Validation des applications communicantes embarquées

Durée : 3h
Documents : tous documents autorisés

Veuillez respecter les notations introduites dans l'énoncé.
Les parties 1 et 2 sont indépendantes, veuillez les rédiger sur des copies séparées.
Des explications pertinentes en français seront **très** appréciées, et il est indispensable de **justifier** vos réponses.

---

1 - Modélisation transactionnelle et SystemC (10 points)

On considère une application de traitement d'image qui utilise notamment deux micro-processeurs ayant accès à une mémoire commune. Les deux processeurs écrivent sur des zones disjointes de la mémoire. À la fin de son traitement, le processeur 1 met à disposition du processeur 2 l'image qu'il vient de finir de calculer en l'écrivant en mémoire. Le processeur 2 est originellement en attente d'une image disponible en provenance du processeur 1. Lorsqu'une image est disponible, il la recopie dans sa zone mémoire et réalise un traitement sur celle-ci puis se remet en attente d'une autre image (on ne se préoccupe pas de savoir comment le résultat du traitement par le processeur 2 est communiqué vers l'extérieur).

Pour synchroniser les deux processeurs, on souhaite utiliser un composant Mailbox, dont la spécification est fournie en annexe.

Note : chaque processeur dispose d'une unique entrée interruption nommée `irq`, active sur front montant. Les autres entrées/sorties (processeur et mémoire) sont les mêmes que pour les composants vus en TP.

▷ **Question 1 (3 points) :**

En utilisant les conventions graphiques vues en cours, faire un schéma d'ensemble de la plate-forme (mettre une légende pour indiquer la signification de chaque symbole). Expliquer comment intégrer le composant Mailbox et comment l'utiliser, de façon concise mais précise. Décrire l'utilisation sur le cas de deux échanges consécutifs d'images.

▷ **Question 2 (5 points) :**

Écrire en SystemC (fichier .h et fichier .cpp) la classe Mailbox implémentant la spécification donnée en annexe.
Vous utiliserez les mêmes éléments de la bibliothèque TLM que ceux vus en cours et en TPs. On supposera l'existence de deux types `ADDRESS_TYPE` et `DATA_TYPE`, chacun codés sur 32 bits. Enfin, vous considérerez que les sorties d'interruptions sont à `false` en début de simulation (pas d'initialisation nécessaire).

On souhaite maintenant écrire en langage C le logiciel embarqué sur le processeur **1**. On suppose l'existence des fonctions suivantes :

- `void production_image()` : calcule une nouvelle image et la met à disposition dans la mémoire commune,
- `void write_mem(a, d)` : écrit la donnée `d` à l'adresse `a` sur le bus auquel est relié le processeur,
- `int read_mem(a)` : lit une donnée à l'adresse `a` sur le bus auquel est relié le processeur, et la retourne.

Enfin, on supposera que la fonction `void irq_handler()` sera le gestionnaire d'interruption et `int main()` la fonction principale. Le port esclave (cible) du composant Mailbox est à l'adresse `0x1000` sur le bus principal.

▷ **Question 3 (2 points) :**

Expliquez ce que doit faire le processeur 1 pour produire des images en boucle et les communiquer au processeur 2.
Écrire le logiciel embarqué du processeur 1 : donner le contenu de la fonction principale et du gestionnaire d'interruption. Vous introduirez les variables qui vous semblent nécessaires.

2 - Validation (10 points)

Dans cette partie, on s'intéresse à deux processus (un écrivain et un lecteur) qui échangent une donnée simple (par exemple un entier) par l'intermédiaire d'une boîte aux lettres assimilable à une file d'attente à une seule case. L'idée est d'assurer un fonctionnement global tel que : l'écrivain ne peut écrire une nouvelle valeur que si le lecteur a lu la précédente ; le lecteur ne peut lire une nouvelle valeur que si l'écrivain en a produit une nouvelle.

La boîte aux lettres est décrite par l'automate interprété de la figure **??**-(a). On ne tient pas compte de la valeur échangée, mais seulement des effets de synchronisation. Les conditions de transitions peuvent être : des actions comme `lire`, `ecrire`, ou bien des tests sur des variables `ITP2V` et `ITV2P` qui représent des interruptions. Les actions associées aux transitions sont des affectations à ces deux variables. Les interruptions sont donc considérées comme des variables partagées, et les automates se synchronisent par mémoire partagée. La boîte aux lettres prévient quand elle accepte une valeur en écriture, en envoyant l'interruption `ITV2P` (*vide* vers *plein*). Inversement, elle prévient quand la valeur qu'elle contient est lue en envoyant `ITP2V`.

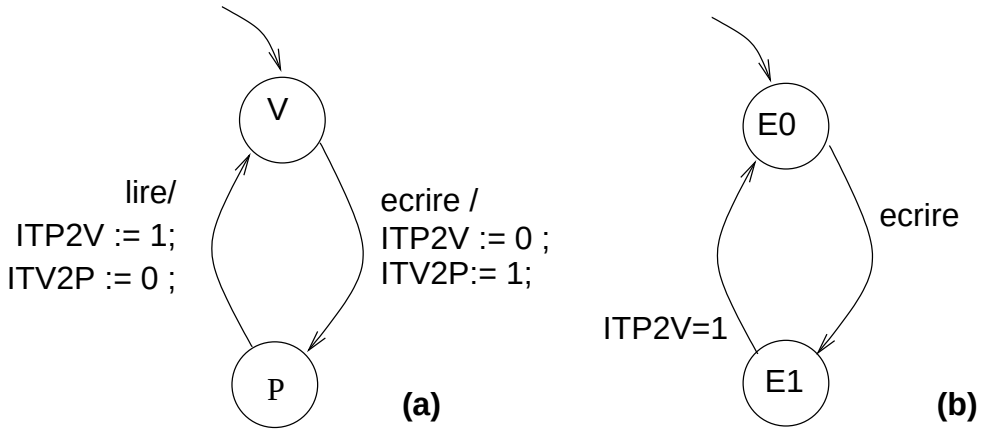

Figure 1: Comportement de la boîte aux lettres (a) et du processus écrivain (b).

La figure **??**-(b) décrit le comportement d'un processus écrivain qui respecte le protocole imposé par la boîte aux lettres : il écrit puis attend de voir `ITP2V` avant d'écrire de nouveau.

Le parallélisme d'exécution est représenté par un produit asynchrone défini comme suit : on construit un automate global dont les états sont des quintuplets : (état de l'écrivain, état du lecteur, état de la boîte aux lettres, valeur de `ITP2V`, valeur de `ITV2P`). Les interruptions sont initialement à 0. Les transitions étiquetées par `lire` (resp. `ecrire`) ne peuvent être exécutées que si tous les processus dans lesquels cette étiquette apparaît exécutent en même temps une transition étiquetée par `lire` (resp. `ecrire`). Les transitions conditionnées par un test sur `ITP2V` ou `ITV2P` ne peuvent être exécutées qu'à partir d'un état global où la condition est vraie.

▷ **Question 4 (1.5 points) :**

Sur le même modèle que l'écrivain, donner l'automate d'un processus lecteur qui respecte le protocole de la boîte aux lettres.

On veut pouvoir analyser les systèmes qui contiennent la boîte aux lettres, et être capable de détecter les cas où l'un des deux processus au moins ne respecte pas le protocole.

▷ **Question 5 (1.5 points) :**

Ajouter un ou plusieurs états d'erreur à l'automate de la boîte aux lettres, ainsi que les transitions associées, de manière à ce que ces états d'erreur soient atteints quand l'un des processus ne respecte pas le protocole.

▷ **Question 6 (2 points) :**

Construire le produit des 3 automates (avec la boîte à lettres obtenue à la question **??**).

▷ **Question 7 (2 points) :**

De quel type ("safety" ou "liveness") est la propriété : "*les 2 processus respectent le protocole*" ? Quelles techniques automatiques peuvent être utilisées pour la prouver ? Est-ce que le fait d'ignorer la valeur échangée empêche de prouver la propriété ci-dessus ? Justifiez soigneusement vos réponses.

▷ **Question 8 (2 points) :**

On désire appliquer des techniques de model-checking énumératif à cet exemple. Quel est le nombre d'états potentiels du système ? Combien sont accessibles ? Vaut-il mieux utiliser une technique en avant ou en arrière ?

▷ **Question 9 (1 points) :**

On désire appliquer des techniques de model-checking symbolique. Expliquez comment vous codez le système en variables booléennes. En particulier, combien faut-il de variables ?

Annexe - Documentation du composant Mailbox

[systemc]c++[iso]c++ morekeywords=SC$_M ODULE, SC_C TOR, SC_T HREAD, SC_M ETHOD, SC_H AS_P ROC$

language=[systemc]c++ commentstyle=

keywordstyle=

basicstyle= showstringspaces=false

Le composant Mailbox 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 Mailbox fournit la logique et les entrées/sorties nécessaires à la synchronisation de deux modules maîtres pour un échange unidirectionnel maître 1 vers maître 2 d'une donnée sur 32 bits.

Entrées/Sorties

Le composant Mailbox possède les entrées/sorties suivantes :

- Interface esclave compatible AMBA
- Sortie d'interruption int_sender pour le composant maître 1 (émetteur)
- Sortie d'interruption $int_receiver$ pour le composant maître 2 (récepteur)

Fonctionnement interne

Initialement les deux sorties d'interruption du composant Mailbox sont à 0 (ou $false$). Lors d'une écriture de valeur sur le registre $DATA_REG$, une interruption sur $int_receiver$ est émise (passage à 1 ou $true$), la sortie d'interruption int_sender est réinitialisée (à 0 ou $false$) et la valeur est stockée dans le registre. Lors d'une lecture de valeur sur le registre $DATA_REG$, une interruption sur int_sender est émise (passage à 1), la sortie d'interruption $int_receiver$ est réinitialisée (à 0) et la valeur du registre renvoyée.

Récapitulatif des registres

Adresse relative	**Type**	**Taille**	**Valeur initiale**	**Nom**	**Description**
$0x00$	Lecture/Écriture	32 bits	$0x00000000$	$DATA_REG$	Registre de données

Utilisation du composant

Le registre de données contient la valeur à échanger entre les deux composants maîtres. L'écriture d'une valeur dans ce registre déclenche une interruption sur $int_receiver$ et remet à zéro l'interruption sur int_sender. De façon symétrique, une lecture du registre provoque une interruption sur int_sender et une remise à zéro de l'interruption de $int_receiver$.