Modélisation Transactionnelle des Systèmes sur Puces

Ensimag 3A, filière SEOC

Année scolaire 2018-2019

Consignes :

- durée : 2h ;
- aucun document autorisé hormis une feuille A4 manuscrite recto-verso ;
- 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 Question de cours sur C++

1.1 Héritage en C++

On considère le programme suivant :

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

struct Base {
        const char * nv() { return "A"; }
        virtual const char * v() { return "B"; }
};

struct Derived : public Base {
        const char * nv() { return "C"; }
        virtual const char * v() { return "D"; }
};

int main() {
        Derived d;
```

```
        cout << "d.nv() = " << d.nv() << endl;
        cout << "d.v() = " << d.v() << endl;

        Base *pD = &d;
        cout << "pD->nv() = " << pD->nv() << endl;
        cout << "pD->v() = " << pD->v() << endl;

        Base b = d;
        cout << "b.nv() = " << b.nv() << endl;
        cout << "b.v() = " << b.v() << endl;
}
```

Question 1 (1.5 points) *Qu'affiche ce programme ? Expliquez brièvement chaque ligne de la sortie.*

```
d.nv() = C
d.v() = D
pD->nv() = A
pD->v() = D
b.nv() = A
b.v() = B
```

1.2 Méthodes virtuelles en C++

On considère le programme (incorrect) suivant :

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

struct Base {
        virtual void f() = 0;
};

struct D1 : public Base {
        virtual void g() { cout << "appel de g()" << endl; }
};

struct D2 : public Base {
        virtual void f() { cout << "appel de f()" << endl; }
};

int main() {
        D1 d1;
```

```
        D2 d2;
}
```

Question 2 (1.5 points) *Quelle(s) ligne(s) de ce programme lève(nt) une erreur ? Indiquez la cause de l'erreur.*

```
g++ pure-virtual.cpp -o pure-virtual -Wall -Wextra
pure-virtual.cpp: In function ‘int main()’:
pure-virtual.cpp:17:12: error: cannot declare variable ‘d1’ to be of abstract type ‘D1’
   17 |         D1 d1;
      |            ^~
pure-virtual.cpp:8:8: note:   because the following virtual functions are pure within ‘D1’:
    8 | struct D1 : public Base {
      |        ^~
pure-virtual.cpp:5:22: note:     ‘virtual void Base::f()’
    5 |         virtual void f() = 0;
      |                      ^
make[2]: *** [Makefile:4 : pure-virtual] Erreur 1
```

```
   3
```

2 Modélisation du temps et ordonnancement en SystemC

On considère le programme suivant :

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

using namespace std;
using namespace sc_core;

SC_MODULE(A) {
        void f() {
                cout << "f0" << endl;
                sleep(1);
                cout << "f1" << endl;
                wait(2, SC_SEC);
                cout << "f2" << endl;
        }
        void g() {
                cout << "g0" << endl;
                wait(1, SC_SEC);
```

```
                cout << "g1" << endl;
                sleep(2);
                cout << "g2" << endl;
        }
        SC_CTOR(A) {
                SC_THREAD(g);
                SC_THREAD(f);
        }
};

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

        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 deux exécutions possibles de ce programme (*i.e.* l'ordonnanceur a la liberté de choisir entre ces deux exécutions).

Question 3 (1 point) *Expliquez le choix de l'ordonnnanceur SystemC, et donnez les deux exécutions possibles.*

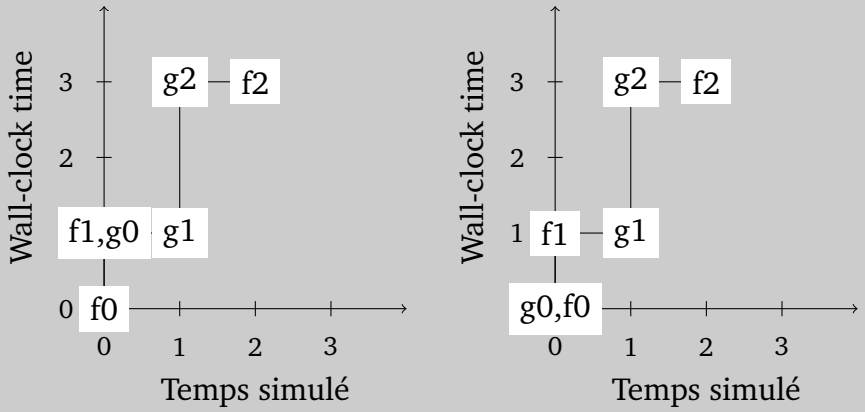
Au lancement de la simulation, f et g sont tous deux éligibles. L'ordonnanceur peut les exécuter dans n'importe quel ordre. Les deux exécutions suivantes sont possibles :

```
f0
f1
g0
g1
g2
f2
```

et

```
g0
f0
f1
g1
g2
f2
```

Question 4 (2 points) *Représentez l'une des deux exécutions sur un graphique à deux dimensions. On mettra le "wall-clock time" sur l'axe des abscisses et le temps simulé sur les ordonnées. 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, 2 secondes de temps simulé pour une exécution de 3 secondes en wall-clock time.

On considère maintenant le programme suivant :

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

using namespace std;
using namespace sc_core;

SC_MODULE(B) {
        sc_event e;
        void f() {
                wait(e);
                cout << "f" << endl;
                e.notify();
        }
        void g() {
                cout << "g0" << endl;
                wait(1, SC_SEC);
                cout << "g1" << endl;
                e.notify();
                wait(1, SC_SEC);
                cout << "g2" << endl;
                wait(e);
                cout << "g3" << endl;
        }
        SC_CTOR(B) {
```

```
                SC_THREAD(g);
                SC_THREAD(f);
        }
};

int sc_main(int, char**)
{
        B b("b");

        sc_start();

        return 0;
}
```

Question 5 (1.5 points) *Donnez une exécution possible de ce programme. Est-ce la seule ? Qu'est-ce qui provoque la fin de la simulation sur cet exemple ? Comment aurait-on pu éviter cela ?*

4.5

3 Implantation d'un accélérateur matériel

Dans cette partie, nous allons ajouter un accélérateur matériel à la plate-forme du TP3 sur laquelle le logiciel exécute le « jeu de la vie ». Une opération essentielle de ce programme est de compter le nombre de voisins vivants d'une case, c'est-à-dire le nombre de pixels blancs autour d'un pixel donné. C'est cette opération qui va être accélérée en matériel. Dans la vraie vie, il est peu probable qu'un tel accélérateur matériel soit jamais réalisé, mais nous nous servirons de cet exemple simpliste pour concevoir une manière de déléguer du calcul depuis le logiciel vers du matériel dédié.

L'implantation logicielle du jeu de la vie repose sur la fonction `generation()`, qui applique une itération du jeu sur l'ensemble de l'image :

```
void generation()
{
    int x, y;
    for(x = 0; x < SW_VGA_WIDTH; ++x) {
        for(y = 0; y < SW_VGA_HEIGHT; ++y) {
            int n = neighbors(old_img_addr, x, y);
            // Application des règles utilisant n
            // ...
        }
    }
}
```

La fonction `neighbors` est définie comme suit :

```
// Calcule le nombre de voisins du pixel situé aux coordonnées
// (x, y) dans l'image située à l'adresse img_addr. Renvoie un entier
// entre 0 et 8 inclus.
int neighbors_hard(uint32_t img_addr, int x, int y);
```

Le but de cette partie est de modéliser un accélérateur matériel (composant `Neighbors`), et de l'utiliser dans le logiciel pour faire une implantation accélérée de la fonction `neighbors`. Une spécification partielle du composant `Neighbors` est fournie en annexe.

3.1 Version simplifiée sans interruptions

Dans cette partie, on fait l'hypothèse simplificatrice que le calcul du nombre de voisins est fait en temps nul par le composant `Neighbors`. Cette hypothèse peut être utilisée dans le modèle TLM du composant et dans le logiciel embarqué.

Question 6 (1 point) *Expliquez les rôles respectifs des sockets initiateur et cible du composant `Neighbors`.*

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

```
class Neighbors : public sc_core::sc_module {
public:
        ensitlm::target_socket<Neighbors> target;
        ensitlm::initiator_socket<Neighbors> 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(Neighbors) : m_neighbors(0xbad) {/* */};
private:
        uint32_t m_x, m_y, m_img_addr, m_neighbors;
        uint32_t neighbors(uint32_t img_addr, uint32_t x, uint32_t y);
};
```

On suppose que la fonction `neighbors` est déjà implantée (elle fait les accès mémoires nécessaires et renvoie le nombre de voisins du pixel défini par `img_addr`, `x` et `y`).

On donne un squelette d'implantation des fonctions `read` et `write`:

```
tlm::tlm_response_status
Neighbors::write(ensitlm::addr_t a, ensitlm::data_t d)
{
```

```
        switch (a) {
        case NEIGHBORS_X:
                m_x = d;
                break;
        case NEIGHBORS_Y:
                m_y = d;
                break;
        case NEIGHBORS_IMG_ADDR:
                m_img_addr = d;
                break;
        case NEIGHBORS_START:
                // ...
                break;
        case NEIGHBORS_GET_NEIGHBORS:
                SC_REPORT_ERROR(name(), "<some error message>");
                return tlm::TLM_COMMAND_ERROR_RESPONSE;
        default:
                SC_REPORT_ERROR(name(), "<some error message>");
                return tlm::TLM_ADDRESS_ERROR_RESPONSE;
        }
        return tlm::TLM_OK_RESPONSE;
}

tlm::tlm_response_status
Neighbors::read(ensitlm::addr_t a, ensitlm::data_t& d)
{
        switch (a) {
        case NEIGHBORS_X:
                // ...
                break;
        case NEIGHBORS_Y:
                // ...
                break;
        case NEIGHBORS_IMG_ADDR:
                // ...
                break;
        case NEIGHBORS_GET_NEIGHBORS:
                // ...
                break;
        default:
                SC_REPORT_ERROR(name(), "<some error messages>");
                return tlm::TLM_ADDRESS_ERROR_RESPONSE;
        }
        return tlm::TLM_OK_RESPONSE;
}
```

Question 7 (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">).

Question 8 (1 point) *Proposez une implantation pour les cas NEIGHBORS_X, NEIGHBORS_Y et NEIGHBORS_IMG_ADDR dans la méthode read.*

Question 9 (1 point) *Proposez une implantation pour les cas NEIGHBORS_GET_NEIGHBORS des méthodes read et write.*

Question 10 (1 point) *Que faut-il ajouter à la fonction sc_main ?*

Question 11 (1 point) *Est-il nécessaire de modifier la couche d'abstraction du matériel (hal.h), et pourquoi ? Si oui, quelles modifications faut-il faire ?*

Question 12 (1 point) *Donnez une implantation de la fonction neighbors utilisant le composant matériel Neighbors.*

3.2 Version avec gestion des interruptions

La version proposée ci-dessus a l'avantage d'être simple à modéliser, mais n'est pas très 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 Neighbors va notifier le processeur via une interruption quand le calcul est terminé.

Question 13 (1 point) *Que faut-il ajouter à l'interface du composant Neighbors pour permettre ceci ? (ajouter des registres ? des ports (entrée ou sortie) ? des sockets (initiateur ou cible) ?*

Question 14 (1 point) *Pour adapter le comportement du composant, il suffit d'une ligne de code à ajouter dans la bonne méthode. Que faut-il ajouter, et à quel endroit ?*

Question 15 (1 point) *À quel composant le signal d'interruption venant du composant Neighbors sera-t-il connecté ?*

Question 16 (1 point) *Proposez une modification du logiciel embarqué qui permette une gestion correcte des interruptions.*

Question 17 (1 point) *La version simplifiée développée dans la section 3.1 fait une hypothèse simplificatrice à la fois sur le modèle du matériel et sur l'implantation du logiciel. Que risque-t-il de se passer si on fait tourner le logiciel développé avec cette méthode sur le vrai système ? Pourquoi ?*

Question 18 (1.5 points) *Quel type de tests faudrait-il faire pour avoir la garantie que le logiciel développé sur la plate-forme TLM marche effectivement sur le système réel quel que soit le temps mis par le composant matériel pour effectuer le calcul ?*

13.5
21

Annexe

"Neighbors" Hardware IP Data-Sheet ACME Inc *

Le composant "Neighbors" est un module à la fois initiateur et cible destiné à être connecté au bus système.

Fonctionnalités

Le composant "Neighbors" permet de calculer le nombre de voisins, c'est-à-dire de pixels blancs, d'un pixel donné. Pour faire ce calcul, le composant a besoin des coordonnées du pixel (`x` et `y`), et de l'adresse de base de l'image (`img_addr`). Le résultat du calcul est obtenu en faisant une lecture sur un registre du composant (`GET_NEIGHBORS`).

Connexion au bus système

Le composant `Neighbors` possède les entrées/sorties suivantes :

- Interface initiateur compatible bus système
- Interface cible compatible bus système

Fonctionnement interne

Lorsqu'une écriture est réalisée dans le registre `START`, le composant calcule le nombre de voisins du pixel aux coordonnées spécifiées par les registres `X`, `Y` (pour ses coordonnées) et `IMG_ADDR` (pour l'adresse du premier pixel de l'image). Le résultat du calcul est mis à disposition dans le registre `GET_NEIGHBORS`. Si une lecture est faite sur `GET_NEIGHBORS` après l'écriture sur `START`, mais avant que le calcul ne soit terminé, la lecture renverra la valeur `0xbad`.

*Traduction française SGDG.

Registres et décalages associés (*register map*)

Adresse relative	Type	Nom	Description
`0x00`	Lecture / Écriture	`X`	Abscisse du pixel
`0x04`	Lecture / Écriture	`Y`	Ordonnée du pixel
`0x08`	Lecture / Écriture	`IMG_ADDR`	Adresse de départ de l'image
`0x0c`	Lecture / Écriture	`START`	Démarrage du calcul
`0x10`	Lecture seule	`GET_NEIGHBORS`	Récupération du résultat

Valeurs des registres `X`, `Y` et `IMG_ADDR`

Ces 3 registres sont des registres généraux, les valeurs contenues sont utilisées par les calculs internes, mais les accès à ces registres ne déclenchent pas d'action dans le composant.

Valeurs du registre `START`

Une écriture dans ce registre positionne immédiatement la valeur de `GET_NEIGHBORS` à `0xbad`, et démarre le calcul du nombre de voisins.

Valeurs du registre `GET_NEIGHBORS`

Ce registre contient le résultat du dernier calcul, ou bien `0xbad` si un calcul est en cours.

Gestion des interruptions

Cette section est volontairement omise de la documentation. Vous pouvez l'ignorer dans la partie « Version simplifiée sans interruptions », et vous devrez vous en passer pour la partie « Version avec gestion des interruptions ».