2010

Rapport Livrable n°2

Thibault COUDERT ; Thomas TYGREAT

EISTI

14/04/2010

Algorithmes

```
Detecter_Defaut_Cache_LRU( t_cache mc , adresse @ ) : booleen
Variables :
        index : entier
Debut

index := conv_bin2dec (@(2 .. 4))

Pour i := 0 à 15 Faire
        Si mc[index][i].bit_validite = '1' Alors
                Si mc[index][i].etiquette = @(5..10) Alors
                        // MAJ du tag
                        mc[index][i].tag_LRU := 0
                        // MAJ des tags
                        Pour i := 0 à 15 Faire
                                Si mc[index][i].bit_validite = '1' Alors
                                        mc[index][i].tag_LRU := mc[index][i].tag_LRU + 1
                                FinSi
                        FinPour
                        retourner Faux
                FinSi
        FinSi
FinPour

retourner Vrai

Fin
```

```
Detecter_Defaut_Cache_LFU( t_cache mc  , adresse @) : booleen
Variables :
        index : entier

Debut

index := conv_bin2dec(@(2 .. 4))

Pour i := 0 à 15 Faire
        Si mc[index][i].bit_validite = '1' Alors
                Si mc[index][i].etiquette = @(5..10) Alors
                        // MAJ du tag
                        mc[index][i].tag_LRU := mc[index][i].tag_LRU + 1

                        retourner Faux
                FinSi
        FinSi
FinPour

retourner vrai

Fin
```

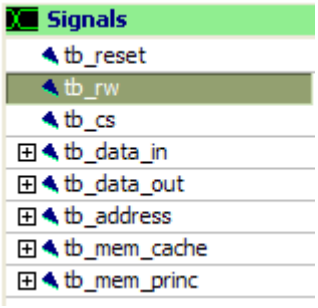
```
Charger_En_Memoire_Cache_LRU( t_cache mc , main_memory mm    , adresse @ ,
t_bloc b) :
Variables :
        index , pos_LRU : entier
Debut
        index := conv_bin2dec(@(2..4))

pos_LRU := 0
Pour i := 0 à 15 Faire
        // Cas ou la ligne n'est pas affmc[index]tée
        Si mc[index][i].bit_validite = '0' Alors
                mc[index][i].bloc := b
                mc[index][i].etiquette := @(5..10)
                mc[index][i].tag_LRU := 0
                mc[index][i].bit_validite := '1'
                pos_LRU := -1
                break // SORT DE LA BOUCLE POUR
        Sinon
                Si( mc[index][i].tag_LRU > mc[index][pos_LRU].tag_LRU)Alors
                        pos_LRU := i
                FinSi
        FinSi
FinPour

// Si toutes les lignes étaient affectées
Si pos_LRU <> -1 Alors
                // Si on a modifié le contenu du bloc, on le recharge en mm avant de l'écraser
                Si mc[index][pos_LRU].maj = '1' Alors
                        @2 := concat( mc[index][pos_LRU].etiquette , @(2 .. 4)  )
                        pos_mm := conv_bin2dmc[index]( @2 )
                        mm(pos_mm) := mc[index][pos_LRU].bloc
                FinSi
                // Ensuite on charge la nouvelle donnée
                mc[index][pos_LRU].bloc := b
                mc[index][pos_LRU].etiquette := @(5..10)
                mc[index][pos_LRU].tag_LRU := 0
FinSi

// MAJ des tags
Pour i := 0 à 15 Faire
        Si mc[index][i].bit_validite = '1' Alors
                mc[index][i].tag_LRU := mc[index][i].tag_LRU + 1
        FinSi
FinPour

Fin
```

```
Charger_En_Memoire_Cache_LFU ( t_cache mc , main_memory mm    , adresse @ ,
t_bloc b) :
variables :
        index , pos_LFU : entier
Debut

        index := conv_bin2dec(@(2..4))

pos_LFU := 0
Pour i := 0 à 15 Faire
        // Cas ou la ligne n'est pas affectée
        Si mc[index][i].bit_validite = '0' Alors
                mc[index][i].bloc := b
                mc[index][i].etiquette := @(5..10)
                mc[index][i].tag_LFU := 1
                mc[index][i].bit_validite := '1'
                pos_LFU := -1
                break // SORT DE LA LOOP POUR
        Sinon
                Si( mc[index][i].tag_LFU < mc[index][pos_old].tag_LFU)Alors
                        pos_LFU := i
                FinSi
        FinSi
FinPour

Si pos_LFU <> -1 Alors
                // Si on a modifié le contenu du bloc, on le recharge en mm avant de l'écraser
                Si mc[index][pos_LFU].maj = '1' Alors
                        @2 := concat( mc[index][pos_LFU].etiquette , @(2 .. 4)  )
                        pos_mm := conv_dmc[index]2bin( @2 )
                        mm(pos_mm) := mc[index][pos_LFU].bloc
                FinSi
                // Ensuite on charge la nouvelle donnée
                mc[index][pos_LFU].bloc := b
                mc[index][pos_LFU].etiquette := @(5..10)
                mc[index][pos_LFU].tag_LFU := 1
FinSi


Fin
```

Tests du composant

1. Structure

Pour vérifier le bon fonctionnement de notre mémoire avec cache, on lance le test-bench et l'on affiche le contenu de la mémoire cache, ainsi que les entrées/sorties.

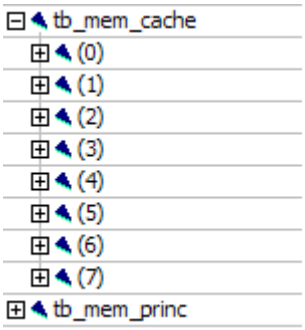


Les signaux affichés sont :

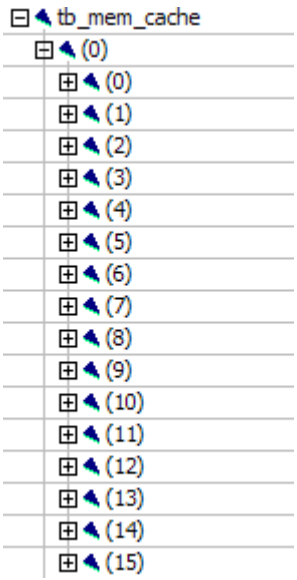
- Le reset qui réinitialise la mémoire dés qu'il est activé
- Le bit rw qui dit s'il s'agit d'une opération de lecture ou d'écriture
- Le Chip-Select qui active la mémoire cache
- Le data_in qui correspond au mot à écrire
- Le data_out qui est un mot de 8 bits et qui correspond au résultat de la lecture
- La mémoire cache
- La mémoire principale

Composition de la mémoire cache :

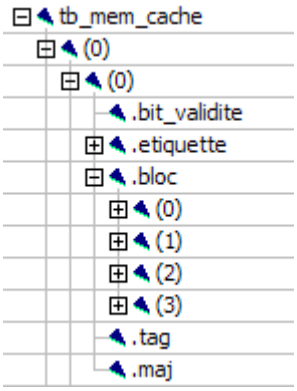
Les 8 ensembles de la mémoire cache :



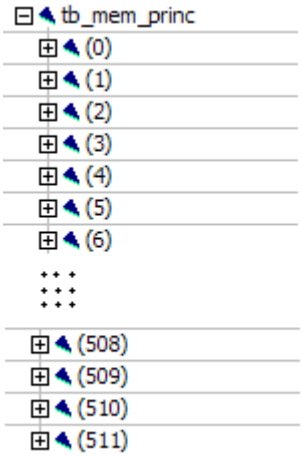
Les 16 blocs d'un ensemble :



Détail d'un bloc :



Composition de la mémoire principale :

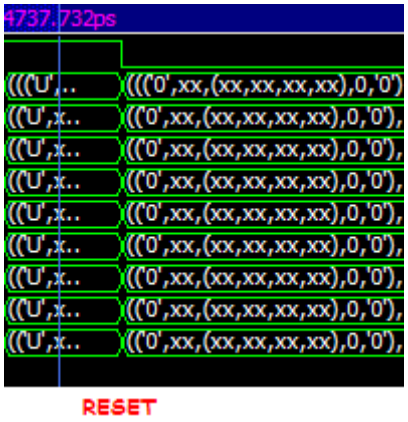


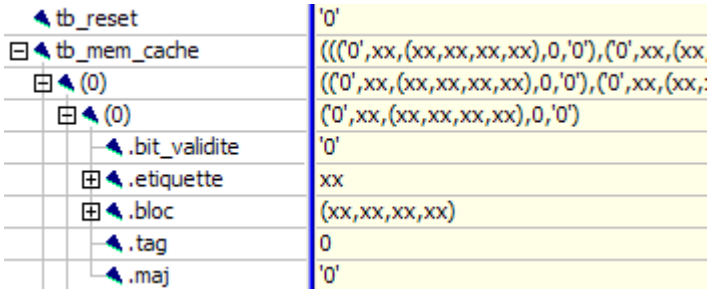
2. Initialisation

On initialise la mémoire cache et la mémoire principale en activant le 'reset'. (actif sur niveau bas).

Initialisation de la mémoire cache :

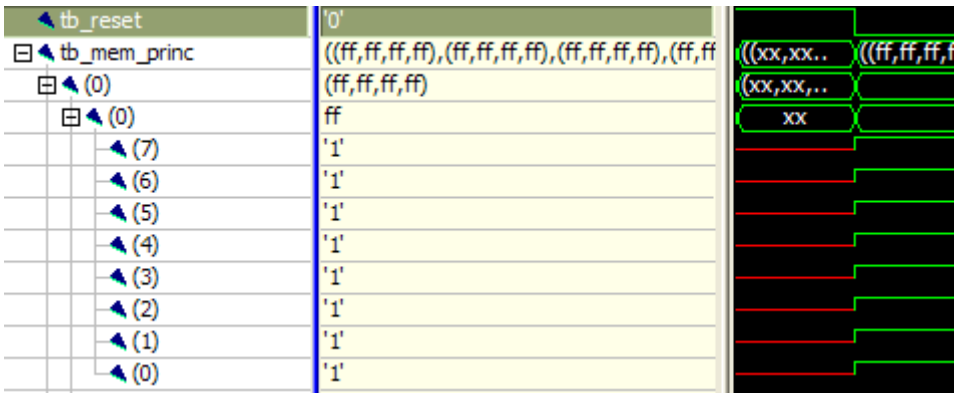
Le contenu initial de la mémoire cache en soit n'est pas important, car il ne sera jamais lu. En effet, le bit de validité empêche tout accès à un bloc de la mémoire cache qui n'a pas encore été chargé. C'est pourquoi on initialise les blocs de la mémoire cache en affectant à chaque std_logic la valeur indéterminée : 'X'





De plus, le bit de validité est initialisé à zéro car pour l'instant aucun bloc n'est chargé en mémoire cache. Le tag de gestion de cache est initialisé à zéro (cas de la stratégie LFU). On initialise l'étiquette de la même manière que les blocs, avec des 'X'.

Initialisation de la mémoire principale :



Tous les bits de la mémoire principale sont initialisés à la valeur '1'.

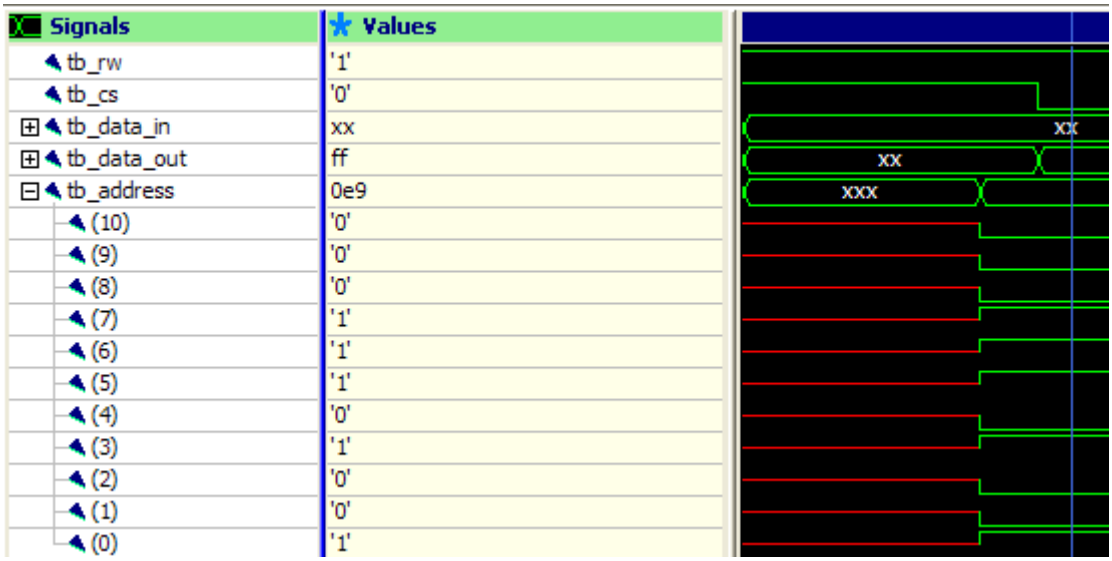
3. Chargement en mémoire et lecture

Maintenant que la mémoire est correctement initialisée, essayons de charger un bloc de la mémoire cache vers la mémoire principale. Nous allons vérifier deux choses :

- ✓ Est-ce que la donnée est chargée en mémoire à l'endroit attendu ?
- ✓ Est-ce que les tags de gestion sont correctement mis à jour ?

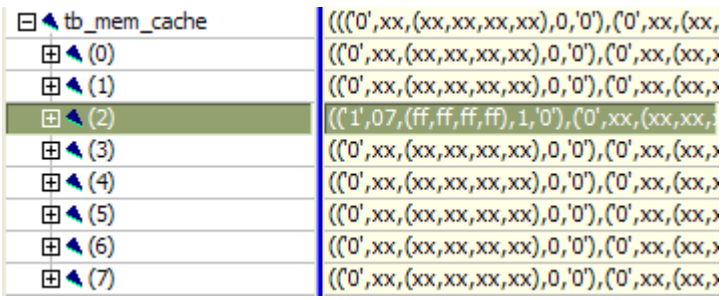
Supposons que l'on veuille lire le mot dont l'adresse est la suivante :

1 0	Offset	Il s'agit du mot n°1 dans le bloc
0 1 0	Index	Le bloc est situé dans l'ensemble n°2
1 1 1 0 0 0	Etiquette	L'étiquette associée au mot est : '111000'



Lecture d'un mot : le mode 'read' est activé et l'adresse est renseignée. Au moment où le chip-select passe à l'état bas (front descendant), le data_out prend la valeur du mot lu.
Le mot lu est (ff)$(FF)_8 = (11111111)_2$

Vérifions que le mot est bien chargé en mémoire vive. Il doit se trouver dans l'ensemble n°2.



En effet, le bit de validité de la première ligne de l'ensemble n°2 est passé à '1', ce qui signifie qu'un bloc a été chargé en mémoire cache.

Vérifions maintenant que le bloc a été correctement chargé :

(2)	(('1',07,(ff,ff,ff,ff),1,'0'),('0',xx,(
(0)	('1',07,(ff,ff,ff,ff),1,'0')
.bit_validite	'1'
.etiquette	07
(5)	'0'
(4)	'0'
(3)	'0'
(2)	'1'
(1)	'1'
(0)	'1'
.bloc	(ff,ff,ff,ff)
.tag	1
.maj	'0'

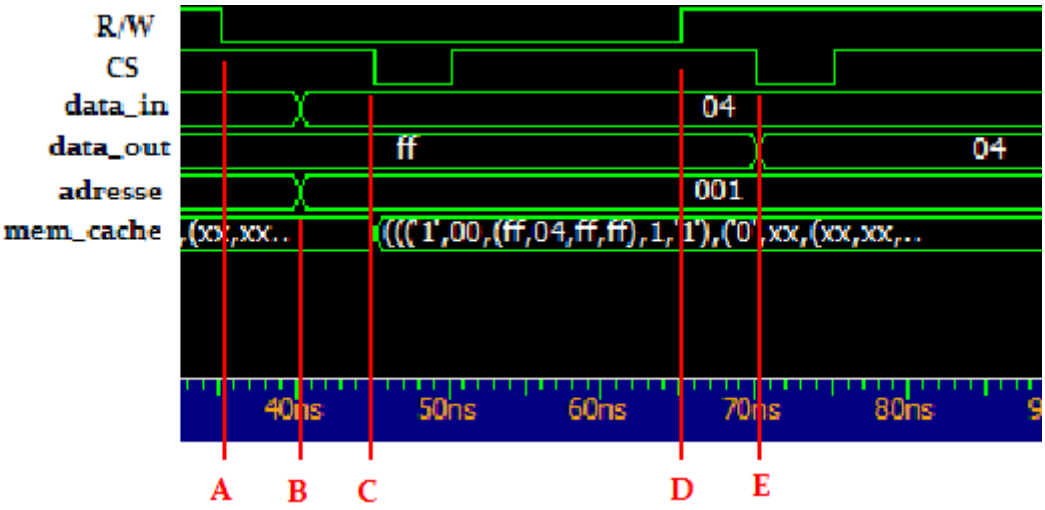
En effet, les 4 mots du bloc contiennent la valeur (FF), ce qui est logique puisque tous les mots de la mémoire principale on été initialisés à cette valeur.

De plus, l'étiquette a été mise à jour et correspond bien à l'étiquette que nous avions observée dans l'adresse.

4. Ecriture

Testons maintenant l'écriture en mémoire. On veut écrire le mot $(0000100)_2 = 4_{10}$

Voici le chronogramme correspondant :



L'opération se fait en 5 temps :

A. On passe en mode écriture
B. On charge l'adresse et le mot à écrire dans data_in
C. Activation du chip-select qui entraine l'écriture en mémoire cache
D. Passage en mode lecture
E. Activation du chip-select qui entrain la lecture

On lit le mot que l'on vient d'écrire en mémoire cache et l'on vérifie qu'il a bien été écrit en lisant à la même adresse. On observe en effet que data_out = 4, ce qui correspond au mot écrit.

Il reste une vérification à faire : est-ce que les modifications apportées lors d'une écriture sont effectuées également dans la mémoire principale ? Il faut savoir que notre stratégie de mise à jour après une écriture est le write-through, c'est-à-dire que l'on ne met à jour que lorsque le bloc en question est remplacé dans la mémoire cache. C'est pourquoi il nous faut remplir entièrement l'ensemble courant afin que le bloc soit écrasé et que l'on puisse vérifier s'il a bien été mis à jour. On choisit de remplir l'ensemble n°2.

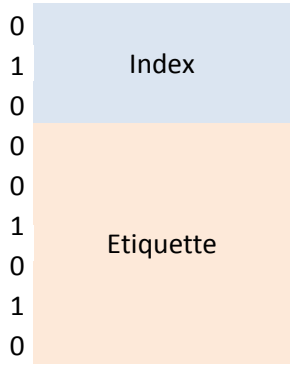
(2)	(('1',14,(ff,43,ff,ff),16,'1'
(0)	('1',14,(ff,43,ff,ff),16,'1')
(1)	('1',16,(ff,43,ff,ff),15,'1')
(2)	('1',37,(ff,43,ff,ff),14,'1')
(3)	('1',3f,(ff,43,ff,ff),13,'1')
(4)	('1',20,(ff,43,ff,ff),12,'1')
(5)	('1',30,(ff,43,ff,ff),11,'1')
(6)	('1',23,(ff,43,ff,ff),10,'1')
(7)	('1',32,(ff,43,ff,ff),9,'1')
(8)	('1',26,(ff,43,ff,ff),8,'1')
(9)	('1',17,(ff,43,ff,ff),7,'1')
(10)	('1',25,(ff,43,ff,ff),6,'1')
(11)	('1',05,(ff,43,ff,ff),5,'1')
(12)	('1',12,(ff,43,ff,ff),4,'1')
(13)	('1',36,(ff,43,ff,ff),3,'1')
(14)	('1',33,(ff,43,ff,ff),2,'1')
(15)	('1',22,(ff,43,ff,ff),1,'1')

Voici l'état de l'ensemble n°2 de notre mémoire cache, une fois rempli. Tous les bits de validité ont la valeur 1.

On observe au passage la valeur des tags de gestion (ici stratégie LRU). Au prochain ajout, la ligne qui doit être remplacée est celle qui a été utilisée le plus longtemps, c'est-à-dire celle qui a le plus grand tag. Ici, c'est la première ligne, dont le tag de gestion vaut 16.

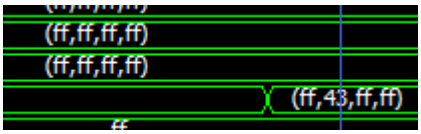
On charge à nouveau un bloc dans cet ensemble et on vérifie en mémoire principale si la valeur du bloc remplacé a bien été mise à jour :

Voici l'adresse du bloc reconstituée (étiquette+index) :



Ce qui fait, converti en binaire :
$2^1+2^5+2^7 = 162$
On doit donc observer une mise à jour dans le bloc 162 de la mémoire principale. Vérifions sur le chronogramme :

(162)	(ff,43,ff,ff)
(0)	ff
(1)	43
(2)	ff
(3)	ff



(2)	(('1',27,(ff,81,ff,ff),1,'1'),('
(0)	('1',27,(ff,81,ff,ff),1,'1')
(1)	('1',16,(ff,43,ff,ff),16,'1')
(2)	('1',37,(ff,43,ff,ff),15,'1')
(3)	('1',3f,(ff,43,ff,ff),14,'1')
(4)	('1',20,(ff,43,ff,ff),13,'1')
(5)	('1',30,(ff,43,ff,ff),12,'1')
(6)	('1',23,(ff,43,ff,ff),11,'1')
(7)	('1',32,(ff,43,ff,ff),10,'1')
(8)	('1',26,(ff,43,ff,ff),9,'1')
(9)	('1',17,(ff,43,ff,ff),8,'1')
(10)	('1',25,(ff,43,ff,ff),7,'1')
(11)	('1',05,(ff,43,ff,ff),6,'1')
(12)	('1',12,(ff,43,ff,ff),5,'1')
(13)	('1',36,(ff,43,ff,ff),4,'1')
(14)	('1',33,(ff,43,ff,ff),3,'1')
(15)	('1',22,(ff,43,ff,ff),2,'1')

Au passage, vérifions que la bonne ligne a bien été remplacée (la première) et que les tags LRU sont bien mis à jour (ici, la nouvelle valeur est '1' et tous les autres ont été incrémentés)