Modélisation Transactionnelle des Systèmes sur Puces

Ensimag 3A, filière SLE

Janvier 2012

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 Question de cours

Question 1 (1 point) *Qu'est-ce qu'un socket dans la bibliothèque TLM 2 ? En quoi est-ce différent d'un* `sc_port` *?*

Un socket est un couple `sc_port` + `sc_export`, qui permet d'envoyer et de recevoir des transactions.

Question 2 (1 point) *On considère les trois fonctions C++ suivantes :*

```
void f1(      string   s) { cout << "Contenu de s : " << s << endl; }
void f2(const string   s) { cout << "Contenu de s : " << s << endl; }
void f3(const string & s) { cout << "Contenu de s : " << s << endl; }
```

Quelle version sera la plus efficace ? Quelle version est recommandée en C++ ? Pourquoi ?

Le passage par référence est le plus efficace. Le `const &` est la forme recommandée en C++, car elle est à la fois efficace et sûre.

Question 3 (1 point) *Le morceau de code suivant lève une erreur sur l'affectation* `m_x = x` *(`assignment of read-only data-member 'A::m_x'`) :*

```
class A {
        A(int x) {
                m_x = x;
        }
        const int m_x;
};
```

Proposez une alternative pour initialiser `m_x` *(qui doit rester constant).*

On peut utiliser le chaînage de constructeurs :

```
A(int x) : m_x(x) {};
```

Question 4 (1.5 points) *La plateforme du TP3 permettait d'exécuter le "jeu de la vie". En simulation, l'affichage se passait correctement, pourtant sur la plateforme FPGA, vous avez pu remarquer des problèmes. Quels étaient ces problèmes et à quel moment se produisaient-ils ? Pourquoi ne les a-t-on pas vu en simulation ? Que faudrait-il faire pour corriger le problème sur la plateforme physique ? Que faudrait-il faire pour voir ces problèmes en simulation ?*

L'affichage scintille pendant le calcul. Le problème est qu'il y a des conflits sur le bus, et le controleur VGA n'arrive pas à lire les pixels à afficher à temps. On ne les a pas vus en simulation car le bus n'est pas timé. Il aurait fallu modéliser un OPB et sa politique d'arbitrage précisément (TLM-AT ou CABA).

Question 5 (1 point) *Dans la plateforme du TP2 (affichage d'une image contenue en ROM sur un LCD), vous deviez mettre en place une gestion d'interruption. À quoi sert cette interruption sur la puce ? Comment l'avez-vous mise en œuvre sur la plateforme TLM ?*

L'interruption sert au LCDC à signaler qu'une image vient d'être affichée. La mise en œuvre est faite avec une `SC_METHOD` sensible aux fronts montant du signal d'interruption (`sc_signal`).

Question 6 (1.5 points) *Expliquez brièvement l'intérêt de mesurer la consommation d'énergie d'une puce au niveau TLM. Quelles sont les techniques utilisables ?*

Les puces modernes ont une politique de gestion de l'énergie fortement dépendante du logiciel embarqué. Il est important de fournir au développeur logiciel des outils pour mesurer la consommation d'une exécution du logiciel. On peut le mettre en œuvre avec des annotations dans le code TLM, en précisant l'état d'énergie dans lequel se trouve un composant à chaque instant.

2 Ajout d'une fonctionnalité au programme du TP3

On cherche à diminuer la consommation électrique de la plateforme du TP3. On suppose que la plateforme dispose déjà d'un moyen efficace d'attendre une interruption

(i.e. une attente d'interruption consomme moins d'énergie qu'un calcul). Quand le calcul d'une image est terminé, le processeur va attendre la prochaine interruption du timer pour faire le prochain calcul. Malheureusement, le contrôleur VGA va tout de même réveiller le processeur à chaque image affichée, pour lui faire exécuter le code suivant :

```
void vga_isr() {
        if(refresh) {
                refresh = 0;
                printf("vga_isr: double buffer flip\r\n");
                write_mem(VGA_BASEADDR + VGA_CFG_OFFSET, old_img_addr);
        }
}
```

Question 7 (1 point) *Expliquez ce que fait ce morceau de code, et à quoi il sert.*

Ce morceau de code implémente le double-buffer. Si refresh est à vrai (ce qui est le cas après que le timer a expiré), il fait pointer le controleur VGA sur la nouvelle image.

Avec `refresh == 0`, ce morceau de code n'a aucun effet, mais il provoque tout de même un réveil du processeur, donc une petite consommation d'énergie supplémentaire. Pour résoudre ce problème, on peut demander à masquer l'interruption du VGA tant qu'une image n'est pas calculée.

Pour cela, on va utiliser le contrôleur d'interruption (qui n'était modélisé que de manière très incomplète dans votre squelette de code). Le composant IntC est décrit en annexe, c'est une version simplifiée du composant IntC fourni par Xilinx.

On suppose définies les constantes suivantes :

```
#define INTC_ISR 0x00
#define INTC_IER 0x08
#define INTC_IAR 0x12
#define INTC_BASE_ADDR 0x41200000
```

2.1 Implémentation du composant

Le composant Intc est étendu de la manière suivante :

```
SC_MODULE(Intc) {
        // Socket ajouté :
        ensitlm::target_socket<Intc> target;

        SC_CTOR(Intc);

        // Ports d'entrée-sortie inchangés :
        sc_core::sc_in<bool> in0;
        sc_core::sc_in<bool> in1;
        sc_core::sc_out<bool> out;
```

```
        // Nouvelles fonctions :
        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);

private:
        void process_in_irq(int N);
        void process_in_irq0() {process_in_irq(0);}
        void process_in_irq1() {process_in_irq(1);}
        void send_irq();

        // IRQ non-masquée
        ensitlm::data_t m_enabled_it;
        // IRQ recue et non-acquitée
        ensitlm::data_t m_active_it;

        bool irq_to_send_notified;
        sc_core::sc_event irq_to_send;
};
```

On implémente la fonction `send_irq()` comme ceci :

```
void Intc::send_irq() {
        while (true) {
                while (!irq_to_send_notified) {
                        wait();
                }
                irq_to_send_notified = false;
                out.write(true);
                wait(20, sc_core::SC_NS);
                out.write(false);
                wait(20, sc_core::SC_NS);
        }
}
```

Question 8 (1 point) *On souhaite faire en sorte qu'une interruption sur* ***in0*** *appelle* `process_in_irq0` *et qu'une interruption sur* ***in1*** *appelle* `process_in_irq1`*. Par ailleurs, la fonction* `send_irq` *doit être enregistrée comme un processus SystemC sensible à l'événement* `irq_to_send`*. Que faut-il écrire dans le constructeur ?*

Par exemple :

```
SC_THREAD(send_irq);
sensitive << irq_to_send;
```

```
#define DECLARE_PROCESS_IRQ(n) \
        SC_METHOD(process_in_irq ## n); \
        sensitive << in ## n.pos()

        DECLARE_PROCESS_IRQ(0);
        DECLARE_PROCESS_IRQ(1);
```

Question 9 (1 point) *La fonction* `process_in_irq(int N)` *enregistre l'interruption numéro* `N` *comme active dans la variable interne* `m_active_it` *puis, si besoin, réveille le thread* `send_irq`*. Implémentez cette fonction.*

```
void Intc::process_in_irq(int N) {
        const ensitlm::data_t mask = (1 << N);
        m_active_it |= mask;
        send_irq_maybe(mask);
}

void Intc::send_irq_maybe(ensitlm::data_t mask) {
        if (m_enabled_it & m_active_it & mask) {
                irq_to_send_notified = true;
                irq_to_send.notify();
        }
}
```

On donne un squelette pour la fonction `read` :

```
tlm::tlm_response_status
Intc::read(ensitlm::addr_t a, ensitlm::data_t& d)
{
        switch(a) {
        case INTC_ISR:
                // ...
                break;
        case INTC_IER:
                // ...
                break;
        case INTC_IAR:
                return tlm::TLM_COMMAND_ERROR_RESPONSE;
                break;
        default:
                return tlm::TLM_ADDRESS_ERROR_RESPONSE;
```

```
        }
        return tlm::TLM_OK_RESPONSE;
}
```

Question 10 (0.5 point) *Expliquez en quelques mots à quoi correspondent les 3 valeurs de retour possibles.*

- `tlm::TLM_COMMAND_ERROR_RESPONSE` : lecture dans un registre write-only.
- `tlm::TLM_ADDRESS_ERROR_RESPONSE` : lecture à une adresse où ne se trouve aucun registre.
- `tlm::TLM_OK_RESPONSE` : la lecture s'est bien passée.

Question 11 (1 point) *Donnez le code correspondant aux cas* `INTC_ISR` *et* `INTC_IER`*.*

```
        case INTC_ISR:
                d = m_active_it;
                break;
        case INTC_IER:
                d = m_enabled_it;
                break;
```

Question 12 (1.5 points) *Implémentez les cas* `INTC_IER` *et* `INTC_IAR` *de la fonction* `Intc::write`*.*

```
        case INTC_IER: {
                const ensitlm::data_t old_d = d;
                m_enabled_it = d;
                // On n'envoie une IRQ que si l'une de celles qu'on
                // vient de démasquer est active
                send_irq_maybe(d & ~(old_d));
                break;
        }
        case INTC_IAR:
                m_active_it &= ~d;
                send_irq_maybe(d);
                break;
```

2.2 Intégration dans la plateforme

Question 13 (1 point) *Quelle(s) ligne(s) faut-il ajouter à la fonction* `sc_main()` *?*

```
        bus.initiator(intc.target);

        bus.map(intc.target,     0x41200000, 0x00010000);
```

2.3 Écriture du logiciel embarqué

On donne ci-dessous une version simplifiée de la structure du logiciel embarqué :

```
void vga_isr() {
        if(refresh) {
                refresh = 0;
                printf("vga_isr: double buffer flip\r\n");
                write_mem(VGA_BASEADDR + VGA_CFG_OFFSET, old_img_addr);
        }
}

void timer_0_isr() {
        printf("timer_0_isr\r\n");
        game_of_life();

        uint32_t tmp = old_img_addr;
        old_img_addr = new_img_addr;
        new_img_addr = tmp;
        refresh = 1;
}

void timer_1_isr() {
        printf("timer_1_isr\n");
}

int __start() {
        printf("-- boot complete -- \r\n");
        refresh = 0;
        // ...
}

void __interrupt() {
        // ...
        if(vga_irq) vga_isr();
        if(timer_0_csr & TIMER_INTERRUPT) timer_0_isr();
```

```
        if(timer_1_csr & TIMER_INTERRUPT) timer_1_isr();
}
```

On souhaite implémenter un mini-driver pour notre contrôleur d'interruption, avec les fonctions suivantes (à utiliser dans le logiciel embarqué) :

```
// Initialiser le composant Intc
static inline void init_intc();

// Acquiter l'interruption numéro n
static inline void ack_it(int n);

// Masquer les interruptions venant du VGA
static inline void disable_vga_it();

// Dé-masquer les interruptions venant du VGA
static inline void enable_vga_it();
```

Dans la version précédente, il n'y avait aucune initialisation du composant IntC, et on supposait qu'aucune interruption n'était masquée au démarrage. L'acquittement d'interruption au niveau du composant IntC n'était pas non plus implémenté.

Question 14 (3 points) *Proposez une implémentation pour ces 4 fonctions.*

```
static inline void ack_it(int n) {
        write_mem(INTC_BASEADDR + INTC_IAR, (1 << n));
}

static inline void init_intc() {
        write_mem(INTC_BASEADDR + INTC_IER, 0xFFFFFFFF);
}

static inline void disable_vga_it() {
        uint32_t ier = read_mem(INTC_BASEADDR + INTC_IER);
        ier &= ~(0x1);
        write_mem(INTC_BASEADDR + INTC_IER, ier);
}

static inline void enable_vga_it() {
        uint32_t ier = read_mem(INTC_BASEADDR + INTC_IER);
        ier |= (0x1);
        write_mem(INTC_BASEADDR + INTC_IER, ier);
}
```

Question 15 (1 point) *Étant donné que le squelette du TP3 ne faisait aucune initialisation ni aucun acquittement, le logiciel tel qu'il était écrit ne marchera plus sur la nouvelle version du composant IntC. Quelles modification doit-on faire pour que le logiciel re-marche ?*

On ajoute `init_intc();` au début de la fonction `__start();`, `ack_it(0);` au début de `vga_isr()` ; et `ack_it(1);` au début des deux `timer_*_isr()`.

Question 16 (1 point) *On souhaite maintenant implémenter l'optimisation proposée dans cette section, c'est à dire masquer l'interruption du composant VGA tant que celle-ci n'est pas utile. Quelle modification doit-on faire au logiciel embarqué ?*

On ajoute `disable_vga_it();` à chaque fois qu'on fait `refresh = 0`, et `enable_vga_it();` à chaque fois qu'on fait `refresh = 1`.

Question 17 (1 point) *La version du logiciel embarqué obtenue avec cette optimisation sera-t-elle plus rapide en simulation ? Pourquoi ?*

Oui, légèrement plus rapide (5% d'après mes mesures). En effet, on évite de réveiller un processus SystemC pour pas grand chose quand le VGA lance une interruption.

```
20
20
```

Annexe

Contrôleur d'interruption (IntC) Documentation

Le composant IntC 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 IntC permet l'interfaçage de plusieurs périphériques générant des interruptions avec un bloc maître ne gérant qu'une seule entrée d'interruption (généralement un processeur).

Entrées/Sorties

Le composant IntC possède les entrées/sorties suivantes :

- Interface esclave compatible OPB
- Deux entrées d'interruptions `int_in_0` et `int_in_1`
- Une sortie d'interruption `int_out`

Les interruptions sont considérées actives sur front montant. Une IRQ est une impulsion courte (i.e. le signal redescend peu de temps après l'émission de l'interruption)

Fonctionnement interne

Initialement la sortie d'interruption du composant est à 0 (ou `false`). Lorsqu'une interruption est reçue sur `int_in_0` ou `int_int_1`, l'interruption est dans tous les cas considérée en interne dans le composant comme active, et si l'interruption n'est pas masquée, elle est ré-émise sur la sortie `int_out`. Le registre `ISR` est positionné de façon à identifier le ou les émetteurs à l'origine de l'interruption émise par l'IntC.

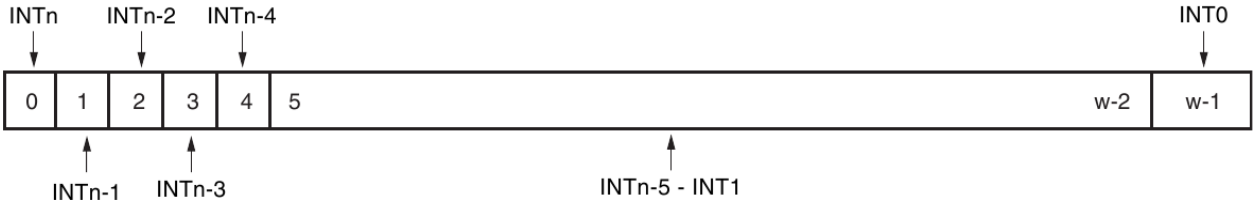
On peut empêcher l'émission d'une interruption sur `int_out` en masquant l'interruption avec le registre `IER`, mais si une interruption est reçue pendant qu'elle est masquée, elle sera ré-émise immédiatement après dé-masquage par l'écriture d'un 1 dans le bit correspondant de `IER`.

Pour réinitialiser ("clear") une interruption reçue, il faut l'acquitter en écrivant 1 dans le bit correspondant du registre `IAR`. Si d'autres signaux d'interruption en entrée sont actifs (et non-acquités) à ce moment, une autre interruption est émise sur la sortie. Par exemple, si on reçoit une interruption sur l'entrée 0, puis sur l'entrée 1, puis qu'on acquitte l'interruption de l'entrée 0, alors une interruption sera générée en sortie pour signaler l'interruption de l'entrée 1 au moment de cet acquittement.

Récapitulatif des registres

Adresse relative	Type	Valeur initiale	Nom	Description
0x00	Lecture seule	0	ISR	Interrupt Status Register
0x08	Lecture / Écriture	0	IER	Interrupt Enable Register
0x12	Écriture seule	0	IAR	Interrupt Acknowledge Register

Ces 3 registres ont la structure suivante :



Chaque bit du registre correspond à une source d'interruption possible (nous nous limiterons à 2 sources ici). Les deux bits du registre sont alignés sur les bits de poids faible. Le bit 0 correspond à la source `int_in_0` et le bit 1 à la source `int_in_1`.

Valeurs du registre `ISR`

Chaque bit vaut 1 si une interruption a été reçue (et non acquittée) sur l'entrée correspondante.

Valeurs du registre `IER`

Le registre est initialisé à 0. Écrire 1 dans un bit dé-masque l'interruption correspondante, et écrire 0 la masque. Si une interruption a été reçue et non acquittée au moment où un 1 est écrit, une interruption est émise sur la sortie.

Ce registre est conçu pour être utilisé par une séquence "lecture/modification/écriture" par le processeur (on ne peut pas adresser les bits un par un, donc pour écrire sur le bit N, on lit l'ensemble, on modifie le bit, et on écrit le résultat).

Valeurs du registre `IAR`

Écrire 1 dans un bit acquitte l'interruption correspondante. Écrire 0 dans un bit ne fait rien.