2025-09-17 DDR

Table of contents

Objectif de la note

Cette note donne quelques éléments d'information et de justification concernant le simulateur utilisé dans les travaux de recherche automatique d'interférences.

Elle décrit brièvement

- le comportement de certains composants liés à la gestion mémoire (caches, DDR, contrôleur DDR, interconnect),
- le fonctionnement du simulateur et certaines améliorations que nous pourrions apporter à celui-ci en regard des caractéristiques de la plateforme T2080.

Rappels concernant le comportement de la mémoire DDR et des contrôleurs mémoire

Le comportement d'une DDR (DDR3) est spécifié dans le standard JEDEC JESD79-3C [1], qui est un document comportant plus de 200 pages. Une synthèse est disponible dans la thèse [2], Section 3.3. L'intérêt de ce document est qu'il vise explicitement l'analyse d'interférences. Les éléments d'information qui suivent sont issus de ce document.

DDR

Les points les plus importants sont rappelés dans la liste ci-dessous :

- La mémoire est décomposée en blocs appelé "banques" (banks) qui peuvent être accédées de manière simultanées.

- Chaque banque est associée à un "row buffer" qui joue le rôle de cache pour une ligne mémoire (row). Lors d'un changement de ligne, le "row buffer" doit être rechargé.
  - Les données concernant l'organisation mémoire (le passage d'une adresse physique aux "adresses " des éléments micro-architecturaux) ne sont pas toujours disponibles et doivent être parfois obtenues par rétro-engineering en exploitant notamment les latences observées.
  - Par exemple, pour le SITARA AM5278, une adresse physique est décodée ainsi : `RowSize | RankSize | BanksSize | ColumnSize | BusSize`
  - Il est parfois possible d'obtenir des informations à partir de compteurs de performance.
- Concernant les latences, on peut retenir les points suivants :
  - l'accès à différentes lignes de la même banque conduit à d'importants délais en raison du coût de changement de ligne (*row miss*)
  - les accès simultanés à une ligne ouverte d'une banque donnée (i*ntra-bank interference*) entraînent plus de délais que l'accès simultané à différentes banques (*inter-bank interference*).
- La DDR me en oeuvre les commandes suivantes
  - Write (**WR**)
  - Read (**RD**)
  - Active (**ACT**) pour activer une ligne fermée (closed row) ; cette opération chaque le buffer de ligne avec une nouvelle ligne. Elle dure tRCD
  - Precharge (**PRE**) pour désactiver une ligne ouverte pour une banque donnée ; cette opération écrit les information du buffer de ligne (row buffer) courant vers la mémoire. Elle dure tRP.
  - Refresh (**REF**). Cette opération a pour objectif de rafraichir les cellules mémoire.
- Les acccès mémoire se font en burst de longueur BL x 8 bytes. Etant donnée que la mémoire est de type Double Rate (DDR : elle fournit des données sur les fronts montants et descendants de l'horloge), la durée d'un burst est BL/2. Les données du burst n'arrivent pas instantanément, mais après une latence CAS Latency (CL) pour une lecture (RD) et une latence CAS Write Latency (CL) pour une écriture (WR).
- À chaque changement de ligne pour une banque, une opération PRE de durée tRP et ACT de durée tRCD sont réalisées. (voir la figure 3.8).
- Périodiquement, une opération de rafraichissement mémoire (REF) doit avoir lieu. La période est tRFC. Lors de cette opération, tous les row buffers des banques mémoire doivent être préchargés

- La machine à états de la DDR est la suivante :

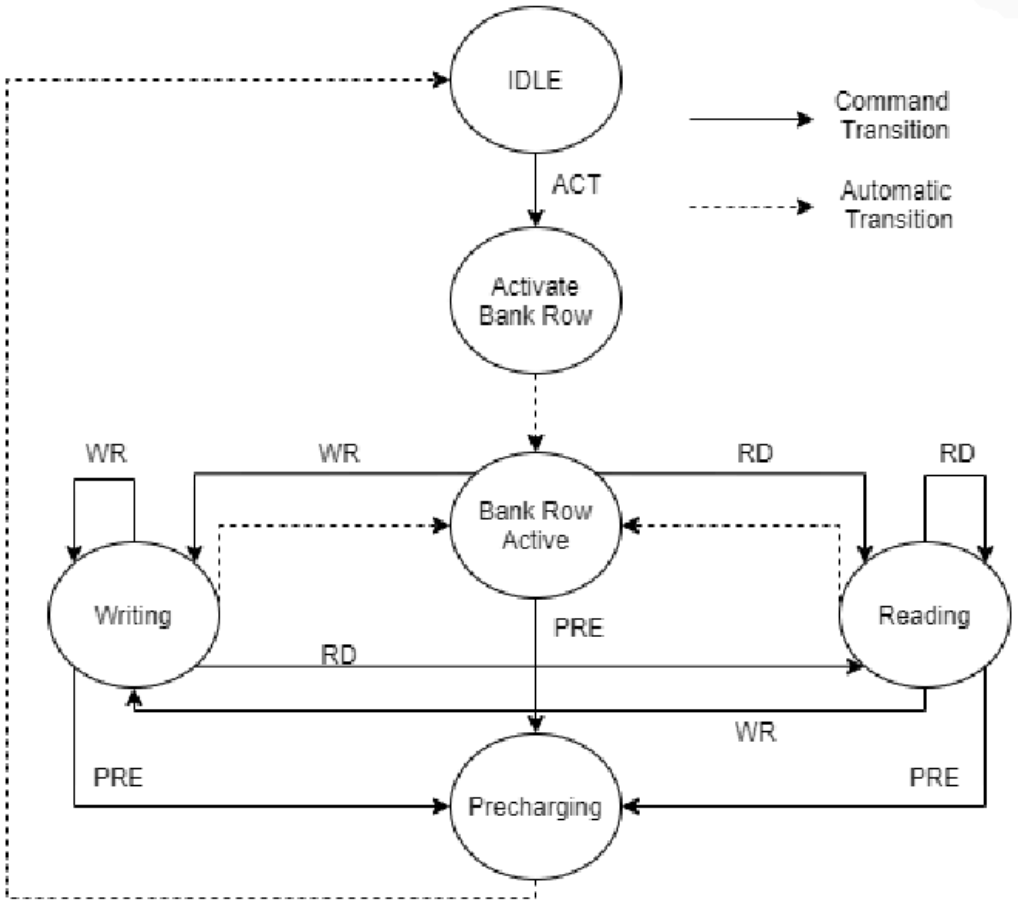


Figure 3.9: Simplified state diagram assuming a single bank.

- La durée minimale entre deux RDs ou Wrs est de tCCD (normalement égal à BL/2)
- Lors d'une transition RD => WR, 2 cycles
- Lors d'une transition WR => RD, pénalité de tWTR
- Voir le diagramme ci-après:

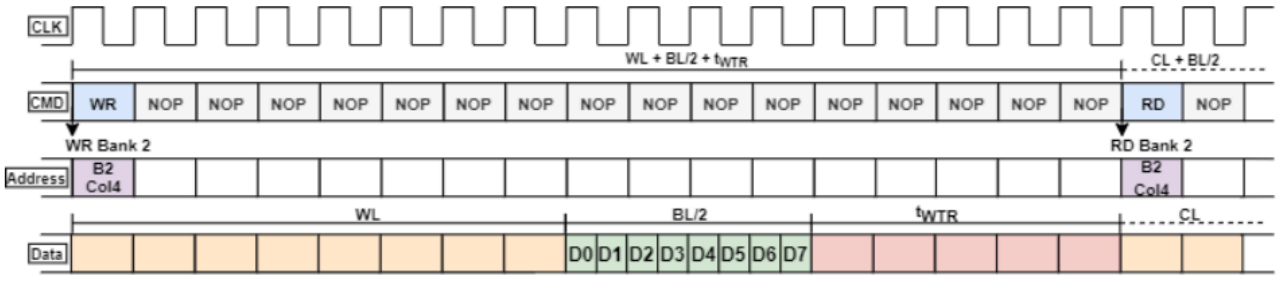


Figure 3.10: WR to RD transmission example.

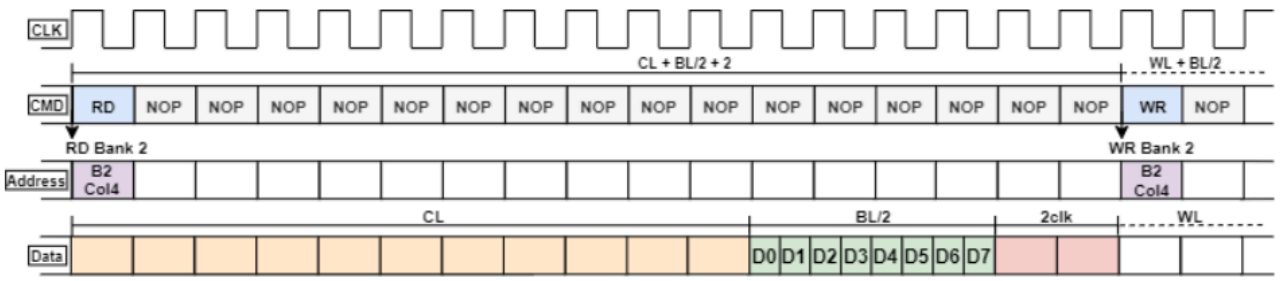


Figure 3.11: RD to WR transmission example.

- Autres contraintes
  - ACT => ACT inter-bank : Le délai entre deux ACT consécutifs ne doit pas dépasser tRRD
  - ACT => PRE : pour une séquence intra-bank, le dt entre ACT et PRE ne doit pas excéder tRAS
  - ACT => ACT intra-bank : Le délai entre deux ACT vers la même bank bne doit pas dépasser tRC
  - RD/WR => PRE
    - Le délai entre RD et PRE doit être au minimum de tRTP
    - Le délai entre WR et PRE doit être au minimum de tWR
  - Il faut émettre 8 REF dans une fenêtre de taille 8xtREFI
- Se référer au document [2] concernant les optimisations pouvant être mises en oeuvre pour optimiser les latences.

Contrôleur mémoire

- Le contrôleur mémoire est en charge de réordonner les requêtes d'accès mémoire pour maximiser le débit d'accès mémoire.
- Il met en œuvre plusieurs files d'attentes : une file pour les commandes, une file pour les données en lecture, une file pour les données en écriture
- Certains chip mettent en oeuvre plusieurs contrôleurs.
- Règles de priorisation
  - Prioriser les RD sur les WR de façon à minimiser les temps d'attente, mais en préservant la cohérence mémoire (WR @x RD @x doit être exécuté dans cet ordre)
  - Prioriser les lignes qui sont déjà dans les row_buffer afin d'éviter la pénalité liée à la réalisation d'une séquence PRE=>ACT.
  - Traitement des RD et WR en "batch" de façon à éviter l'overhead lié à la transition entre RD et WR
- Il existe aussi des mécanismes permettant de prévenir les problèmes de famine.

Etat du simulateur et proposition d'améliorations

Le code du simulateur est disponible sur le serveur gitlab du projet, à l'adresse https://forge-csec.pf.irt-saintexupery.com/AIxIA/memory_simulator. Le code est documenté. Il faut donc se référer au code pour plus d'information.

> ⚠ **T2080**
>
> D'une manière générale, le T2080 est un processeur très complexe. Vouloir modéliser finement l'ensemble de ses mécanismes est clairement hors de portée, sans compter le fait que la documentation disponible hors NDA est insuffisante.
> Il faut donc limiter le modèle de simulation aux mécanismes les plus importants / génériques de façon à valider l'approche sur un modèle "à peu près" représentatif.

Modélisation des cœurs (class "core")

Etat du simulateur

La simulation des cœurs est limitée à la prise en compte des instructions effectuant des accès mémoire (load et store). En d'autres termes, le jeu d'instructions est réduit à des opérations de lecture ("read") et d'écriture ("write") en mémoire puisque, en définitive, seuls les accès mémoire sont simulés.

Pour chaque coeur, une simple boucle modélise le cycle "fetch et execute".

À chaque itération de la boucle (donc, chaque cycle processeur), le coeur peut

- attendre la fin d'une opération de lecture ;
- exécuter un "read" (load) ;
- exécuter un "write" (store).
- exécuter une instruction n'effectuant pas d'accès mémoire, ce qui, dans le cas du simulateur, revient à ne rien faire ;

Le choix entre les trois types d'opération (tout instruction n'effectuant pas d'accès mémoire, un LD ou un ST) est choisi aléatoirement.

Ce modèle est évidemment très simplifié par rapport au fonctionnement réel du coeur du T2080 (voir ci-après)

- Toute instruction prend un cycle
- L'attente de l'achèvement d'un accès mémoire en lecture pour poursuivre l'exécution du code ne correspond pas à la réalité car le processeur dispose de divers mécanismes permettant justement de masquer ces latences. Cette attente n'est nécessaire que pour préserver les vraies dépendances enter accès (par ex. une séquence LD @a, WR @a doit être préservée lors de l'exécution pour maintenir la sémantique du programme).

Le T2080

Le coeur e6500 du T2080 est un coeur complexe [3]. Il peut décoder 2 instructions et achever deux instruction par thread matériel (il y a 2 threads matériel par coeur) et par cycle. L'exécution des instruction se fait *out-of-order*. Il dispose de plusieurs unités de traitement lui permettant d'exécuter jusqu'à 12 instructions en parallèle. Deux unités load/store (une par thread) permettent de masquer les latences mémoire (ce qui signifie notamment que l'ordre des accès "perçus" au niveau des bus n'est pas celui des instructions). À noter que certains événements liés au fonctionnement de cette unité peut être observés via les registres de performance.

On peut aussi mentionner l'existence d'une MMU (Memory Management Unit) qui détermine l conversion d'une adresse logique (telle que vue par le programmeur) en adresse physique (telle que vue par le matériel).

Améliorations possibles

Le simulateur ne représente que de façon très simpliste le fonctionnement d'un coeur e6500.
La liste des mécanismes mis en oeuvre dans le coeur est longue et leur complexité est grande. Il n'est donc absolument pas envisageable de les reproduire dans notre simulateur.

Un simulateur tel que [gem5](gem5: The gem5 simulator system) modélise ces mécanismes (par ex. l'exécution out-of-order avec le modèle [O3CPU](gem5: Out of order CPU model)), même s'il n'existe pas de modèle spécifique du coeur e6500.

A minima, le simulateur pourrait être amélioré en vérifiant, à chaque cycle, si l'exécution d'une instruction est compatible des opérations mémoire en cours de réalisation. Ceci permettrait notamment de simuler le parallélisme d'exécution des requêtes en lecture.

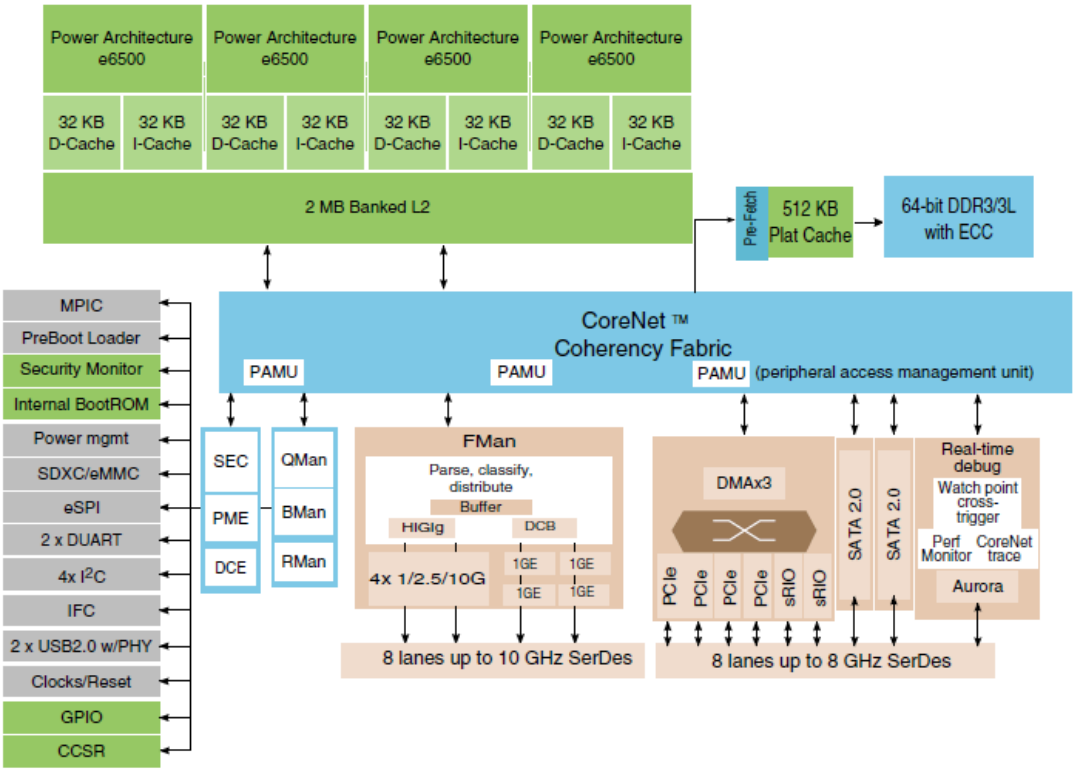
Modèle des caches

Etat simulateur

- Chaque niveau de cache (classe "ClassLevel") est configurable en taille totale, taille d'une ligne de cache et associativité.
- Par défaut le comportement est de type "write_back", c'est à dire que l'écriture dans la mémoire de niveau inférieur a lieu lors de l'éviction de la ligne de cache. Lors de l'opération d'écriture, la ligne est simplement marquée "dirty" et lors de l'éviction de la ligne, celle-ci est écrite dans la mémoire de niveau inférieur .
- Il est aussi est possible de mettre en oeuvre un comportement de type "write-through" dans lequel l'écriture dans la mémoire de niveau inférieur a lieu immédiatement.
- La gestion de l'éviction des lignes de cache est réalisée par un PLRU (classe PLRU) dont le rôle est de déterminer la ligne à remplacer en fonction de l'état courant du cache. Idéalement, on souhaiterait mettre en oeuvre un comportement de type LRU (Least Recently Used), qui consisterait à éliminer la ligne utilisée le moins récemment afin d'exploiter au mieux le principe de localité. Cependant, cette stratégie est coûteuse à mettre en oeuvre et on préfère souvent utiliser un mécanisme plus simple appelé Pseudo-LRU qui repose sur un arbre binaire.
- Cet algorithme comporte :
  - une fonction permettant de maintenir la structure de données (arbre binaire) qui permettra de choisir le prochaine ligne à évincer en fonction des accès mémoire ("update_on-access")
  - une fonction permettant de choisir la prochaine ligne de cache à évincer ("get_victim") à partir de l'information contenue dans l'arbre binaire.
- Le cache de dernier niveau (L2 dans le code actuel) est partagé par les deux hiérarchies mémoire.
- Il n'y a pas de mécanisme de gestion de la cohérence de cache.

T2080

- Sur le T2080, la structure et les caractéristiques des caches sont données ci-après. Aucune information n'est donnée sur la politique de gestion (pLRU ou autre).



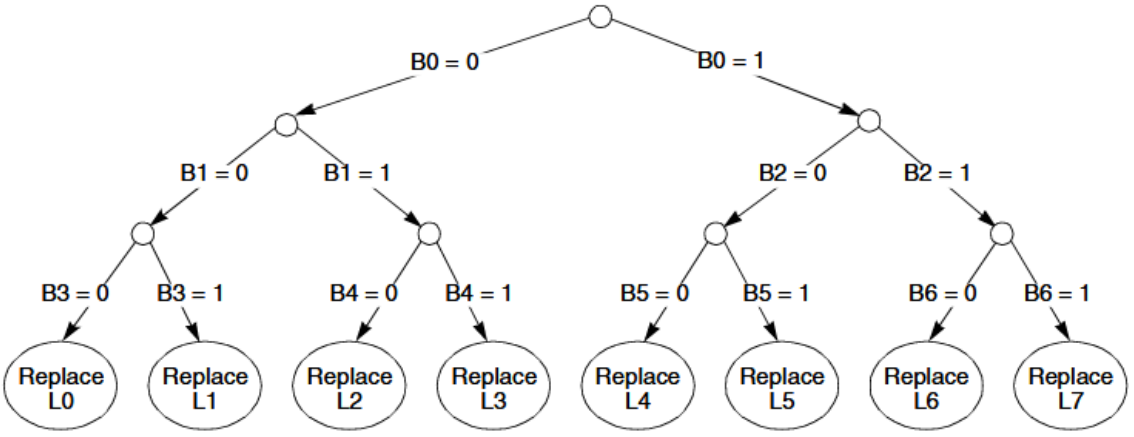
Les caractéristiques des caches du T2080 sont les suivantes (selon le T2080 Reference Manual [2], plus de détails sont donné dans l'E6500 Reference Manual [3]) :

- L1 caches, running at core frequency
  - 32 KB instruction, 8-way set-associative
  - 32 KB data, 8-way set-associative
  - Each with data and tag parity protection
- L2 cache
  - 2 MB cache with ECC protection (data, tag, & status)
  - Operates at same frequency as the CPUs in the cluster
  - 64-byte cache line size
  - 16 way, set associative
  - **4 banks, supporting up to four concurrent accesses.**
    - Each bank can be configured as exclusive to a vCPU or shared
    - A vCPU can exclusively own one or more banks
    - Ways in each bank can be configured in one of several modes
    - Flexible way partitioning per vCPU
      - I-only, D-only, or unified
  - Supports direct stashing of datapath architecture data into L2

La description détaillée du comportement des caches du T2080 est donnée dans le manuel de référence du coeur e6500 [3, §5.7]. À nouveau, le comportement est complexe et, même si le coeur met en oeuvre -- par exemple -- un algorithme de remplacement de type PLRU (voir figures suivantes), les différences de fonctionnement sont nombreuses.

Table 5-3. L1 PLRU replacement way selection

PLRU Bits						Way Selected for Replacement
B0	0	B1	0	B3	0	L0
	0		0		1	L1
	0		1	B4	0	L2
	0		1		1	L3
	1	B2	0	B5	0	L4
	1		0		1	L5
	1		1	B6	0	L6
	1		1		1	L7



- Le comportement du cache L2 ([3, §5.8] est lui aussi plus beaucoup complexe que ce qui est mis en œuvre dans le simulateur. Au delà de la complexité, un point majeur réside dans le fait que le cache L2 est unifié [3, §5.8.3.3], c'est-à-dire qu'il contient données **et** instructions.
- Le cache L2 met aussi en oeuvre un mécanisme de cohérence de type MESI dont le comportement est brièvement rappelé sur la figure ci-après :

Table 5-5. Cache line state definitions

Name	Description
Modified (M)	The line in the cache is modified with respect to main memory. It does not reside in any other coherent cache.
Exclusive (E)	The line is in the cache, and this cache has exclusive ownership of it. It is in no other coherent cache and it is the same as main memory. This processor may subsequently modify this line without notifying other bus masters.
Shared (S)	The addressed line is in the cache, it may be in another coherent cache, and it is the same as main memory. It cannot be modified by any processor.
Invalid (I)	The cache location does not contain valid data.

- On rappelle que le mécanisme de cohérence de cache permet d'assurer une vue cohérente de la mémoire entre les différents coeurs.
  - Dans le cas où on a plusieurs coeurs disposant chacun d'un cache local (par ex., le cache L1), les données correspondant à une même adresse en mémoire centrale peuvent apparaître dans plusieurs d'entre-eux.
  - Sans protocole particulier, si un coeur vient à modifier une donnée en mémoire centrale, les données correspondant à cette même adresse situées dans les caches des autres coeurs vont devenir incohérentes.
  - Cette incohérence peut être évitée par des moyens logiciels ou matériels (mécanismes de cohérence de cache). Dans le cas d'une gestion matérielle, une solution est de mettre en oeuvre un mécanisme de "snooping" qui consiste à invalider l'entrée de cache correspondant à une adresses en mémoire centrale lorsqu'un autre coeur vient à en modifier le contenu.

    - Ce mécanisme repose (de façon simpliste) sur l'observation des transactions ayant lieu sur l'interconnect. Lorsqu'un coeur modifie une valeur en mémoire ce mécanisme entraine (par ex.) l'invalidation automatique des entrées des caches correspondantes. L'invalidation est une solution ; d'autres "protocoles de cohérence de cache" existent. Ces opérations induisent un *overhead* sur le trafic de l'interconnect ; l'impact devrait être évalué. **Dans un modèle simplifié, on peut ignorer ce mécanisme en considérant qu'il y a une ségrégation de niveau logiciel : deux coeurs n'accèdent pas à des zones mémoires partagées.**
- Outre les caches intégrés au coeur e6500, il existe aussi un CoreNetPlatform cache (CPC) de **512Kb**** :
  The CoreNet platform cache (CPC) is a CoreNet-compliant target device that functions as a general purpose write-back cache, I/O stash and memory mapped SRAM device, or a combination of these functions. As a general purpose cache, it manages allocations and victimizations to improve read latency and bandwidth over accesses to backing store (for example, DRAM). As an I/O stash, it can accept and allocate writes from an I/O device in order to reduce latency and improve bandwidth for future read operations to the same address. As an SRAM device, it acts as a low-latency, high-bandwidth memory that occupies a programmable address range.
  The CPC connects between the CoreNet coherency fabric (CCF) and the memory controller in an inline configuration and cannot function as a lookaside cache. Therefore, the CoreNet platform cache can only cache address ranges that are present in the memory controller behind it. This limitation does not apply to SRAM, which does not have backing store.
    - Ce cache est configurable. On lit notamment :
      *Configurable pseudo-least recently used (PLRU), streaming PLRU with aging, streaming PLRU without aging, and first-in/first-out (FIFO) replacement policies with programmable allocation policy and update options"

Possibilités d'amélioration

Etant donnée la complexité du T2080 et l'impact potentiel d'un seul de ses mécanismes sur l'état du système mémoire et sur les latences (effets "en cascade"), une amélioration ponctuelle ou partielle du modèle de simulation n'en améliorera pas la représentativité de façon significative.

Aussi, nous ne proposons pas d'amélioration pour l'instant.

Modèle de la DDR (class DDRMemory)

Etat actuel

- La mémoire est constituée de plusieurs banques ("banks") dont chacune dispose d'un "row buffer" (une forme de cache). Chaque banque est gérée par une machine à états temporisée qui reprend celle décrite dans [1] (voir figure plus haut)
- L'état des banques de la DDR détermine les dates de complétion des opérations. Les requêtes sont émises par le contrôleur DDR.

Le T2080

La datasheet du T2080 donne les informations suivantes (voir [2 §1.3.6] pour la liste complète) :

- One 32-/64-bit DDR3/3L SDRAM memory controllers with ECC and interleaving support Memory pre-fetch engine
- Support x8 and x16 memory widths
- Programmable support for single-, dual-, and quad-ranked devices and modules
- Support for both unbuffered and registered DIMMs
- 4 chip-selects
- 40-bit address support, up to 1 TB memory
- The SoC can be configured to retain the currently active SDRAM page for **pipeline burst accesses**. Page mode support of up to 64 simultaneously open pages can dramatically reduce access latencies for page hits. Depending on the memory system design and timing parameters, page mode can save up to ten memory clock cycles for subsequent burst accesses that hit in an active page.

Possibilités d'amélioration

- Vérification du respect des contraintes temporelles (une seule vérification est réalisée dans le contrôleur)
- Traitement des *bursts* qui ont un impact majeur sur les latences.

Modèle du contrôleur mémoire

Etat actuel

- (Le modèle de la DDR et celui du contrôleur mémoire sont couplés.)
- À chaque cycle :
  - le contrôleur détermine les requêtes dont le traitement est achevé (c-à-d la date courant est supérieure à la date de complétion prévue). Dans ce cas, et pour une requête de type "read", il appelle un "callback" qui signale au coeur que l'accès est achevé? C'est ce callback qui est utilisé par le coeur pour débloquer l'exécution d'une nouvelle instruction. Aucun callback n'est prévu dans le cas d'une requête en écriture.
  - le contrôleur traite les requêtes situées dans sa file d'entrée. Pour chaque requête, il détermine si celle-ci peut être traitée par la DDR selon son état (géré par la machine à état de la DDR) en assurant les contraintes de délais minimum (cf [1]). Puis il détermine la "meilleure" requête à traiter en fonction d'un classement établi pour privilégier les requêtes aboutissant à un "row hit", les requêtes WR sur les requêtes RD et l'ordre d'arrivée (FIFO). Enfin, il transmet la "meilleure" requête à la DDR.
- Le modèle reprend les "grandes lignes" des mécanismes généraux décris dans [1].

Eléments concernant le T2080

La datasheet du T2080 donne les informations suivantes (voir [2 §1.3.6] pour la liste complète) :

- One 32-/64-bit DDR3/3L SDRAM memory controllers with ECC and interleaving support Memory pre-fetch engine
- Support x8 and x16 memory widths
- Programmable support for single-, dual-, and quad-ranked devices and modules
- Support for both unbuffered and registered DIMMs
- 4 chip-selects
- 40-bit address support, up to 1 TB memory
- The SoC can be configured to retain the currently active SDRAM page for **pipeline burst accesses**. Page mode support of up to 64 simultaneously open pages can dramatically reduce access latencies for page hits. Depending on the memory system design and timing parameters, page mode can save up to ten memory clock cycles for subsequent burst accesses that hit in an active page.

Possibilités d'amélioration du simulateur

- S'assurer de la cohérence des délais. Pas d'amélioration significative nécessaire.

Modèle de l'Interconnect

Etat actuel

- Le modèle de l'interconnect est trivial. Il consiste à maintenir une file d'attente (FIFO) des requêtes mémoire et à exécuter les requêtes dans l'ordre d'arrivée après un délai correspondant à la une latence d'accès minimale. À noter que l'on rajoute un délai aléatoire de 2 cycles max.
- À chaque cycle d'horloge (fonction "tick"), on dépile un certain nombre de requêtes parmi celles pouvant être exécuté au cycle courant que l'on transmet à la mémoire. Le nombre de requêtes dépilé à chaque cycle est déterminé par la bande passante du bus (test "processed < self.bandwidth") et comptage du nombre de requête traitées (ligne 141).
- Les requêtes qui n'ont pas pu être traitées sont mises dans la file d'attente des requêtes à exécuter aux cycles suivants
- Il est important de noter que, tel que le code d'appel à l'horloge est réalisé, la fréquence d'horloge de l'interconnect et des "cœurs" sont les mêmes. On pourrait introduire des horloges de périodes différentes en appelant les fonction "clock" sur des cycles différents.

Eléments concernant le T2080

- L'interconnect du T2080 est le "CoreNet". Le manuel de référence du T2080 donne très peu d'information sur le sujet (cf. §1.3.5). Il s'agit d'un composant très complexe.
- On peut noter que la fréquence du cornet est de 700MHz

Possibilités d'amélioration du simulateur

- Prise en compte du trafic lié aux activités de *snooping*?
- Prise en compte des mécanismes d'évitement de la famine?
- Prise en compte des différences de fréquence entre les CPU, DDR et corenet?

Références

[1] A. Mascareñas-Gonzàlez, ‘DDR SDRAM Interference minimization via task and memory mapping in a multi-objective optimization context on heterogeneous MPSoCs’, Institut Supérieur de l’Aéronautique et de l’Espace, Toulouse, 2022.
[2] T2080 Integrated Multicore Communications Processor Family Reference Manual’, NXP, Reference manual, https://community.nxp.com/pwmxy87654/attachments/pwmxy87654/qoriq-grl/1488/1/T2080_T2081%20Reference%20Manual%20RevC.1.pdf
[3] E6500RM, e6500 Core Reference Manual - Reference Manual’, NXP, Reference manual. Accessed: Sept. 24, 2025. [Online]. Available: https://www.nxp.com/docs/en/reference-manual/E6500RM.pdf