Modélisation Transactionnelle des Systèmes sur Puces

Ensimag 3A, filière SLE

Février 2015

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 en C++

On considère le programme suivant :

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

struct Base {
        int a;
        Base () { a = 42; }
};

struct B : Base {
        int b;
        B () { a = 12; b = 4; }
};

struct C : Base {
        int b;
};

struct D : B, C {
        void f() {
                cout << "a␣=␣" << a << endl;
                cout << "b␣=␣" << b << endl;
        }
```

```
};

int main() {
        D d;
        d.f();
}
```

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 désambiguiser les accès à a et b dans D.

1.2 Templates en C++

On considère la classe template suivante, dans laquelle certains identificateurs ont été remplacés par `D1`, ... `D4` :

```
template<typename T, size_t S>
struct tableau_intelligent /* (enfin pas tant que ça) */ {
        D1 elements[D2];

        D3 get_elem(unsigned index) {
                assert(index < S);
                return elements[index];
        }

        D4 set_elem(unsigned index, T e) {
                assert(index < S);
                elements[index] = e;
        }
};
```

Question 2 (1 point) *Par quoi faut-il remplacer `D1`, `D2`, `D3` et `D4` pour que la déclaration soit proprement typée ?*

```
% 0.25 point chacun.
#define D1 T
#define D2 S
#define D3 T
#define D4 void
```

Question 3 (1 point) *Écrivez un programme principal qui instancie cette classe `tableau_intelligent` en un tableau de 12 flottants, et qui utilise les méthodes `get_elem()` et `set_elem()`.*

```
int main() {
        tableau_intelligent<float, 12> t;
        t.set_elem(41, 14.0);
        cout << t.get_elem(41) << endl;;
}
```

3.5

2 Questions de cours sur TLM et les SoCs

Question 4 (1 point) *Citez 3 nouvelles fonctionnalités des systèmes sur puces fabriqués par STMicroelectronics. Pour chacune, justifiez en une phrase leur intérêt.*

Support matériel pour la sécurité/gestion des droits numériques (demandé par les fournisseurs de contenu)
HEVC : meilleure compression
UHD : meilleur confort visuel pour l'utilisateur
64 bits : nécessaire pour gérer les grosses quantités de données (espace d'adressage plus grand)

Question 5 (1 point) *En quoi SystemC/TLM peut être utile pour la vérification de blocs (IP) matériel ?*

Pour l'environnement de test (intégration d'une IP dans une plateforme minimaliste), pour la génération des tests.

Question 6 (1 point) *Dans le flot complet d'un système sur puce, la même information se trouve à plusieurs endroits : dans le modèle TLM, dans l'implémentation RTL, et dans la documentation technique utilisée entre autres par les développeurs de logiciel. Quelles techniques peuvent être utilisées pour éviter d'avoir à ré-écrire la même chose plusieurs fois ?*

Génération de code à partir de la doc.
Synthèse de haut niveau.

3

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() {
                wait(1, SC_SEC);
                cout << "f1" << endl;
                wait(2, SC_SEC);
                cout << "f2" << endl;
                sleep(1);
                cout << "f3" << endl;
        }
        SC_CTOR(A) {
                SC_THREAD(f);
        }
};

SC_MODULE(B) {
        void g() {
                cout << "g0" << endl;
                sleep(2);
                cout << "g1" << endl;
                wait(4, SC_SEC);
                cout << "g2" << endl;
        }
        SC_CTOR(B) {
                SC_THREAD(g);
        }
};

int sc_main(int, char**)
{
        A a("a");
        B b1("b1");
        B b2("b2");

        sc_start();

        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.

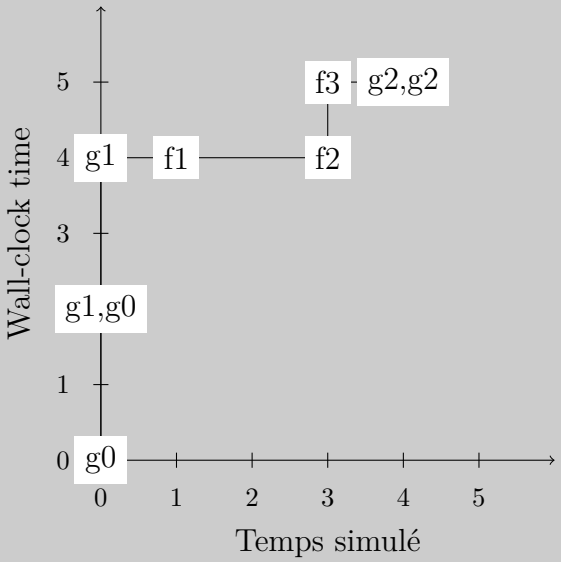
Question 7 (1.5 points) *Combien y a-t-il d'exécution différentes (i.e., produisant des affichages différents) autorisées par SystemC ? Donnez cette ou ces traces d'exécution.*

Au lancement de la simulation, b1.g et b2.g sont éligibles. a.f est aussi éligible mais ne fait rien d'autre qu'entrer dans un wait, donc a.f n'a aucun effet sur l'affichage.

b1.g et b2.g s'exécutent l'un après l'autre sans rendre la main jusqu'au wait. On arrive ensuite au temps simulé t=1, on exécute f1, puis t=3, on exécute f2, puis t=4 on exécute g4 deux fois. Les seuls choix possibles sont entre b1.g et b2.g qui font la même chose, il n'y a qu'un affichage possible.

```
g0
g1
g0
g1
f1
f2
f3
g2
g2
```

Question 8 (2.5 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. Si on néglige le temps de calcul, combien de temps prendra une exécution en "wall-clock time" ? Combien de temps en temps simulé ?*



Dans les 2 cas, 4 secondes de temps simulé pour une exécution de 5 secondes en wall-clock time.

4 Modélisation d'un composant matériel pour implémenter des mutex rapides

L'implémentation classique d'un verrou d'exclusion mutuelle utilise des opérations atomiques (comme test-and-set) sur la mémoire RAM. Il est aussi possible d'accélérer les choses avec des composants matériels gérant directement ces opérations atomiques.

On propose dans cette partie de modéliser un composant "test and set" qui permette de gérer en matériel 128 verrous. Du point de vue logiciel, les opérations se font via des lectures et écritures normales.

Nous utiliserons les mêmes conventions que dans le TP3, mais on suppose que la plateforme a été étendue à plusieurs processeurs.

Le but de cette partie est d'implémenter les fonctions suivantes (pour utilisation dans le logiciel embarqué) :

```
// Verrouille le mutex numéro n (avec n >= 0 et n < 128).
// Si le mutex est déjà verrouillé, boucle tant que le mutex
// n'est pas relaché.
void hw_mutex_lock(int n);

// Relache (déverrouille) le mutex numéro n (avec n >= 0 et n < 128).
void hw_mutex_unlock(int n);

// Tente un verrouillage du mutex numéro n, et renvoie 1 si le
// verrouillage réussit. En cas d'échec, abandonne immédiatement et
// renvoie 0.
int hw_mutex_trylock(int n);
```

Voici un exemple d'utilisation de ces fonctions :

```
hw_mutex_lock(0);
// ... section critique
hw_mutex_unlock(0);
```

Le composant "test and set" est un composant cible sur le bus.

Chaque mutex est un registre dans le composant "test and set". Les registres n'ont que deux valeurs possibles : 0 et 1. 3 opérations différentes sont disponibles sur chaque mutex :

1. Écrire une valeur (0 ou 1)
2. Lire une valeur (0 ou 1)
3. Le test-and-set a proprement parler :
   - si le registre correspondant au mutex contient la valeur 0, alors on écrit 1 dans le registre et l'opération renvoie 1.
   - Si le registre correspondant au mutex contient la valeur 1, alors la valeur n'est pas modifiée et l'opération renvoie 0.

Les opérations 1 et 2 se font via les opérations read et write du bus : pour accéder au mutex n, on fait une lecture ou une écriture à l'offset $n * 4$ sur le composant test and set. L'opération 3 se fait via une opération read : lire à l'offset $n * 4 + 128 * 4$ provoque un test-and-set sur le mutex n. La valeur lue est le résultat de l'opération (0 ou 1).

Écrire à un offset supérieur à $128 * 4$ ou lire à une adresse supérieure à $128 * 4 * 2$ est interdit.

Initialement, tous les registres du composant contiennent la valeur 0.

Le composant est mappé à l'adresse 0x40700000 dans la plateforme. La constante suivante est définie :

```
#define TESTANDSET_BASEADDR 0x40700000
```

4.1 Logiciel embarqué

Question 9 (2 points) *En utilisant l'API d'abstraction du matériel du TP3 (hal.h, vu en cours), proposez une implémentation pour les fonctions* `hw_mutex_lock(int n)`, `hw_mutex_unlock(int n)` *et* `hw_mutex_trylock(int n)`.

```
int hw_mutex_trylock(int n) {
        return read_mem(TESTANDSET_BASEADDR + (4*128) + 4*n);
}

void hw_mutex_lock(int n) {
        while (!hw_mutex_trylock(n)) {
                cpu_relax();
        }
}
```

```
void hw_mutex_unlock(int n) {
        write_mem(TESTANDSET_BASEADDR + 4*n, 0);
}
```

4.2 Modélisation du matériel

Un point clé pour le fonctionnement du composant "test and set" est l'atomicité : si plusieurs modules tentent d'accéder au composant en même temps, les accès ne sont jamais entrelacés.

Question 10 (0.5 point) *Dans l'implémentation matérielle, quel mécanisme permet de garantir cette atomicité ?*

L'arbitrage sur le bus qui fait que deux initiateurs ne peuvent accéder au même port cible en même temps.

Question 11 (0.5 point) *Dans le modèle SystemC/TLM, quel mécanisme permet de garantir cette atomicité ?*

La sémantique de co-routine de SystemC.

On propose de squelette de code suivant :

```
// Fichier testandset.h
#define TESTANDSET_NBSLOTS 128
#define TESTANDSET_ADDRSIZE (TESTANDSET_NBSLOTS * sizeof(ensitlm::data_t))

class TESTANDSET : public sc_core::sc_module {
public:
        ensitlm::target_socket<TESTANDSET> target;

        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);
};

// Fichier testandset.cpp
tlm::tlm_response_status
TESTANDSET::write(ensitlm::addr_t a, ensitlm::data_t d)
{
        if (a >= TESTANDSET_ADDRSIZE) {
                return tlm::TLM_ADDRESS_ERROR_RESPONSE;
```

```
        }
        // ...
        return tlm::TLM_OK_RESPONSE;
}

tlm::tlm_response_status
TESTANDSET::read(ensitlm::addr_t a, ensitlm::data_t& d)
{
        // ...
        return tlm::TLM_OK_RESPONSE;
}
```

Question 12 (0.5 point) *À quoi sert le test* `if (a >= TESTANDSET_ADDRSIZE)` *en début de méthode* `write` *?*

Question 13 (1 point) *Faut-il ajouter quelque chose à la déclaration de classe dans* `testandset.h` *pour implémenter le comportement du composant dans* `read` *et* `write` *? Si oui, écrivez le code à ajouter.*

```
        SC_CTOR(TESTANDSET) {
            memset(locks, 0, sizeof(TESTANDSET::locks));
        }
private:
        bool locks[TESTANDSET_NBSLOTS];
```

Question 14 (1 point) *Proposez une implémentation pour la partie manquante de la méthode* `write`*.*

```
        locks[a / 4] = (d == 1); // Attention au /4.
```

Question 15 (2.5 points) *Proposez une implémentation pour la partie manquante de la méthode* `read`*.*

```
        if (a >= TESTANDSET_ADDRSIZE) {
                a = a - TESTANDSET_ADDRSIZE;
                if (a >= TESTANDSET_ADDRSIZE) {
                        return tlm::TLM_ADDRESS_ERROR_RESPONSE;
                }
                // Test and set
                if (locks[a / 4]) {
                        // Failure, already set.
```

```
                        d = 0;
                } else {
                        locks[a / 4] = true;
                        d = 1;
                }
        } else {
                // Read
                d = locks[a / 4];
        }
```

Question 16 (1 point) *Faut-il ajouter quelque chose à la fonction* `sc_main` *? Si oui, écrivez le code à ajouter.*

```
TESTANDSET testandset("testandset");

bus.initiator(testandset.target);

bus.map(testandset.target, TESTANDSET_BASEADDR, 128*2*4);
```

4.3 Extension possible

Notre implémentation du mutex en utilisant le composant "test and set" est assez inefficace dans le cas où un processus tente d'accéder à un mutex déjà verrouillé et que l'attente est longue (i.e. si les sections critiques sont longues).

Question 17 (1 point) *Quels problèmes de performances sont posés par ce cas ?*

Attente active, donc utilisation du processeur, et surcharge du bus et du composant testandset pendant toute la période d'attente. Les autres processus auraient pu utiliser le CPU, mais même les autres composants sont ralentis par l'occupation du bus.

Question 18 (1 point) *Quelle(s) solution(s) proposez-vous pour améliorer les performances dans ce cas (pour la plateforme du TP3 et/ou pour une plateforme plus complexe) ? Il n'est pas demandé de les implémenter.*

fall-back sur un autre mécanisme (e.g. appel système si on a un OS) après n tour de boucles.

exponential backoff côté logiciel : si on voit que le verrou n'est pas disponible, on attend un peu, en multipliant par 2 la période d'attente à chaque tour de boucle.

gestion des priorités entre process s'il y a du multi-tâche : on rend la main à chaque tour de boucle.

mécanisme à base d'interruption, ou déverrouiller un mutex enverrai une interruption au CPU qui en a fait la demande.

15
21.5