Validation des systèmes embarqués

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 décodage vidéo qui utilise deux décodeurs matériels: l'un pour la vidéo, et l'autre pour l'audio. Les décodeurs indiquent la fin d'un traitement (fin de décompression d'une image pour le décodeur vidéo, et fin de décompression d'un paquet audio pour le décodeur audio) par une interruption chacun. Ces décodeurs doivent être pilotés par un processeur qui ne possède qu'une seule entrée d'interruption. Pour ce faire on souhaite utiliser un contrôleur d'interruption, dont la spécification est fournie en annexe.

Note : les interruptions sont considérées actives sur front montant.

▷ **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 ITC et comment l'utiliser, de façon concise mais précise. Quel composant additionnel est-il nécessaire?

▷ **Question 2 (5 points) :**

Écrire en SystemC (fichier .h et fichier .cpp) la classe ITC 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 la sortie d'interruption est à `false` en début de simulation (pas d'initialisation nécessaire).

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

- `void decodage_image()` : programme un décodage vidéo sur le décodeur matériel correspondant,
- `void decodage_son()` : programme un décodage audio sur le décodeur matériel correspondant,
- `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 ITC est à l'adresse `0x1000` sur le bus principal.

▷ **Question 3 (2 points) :**

Écrire le logiciel embarqué du processeur de façon à garantir le schéma suivant:

1. Décodage Vidéo en parallèle avec décodage audio
2. Attente de la fin des deux décodages
3. Retour au point 1

2 - Validation (10 points)

2.a - Sûreté ou vivacité ? (2 points)

▷ **Question 4 (2 points) :**

Parmi les propriétés suivantes, lesquelles sont des propriétés de sûreté (safety) et lesquelles sont des propriétés de vivacité (liveness) ? Justifiez vos réponses.

1. A partir de tout état du système, atteignable depuis l'état initial, il est possible de retourner à l'état initial.
2. Si l'on observe les sorties `a`, `b` et `c` du système, tout intervalle entre un instant où `a` est vrai et un instant où `c` est vrai contient au moins une occurrence de `b`.
3. La durée pendant laquelle le système émet sa sortie `a` continûment est d'au plus 10 instants.
4. Dans un système distribué, si la machine `M1` envoie un message à la machine `M2`, la machine `M2` le reçoit toujours avant le prochain envoi de message de `M1` à `M2`.

2.b - Une classe de modèles de systèmes (5 points)

On considère des systèmes informatiques qui sont fidèlement représentés par des modèles comme celui de la figure 1. Le nombre d'états est fini, le nombre de variables (`x, y, ...`) est fini, mais les variables prennent leurs valeurs dans un domaine infini, les entiers naturels $\mathbb{N}$. Toutes les variables valent initialement 0. Les transitions de l'automate portent, soit des conditions, soit des affectations. Les conditions sont de la forme `v != v'` où `v` et `v'` sont deux variables dans l'ensemble `x, y, ...` et `!=` denote l'opérateur "différent", comme en C. Les affectations sont de la forme `v1 = f (v1, v2, v3, ...)` où l'une des variables prend une valeur calculée d'après les valeurs de toutes les variables (y compris elle-même) avec une fonction.

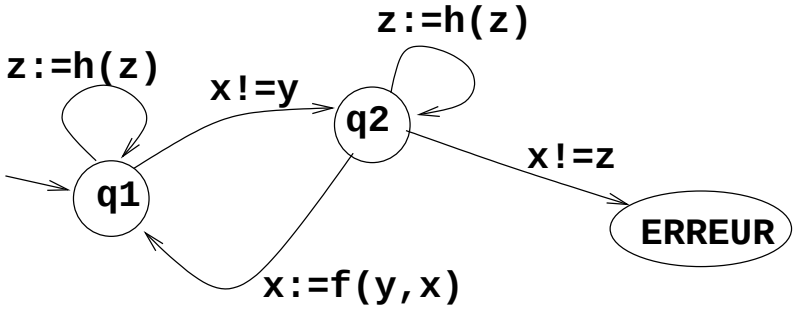

Figure 1: Une classe de modèles à analyser

On suppose que l'on s'intéresse à des propriétés de sûreté uniquement, et que la modélisation du système et des propriétés de sûreté à prouver a conduit à définir un unique état d'erreur, parmi tous les états du système. On suppose de plus qu'il y a une seule transition qui mène à l'état d'erreur, et qu'elle est étiquetée par une condition. Comme d'habitude on se demande s'il existe un chemin depuis l'état initial jusqu'à l'état d'erreur.

Le but de cet exercice est de s'interroger sur les méthodes qui permettent de répondre à cette question, de manière exacte ou approchée. Cela dépend de la forme des fonctions `f, g, ...`.

On suppose que les fonctions `f, g, ...` sont toutes de la forme : `(arith (x, y, ...)) mod k`, c'est-à-dire : une formule arithmétique quelconque (avec les opérateurs `+, *`) sur les variables et des constantes entières positives, le tout modulo `k`. `k` est un entier naturel, et sa valeur est la même dans tout l'automate.
Exemple d'affectation : `x := ((x+y*z+2)*(x+1)) mod k`.

▷ **Question 5 (1 point) :**

1. Montrer que $(x + y) \bmod k = (x \bmod k + y \bmod k) \bmod k$, pour tous $x, y, k \in \mathbb{N}^*$.
2. Sans faire de démonstration, exprimer une propriété similaire pour l'opérateur de multiplication.

On s'intéresse aux états complets des systèmes décrits par des automates de la forme donnée Figure 1. Rappelons qu'un état complet est formé d'un état de l'automate plus une valuation des variables. Comme les variables sont dans un domaine infini, le nombre d'états complets est également infini. On cherche à se ramener à une abstraction finie.

On suggère d'analyser les automates de la forme décrite Figure 1 en calculant, pour chaque état de l'automate, le vecteur $(x \bmod k, y \bmod k, z \bmod k, ...)$, c'est-à-dire en ne conservant que l'information modulo k de chacune des variables.

▷ **Question 6 (1 point) :**

Pour un automate à n états et m variables, combien y a-t-il d'états complets ainsi abstraits ? Justifiez votre réponse.

On étudie maintenant comment se comporte cette abstraction vis-à-vis des affectations qui apparaissent dans l'automate.

▷ **Question 7 (1 point) :**

— Si l'on se trouve dans un état `q` de l'automate, avec un vecteur abstrait $(x_a, y_a, z_a, ...)$, et que l'on prend une transition de `q` à `q'` étiquetée par l'affectation `x := ((x+y*z+2)*(x+1)) mod k`, expliquez comment calculer le vecteur abstrait associé à l'état `q'`, c'est-à-dire après l'affectation.
— Qu'en pensez-vous ?

On étudie ensuite comment se comporte l'abstraction vis-à-vis des tests qui apparaissent dans l'automate.

▷ **Question 8 (2 points) :**

— Si l'on se trouve dans un état `q` de l'automate, avec un vecteur abstrait $(x_a, y_a, z_a, ...)$, et que l'on essaie de prendre la transition de `q` à `Erreur` étiquetée par le test `x != z`, expliquez comment le vecteur abstrait permet de déterminer si la transition est exécutable, et donc si l'état d'erreur est accessible.
— Si vous déclarez que l'état d'Erreur est inaccessible sur la base de vos raisonnements abstraits, que pouvez-vous en déduire sur le fait que l'état d'Erreur est effectivement inaccessible dans le système d'origine ? Justifiez soigneusement.
— reprenez les deux points ci-dessus en supposant maintenant que les conditions qui apparaissent dans l'automate sont de la forme `x == y` (des tests d'égalité).

2.c - Compréhension globale (3 points)

Un vendeur d'outils informatiques pour la CAO de systèmes électroniques, appelons-le X[1], présente un outil Y pour la vérification séquentielle des comportements d'un circuit décrit au niveau RTL et d'un modèle de plus haut niveau décrit par exemple en SystemC. On lit par exemple sur le serveur web du vendeur X :

1. *Y verifies that RTL implementations match hardware intent captured in system-level models written in SystemC and C++*
2. *Y verifies that RTL (Verilog and VHDL) optimizations for power, timing and area do not introduce functional defects*
3. *Y verifies block-level clock gating changes in Verilog and VHDL designs.*
4. *...*

Si vous étiez un industriel en charge du développement de circuits, seriez-vous convaincus par cet argumentaire ? Pour cela, répondez aux deux sous-questions ci-dessous.

▷ **Question 9 (4 points) :**

(a) Considérons le point 2 ci-dessus (les transformations d'une description RTL de circuit pour faire des optimisations). Expliquer de manière précise quelles techniques et outils de vérification formelle vus en cours doivent être mis en œuvre dans l'outil Y pour prouver qu'un circuit et un circuit transformé ont le même comportement. En particulier :

- Expliquer ce que peut être un "modèle" de circuit décrit au niveau RTL, parmi les catégories suivantes : modèle purement booléen, automate interprété général, quelque chose d'intermédiaire, ...
- En déduire quel type de méthode de vérification peut convenir (model-checking, interprétation abstraite, ...) et quelle est la qualité des résultats obtenus (exacts, approchés, ...). Par exemple, si la méthode de vérification déclare que les deux circuits sont équivalents, que faut-il en penser ?
- Discuter du coût en temps, et donc de la taille des circuits analysables

Justifier soigneusement.

(b) Même question pour le point 1 ci-dessus, c'est-à-dire la comparaison entre une description de circuit en SystemC/C++ et la description du même circuit au niveau RTL. Si vous pensez que votre réponse dépend éventuellement de la forme des descriptions SystemC que l'on considère, expliquez-le.

[1] Ce vendeur, et l'outil décrit par la suite, existent réellement.