E.N.S.I.M.A.G.
3ème année
Janvier 2013

Examen

Modélisation et Vérification de Systèmes Concurrents et Temps-Réel

Durée 2 heures
Tous documents autorisés

Avertissement : Il vous est demandé d'apporter le plus grand soin dans la rédaction. Vous serez jugés plus sur la QUALITÉ que sur la QUANTITÉ de vos réponses.

Le barême est donné à titre purement indicatif.

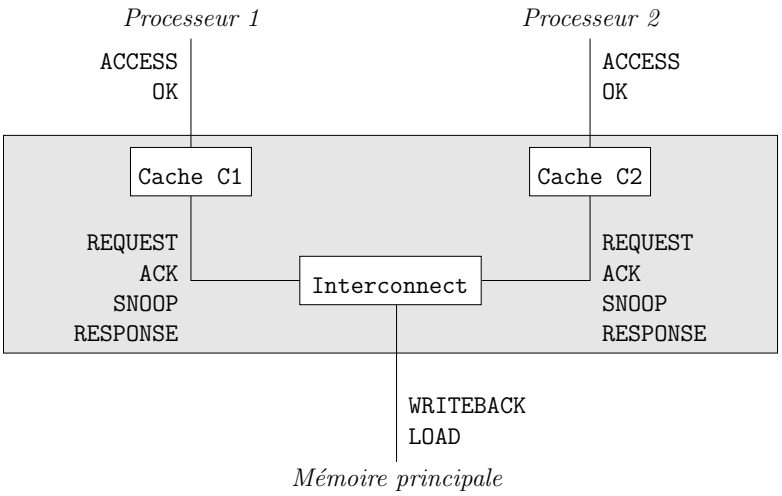
Partie I Modélisation en LNT (10 points)

Dans un système multiprocesseur qui possède une mémoire cache séparée pour chaque processeur, il est possible qu'une même donnée soit dupliquée dans la mémoire principale et dans la mémoire cache de plusieurs processeurs. La cohérence de cache s'assure que toute modification d'une donnée est propagée à travers le système de manière cohérente et efficace. Les données sont stockées dans les caches sous la forme de blocs de taille fixe appelés lignes de cache.

On considère une version simplifiée d'un protocole de cohérence de cache inspiré du protocole MESI, un des protocoles de cohérence de cache les plus courants. Dans ce protocole, chaque ligne de cache est étiquetée par un des quatre états suivants :

- **Modified** (M) : La donnée contenue dans la ligne de cache est présente seulement dans le cache courant et sa valeur en mémoire principale n'est plus à jour. La donnée devra être copiée dans la mémoire principale avant de pouvoir être lue par les autres processeurs. A ce moment, la ligne de cache passera dans l'état Exclusive.
- **Exclusive** (E) : La donnée contenue dans la ligne de cache est présente seulement dans le cache courant et sa valeur en mémoire est à jour. Elle peut passer dans l'état Shared à n'importe quel moment, en réponse à une requête de lecture en provenance d'un autre processeur. Elle peut aussi passer dans l'état Modified, en réponse à une écriture du processeur courant.
- **Shared** (S) : La donnée contenue dans la ligne de cache peut être présente dans d'autres caches du système et sa valeur est à jour. Elle peut passer dans l'état Invalid à n'importe quel moment, en réponse à une requête d'écriture en provenance d'un autre processeur.
- **Invalid** (I) : La ligne de cache ne contient pas de donnée valide ; la ligne de cache n'est pas utilisée.

Dans cet examen, on considèrera uniquement la partie du système constituée des mémoires cache de deux processeurs et d'un *interconnect* (cas plus général qu'un bus, par exemple un réseau sur puce) sur lequel les données sont échangées, comme dessiné sur la figure suivante. En particulier, la mémoire principale et les processeurs ne seront pas modélisés.



De plus, pour abstraire la spécification, on ne modélise pas les valeurs des données, qui restent implicites.

On donne les types, fonctions et channels LNT suivants:

```
type OPERATION_T is READ, WRITE with "==", "<>" end type

type CACHE_T is C1, C2 with "==", "<>" end type

function OTHER (CACHE: CACHE_T) : CACHE_T is
   case CACHE in
     C1 -> return C2
   | C2 -> return C1
   end case
end function

type LINE_T is L1, L2 with "==", "<>" end type

type PENDING_T is
   NONE, PENDING (OP: OPERATION_T, LINE: LINE_T)
   with "=="
end type

type STATUS_T is M, E, S, I with "==", "<>" end type

channel PROCESSOR_CHANNEL is
   (CACHE_T, OPERATION_T, LINE_T), (CACHE_T)
end channel

channel REQ_CHANNEL is (CACHE_T, OPERATION_T, LINE_T) end channel

channel ACK_CHANNEL is (CACHE_T, STATUS_T) end channel

channel MEMORY_CHANNEL is () end channel
```

Question I.1 Modélisation d'un cache

Les caches sont modélisés par des instances du processus `CACHE` suivant, paramétrées respectivement par `C1` et `C2`.

```
process CACHE [ACCESS, OK: PROCESSOR_CHANNEL, REQUEST, SNOOP: REQ_CHANNEL,
               ACK, RESPONSE: ACK_CHANNEL] (ID: CACHE_T) is
   var LINE: LINE_T, PENDING: PENDING_T, STATUS: STATUS_T, OP: OPERATION_T, L: LINE_T in
      LINE := any LINE_T;
      PENDING := NONE;
      STATUS := I;
      loop
         select
            case PENDING in
              NONE ->
                  ACCESS (ID, ?OP, ?L);
                  PENDING := PENDING (OP, L)
            | PENDING (OP, L) ->
                  case OP in
                    READ ->
                        if (L == LINE) and (STATUS <> I) then
                           OK (ID)
                        else
                           REQUEST (ID, READ, L);
                           ACK (ID, ?STATUS);
                           LINE := L;
                           OK (ID)
                        end if
                  | WRITE ->
                        if (L == LINE) and ((STATUS == M) or (STATUS == E)) then
                           OK (ID);
                           if STATUS == E then STATUS := M end if
                        else
                           REQUEST (ID, WRITE, L);
                           ACK (ID, ?STATUS);
                           LINE := L;
                           OK (ID)
                        end if
                  end case;
                  PENDING := NONE
            end case
         []
            SNOOP (ID, READ, ?L);
            if (L == LINE) then
               RESPONSE (ID, STATUS);
               if (STATUS == M) or (STATUS == E) then
                  STATUS := S
               end if
            else
               RESPONSE (ID, I)
            end if
         []
            SNOOP (ID, WRITE, ?L);
            if (L == LINE) then
               RESPONSE (ID, STATUS);
               STATUS := I
            else
               RESPONSE (ID, I)
            end if
         end select
      end loop
   end var
end process
```

On considère le processus EXERCICE suivant :

```
process EXERCICE [ACCESS, OK: PROCESSOR_CHANNEL, REQUEST, SNOOP: REQ_CHANNEL,
                  ACK, RESPONSE: ACK_CHANNEL] is
   par ACCESS, OK, SNOOP in
      ACCESS (C1, READ, L1); OK (C1)
   ||
      CACHE [ACCESS, OK, REQUEST, SNOOP, ACK, RESPONSE] (C1)
   end par
end process
```

Dessiner le LTS correspondant au processus EXERCICE.

Indication : Ce LTS a de l'ordre d'une dizaine d'états et une dizaine de transitions.

Question I.2 Modélisation de l'interconnect

L'interconnect a le comportement cyclique suivant :

- attente sur la porte REQUEST d'une requête en provenance d'un cache CI concernant une opération OP sur une adresse mémoire LINE ;
- transmission sur la porte SNOOP (en français, *surveiller*) de l'opération OP et de l'adresse LINE au cache de l'autre processeur ;
- attente sur la porte RESPONSE de la réponse du cache de l'autre processeur, qui indique le statut SJ d'une éventuelle copie de la donnée stockée à l'adresse mémoire LINE ;
- si OP est l'opération READ :
  - si SJ est l'état M, écriture sur la porte WRITEBACK pour informer la mémoire de la nouvelle valeur de la donnée à l'adresse LINE ;
  - si SJ est l'état I, chargement depuis la mémoire sur la porte LOAD de la donnée courante stockée à l'adresse LINE ;
  - sinon, passage à l'étape suivante ;
- si OP est l'opération WRITE : passage à l'étape suivante ;
- envoi sur la porte ACK au cache CI d'un acquittement indiquant le nouvel état SI de la ligne de cache :
  - si OP est l'opération READ et si SJ est différent de I, alors le cache CI devra passer dans l'état S ;
  - si OP est l'opération READ et si SJ est l'état I, alors le cache CI devra passer dans l'état E ;
  - si OP est l'opération WRITE, alors le cache CI devra passer dans l'état M.

Compléter la définition du processus INTERCONNECT ci-dessous :

```
process INTERCONNECT [LOAD, WRITEBACK: MEMORY_CHANNEL, REQUEST, SNOOP: REQ_CHANNEL,
                      ACK, RESPONSE: ACK_CHANNEL]
is
   var CI, CJ: CACHE_T, OP: OPERATION_T, LINE: LINE_T, SI, SJ: STATUS_T in
      -- partie à compléter (25 lignes environ)
      ...
   end var
end process
```

Note : On respectera bien le typage (channels) lors de l'utilisation des portes `LOAD`, `WRITEBACK`, `REQUEST`, `SNOOP`, `ACK` et `RESPONSE`.

Question I.3 Modélisation du système complet

Compléter la définition du processus `MAIN` ci-dessous, modélisant l'architecture du système constitué des deux caches d'identificateurs respectifs C1 et C2 et de l'interconnect, en considérant les opérations `REQUEST`, `ACK`, `SNOOP` et `RESPONSE` comme des opérations internes.

```
process MAIN [ ... (* déclaration des portes à compléter *) ] is
  -- partie à compléter (10 lignes environ)
  ...
end process
```

Partie II Automates temporisés (5 points)

On considère une extension temporisée d'un cache de nom `ID`. Le cache stocke la ligne de cache d'adresse mémoire `LINE` dont l'état selon le protocole MESI est `STATUS`.

On distingue deux cas, selon s'il y a une opération du processeur en attente. Dans les deux cas, le cache est prêt à repondre à une requête se surveillance venant de l'interconnect. Ainsi, le cache répond à un rendez-vous "`SNOOP (ID, ?OP, ?L)`" en moins de 2 unités de temps avec un rendez-vous "`RESPONSE (S)`", où `S` vaut `STATUS` si `L` est égale à `LINE` ou `I` sinon ; en même temps, le cache met à jour la variable `STATUS` en utilisant la fonction "`update_snoop (LINE, L, STATUS, OP)`".

Dans le cas où le cache n'a pas d'opération en attente, il est prêt à recevoir, en plus d'un `SNOOP`, une demande d'opération `OP` sur la ligne de cache `LINE` par un rendez-vous "`ACCESS (ID, ?OP, ?LINE)`", et il mémorise l'opération (respectivement, l'adresse) dans la variable `PENDING_OP` (respectivement `PENDING_LINE`).

Dans le cas où le cache a une opération en attente, on distingue deux sous-cas :

- Si `PENDING_LINE` est égal à `LINE` et le prédicat "`direct (OP, STATUS)`" est vrai, alors il répond en moins d'1 unité de temps par un rendez-vous "`OK (ID)`" et, en même temps, il met à jour la variable `STATUS` en utilisant la fonction "`update_ok (STATUS, OP)`".
- Sinon, le cache transmet la demande à l'interconnect par un rendez-vous "`REQUEST (ID, OP, LINE)`". Ensuite, il se met en attente de l'acquittement de l'interconnect par un rendez-vous "`ACK (ID, ?STATUS`", qui met à jour la variable `STATUS`. Si après 6 unités de temps aucun acquittement n'a été reçu, le cache réitère la requête. Après réception d'un acquittement, il met à jour la variable `LINE` en lui affectant `L` et répond en moins d'1 unité de temps au processeur par un rendez-vous "`OK (ID)`".

Dessiner l'automate temporisé d'un cache, en utilisant la notation suivante pour les étiquettes des transitions :

$$g \ / \ \mathtt{G}\ (O_1, ..., O_{n\geq 0})\ ;\ a$$

où :

- "g" est une garde ; on utilise "*true*" pour une garde toujours vraie

- "`G` (O_1, ..., $O_{n\geq 0}$)" est un rendez-vous, possiblement avec les offres "O_1, ..., $O_{n\geq 0}$", comme en LNT
- "a" est une affectation d'horloges et de variables locales (exécutée après le rendez-vous)

Le temps entre une demande d'accès sur la porte `ACCESS` et la réponse correspondante sur la porte `OK` est-il borné ? Justifiez votre réponse.

Note : la fonction "`update_snoop (LINE, L, STATUS, OP)`" peut être définie par le pseudo-code suivant :

```
function update_snoop (LINE, L: LINE_T, STATUS: STATUS_T, OP: OPERATION_T) : STATUS_T is
   if LINE != L then
      return STATUS
   elsif OP == WRITE then
      return I
   elsif (STATUS == M) or (STATUS == E) then
      return S
   else
      return STATUS
   end if
end function
```

Note : la fonction "`update_ok (STATUS, OP)`" peut être définie par le pseudo-code suivant :

```
function update_ok (STATUS: STATUS_T, OP: OPERATION_T) : STATUS_T is
   if OP == READ then
      return STATUS
   else
      return M
   end if
end function
```

Note : le prédicat "`direct (STATUS, OP)`" peut être défini par le pseudo-code suivant :

```
function direct (STATUS: STATUS_T, OP: OPERATION_T) : Bool is
   if OP == READ then
      return (STATUS != I)
   else
      return ((STATUS == M) or (STATUS == E))
   end if
end function
```

Note : cette version d'un cache est une extension du processus `CACHE` de la question Question I.1 ci-dessus, avec les contraintes temporelles suivantes :

- Le cache répond à une requête sur la porte `SNOOP` en moins de 2 unités de temps par une rendez-vous sur la porte `RESPONSE`.
- Si une requête du cache vers l'interconnect sur la porte `REQUEST` n'est pas suivie d'un acquittement sur la porte `ACK` après 4 unités de temps, le cache réitère la requête.

Partie III Logique temporelle (5 points)

On considère l'automate de la Figure 1 (dont 0 est l'état initial). Indiquer pour chacune des huit formules ci-dessous les états de l'automate qui la satisfont.

1. $\langle \texttt{ACCESS} \rangle$ tt
2. $[\texttt{ACK}]$ ff
3. $[\texttt{SNOOP}]$ tt
4. $[\text{tt}^*]\langle \text{tt}^* \cdot \texttt{REQUEST} \rangle$ tt
5. $[\texttt{SNOOP} \cdot (\neg \texttt{RESPONSE})^* \cdot \texttt{SNOOP}]$ ff
6. $\mu X.\langle \texttt{OK} \rangle$ tt $\vee \langle \texttt{ACCESS} \rangle X$
7. $\mu X.\langle \texttt{OK} \rangle X$
8. $\nu X.\langle \texttt{SNOOP} \cdot \texttt{RESPONSE} \rangle X$

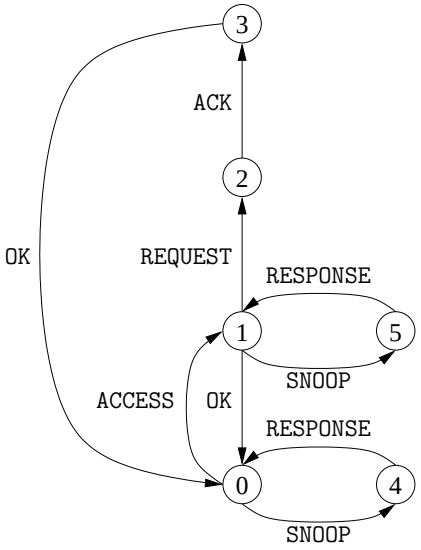


Figure 1: Automate de la Partie III