPORT_ERROR(name(), "<some error message>");
                return tlm::TLM_ADDRESS_ERROR_RESPONSE;
        }
        return tlm::TLM_OK_RESPONSE;
}
```

Question 11 (1 point) *Pour chaque message d'erreur (i.e. les chaque occurrence de* `SC_REPORT_ERROR`*), préciser la cause de l'erreur (par exemple en proposant un message précis et adapté à la place de* `"<some␣error␣message">`*).*

Dans les deux cas, c'est un accès à une adresse qui ne correspond à aucun registre.

Question 12 (1 point) *Proposez une implémentation pour les cas* `DMA_ADDR`*,* `DMA_VALUE` *et* `DMA_NUM` *dans la méthode* `write`*.*

Question 13 (1 point) *Proposez une implémentation pour les cas* `DMA_START` *des méthodes* `read` *et* `write`*.*

```
... write(...) { ...
        case DMA_ADDR:
                m_addr = d;
                break;
        case DMA_VALUE:
                m_value = d;
                break;
        case DMA_NUM:
                m_num = d;
                break;
        case DMA_START:
                run(m_addr, m_value, m_num);
                break;

... read(...) { ...
        case DMA_START:
                d = 0;
                break;
```

Question 14 (1 point) *Que faut-il ajouter à la fonction* `sc_main` *?*

```
        Dma dma("dma");
        bus.initiator(dma.target);
        bus.target(dma.initiator);
        bus.map(dma.target,      DMA_BASEADDR,      DMA_SIZE);
```

Question 15 (0.5 point) *Est-il nécessaire de modifier la couche d'abstraction du matériel (*`hal.h`*), et pourquoi ? Si oui, quelles modifications faut-il faire ?*

Non, rien à changer. Les accès au DMA se font via `read_mem` et `write_mem`.

Question 16 (1 point) *Donnez une implémentation de la fonction* ***`hard_memset`*** *du logiciel, utilisant le composant matériel* ***`DMA`****.*

```
void hard_memset(uint32_t ptr, int value, size_t num) {
        write_mem(DMA_BASEADDR + DMA_ADDR, ptr);
        write_mem(DMA_BASEADDR + DMA_VALUE, value);
        write_mem(DMA_BASEADDR + DMA_NUM, num);
        write_mem(DMA_BASEADDR + DMA_START, 1);
}
```

4.2 Version avec gestion propre des interruptions

La version proposée ci-dessus a l'avantage d'être simple à modéliser, mais n'est pas réaliste du point de vue du vrai système : le calcul fait au moment de l'accès au registre `START` est supposé instantané. Pour pouvoir nous passer de cette hypothèse simplificatrice, le composant doit fournir un moyen de savoir quand le calcul est terminé, et le logiciel doit utiliser cette information pour attendre le résultat du calcul. La solution proposée est que le composant `DMA` va notifier le processeur via une interruption quand le calcul est terminé.

Question 17 (1 point) *Que faut-il ajouter à l'interface du composant* ***`DMA`*** *pour permettre ceci ? (ajouter des registres ? des ports (entrée ou sortie) ? des sockets (initiateur ou cible) ?*

Un port d'IRQ en sortie, et un registre `FINISHED` qui permet de savoir si l'opération est terminée.

Question 18 (2 points) *Pour adapter le comportement du composant, faut modifier la manière de gérer les transactions sur le registre* ***`START`****. Quelles modifications faudrait-il faire dans le composant* ***`DMA`*** *? Répondez au choix en écrivant du code ou en décrivant précisément les constructions SystemC à utiliser (quel processus fait quoi, quelle méthode appelle qui, ... ?).*

Il faut ajouter un processus SystemC qui exécute la méthode C++ `run` quand elle reçoit un évènement `start`. Le traitant de transaction de `write` pour `START` notifiera cet évènement pour réveiller le processus. Un `wait` suivra l'appel à `run` puis le signal d'IRQ est levé.

Question 19 (1 point) *À quel composant le signal d'interruption venant du composant* ***`DMA`*** *sera-t-il connecté ?*

Au contrôleur d'interruption (ou directement au CPU si on n'en a pas).

Question 20 (1 point) *Proposez une modification du logiciel embarqué qui permette une gestion correcte des interruptions.*

Il faut ajouter une attente d'interruption à la fin de la fonction memset :

```
while (!read_mem(DMA_BASEADDR + DMA_FINISHED)) {
        wait_for_irq();
}
```

Question 21 (2 points) *La fonction* `memset` *de la bibliothèque C est "thread-safe" (peut être appelée par deux fils d'exécutions concurrents). Est-ce le cas avec* `soft_memset` *? Et avec* `hard_memset` *? Si non, proposez une correction.*

`soft_memset` est thread-safe (seulement des variables locales), mais `hard_memset` ne l'est pas : si deux threads tentent d'accéder au même composant `DMA` en concurrence, ils vont accéder aux même registres. Il faudrait ajouter un mutex autour de memset par exemple.

26
26

5 Annexe : composant DMA

Le composant DMA est destiné à être connecté à un bus Xilinx On-chip Peripheral Bus (OPB). Il possède les entrées/sorties suivantes :

— Interface initiateur compatible OPB
— Interface cible compatible OPB

Fonctionnalités

Le port cible du DMA permet d'initialiser une zone mémoire avec une donnée particulière.

Fonctionnement interne

Lorsqu'une écriture est réalisée dans le registre `START`, le composant lance l'initialisation de la zone mémoire en recopiant la donnée (1 octet) contenue dans les bits de poids faible du registre `VALUE` sur les `NUM` octets suivant l'adresse spécifié dans le registre `ADDR`.

Récapitulatif des registres

Adresse relative	**Type**	**Nom**	**Description**
`0x00`	Lecture/écriture	`DMA_ADDR`	Adresse de départ
`0x04`	Lecture/écriture	`DMA_VALUE`	Valeur à recopier
`0x08`	Lecture/écriture	`DMA_NUM`	Nombre d'octets à initialiser
`0x0c`	Lecture/écriture	`DMA_START`	Démarrer l'initialisation

Registres `DMA_ADDR`, `DMA_VALUE`, `DMA_NUM`

Permettent de stocker les paramètres de l'initialisation. Seuls les 8 bits de poids faibles de `DMA_VALUE` sont utilisés pendant l'initialisation. Aucune vérification n'est faite sur ces valeurs : la cohérence est de la responsabilité du composant initiateur qui programme le DMA.

Registre `DMA_START`

Écrire dans ce registre lance l'initialisation de la zone mémoire. Lire dans le registre renvoie toujours 0.