Modélisation Transactionnelle des Systèmes sur Puces

Ensimag 3A, filière SLE

Frédéric Pétrot/Julie Dumas — 7 Février 2018

Consignes :

— Durée : 2h,

— Document autorisé : une feuille A4 manuscrite recto-verso seulement,

1 Questions 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 *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 suivant :

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

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

struct A : public F {
  virtual void g() { cout << "appel␣de␣g()" << endl; }
};

struct B : public F {
  virtual void f() { cout << "appel␣de␣f()" << endl; }
};

int main() {
  A a;
  B b;
}
```

Question 2 *Ce programme ne compile pas. Indiquez quelle(s) ligne(s) cause(nt) une erreur et donnez-en la cause.*

```
g++ pure-virtual.cpp -o pure-virtual -Wall -Wextra
pure-virtual.cpp: In function ‘int main()’:
pure-virtual.cpp:17:4: error: cannot declare variable ‘a’ to be of abstract type ‘A’
  A a;
    ^
pure-virtual.cpp:8:8: note:   because the following virtual functions are pure within ‘A’:
 struct A : public F {
        ^
pure-virtual.cpp:5:15: note:  virtual void F::f()
  virtual void f() = 0;
               ^
make[1]: *** [Makefile:4: pure-virtual] Error 1
```

1.3 Macros

On considère le programme C++ ci-dessous :

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

#define FOO(x) \
   do {        \
      f(x+x);  \
   } while (0)

#define BAR(x) \
   {           \
      f(x+x);  \
   }

void f(int x)
{
   (volatile int)x;
}

int main() {
   int a = random()%2; // 0 ou 1
   if (a)
      BAR(a+1);
   else
      FOO(a++);

   return printf("%d\n", a);
}
```

On ne s'intéresse pas à la fonction f.

Question 3 *Ce programme ne compile pas, expliquer pourquoi.*

```
int main() {
  int a = random() % 2;
  if (a) {
    f(a + 1 + a + 1);
  }; else
    do {
      f(a++ + a++);
    } while (0);

  return printf("%d\n", a);
}
```

Le point-virgule derrière l'accolade fait qu'on ne peut avoir de else!

Question 4 *Si l'on inverse les lignes FOO et BAR, il compile, expliquer pourquoi. Il reste néanmoins un problème dont gcc vous informera par un warning. Identifiez-le.*

Le point-virgule fini un "statement", derrière lequel un `return` peut se trouver. Par contre l'expansion de la macro FOO produit un a++ + a++ qui est ambigu, car le moment auquel le ++ est appliqué (après la première ou la seconde lecture de a) n'est pas défini par le standard et pourra donner des résultats différents suivant les implantations. Bref, évitez les instructions à effet de bord comme paramètres de macros!

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) {
  sc_event e;
  void f() {
    cout << "f1" << endl;
    e.notify();
    wait(e);
    cout << "f2" << endl;
    sleep(3);
    e.notify();
    wait(1, SC_SEC);
    cout << "f3" << endl;
    e.notify();
    wait(e);
  }
  void g() {
    cout << "g1" << endl;
    e.notify();
    wait(e);
    cout << "g2" << endl;
    e.notify();
    sleep(1);
    wait(e);
    wait(3, SC_SEC);
    cout << "g3" << endl;
    wait(e);
    e.notify();
  }
  SC_CTOR(A) {
    SC_THREAD(f);
    SC_THREAD(g);
  }
};
```

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

Question 5 *Combien y a-t-il d'exécution différentes (i.e., produisant des affichages différents) autorisées par SystemC ? Donnez ces traces d'exécution.*

Initialement, les deux processus sont éligibles, et peuvent donc s'exécuter dans n'importe quel ordre. On a donc :

```
f1  et  g1
g1      f1
f2      g2
g2      f2
f3      f3
g3      g3
```

Question 6 *Représentez toutes les 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 `f1`, `f2`, `f3`, `g1`, ..., `g4`. Si on néglige le temps de calcul, combien de temps prendra chaque exécution en "wall-clock time" ? Combien de temps en temps simulé ?*

1er cas : 3s après f2, 1s après g2 2nd cas : 3s après g2, 1s après f2

3 Implantation d'un accélérateur matériel

Dans cette partie, nous allons ajouter un accélérateur matériel à la plate-forme du TP2 (affichage d'une image en niveau de gris). L'objectif (pas très crédible) de cet accélérateur est de recopier l'image en place (not comment sur l'utilité) en en calculant si demandé le « négatif ». Pour ce faire, l'accélérateur, qui est un maître/esclave, va lire chaque pixel de l'image un par un, en prendre le complément, et de le réécrire à sa place.

En pseudo code, ça donne ceci, en supposant que `negate` est un booléen :

```
for (int y = 0; y < HEIGHT * WIDTH / 4; y++) {
   read(a, rvalue);
   wvalue = negate ? ~rvalue : rvalue;
   write(a, wvalue);
   a += 4;
}
```

Nous allons maintenant modéliser un composant matériel qui implante ce (délicat) algorithme. Le composant se nomme `Negatif`, possède comme adresse de base sur le bus l'adresse `NEGA_BASEADDR` (macro dont on dispose n'importe où) et contient un unique registre, `negate`, à l'*offset* 0, qui est accessible en écriture seulement ! Écrire un 0 (zéro) dans ce registre fera une copie directe, alors qu'un 1 (un) provoquera la copie avec calcul du négatif. C'est l'action d'écrire qui provoque le lancement de la copie, i.e., pas d'écriture dans ce registre, pas d'action du composant.

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

```
struct Negatif : sc_core::sc_module {
  ensitlm::initiator_socket<Negatif> initiator_socket;
  ensitlm::target_socket<Negatif> target_socket;
  sc_core::sc_event go_go_go;
  SC_CTOR(Negatif);
  void compute();
  tlm::tlm_response_status write(ensitlm::addr_t a, ensitlm::data_t d) {
    switch (a) {
    case ZERO :
      ...
      return tlm::TLM_OK_RESPONSE;
    default:
      SC_REPORT_ERROR(name(), "write␣register␣not␣implemented");
      return tlm::TLM_ADDRESS_ERROR_RESPONSE;
    }
  }

  tlm::tlm_response_status read(ensitlm::addr_t a, ensitlm::data_t& d) {
    ...
  }
  bool negate;
};
```

Question 7 *Expliquez les rôles respectifs des sockets initiateur et cible du composant `Negatif`.*

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

Question 8 *À quoi sert selon vous l'événement `go_go_go` ? Écrivez le code correspondant au `case ZERO` du `switch`.*

L'événement `go_go_go` est utilisé pour lancer le calcul du négatif de l'image.

```
case 0:
  negate = d;
  go_go_go.notify();
  return tlm::TLM_OK_RESPONSE;
```

Nous allons maintenant implanter la méthode `compute` du composant. On en donne une ébauche ci-dessous, dans laquelle `VRAM_BASE` est l'adresse de base de la mémoire vidéo :

```
void Negatif::compute() {
  while (true) {
    ensitlm::addr_t a = VRAM_BASE;
    ensitlm::data_t rvalue, wvalue;
    /* TODO: ... */

    for (int y = 0; y < HEIGHT * WIDTH / 4; y++) {
      /* TODO: negate or not negate ! */
      ...
      a += 4;
    }

    cout << name() << ": finished image" << endl;
  }
}
```

Question 9 *Le calcul effectué étant celui indiqué au début de l'exercice, proposez une encapsulation SystemC qui respecte les spécifications énoncées sur le lancement d'une copie et l'éventuel calcul du négatif lors de celle ci.*

```
void Negatif::compute() {
  ensitlm::addr_t a = VRAM_BASE;
  ensitlm::data_t rvalue, wvalue;
  while (true) {
    wait(go_go_go);

    for (int y = 0; y < HEIGHT * WIDTH / 4; y++) {
      initiator_socket.read(a, rvalue);
      wvalue = Negatif::negate ? ~rvalue : rvalue;
      initiator_socket.write(a, wvalue);
      a += 4;
    }

    cout << name() << ": finished image" << endl;
  }
}
```

Question 10 *Ajoutez à la fonction `sc_main` le nécessaire à l'instanciation de ce composant.*

```
  Negatif negatif("Negatif");
  negatif.initiator_socket.bind(bus.target);
  bus.initiator.bind(negatif.target_socket);
  bus.map(negatif.target_socket, NEGA_BASE,   4);
```

On donne ci-dessous le corps de la fonction `generator` telle que définie dans le TP2. La fonction `test`, donnée, produit l'image en RAM vidéo, tout comme dans le TP.

```
void Generator::compute() {
  socket.write(LCDC_BASE + LCDC_ADDR_REG, VRAM_BASE);
  socket.write(LCDC_BASE + LCDC_START_REG, 0x000000001);

  while (true) {
    if (!interrupt)
      wait(interrupt_event);
    interrupt = false;

    socket.write(LCDC_BASE + LCDC_INT_REG, 0x00000000);
    test(0); // Transfère l'image en RAM Vidéo
  }
}
```

Question 11 *Que faut-il ajouter dans ce code pour qu'une copie ait lieu à chaque nouveau transfère de l'image, et que 64 images affichées soient « directes » puis 64 « negatives », et cela continuement ? Astuce : jouez avec les bits !*

```
  uint32_t i = 0;
  while (true) {
    if (!interrupt)
      wait(interrupt_event);
    interrupt = false;

    socket.write(LCDC_BASE + LCDC_INT_REG, 0x00000000);
    test(0); // Transfère l'image en RAM Vidéo
    socket.write(NEGA_BASE + 0, i++ & 64);
  }
```

Total partie 3.1 : 0