REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE MOULOUD MAMMERI DE TIZI-OUZOU

FACULTE DE GENIE ELECTRIQUE ET INFORMATIQUE

DEPARTEMENT D'INFORMATIQUE

Mémoire

de fin d'études

En vue de l'obtention du diplôme Master en informatique.

Spécialité : Réseaux, Mobilité et Systèmes embarqués.

Thème :

Evaluation de performances d'une mémoire Scratchpad pour les systèmes embarqués multicores

Proposé et dirigé par:

Mr DAOUI MEHAMMED

Réalisé par :

Melles LOUNIS SONIA

IMOUN NAHIDA

Promotion 2017/2018

Remerciements

Nous tenons à remercier notre promoteur Monsieur Daoui Mehammed, pour la qualité de son encadrement, et son suivi durant toute la durée du projet. Nous tenons, également, à lui exprimer notre reconnaissance pour le temps qu'il nous a consacré, ainsi que pour ses encouragements.

Que les membres du jury trouvent ici nos plus vifs remerciements pour avoir accepter d'honorer par leur jugement notre travail.

Notre profonde gratitude et sincères remerciements vont à tous les professeurs qui nous ont suivis durant notre parcours d'étude.

Dédicace

Je dédis ce modeste travail :

A mes très chers parents ;

A mon frère ;

A mes sœurs ;

Et à toute ma famille et mes amis (es) surtout ma binôme Nahida et tous ceux qui me sont chers au monde.

SONIA

Dédicace

Je dédis ce modeste travail :

A mes très chers parents ;

A mes deux frères ;

Et à toute ma famille et mes amis (es) surtout ma binôme Sonia et tous ceux qui me sont chers au monde.

NAHIDA

Sommaire

Chapitre II : Les Mémoires dans les Systèmes Embarqués

Chapitre III : Techniques de gestion de la mémoire Scratchpad

Chapitre IV : Réalisation

Liste des figures

Liste des tableaux

Introduction Générale

Introduction générale

La conception de système embarqué utilise une méthode dans laquelle la mise en place de la plate-forme matérielle et celle des parties logicielles sont soumises à des interactions très fortes. En effet, ces systèmes (téléphones portables, agendas personnels électroniques) doivent répondre à des contraintes spécifiques de place, de consommation et de performances. Près de la moitié de la surface des systèmes actuels est dédiée à la mémoire, il est extrêmement important de prendre en compte les optimisations et les communications dans les hiérarchies de mémoires ou bien vers les réseaux d'interconnexion processeur/mémoire afin d'améliorer la vitesse d'exécution et de réduire la consommation de ces systèmes.

L'accès à la mémoire constitue un goulet d'étranglement à la vitesse du processeur. D'autant plus que l'on a besoin de plus en plus de mémoire larges mais à temps d'accès forcément plus long et une consommation d'énergie supérieure. Des études ont montré que le système mémoire est responsables de 50% à 75% de la consommation d'énergie et occupe une surface considérable sur circuit **[1].**

Dans ce cadre, les mémoires Scratch-pad (SPM) sont de plus en plus utilisés et commencent à concurrencer les mémoires cache traditionnelles. La mémoire scratch-pad est une petite zone mémoire rapide (SRAM) gérée directement par le logiciel. Ses avantages sont nombreux : taille plus faible de 34% par rapport à un cache, consommation énergétique moins de 40% par rapport au cache **[2].**

L'objectif de notre travail est la simulation d'une architecture embarquée multi-cores dans ce contexte, notre étude est axée sur l'évaluation de performances de la mémoire Scratch-pad par l'implémentation d'un dispositif matériel facilitant la gestion dynamique de cette dernière.
Afin de bien mener cette étude, nous avons structuré notre travail en quatre chapitres :

- Le chapitre I donnera une brève présentation des systèmes embarqués ainsi que leurs contraintes de construction et les différentes architectures embarquées. Ce chapitre présente le domaine où intervient notre étude.

- Le chapitre II quant-à-lui, va décrire les différents types de mémoires dans les systèmes embarqués ainsi que leurs caractéristiques

- Le chapitre III présentera les technologies de la mémoire scratch-pad. On introduira également le nouveau dispositif hardware pour la gestion de la mémoire scratch-pad.

- Le chapitre IV sera consacré à la simulation de notre système ainsi que son évaluation. En effet nous nous intéresserons aux deux méthodes d'allocation pour effectuer une comparaison entre eux afin de prouver la performance de la mémoire scratch-pad en terme de temps d'exécution.

Chapitre I

Introduction aux Systèmes Embarqués

I.1 Introduction

Les systèmes embarqués appartiennent à un domaine en très forte expansion. Néanmoins ils sont de plus en plus soumises à des fortes contraintes fonctionnelles (Puissance de calcul, consommation d'énergie..) et non fonctionnelles (temps de mise sur le marché, forte croissance de la quantité de la production). L'exemple le plus illustrant de ces systèmes est le téléphone mobile. Ce système de communication a révolutionné le monde car il permet aujourd'hui de passer des appels, prendre des photos, et même visualiser des vidéos. Il existe d'autres exemples de systèmes embarqués, tels que contrôle de procédés manufacturiers (usines), la navigation aérienne, supervision de patients, les robots mobiles et autonomes sur lesquels un système de vision temps Réel est implanté.

La plupart des systèmes embarqués sont des systèmes On Chip et sont le résultat de l'évolution technologique qui a permis l'intégration de plusieurs composants électroniques sur une et même puce.

L'architecture de ces systèmes a évolué en faisant intégrer des dizaines, voire des centaines de cœurs de traitement autour d'un réseau de communication sur une seule puce, Il s'agit d'une évolution des SoCs vers les multiprocesseurs sur Puce MPSoCs.

Dans ce chapitre, nous présenterons les concepts des systèmes embarqués, les contraintes de construction des systèmes embarqués et les différentes architectures embarquées.

I.2 Système embarqué

I.2.1 Historique des Systèmes Embarqués [3]

L'un des premiers systèmes embarqués reconnu est le « Apollo Guidance Computer », développé en 1961 au MIT[1] Instrumentation Laboratory. Composé d'environ un millier de circuits intégrés identiques (portes NAND).

Un autre système embarqué est le « Autonetics D-17 guidance computer » qui servait de système de contrôle aux missiles nucléaires américains LGM-30 Minuteman, développé à partir de 1962, il était basé sur des transistors et contenait un disque dur comme support de stockage.

En 1971, Intel produit le premier microprocesseur au monde. Cette puce, le 4004 (4 bits, composé de 2300 transistors), a été conçue suite à une demande de la compagnie japonaise Busicom. Premier circuit générique, personnalisable par logiciel. Le microprocesseur a été un succès immédiat, et son utilisation a augmenté régulièrement au cours de la prochaine décennie.

Une multitude de systèmes embarqués a vu le jour depuis. Nous citerons quelques-uns ci après:

- 1972 : lancement de l'Intel 8008, premier microprocesseur 8 bits (48 instructions, 800kHz).

[1] MIT: Massachusetts Institute of Technology.

- 1974 : lancement du 8080, premier microprocesseur largement diffusé. 8 bits, (64KB d'espace adressable, 2MHz - 3MHz).

- 1978 : création du Z80, processeur 8 bits.

- 1979 : création du MC68000, processeur 16/32 bits.

I.2.2 Définition du système embarqué [4]

Un système embarqué peut être défini comme un système électronique et informatique autonome, qui est dédié à une tâche bien précise, possédant des contraintes d'ordre spatial (taille limitée) et énergétique (consommation limitées). Le terme de « Système Embarqué » désigne aussi bien le **matériel** que le **logiciel** utilisé.

Les systèmes embarqués exécutent des tâches prédéfinies et ont un cahier des charges contraignant à remplir, qui peut être d'ordre :

- D'espace mémoire limité de l'ordre de quelques Mo maximum.

-De consommation énergétique le plus faible possible, due à l'utilisation de sources autonomes, batteries, panneaux solaires.

- Temporelle, dont le temps d'exécution de tâches est déterminé.

Des compétences pluridisciplinaires en termes d'électronique (de commande et de puissance), d'automatique et d'informatique sont nécessaires pour concevoir de tels systèmes.

I.2.3 Les composants d'un système embarqué [5]

Un système embarqué est composé d'une partie matérielle semblable à celle des ordinateurs standards, en effet, on y retrouve les mêmes composants (processeur, mémoires, dispositifs d'entrée/sortie) et d'une partie logicielle.

D'autres composants viennent compléter l'architecture simplifiée dont ces composants répondent aux caractéristiques spécifiques.

Dans un système embarqué on retrouve en générale les composants suivant :

- Capteurs (antennes, etc.) généralement analogiques couplés à des convertisseurs analogique/numérique.

- Des actionneurs (LED[2], etc.) couplés à des convertisseurs numérique /analogique.

- CPU[3] (processeur embarqué et ses entrées/sorties).

[2] LED : light-emitting diode
[3] CPU : Central Processing Unit.

- Interfaces Homme-Machine IHM[4] (écran LCD, etc.) permettant à l'utilisateur d'interagir avec le système.

- Un circuit FPGA[5] ou ASIC[6] jouant le rôle de coprocesseur.

- Ports de diagnostic utilisés pour la configuration du système et le débogage (JTAG[7], etc.)

L'architecture typique d'un système embarqué peut être représentée par le schéma de la figure I.1.

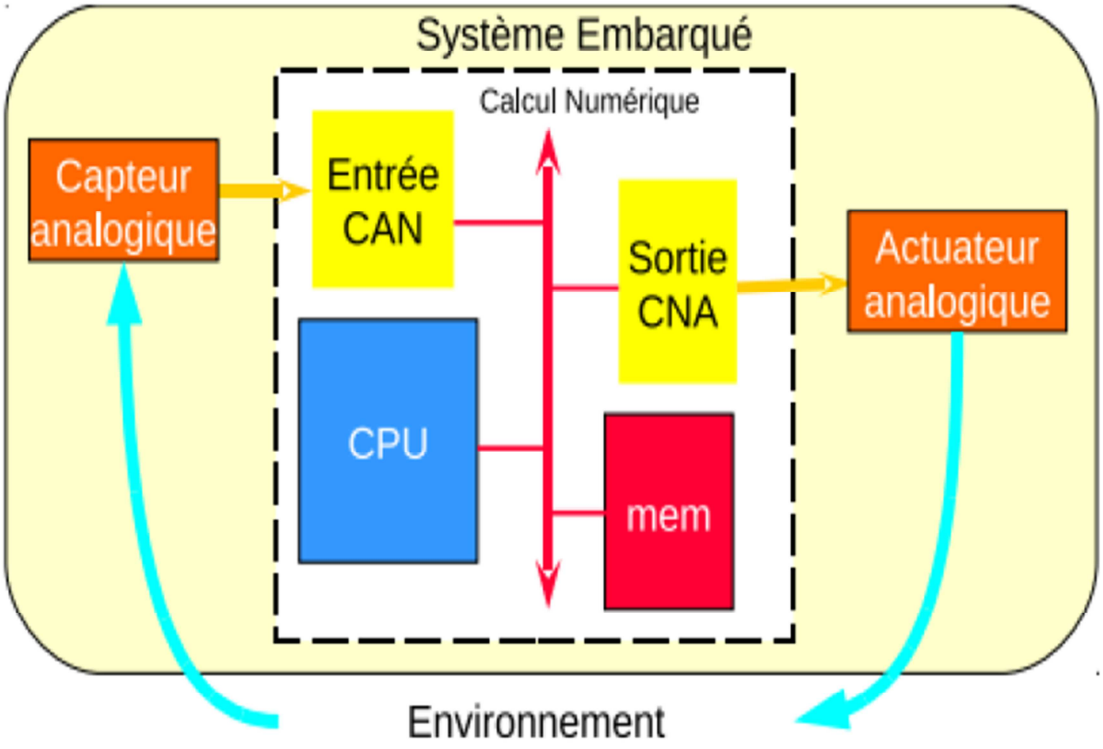


Figure I.1 Architecture typique d'un système embarqué **[6]**

[4]IHM : Interface homme machine
[5] ASIC (Application-Specific Integrated Circuit) : Circuit intégré configure pour réaliser une tâche spécifique.
[6]FPGA : (Field-Programmable Gate Array) : Circuit intégré composé de nombreuses cellules logiques élémentaires
[7]JTAG (Joint Test Action Group) : utilisé au lieu du terme générique Boundary Scan pour désigner le port de teste des cartes électroniques.

I.3 Les type de systèmes embarqués [7]

Nous pouvons énumérer les types de systèmes embarqués, et cela en tenant compte de leur rôle, comme suit :

- **Calcul général :** Ce sont des applications similaires aux applications de bureau mais empaquetées dans un système embarqué. Exemple : jeu vidéo, etc.

- **Contrôle de systèmes :** Des applications dédiées au contrôle de systèmes en temps réel. Exemple : moteur d'automobile, traitement chimique, traitement nucléaire, système de navigation aérien.

- **Télécommunication et réseaux :** Transmission d'information et commutation.
  Exemple : téléphone routeur, pare-feu, etc.

I.4 Les contraintes de construction des systèmes embarqués

Le terme système embarqué dénote un système autonome, disposant de l'ensemble des éléments physiques nécessaires à son fonctionnement (processeur, mémoire, périphériques d'entrées/sorties) et en forte interaction avec l'environnement extérieur dans lequel il évolue. Les contraintes lors de la construction de tels systèmes sont relatives au temps, imposées par l'interaction du système avec son environnement, et aussi des ressources limités (énergie, mémoire). À ces contraintes s'ajoutent des contraintes de sûreté de fonctionnement plus ou moins importantes selon le type d'utilisation du système embarqué.

I.4.1 Contraintes de temps

Beaucoup de systèmes embarqués interagissent directement avec leur environnement via des capteurs/actionneurs ou un réseau de communications sans fil. Ces interactions contraignent les temps de réponse du système embarqué de manière plus ou moins forte selon le domaine d'applications visé. On parle alors de système temps-réel**,** dans le sens où le système ne doit pas simplement délivrer des résultats exacts, il doit les délivrer dans des délais imposés **[8]**.

a) Temps-réel strict

Dans les systèmes temps-réel strict, ou dur, le non-respect des contraintes temporelles du système, constitue une défaillance de l'application. Dans le cadre d'applications critiques, telles que par exemple le contrôle de centrales nucléaires, une telle défaillance peut avoir des conséquences catastrophiques, telles que la mise en danger de vies humaines ou des pertes financières importantes **[9].**

b) temps-réel souple

Dans un système temps-réel souple, le non-respect des contraintes temporelles du système est acceptable. Cette tolérance est acceptée car les applications concernées ne relèvent pas du domaine des applications critiques. Les exemples typiques d'applications ayant des contraintes temps-réel souples sont les applications multimédias à flux continus. Le système vise au respect des contraintes temporelles dans la délivrance des flux de données afin de garantir la qualité des images et du son ; toutefois, les contraintes temporelles peuvent être adaptées puisque une dégradation de la qualité des données ne sera que faiblement perçue par l'utilisateur **[9].**

I.4.2 Consommation énergétique

Une grande majorité des systèmes embarqués (téléphones cellulaires, ordinateurs de poche,...) sont confrontés au problème de l'autonomie. Aussi, afin d'étendre l'autonomie de fonctionnement d'un système deux approches complémentaires sont possibles : augmenter la capacité de stockage des batteries ou réaliser un système embarqué à faible consommation énergétique. Dans le cadre de cette dernière approche, plusieurs méthodes sont alors envisagées qui touchent à la fois le domaine de l'électronique et du logiciel : la conception de composants électroniques consommant le minimum d'énergie, l'optimisation du logiciel afin de diminuer le coût énergétique de son exécution **[9].**

I.4.3 Mémoire

La mémoire est une ressource limitée dans un grand nombre de systèmes embarqués (de quelques Kilo-octets dans une carte à puce à quelques Mégaoctets dans un téléphone portable), et par conséquent une bonne utilisation de la ressource mémoire est cruciale pour ces systèmes. Les méthodes permettant d'utiliser des systèmes à faible capacité mémoire se basent sur l'utilisation de codes interprétés compacts, comme par exemple le bytecode Java et à l'utilisation d'algorithmes de compression **[10].**

I.4.4 Tolérance aux fautes

Certains systèmes embarqués doivent pouvoir remplir leurs fonctions malgré la présence de fautes, qu'elles soient d'origine physique ou humaine. Les moyens pour la sûreté de fonctionnement, et plus spécifiquement les méthodes de tolérance aux fautes, permettant au système de remplir ses fonctions malgré des fautes pouvant affecter ses composants.

Par exemple, en ce qui concerne les fautes d'origine physique, il est nécessaire de détecter les erreurs, puis d'effectuer un recouvrement d'erreur permettant au système de continuer à remplir ses fonctions malgré l'erreur par la reprise de son exécution à partir d'un état sauvegardé au préalable (point de reprise) **[9]**.

I.5 Architecture des systèmes embarqués

Dans cette section, nous présentons les architectures supportées par trois générations

I.5.1 Les systèmes embarqués de première génération [11]

- **Partie matérielle des systèmes embarqués de première génération**

Les premiers systèmes embarqués étaient très simples, constitués d'un processeur. Ce type d'architecture est représenté sur la figure I.2. Les communications de cette architecture se situent au niveau du bus du processeur et sont de type maître/esclave : le processeur est le maître et les périphériques sont les esclaves

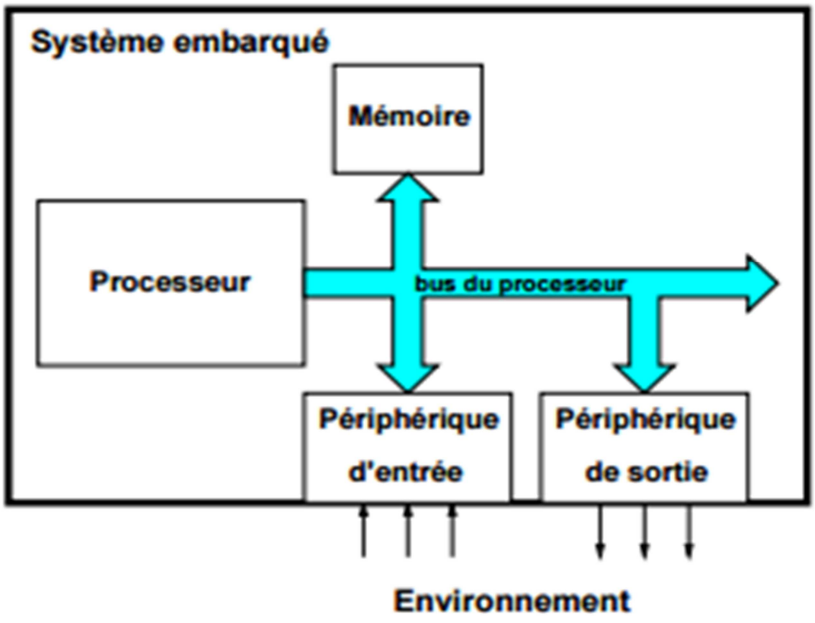


Figure I.2 Architecture embarquée de première génération [3].

Les périphériques de ces architectures étaient essentiellement des capteurs et des actionneurs. Le processeur est dédié au calcul et au contrôle de l'ensemble du système.

- **Partie logicielle des systèmes embarqués de première génération**

  - Ne devant pas exécuter de nombreuses opérations simultanées (le nombre de périphériques et de fonctions étant restreint),

  - les parties logicielles étaient constituées d'un seul programme et décrite directement en langage d'assemblage.

I.5.2 Les systèmes embarqués de deuxième génération [11]

Les premiers systèmes embarqués ne pouvaient fournir que des fonctions simples. Leur architecture ne peut pas supporter les fonctionnalités requises pour les systèmes embarqués actuels à qui il est demandé non seulement d'effectuer du contrôle, mais aussi des calculs complexes tels que ceux requis pour le traitement numérique du signal

- **Partie matérielle des systèmes embarqués de deuxième génération**

L'architecture des systèmes embarqués de deuxième génération est composée d'un processeur central, de nombreux périphériques, et souvent de quelques processeurs annexes contrôlés par le processeur central. Le processeur central est dédié au contrôle de l'ensemble du système. Les processeurs annexes sont utilisés pour les calculs ; il s'agit souvent de processeurs spécialisés comme les DSP[7]. Une telle architecture est représentée figure I.3. Dans une telle architecture, plusieurs bus de communication peuvent être nécessaires : chaque processeur dispose de son bus de communication.

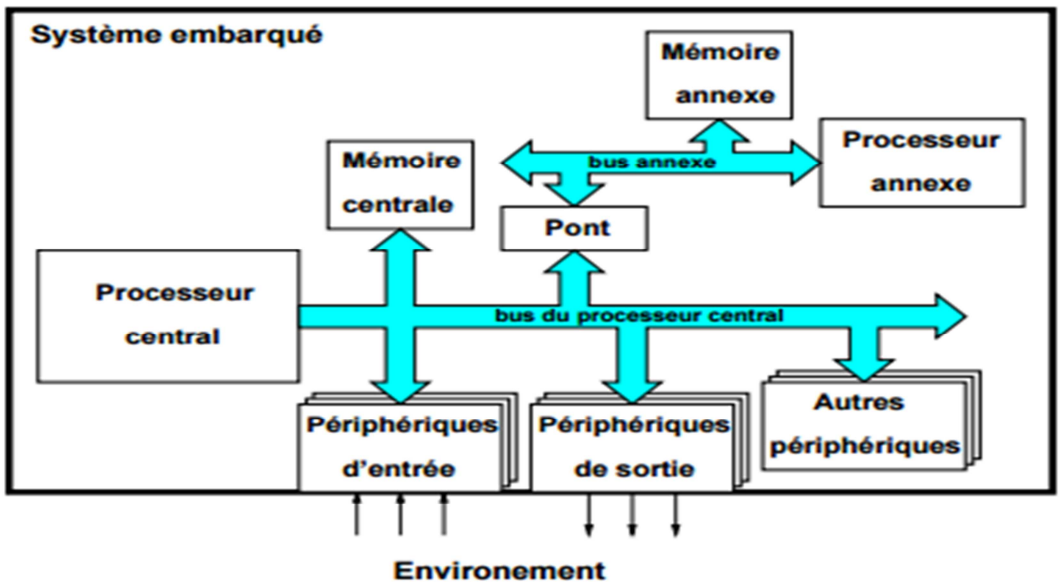


Figure I.3 Architecture embarquée de deuxième génération [3].

[8]DSP :Digital Signal Processor

- **Partie logicielle des systèmes embarqués de deuxième génération**

La partie logicielle des systèmes embarqués de deuxième génération est répartie sur plusieurs processeurs (le processeur principal et les processeurs annexes). Les systèmes actuels sont trop complexes pour pouvoir être gérés par un unique programme sur le processeur principal.

I.5.3 Les systèmes embarqués de troisième génération [11]

Les progrès de l'intégration permettent d'envisager des circuits pouvant contenir plusieurs milliers de portes. Il devient donc techniquement possible de fabriquer des systèmes embarqués pouvant remplir toutes les fonctionnalités souhaitées.

- **Parties matérielles des systèmes embarqués de troisième génération**

Pour pouvoir supporter conjointement les besoins en puissance et en flexibilité, ces architectures comprennent de plus en plus de processeurs, qui peuvent chacun se comporter en maître : l'architecture couramment utilisée, basée sur un processeur central contrôlant le reste du système, n'est donc plus suffisante.

Alors qu'auparavant le goulet d'étranglement était les ressources en calcul, de nos jours il est situé plutôt au niveau des communications. Ce sont elles qui définissent désormais l'architecture, et nos plus les ressources de calcul. La figure I.4 donne des exemples d'architectures centrées sur les communications :

- Architecture à base de bus : ce modèle est basé sur des communications par bus de communication et consomme peu de surface.
- Architecture en barres croisées : ce modèle est basé sur des communications en barres croisées très performantes mais aussi très coûteuses en surface.
- Architecture à base de réseau commuté : ce modèle donne une solution intermédiaire, par réseau commuté.
- Architecture mixte : ce modèle montre qu'il est possible de mixer plusieurs modèles de communication, et d'apporter de la hiérarchie dans l'architecture.

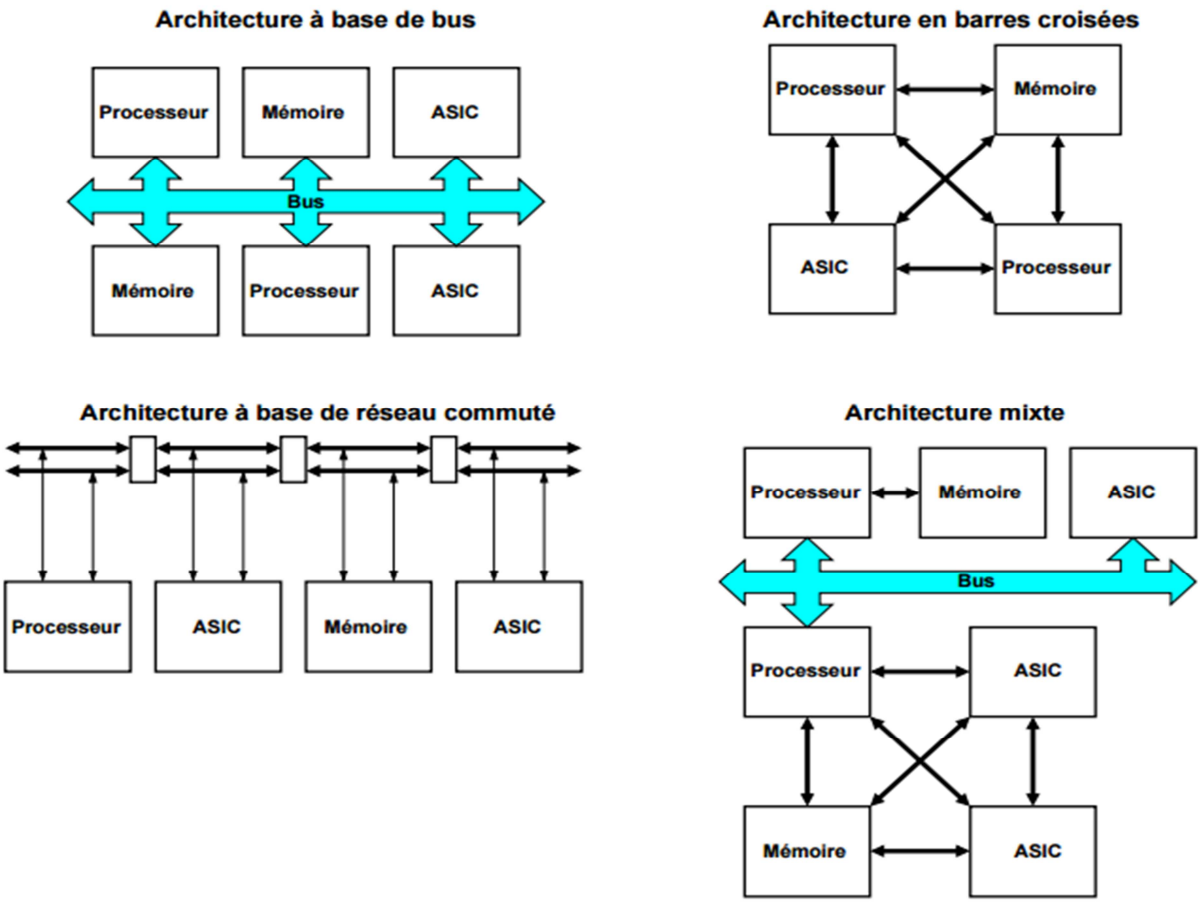


Figure I.4 Architectures embarquées de troisième génération [3].

- **Parties logicielles des systèmes embarqués de troisième génération**

Les parties logicielles ont beaucoup gagné en importance dans les systèmes embarqués. Plusieurs systèmes d'exploitation sont parfois nécessaires pour les divers processeurs de l'architecture. La complexité et la diversité des architectures nécessite d'abstraire les tâches logicielles des détails du matériel. Cette complexité se reporte dans les systèmes d'exploitation, qui deviennent de plus en plus complexes.

I.6 Les Systèmes embarqués monopuces (SoC)

Les progrès réalisés par les fondeurs de circuits permettent maintenant d'envisager l'intégration sur une même puce un système embarqué complet. Ces systèmes mono puce apportent des changements importants dans les flots de conception classiques.
Les systèmes mono puce sont dédiés à des applications spécifiques et taillés sur mesure pour satisfaire seulement les besoins de l'application visée **[11]**.

I.6.1 Définition

Un système sur puce, encore appelé système mono puce ou SoC (pour system on chip), désigne l'intégration d'un système complet intégrant plusieurs composants complexes et hétérogènes dédiés à des applications spécifiques sur une seule puce électronique (puce de silicium) **[11].**

I.6.2 Caractéristiques des Socs

a) L'hétérogénéité

Un Système sur puce est par définition, un système hétérogène du fait qu'il regroupe dans la même puce des composants de différentes natures. Prenant l'exemple du téléphone portable qui est un système hétérogène où la partie analogique (radio) et la partie numérique (traitement de signal) cohabitent sur la même puce **[12].**

b) Spécification

Les systèmes sur puce ciblent toujours une application particulière bien définie, à la différence des systèmes a usages généraux tel que les ordinateurs, on parle souvent de SoC spécifique. Cette caractéristique a fait la différence au niveau de la méthodologie de conception entre les systèmes mono-puce et les systèmes à usage général

Le but de la conception de SoC d'avoir sur la même puce l'ensemble du système afin de raccourcir les chemins de communication.

Les SoC sont aussi moins encombrants et surtout, ils peuvent consommer moins : en effet les données doivent transiter par des chemins beaucoup moins longs, l'énergie nécessaire à cette transmission est donc plus faible **[12].**

I.7 Vers des systèmes embarqués multiprocesseurs

Les avancées technologiques, architecturales, et développement des outils de conception permettent d'intégrer aujourd'hui plusieurs cœurs de processeurs généralistes et de coprocesseurs spécifiques sur une même puce. Les systèmes sur puce SoC se sont rapidement transformés en systèmes multiprocesseurs sur puce MPSoC **[13].**

I.7.1 Définition

Un système multiprocesseurs sur puce est un système qui fait intégrer dans un même circuit (puce) plusieurs processeurs homogènes ou hétérogènes, plusieurs composants complexes et différents tels que des unités de calcul spécifiques programmables et/ou non programmable (ASIC, FPGA) ,des composants de mémorisation variés, périphériques E/S…**[14]**
Ces systèmes viennent comme une réponse aux limites des systèmes mono puce en termes de puissance de calcul, exploitation de parallélisme

Donc nous avons sur une même puce plusieurs processeurs exécutant en parallèle différentes tâches.

I.7.2 Architectures multiprocesseurs sur puce [6]

a) Architecture Homogène

Tous les processeurs présents dans une architecture MPSoC homogène sont identiques. Ainsi, ils sont faciles à programmer par rapport aux plateformes hétérogènes.

b) Architectures Hétérogène

Un Système intégrant de différents types d'éléments de traitements (PEs[9]).On trouve les processeurs généralistes (GP), et les processeurs spécifiques excellant dans leurs domaines de calculs avec des performances moyennes en contrôle tel que les ASIC, FPGA, etc.
La performance de calcul peut être augmentée par l'exploitation des caractéristiques distinctes des différents types d'éléments de traitements.

[9] PEs : processing elements

I.8 Conclusion

Ce chapitre nous a permis de voir que les systèmes embarqués ont certaines particularités en ce qui concerne leur architecture, et cela leur permet de réaliser une tâche bien précise tout en respectant les contraintes. Dans les divers domaines, le développement des systèmes embarqués ne cesse de croître et d'attirer de nouveaux développeurs et les industriels sont en concurrence permanente, ce qui rend le marché de ces systèmes très intéressant.

Le chapitre suivant va être consacré à étudier les différentes mémoires dans les systèmes embarqués.

Chapitre II

Les Mémoires dans les Systèmes Embarqués

II.1 Introduction

Depuis le milieu des années 80, la performance des microprocesseurs n'a cessé d'évoluer (50-100% par an), mais ces améliorations se retrouvèrent confrontés au problème de la modeste évolution des mémoires (à peine 7% par an) qui constitue un goulet d'étranglement. On se retrouve avec des processeurs de plus en plus rapides, mais avec des mémoires 23 fois moins rapides donc ne pouvant pas leur fournir autant de données qu'ils réclament, et le pire est que ce ratio double d'une à deux années environs.

Mais en contre partie, le coût de la mémoire ne cesse de baisser, et les programmeurs jouent sur cet aspect pour combler l'écart en performance processeur/mémoire en exécutant plusieurs taches parallèlement disposant (presque) d'autant de mémoire vive qu'ils veulent, mais pour les systèmes embarqués l'histoire est différente, car en plus du fait d'être un goulet d'étranglement en performance, la mémoire l'est aussi en consommation, à elle seule utilise jusqu'à 70% de la consommation.

C'est dans le contexte des mémoires pour systèmes embarqués que se situe notre axe de recherche où nous faisons une brève présentation des différentes mémoires en premier lieu, dans ce cadre, les Scratch-Pad Memories (SPMs) ou mémoires Scratch-Pad présentent un intérêt considérable de par la facilité et la certitude avec lesquelles leur comportement (temps de réponse notamment) peut être prédit. Il faudra donc les intégrer dans notre plate-forme.

II.2 Les mémoires

II.2.1 Qu'est-ce qu'une mémoire

En informatique, la mémoire est un dispositif électronique qui sert à stocker des informations. La mémoire est un composant essentiel, présent dans tous les ordinateurs, les consoles de jeux, les GPS et de nombreux appareils électroniques.
Les mémoires sont vendues sous forme de pièces détachées de matériel informatique, ou de composants électroniques. Les différences entre les pièces sont la forme, l'usage qui en est fait, la technologie utilisée, la capacité de stockage et le rapport entre le coût et la capacité **[15].**
La mémoire peut être dans le processeur (des registres), interne (Mémoire centrale ou principale) ou externe (Mémoire secondaire).

II.2.2 Caractéristiques des mémoires

a) La capacité d'une mémoire

La capacité (taille) d'une mémoire est le nombre (quantité) d'informations qu'on peut enregistré (mémoriser) dans cette mémoire.

La capacité peut s'exprimer en :

- Bit : un bit est l'élément de base pour la représentation de l'information.
- Octet : 1 Octet = 8 bits

- kilo-octet (KO) : 1 kilo-octet (KO)= 1024 octets = 2^{10} octets
- Méga-octet (MO) : 1 Méga-octet (MO)= 1024 KO = 2^{20} octets
- Géga-octet (GO) :1Géga-octet (GO)=1024 MO = 2^{30} octets
- Téra-octet (To) : 1 téra-octet (To)= 1024 Go =2^{40} octets

b) Volatilité

Comme nous allons le voir plus loin, il existe plusieurs types de mémoires. On peut les classer en deux catégories : les mémoires volatiles set les mémoires non volatiles. Une mémoire est dite volatile si son contenu disparaît ou est effacé lorsqu'on éteint l'ordinateur, alors qu'une mémoire non volatile garde l'information dans tous les cas. Par exemple, lorsqu'on démarre un programme, le microprocesseur place ce programme dans la mémoire volatile; lorsqu'on éteint l'ordinateur, toutes ces données sont immédiatement perdues.

En termes techniques, on parlera de mémoire RAM (random access memory) pour désigner la mémoire volatile et de mémoire ROM (read only memory) pour celle qui est non volatile **[16].**

c) Mode d'accès à l'information (lecture /écriture)

Sur une mémoire on peut effectuer l'opération de :

• lecture : récupérer / restituer une information à partir de la mémoire.

• écriture : enregistrer une nouvelle information ou modifier une information déjà existante dans la mémoire.

- Les mémoires vives offrent les deux modes lecteur/écriture.

- Les mémoires mortes offrent uniquement la possibilité de la lecture (ce n'est pas possible de modifier le contenu) **[17].**

d) La vitesse et le temps d'accès

Le temps d'accès est l'intervalle de temps qui sépare la demande d'information de son obtention .Pour une opération de lecture par exemple, le temps d'accès est le temps qui sépare la demande de la lecture de la disponibilité de l'information. Il s'exprime en nanosecondes (ns), soit un milliardième de seconde (10^{-9}) **[16].**

Le temps d'accès est un critère important pour déterminer les performances d'une mémoire ainsi que les performances d'une machine.

Le temps d'accès des mémoires actuelles est généralement inférieur à 60 ns. Plus le chiffre est faible, plus la vitesse est grande.

e) Mémoires synchrones et asynchrones [18]

- *Mémoire asynchrone* : pour ce type de mémoire, l'intervalle de temps entre deux accès mémoire consécutif n'est pas régulier. Le processeur ne sait donc pas quand l'information qu'il attend est disponible et doit attendre (wait-state) que la mémoire lui transmette les données.

- *Mémoire synchrone* : la cadence de sortie des informations est régulière

II.3 Les type de mémoire

Les différents mémoires peuvent être classés comme indiqué sur le schéma de la figure ci-dessous

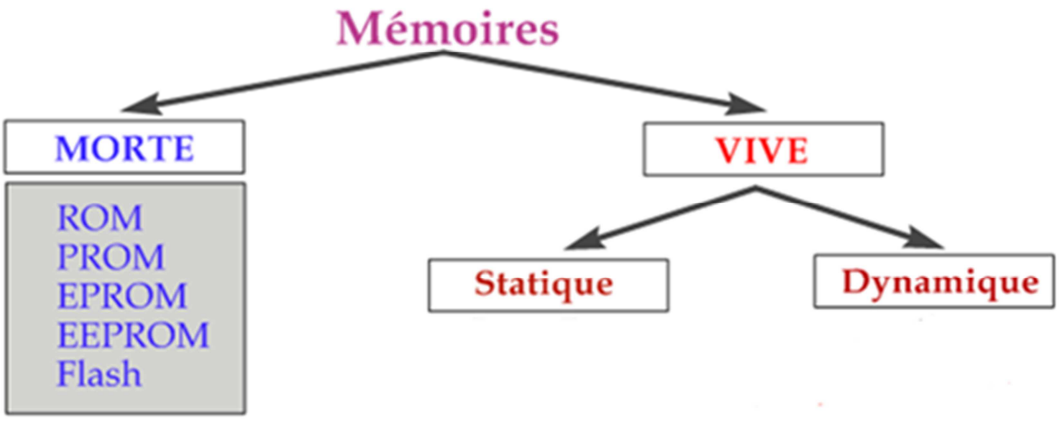


Figure II.1 Classifications des mémoires [18].

II.3.1 les mémoires mortes (ROM)

La mémoire ROM (read only memory), que l'on nomme en français « mémoire morte ou mémoire à lecture seule», est un type de mémoire capable de stocker des données de façon permanente. C'est une mémoire qui ne peut qu'être lue. La ROM est un emplacement idéal pour mettre les instructions de démarrage de l'ordinateur, c'est-à-dire le BIOS[1].Celui-ci contient le programme de démarrage qui a pour but de charger le système d'exploitation en mémoire lors de la mise en route de l'ordinateur.

La capacité de cette mémoire est généralement réduite (de l'ordre de Ko) .Son accès est aléatoire, comme dans le cas de la mémoire RAM.

[1]BIOS : Basic Input Output System, en français : « système élémentaire d'entrée/sortie »

Comme son contenu ne peut être modifié facilement, elle offre une bonne sécurité pour la sauvegarde de programmes spécifiques. Bien que cela soit possible, il est difficile d'altérer ses programmes ou de les contaminer avec un virus **[16].**

❖ **Les différents types de mémoire ROM**

- **La mémoire morte standard (ROM)**

La mémoire ROM standard est un circuit intégré électronique dans lequel les programmes ou les données sont gravés une fois pour toutes lors de la fabrication de la puce électronique. Il est impossible de les modifier par la suite. On peut les comparer à des disques compacts de musique (CD-Audio). Elle existe en général sous une petite taille dans un système embarqué, son temps d'accès est plus lent que les RAM **[16].**

- **La mémoire morte programmable (PROM)**

La mémoire de type PROM (programmable read only memory) est un circuit fabriqué et programmé une fois. Plus précisément c'est des ROM vierges qu'on peut programmer une seule fois. Une fois programmées, il n'est plus possible de les reprogrammer.

- **La mémoire morte reprogrammable (EPROM)**

La mémoire EPROM (ereasable PROM) est une mémoire morte reprogrammable avec un appareil spécial appelé programmateur d'EPROM. Son contenu peut être effacé à l'aide d'un appareil d'exposition aux rayons ultraviolets. L'effacement répété finit par endommager la mémoire elle n'est pas donc conseillée dans les systèmes finaux.

On reconnaît facilement les dispositifs de ce type de mémoire grâce à la petite fenêtre vitrée sur le dessus du circuit intégré. Cette fenêtre est souvent recouverte d'une étiquette pour empêcher la lumière d'entrer dans la puce et éviter ainsi de détériorer la mémoire **[16].**

- **La mémoire EEPROM**

Ce sont des mémoires non volatiles qu'on peut écrire et effacer en appliquant des tensions électriques. Elles existent sur les systèmes embarqués en de très petites tailles (plus petite que la ROM). Elles contiennent en général des données qui peuvent changer au fil du temps mais qui doivent rester après extinction du système.
L'écriture et l'effacement ne nécessite pas de dispositif spécifique comme ce qui est de l'EPROM et peut se faire sur une ligne spécifique. L'effacement répété n'endommage pas l'EEPROM.

- **La mémoire Flash [19]**

La mémoire flash est un compromis entre les mémoires RAM et ROM : c'est une mémoire non volatile, comme les mémoires mortes, mais qui possède par ailleurs les caractéristiques d'une mémoire vive. Les mémoires Flash fournissent une grande densité de sauvegarde comparées aux EEPROM avec un temps d'accès équivalent aux RAMs lentes (DRAM par exemple). Elles sont utilisées pour sauvegarder le logiciel (le Firmware) du système embarqué.

Deux types de mémoires Flash existent, la mémoire NAND et la mémoire NOR

- **La NAND-FLASH**

Cette mémoire offre une plus grande densité de sauvegarde mais lente. Toutefois son interface d'entrée-sortie n'autorise que l'accès séquentiel. Cela tend à limiter au niveau du système sa vitesse effective de lecture, et à compliquer le démarrage direct à partir d'une mémoire NAND. De ce fait elle est moins bien adaptée que la NOR pour exécuter du code machine. Du fait de son prix moins élevé, elle est présente dans de nombreux assistants et téléphones portables.

- **La NOR-FLASH**

Cette mémoire offre une petite densité de sauvegarde mais rapide, elle offre un meilleur comportement aléatoire que sa similaire NAND car elle possède une interface d'adressage permettant un accès aléatoire et rapide à n'importe quelle position.

II.3.2 Les mémoires vivres (RAM)

La mémoire RAM est une mémoire volatile, utilisée par le processeur pour stocker temporairement de l'information et exécuter des instructions. L'information contenue dans une mémoire RAM peut être effacée, remplacée ou recouvrée en tout temps.

Une mémoire RAM est constituée d'un ensemble de cellules mémoires, qui sont étiquetées, par une adresse mémoire. Les informations enregistrées en mémoire sont donc repérées par leurs adresses, c'est-à-dire par leur position physique. Le qualificatif aléatoire de l'accès par le processeur à la mémoire signifie que toutes les cellules mémoires peuvent être accessibles dans un intervalle de temps identique. Nous convenons que le terme accès direct est plus commode **[16].**

On distingue deux catégories, dépendant du type de conception :

- Les RAM statiques

- Les RAM dynamiques

a) La mémoire SRAM

Les mémoires statiques ou SRAM (Static Random Acces Memory), onéreuses et encombrantes, ont l'immense avantage de pouvoir stocker des valeurs pendant une longue période sans devoir être rafraîchies. Cela permet des temps d'accès très court (8-20 ns) **[20].**

Le bit mémoire d'une RAM statique (SRAM) est composé d'une bascule. Chaque bascule contient entre 4 et 6 transistors, ce qui fait qu'elle soit la plus chère et qu'elle occupe beaucoup d'espace par bit mais malgré cela elle est souvent montée sur le processeur même comme mémoire sur puce (On chip memory). Les mémoires SRAM sont rapides, il est possible d'y accéder en un cycle processeur.

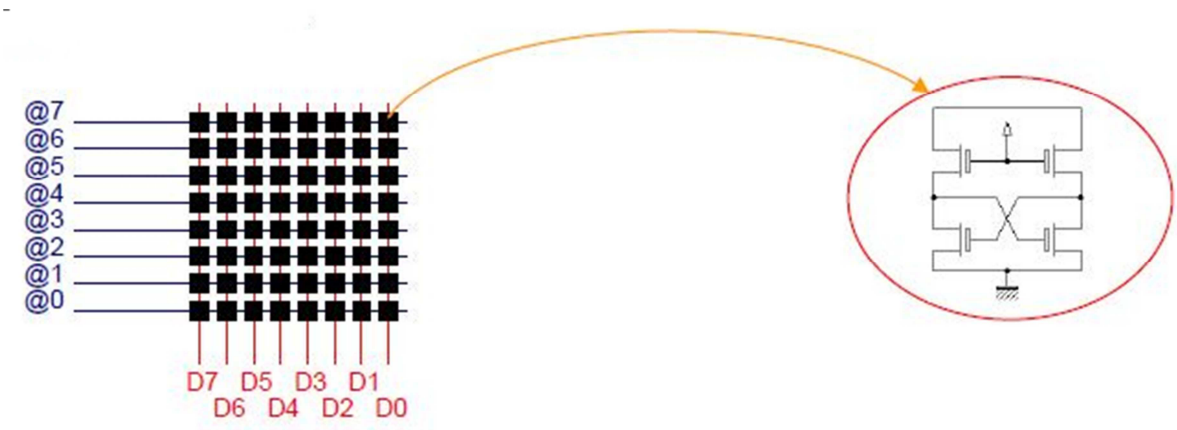


Figure II.2 : Structure d'une mémoire SRAM [22].

b) La mémoire DRAM

Dans la mémoire dynamique ou DRAM (Dynamic Ramdom Access Memory),une cellule mémoire est constituée d'un seul transistor (couplé d'un condensateur), cette simplicité architecturale explique le fait qu'une cellule DRAM occupe le sixième de l'espace mémoire occupé par une cellule SRAM**,[23]**Comme tout condensateur présent des courants de fuites, il se décharge peu à peu et ce risque de fausser des informations contenues en mémoire. Par exemple, si un bit 1 est stocké dans la cellule, il s'efface peu à peu et risque de voir sa valeur à 0 **[20].**

Pour y remédier, on procède régulièrement à la relecture et la réécriture des informations. C'est le rafraîchissement qui a lieu toutes les 15 millisecondes environ. Bien entendu, pendant le rafraîchissement, la mémoire est disponible en lecture comme en écriture, ce qui ralenti les temps d'accès **[20].** Son temps d'accès est couramment de l'ordre de 60 à 70 nanosecondes.

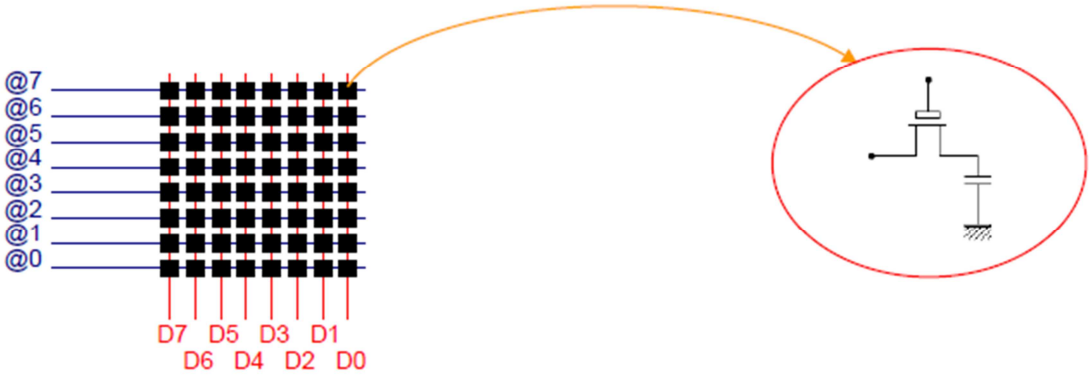


Figure II.3 : Structure de la mémoire DRAM [21].

Le tableau suivant permet de comparer ces deux types mémoires :

Type	Vitesse	Densité	Coût
DRAM	Lente	Haute	Bas
SRAM	Rapide	Faible	Elevé

Tableau II.1 : comparaison entre DRAM et SRAM

II.4 La hiérarchies mémoires

Pour avoir un système avec les meilleures performances et les moindres coûts possibles, il est impératif de penser à une organisation pyramidale de la mémoire, mettre en première ligne juste après les registres du processeur une quantité minimale de la mémoire la plus rapide, puis plus on descend dans la hiérarchie, plus on peut se permettre plus de mémoire.

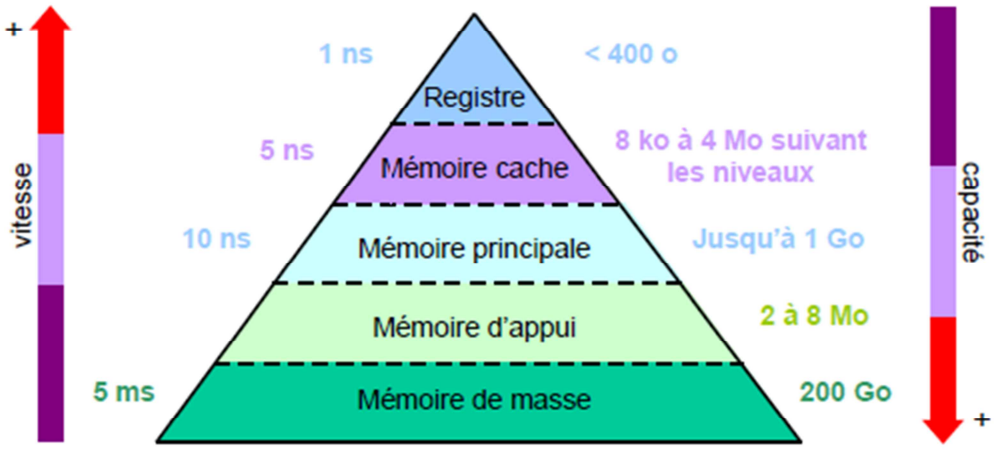


Figure II.4 Hiérarchie de mémoires [21].

Dans un système embarqué une hiérarchie mémoire est l'ensemble des mécanismes automatiques de gestion de la mémoire qui permet d'optimiser des paramètres comme le coût, l'efficacité du système ou encore sa consommation d'énergie.

- **Les registres :** sont les éléments de mémoire les plus rapides. Ils sont situés au niveau du processeur et servent au stockage des opérandes et des résultats intermédiaires.
- **La mémoire cache :** est une mémoire rapide de faible capacité destinée à accélérer l'accès à la mémoire centrale en stockant les données les plus utilisées.
- **La mémoire principale :** est l'organe principal de rangement des informations. Elle contient les programmes (instructions et données) et est plus lente que les deux mémoires Précédentes.
- **La mémoire d'appui :** sert de mémoire intermédiaire entre la mémoire centrale et les mémoires de masse. Elle joue le même rôle que la mémoire cache.
- **La mémoire de masse :** est une mémoire périphérique de grande capacité utilisée pour le stockage permanent ou la sauvegarde des informations **[21].**

II.4.1 La mémoire cache

La mémoire cache a fait son apparition vers les années 80, elle est placée près du processeur, voire greffée sur ce dernier. Elle fait office de tampon entre le processeur et la mémoire vive. Étant plus rapide (10 fois plus rapide) que la RAM, de petite taille. Elle contient les mots mémoire les plus fréquemment utilisés accélérant ainsi l'accès à ces mots. Lorsqu'un pourcentage important de mots nécessaires se trouve en cache, la latence effective de la mémoire est fortement réduite. Pour améliorer à la fois la bande passante et la latence, on peut recourir à des caches multiples.

Une technique de base très efficace, qualifiée de caches séparés, consiste à introduire deux caches distincts : un pour les instructions et un autre pour les données. Les opérations mémoire peuvent donc être réalisées indépendamment dans chaque cache.

Il est à noter que de nombreux processeurs contiennent une mémoire cache intégrée **[23].**

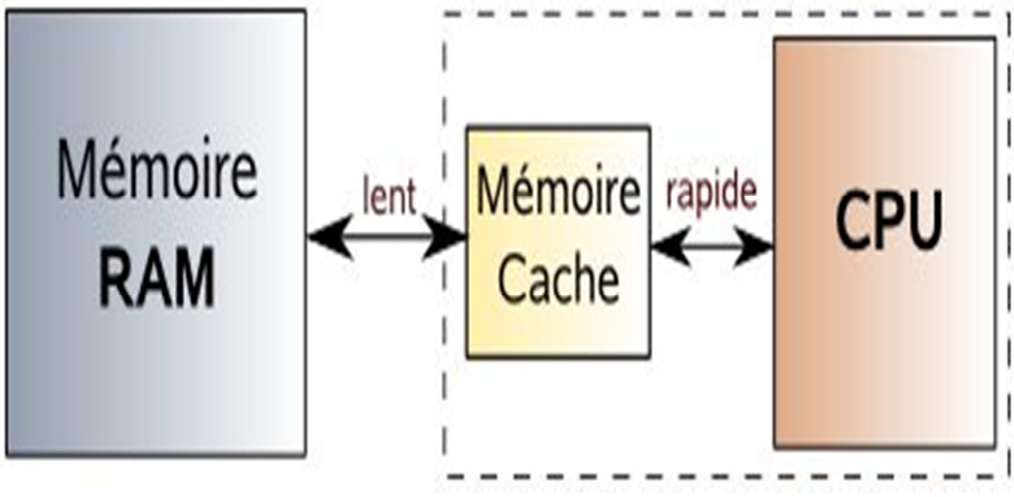


Figure II.5 La mémoire cache [24].

II.4.1.1 Fonctionnement d'un cache

Le cache contient une copie des données originelles lorsqu'elles sont coûteuses (en termes de temps d'accès) à récupérer par rapport au temps d'accès au cache. Une fois les données stockées dans le cache, on y accède directement par le cache plutôt qu'en les récupérant ou en les recalculant, ce qui diminue le temps d'accès moyen.

Le processus fonctionne ainsi :

1. l'élément demandeur (microprocesseur) demande une information ;
2. Suite à cette demande, le cache va vérifier s'il possède cette information. Si oui, il la retransmet au microprocesseur. On parle de « cache hit », c'est-à-dire succès de cache.

   S'il ne possède pas l'information, le cache la demande à l'élément fournisseur (la mémoire vive par exemple). On parle alors de « cache miss », c'est-à-dire défaut de cache.

3. l'élément fournisseur traite la demande et renvoie la réponse au cache ;
4. Le cache stocke l'information reçue pour une utilisation ultérieure et la retransmet à l'élément demandeur au besoin **[24].**

Si les mémoires cache permettent d'accroître les performances, c'est en partie grâce à deux principes qui ont été découverts suite à des études sur le comportement des programmes informatiques :

- **le principe de localité spatiale** : qui indique que l'accès à une donnée située à une adresse *X* va probablement être suivi d'un accès à une zone très proche de *X*. C'est évidemment vrai dans le cas d'instructions exécutées en séquence.
- **le principe de localité temporelle :** qui indique que l'accès à une zone mémoire à un instant donné a de fortes chances de se reproduire dans la suite immédiate du programme. C'est évidemment vrai dans le cas des boucles **[24]**.

II.4.1.2 Organisation générale d'un cache

La mémoire cache ne dispose pas d'un espace d'adressage propre car elle est sensée contenir un fragment de code et de données provenant de n'importe quelle zone mémoire. Il existe plusieurs façons d'organiser l'espace du cache pour contenir ces données. Dans tous les cas, le processeur référence le cache avec l'adresse mémoire de l'emplacement demandé. Le système cache doit pouvoir réaliser une correspondance entre l'espace du cache et l'espace mémoire.

Plusieurs stratégies sont disponibles, ici nous présentons trois:

a) Mémoire cache à correspondance directe

Ici, le cache est vu comme un espace de stockage cyclique. Un bloc de mémoire est placé dans le bloc de cache dont l'index est égal à :(numéro de bloc de mémoire) **mod** (nombre de blocs de cache) **[25].**

Le principal inconvénient de cette méthode est qu'une ligne sera retirée du cache si un accès est réalisé à une autre adresse qui possède le même emplacement dans le cache. Le risque est donc d'avoir un programme qui manipule en boucle deux valeurs qui vont s'exclure alternativement du cache.

L'intérêt de cette méthode est essentiellement la simplicité de sa mise en œuvre et sa vitesse d'exécution. En effet, on sait rapidement si une ligne est présente dans le cache et on sait également quelle ligne remplacer si besoin.

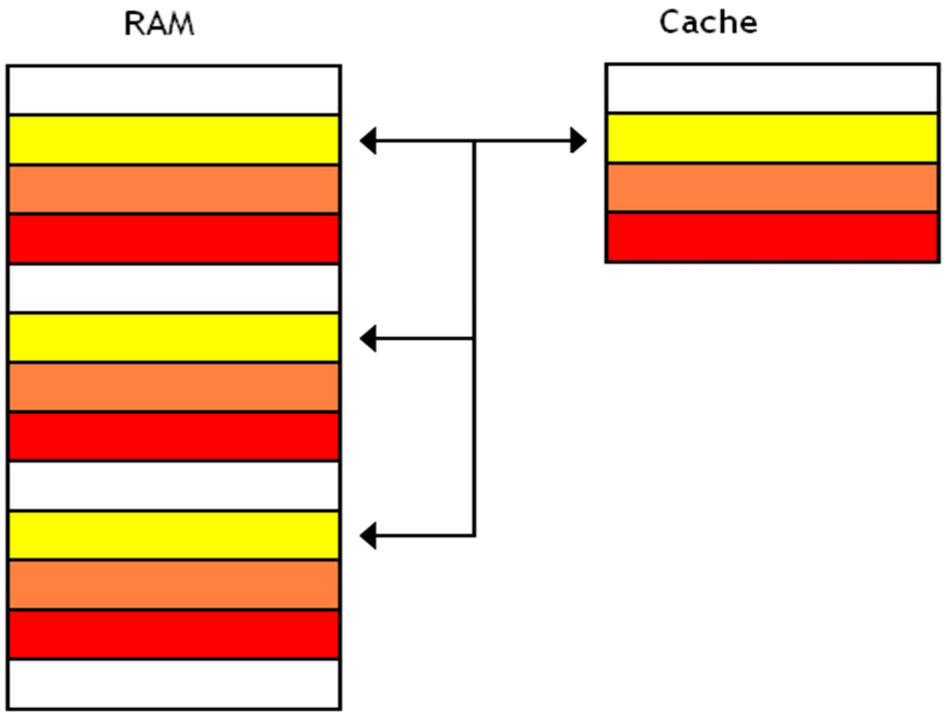


Figure II.6 Exemple d'une organisation directe [26].

b) Cache totalement associatif

Contrairement à l'organisation directe, cette organisation offre plus de flexibilité dans le placement des blocs. Chaque bloc mémoire peut être mis dans n'importe quel bloc cache. L'inconvénient majeur de cette approche est la nécessité, à chaque accès au cache, de comparer le numéro du bloc demandé avec celui de l'ensemble des lignes présentes dans le cache afin de vérifier si la ligne est présentée dans le cache et peut consommer énormément de temps si les indexes étaient comparés séquentiellement.

Au contraire, cette méthode permet de garder dans le cache les lignes qui ont le plus de chance d'être accédées à nouveau et donc de réduire le nombre de défauts de cache.

Lors d'un défaut de cache, l'algorithme de remplacement est appelé afin de déterminer l'emplacement du cache qui doit être utilisé pour accueillir la nouvelle ligne. Il existe tout un ensemble d'algorithmes pour ce faire tels que LRU (Least Recently Used), FIFO, etc. Les politiques les plus courantes sont LRU.

Un cache associatif provoque généralement moins de défauts de cache que la méthode par accès direct, mais elle est aussi plus complexe (algorithmiquement) et donc plus consommatrice en temps. Dès que le nombre de lignes devient trop grand, cette méthode est à écarter.

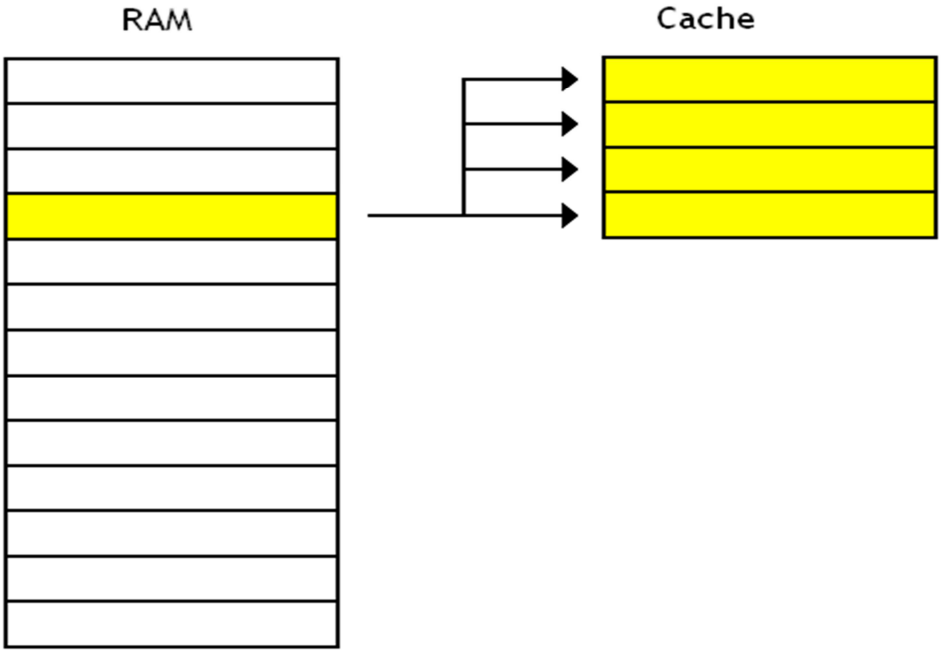


Figure II.7 Exemple d'une organisation associative [26].

c) Cache associatif par ensemble

Les caches précédents ont chacun leur défaut : un taux de succès médiocre pour les premiers, et un temps d'accès trop long pour les autres. Certains caches implémentent une sorte de compromis destiné à trouver un juste milieu : ce sont les caches associatifs par ensemble pour simplifier, Cette solution consiste à adresser de manière directe un ensemble d'emplacements du cache puis d'utiliser le fonctionnement d'un cache associatif sur ce sous-ensemble du cache. Ainsi, on limite les inconvénients du cache associatif par un nombre réduit de lignes à comparer tout en gardant une partie de ses avantages.

C'est donc un compromis entre un cache rapide (direct) et un cache qui fait moins de défauts de cache (associatif).

Un cache associatif par ensemble à N voies est un cache où chaque ensemble contient N emplacements. On dit aussi que son associativité est de N. Un cache direct est un cas particulier où l'associativité est de 1, et un cache entièrement associatif est un cache où l'associativité est égale à la capacité en nombre de lignes du cache.

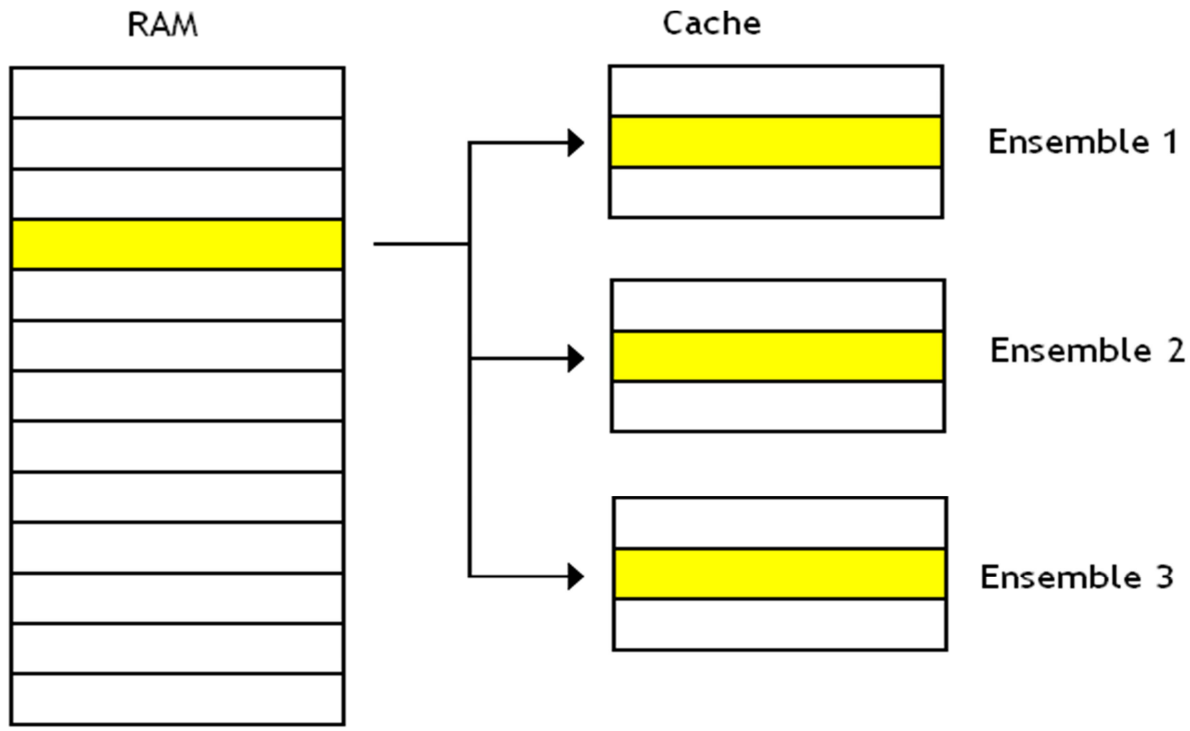


Figure II.8 Exemple d'une organisation associative par ensemble [26].

II.4.1.3 Gestion de l'espace d'un cache

II.4.1.3.1 Recherche d'un bloc de cache

La tâche fondamentale du cache est de chercher, à partir d'une adresse mémoire si le bloc mémoire correspondant est placé en cache. Étant donné un bloc de mémoire, sa recherche est une réussite s'il a été trouvé dans un bloc de cache valide.

II.4.1.3.2 Remplacement d'un bloc de cache

Lorsqu'un défaut de cache survient, le bloc de mémoire manquant doit être obtenu par lecture en mémoire principale et placé en cache.
Il existe plusieurs politiques de remplacement, dont les deux suivantes sont les représentantes canoniques :

-La politique aléatoire

C'est l'algorithme de remplacement le plus simple à implémenter. Le choix de la ligne à remplacer se fait d'une façon complètement aléatoire sans faire attention aux références. Son implémentation peut être obtenue par un compteur cyclique qui sera incrémenté à chaque référence et qui donnera à chaque fois que c'est nécessaire la ligne à remplacer **[27]**.

-La politique du plus ancien (LRU - Least Recently Used).

L'algorithme de remplacement LRU est le plus utilisé dans la gestion mémoire. Il consiste à remplacer la ligne dont la dernière référence est la plus ancienne parmi toutes les lignes du cache. Il est implémenté en utilisant des compteurs associés à chaque ligne du cache. Le compteur est incrémenté à intervalles réguliers et remis à zéro dès que la ligne est référencée. A tout moment, la valeur de chaque compteur indique l'âge de la ligne depuis sa dernière référence. La ligne LRU est celle ayant la valeur de compteur la plus élevée **[27]**.

II.4.2 Vers un design plus simple : les mémoires scratchpads

II.4.2.1 Inconvénients des mémoires caches

Les mémoires cache constituent des dispositifs de stockage améliorant les performances générales des systèmes informatiques dont la capacité relativement faible par rapport à celle de la mémoire principale est rendue possible grâce à la localité temporelle et spatiale exhibée par les applications exécutées. Cependant elles présentent des phénomènes de conflits ce qui interdit l'utilisation de ces mémoires dans les systèmes temps-réel strict sans précautions supplémentaires.

L'évolution des performances d'un cache fut longtemps axée sur l'évaluation des défauts de cache.
L'implantation d'un cache répond à un souci d'efficacité et de rapidité dans l'exécution des programmes. Ainsi, on tend à optimiser (minimiser) certains paramètres tels que :

-La probabilité de ne pas trouver une référence dans le cache (*miss ratio*).

-Le temps d'accès aux données du cache (*Access time*).

-Le temps dû au traitement d'un défaut de cache.

-Le temps et le nombre de mises à jour de la hiérarchie mémoire.

-La pollution du cache.

Il est impossible d'optimiser l'ensemble de ces paramètres. Il est évident que certains compromis doivent être acceptés.

La conception d'un cache reste conditionnée par plusieurs contraintes et compromis; les paramètres importants dans la conception d'un cache restent : l'algorithme de placement qui détermine la correspondance entre les adresses mémoire et les locations cache **[27]**.

De plus, du fait de la gestion explicite des déplacements de données, le temps d'accès est indéterminé dans un cache. En effet, un accès à un cache prend plus ou moins de temps selon si le cache possède la donnée ou est obligé d'aller la chercher en mémoire principale **[28].**

II.4.2.2 La mémoire Scratchpad (SPM)

Une mémoire scrachpad est une mémoire de type RAM située dans le processeur ou à l'extérieur. Comme dans le cas des mémoires cache, son espace de stockage est généralement une mémoire statique. Par rapport à la mémoire principale, elle est de faible capacité et se trouve dans un espace d'adressage séparé (figure II.9). Cependant la mémoire principale et la mémoire scratchpad sont connectées aux mêmes bus d'adresses et de données.

Comme dans le cas des mémoires cache, la mémoire scratchpad a un temps d'accès relativement faible à ses données. Mais à la différence des mémoires cache, la mémoire scratchpad garantit des accès en 1 cycle d'horloge, et n'a pas à traiter des problèmes de conflits. En outre, une telle mémoire présente l'avantage d'exhiber une consommation d'énergie relativement faible.

À la différence d'une mémoire cache, pour laquelle l'adressage est transparent, la gestion d'une mémoire scratchpad doit être assurée par le logiciel souhaitant en tirer parti. Ceci peut être réalisé par le programmeur. Certains compilateurs destinés aux applications embarquées étendent le langage supporté avec des mots clés permettant de placer explicitement des variables en mémoire scratchpad.

Enfin cette gestion peut être implémentée dans le programme à compiler, par le compilateur lui-même. Cette approche automatique a fait l'objet de nombreuses études ces dernières années dans le but d'améliorer le temps moyen d'exécution des programmes et réduire leur consommation d'énergie **[25].**

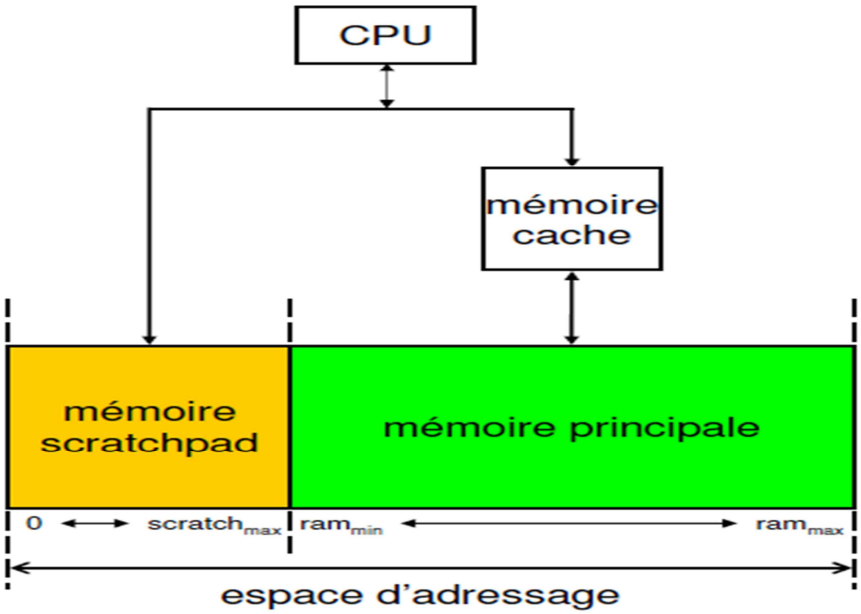


Figure II.9 Position de la mémoire scratchpad dans la hiérarchie mémoire [25].

II.4.2.3 Les points similaire et différents de la mémoire scratchpad et la mémoire cache

Le tableau suivant permet de comparer ces deux types de mémoires :

Les points similaires	▪ connectées aux mêmes bus d'adresses et de données ▪ les mémoires scratchpad (SPM) et cache ont un temps d'accès faible.
Les points différents	▪ la mémoire SPM disposant de son propre espace d'adressage à la différence d'une mémoire cache qui ne possède pas son propre espace d'adressage. ▪ La mémoire scrachpad à une latence d'accès unique (1 cycle) alors qu'un accès à travers la mémoire cache à une latence d'accès variable qui dépend de la présence de la donnée accédée dans la structure interne du cache. ▪ la gestion d'une mémoire scratchpad est assurée par le logiciel, en effet les compilateurs et les systèmes d'exploitations qui choisissent des portions de code et de données les plus sensibles et les mappent dans l'espace SPM par contre dans la mémoire cache sa gestion est assuré par le matériel. ▪ les mémoires cache améliorent la rapidité du programme, elles ne sont pas toujours adéquates dans le cas des systèmes embarqués : elles augmentent la taille du système ainsi que son coût énergétique comparé aux SPMs qui sont plus adapté dans les systèmes embarqués. ▪ Le coût de la mémoire scratchpad est plus bas que la mémoire cache. ▪ Les mémoires scratchpads sont plus éco énergétiques que les caches car elles n'ont pas besoin d'une logique complexe de décodage des adresses, elles peuvent également réduire le nombre d'erreur de conflit dans la cache.

Tableau II.2 comparaison entre cache et Scratchpad

II.5 Conclusion

Tout au long de ce chapitre nous avons présentés quelques notions sur les mémoires tel que: les caractéristiques des mémoires, les types de mémoires et la hiérarchie mémoire .Il est donc extrêmement important de prendre en compte les temps d'accès aux mémoires dans la hiérarchie mémoire afin d'améliorer le temps d'exécution. Cela est motivé par le fait que notre travail s'intègre dans le contexte d'évaluation de performance de la mémoire Scratchpad dans une architecture embarquée multicores.

Une étude consacrée sur ce type de mémoire qui est très recommandé pour les systèmes embarqué et les techniques de gestion de cette mémoire sont faites dans le chapitre suivant.

Chapitre III

Technique de gestion de la mémoire Scratchpad

III.1 Introduction

Les exigences toujours croissantes de la pile logicielle du système embarqué, les limitations du domaine uni-processeur et la mise à l'échelle de la technologie ont poussé au passage vers la technologie multiprocesseur. Un sous-produit des phénomènes multicores est l'intégration rapide des mémoires scratchpad dans la hiérarchie de mémoire en raison de leur prévisibilité accrue, de leur superficie réduite et de leur consommation d'énergie.

De plus, l'adoption de plates-formes multicores motive davantage la nécessité d'environnements multi-tâches, Les ressources système telles que les SPMs doivent être partagées. Le partage des SPMs est une tâche cruciale car elles ont tendance à contenir des données critiques (données couramment utilisées, données sensibles, etc.) et il a été démontré que l'utilisation efficace des SPMs entraîne une meilleur performance d'exécution et de grandes économies d'énergie.

Les approches traditionnelles supposent qu'une application donnée bénéficie d'un accès complet aux ressources sous-jacentes, cependant, dans des environnements multitâches, ces approches ne fonctionneront pas puisque l'état du Système (applications en cours d'exécution, besoins en mémoire) varie.

Des techniques de partage des SPM à puce ont été proposées, mais elles supposent que les applications sont connues à l'avance. La virtualisation de la SPM a été proposée avec l'apparition des plateformes embarquées multicores.

Dans ce chapitre nous présentons le concept de gestion de la SPM, un dispositif hardware qui permet de fournir au système d'exploitation et aux applications une vue simplifiée de la mémoire scratchpad en virtualisant l'accès à cette dernière.

III.2 Etat de l'art

III.2.1 Placement des données en mémoire

La première catégorie comprend les techniques de gestion SPM qui se focalisent sur le placement des données en mémoire en fonction du type de mémoire.

Ces approches tentent de répondre à la question : quelles variables du programme devraient être allouées à quelle mémoire ou banc ? Dans ces techniques, à cause de la taille réduite de la SPM, les variables les moins utilisées sont d'abord allouées aux bancs mémoire lents, alors que les variables les plus fréquemment utilisées sont gardées en mémoire rapide le plus long possible. Ces méthodes utilisent les données des profils d'exécution pour collecter des informations sur les fréquences d'accès (nombre de fois qu'une donnée est accédée) afin de placer les données fréquemment utilisées dans la mémoire rapide et les autre données dans la mémoire lente pour minimisé le temps d'exécution.

D'autres techniques sont considérées comme un raffinement de celles présentées précédemment. En effet, il serait intéressant de conjuguer le meilleur placement de données en respectant les types de mémoires et la localité des accès mémoire.

III.2.2 Travaux connexes [29]

Les SPM sont devenue, au fil des années, une composante critique de la hiérarchie des mémoires et devraient être les mémoires de choix pour les futures plates-formes à plusieurs noyaux. Contrairement aux plates-formes basées sur le cache où les données sont chargées dynamiquement dans le cache avec espoir d'un certain degré de réutilisation en raison de la localisation d'accès, les systèmes basés sur SPM dépendent complètement du compilateur pour déterminer quelles données charger. Le placement des données sur la mémoire est souvent effectué de manière statique par le compilateur par analyse statique ou profil d'application, l'emplacement des données est connu a priori qui augmente la prévisibilité du système.

Panda et al. **[30]** ont essayé d'allouer toutes les variables scalaires sur les SPM. Ils ont identifié des matrices candidates pour le placement sur les SPM en fonction du nombre d'accès aux matrices et de leur taille.

Verma et al. **[31]** ont examiné les tableaux d'une application et identifié les candidats pour le fractionnement avec l'objectif final de trouver un point de partage optimal afin de mapper la zone la plus utilisée du tableau vers SPM.

Poletti et al **[32]** a proposé un gestionnaire de mémoire qui prend en charge l'allocation dynamique de l'espace SPM, qui prend en charge l'allocation par blocs.

Verma et al **[33]** ont proposé trois différentes stratégies de partage de SPM pour prendre en charge le mappage des applications sur MPSoC (par exemple, temporel, spatial et hybrides).

III.2.3 Politique d'allocation de SPM [29]

Nous définissons quatre politiques d'allocation basées sur des blocs :

1. **L'allocation temporelle**

   Qui consiste à partager temporairement l'espace SPM entre les différentes applications. Pour partager l'espace SPM, cette stratégie échange le contenu du SPM sur un commutateur de contexte pour éviter les conflits avec d'autres tâches.

2. **L'allocation dynamique**

   Le schéma d'allocation dynamique mappe les données à l'espace SPM tant qu'il y a de l'espace et ne donne pas la priorité à ce qui se passe sur puce et ce qui se passe hors de puce. S'il n'y a pas d'espace sur la puce (dans l'espace SPM), alors les données sont mappées hors-puce (DRAM).

3. Allocation spatiale

L'allocation spatiale, qui divise également l'espace SPM entre les différentes applications s'exécutant sur le système. Elle alloue une partie de l'espace SPM à chaque application, dans ce cas, les blocs plus grands pour chaque application ne rentreraient pas dans leur espace SPM affecté, de sorte que la politique d'allocation était obligée d'allouer moins Blocs critiques sur l'espace SPM.

4. politique d'allocation axée sur les priorités

Il est possible d'avoir différentes politiques axée sur les priorités, nous avons cité deux types axés sur les données ou les blocs, et axé sur l'application.

-l'approche basée sur les données peuvent avoir des priorités individuelles, ce qui permet de décidé au moment de l'exécution où chaque bloc doit être mappé.

-l'approche axée sur l'application ou chaque application a des exigences en temps réel et possède une priorité au niveau SPM, ce qui est utile lorsque le programmeur souhaite disposer d'un espace spécifique d'un type donné.

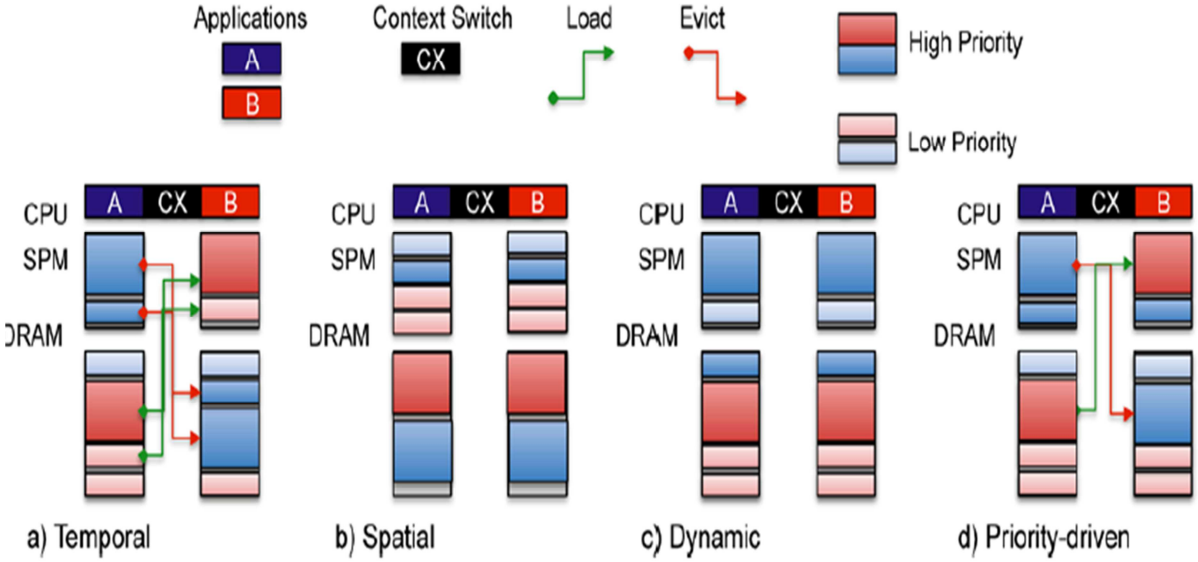


Figure III.1 Politiques d'allocation de mémoire contrôlées par logiciel [29].

III.2.4 Les différentes technologies de la mémoire scratchpad

III.2.4.1 Une architecture de système basée sur une mémoire virtuellement partagée (VS-SPM) [34]

Dans cette architecture, nous avons un système sur puce (SoC) et un hors-puce DRAM qui peut contenir des données ainsi que des instructions. Le SoC détient des processeurs multiples avec leurs SPM locaux, un mécanisme de communication / synchronisation entre processeurs, un circuit d'horloge et certains ASIC.

Les SPMs des processeurs individuels constituent un VS-SPM. Chaque processeur a un accès rapide à sa propre SPM ainsi aux SPM des autres processeurs. En ce qui concerne les SPM d'autres processeurs sont appelés SPM distants.

L'accès à des SPM distants est possible en utilisant des liens de communication sur puce rapides entre processeurs. L'accès à la DRAM hors-puce, cependant, est très coûteux en termes de latence et d'énergie. Étant donné que l'énergie d'accès et la latence de VS-SPM sont beaucoup plus faibles que les valeurs correspondantes de DRAM, il est important de s'assurer que le VS-SPM satisfait autant de demandes de données (réalisées par les processeurs) que possible.

Cette architecture apporte des contributions plus précisément des techniques de compilation efficaces pour les applications embarquées dominées par des tableaux. En effet, un tel compilateur peut augmenter les opportunités de partage de données lorsque plusieurs processeurs fonctionnent sur un ensemble de tableaux en parallèle, à titre exemple dans le traitement d'une boucle parallèle, tous les processeurs du système participent au calcul et exécutent chacun un sous-ensemble d'itérations de la boucle.

Cette approche est orientée vers l'élimination des accès hors-chip DRAM causés par la communication entre processeurs.

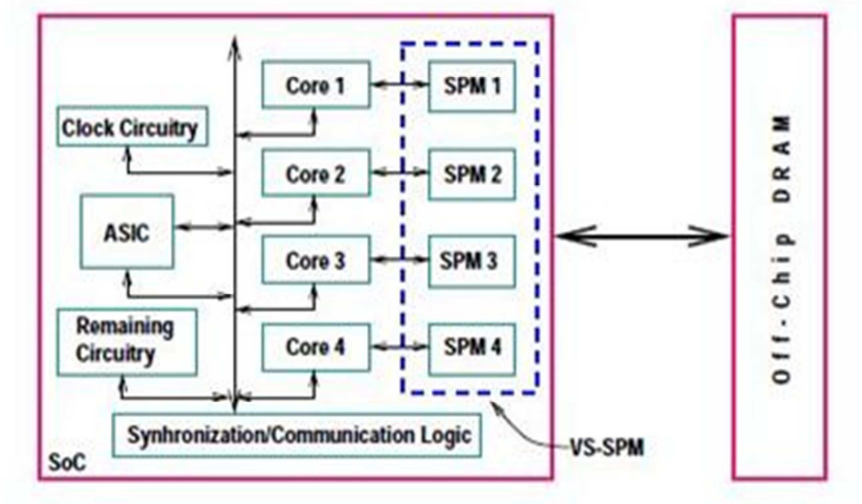


Figure III.2 Une architecture de système basée sur VS-SPM [34].

III.2.4.2 Virtualisation des mémoires distribuées ScratchPad pour une faible puissance et une exécution d'application approuvée [29]

Dans cette approche, ils ont introduit le concept de SPMVisor, une couche matérielle / logicielle qui virtualise l'espace mémoire Scratchpad. Les contributions de cette approche sont les suivantes:

- ✓ Faciliter l'utilisation des SPMs distribués de manière efficace, transparente et économe en énergie des ressources mémoire sur puce.
- ✓ Le concept de ScratchPad Memories virtuel (vSPM), permettant aux programmeurs une vue transparente de l'espace mémoire sur puce.
- ✓ Un mécanisme dynamique et efficace de gestion des ressources fondé sur l'idée d'une allocation basée sur les politiques de privilège d'application et les métriques de priorisation de niveau de données pour gérer efficacement la mémoire on-chip.

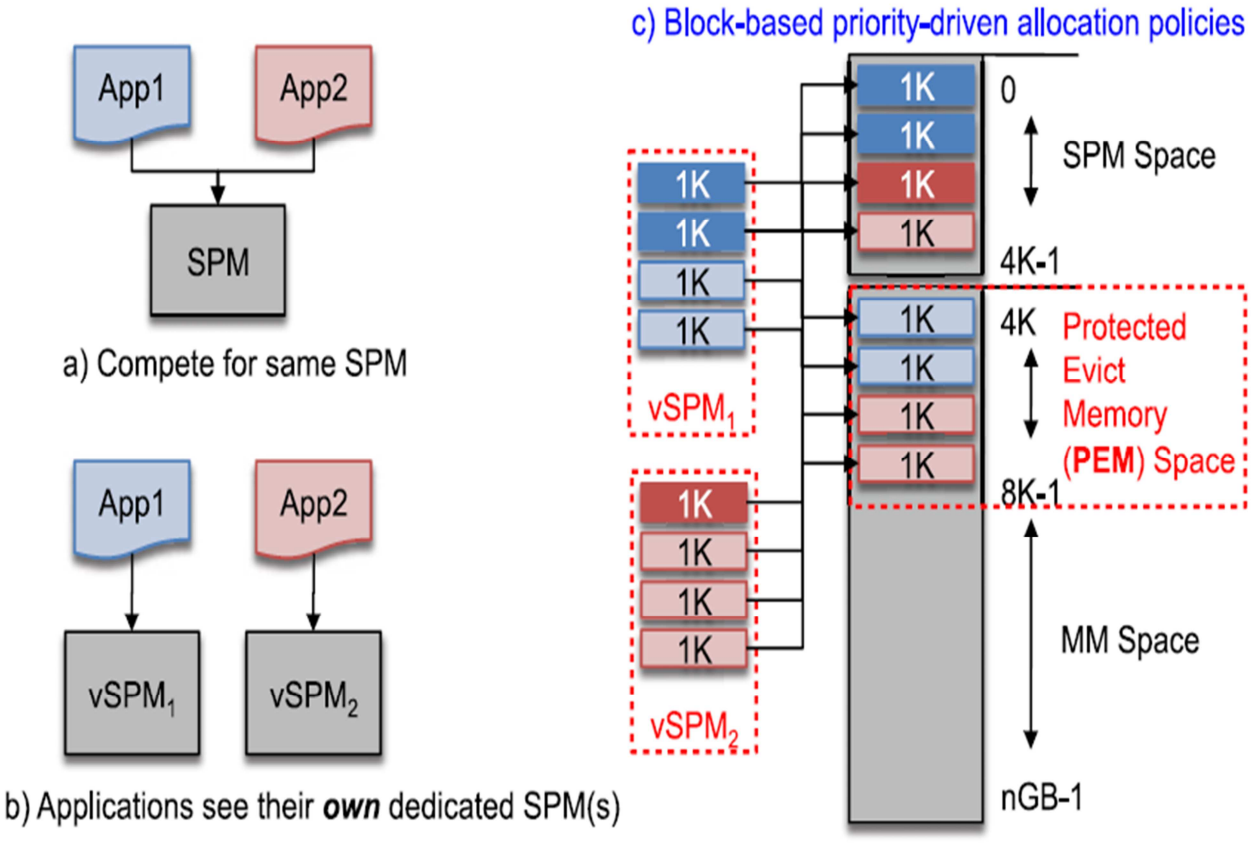


Figure III.3 Virtualisation des mémoires distribuées [29].

Par conséquent, si le système souhaite exécuter plusieurs applications avec le support de SPM, il y aura une contention pour l'espace SPM physique. La figure III.4 représente la vue d'ensemble de cette solution, de sorte que chaque application peut voir son propre SPM plutôt que de rivaliser avec d'autres pour l'espace. Pendant ce temps, la couche de virtualisation priorise dynamiquement quelles données vont sur puce et hors puce.

Cette approche permet de la translation d'adresse comme celui de la mémoire virtuelle classique, par exploitation de la notion d'IPA, qui signifie l'adresse physique intermédiaire et est utilisé pour adresser l'espace vSPM. L'idée est que les adresses virtuelles de l'application sont traduites par l'unité MMU[1] de la CPU et génèrent des adresses physiques intermédiaires qui sont par la suite traduite ou translater par le SPMvisor et génèrent les adresses physiques pour pointer vers les blocs physiques.

En outre, le SPMvisor utilise une logique complexe de décodage des adresses. On relève que ces résultats de simulation sont apparemment satisfaisants, quoi que nous pensions que cette architecture nécessite trop de ressources matérielles et logicielles pour un système embarqué.

Pour mieux exploiter les performances de la mémoire scrachpad et pallier aux insuffisances des approches citées ci-dessus, nous avons introduit un nouveau dispositif hardware pour la gestion d'une mémoire scratchpad. Dans la section qui suit nous allons présenter l'architecture fonctionnelle de notre circuit.

III.3 Circuit de gestion de la SPM

Notre dispositif sera utilisé dans le contexte d'un système embarqué ayant une architecture multiprocesseur. Il permet de fournir au système d'exploitation et aux applications une vue simplifiée de la mémoire Scratchpad en virtualisant l'accès à cette dernière, en effet il peut allouer dynamiquement un espace SPM virtuel pour le code et les données des applications de façon transparente en exploitant l'espace libre des mémoires SRAM on-chip et off-chip et en émulant la SPM dans la DRAM dans le cas d'insuffisance d'espace SRAM.

III.3.1 Objectif de circuit de gestion SPM

L'objet de ce dispositif matériel est de faciliter la gestion dynamique de la SPM en permettant à une tâche de satisfaire sa demande en SPM en utilisant la mémoire la plus performante en fonction de la disponibilité. Ce circuit permet de virtualiser l'espace de la SPM en exploitant les hiérarchies de mémoires du système telles que les SRAM on-chip, les SRAM off-chip et les DRAM. Quand une tâche en exécution demande un espace SPM, notre dispositif lui crée, de façon transparente, un espace SPM virtuel qui peut être logé dans :

1. La mémoire SRAM on-chip partagée par tous les processeurs
2. La mémoire SRAM off-chip commune aux processeurs si la SRAM on-chip est pleine
3. Et enfin dans la mémoire DRAM si aucun espace n'est libre dans les SRAMs.

[1]MMU :memory management unit

III.3.2 Principe de fonctionnement du circuit de gestion de la SPM

Notre solution permet de séparer le choix des éléments à mettre dans la SPM de sa propre gestion. Le choix des données et les codes à mettre en SPM est laissée à l'appréciation du logiciel. La gestion proprement dite de cette SPM est réalisée par matériel. De plus, la taille de la SPM étant limitée, il arrive que l'on ne puisse pas satisfaire la demande d'une tâche. Au lieu de mettre cette tâche en attente, le matériel émule la SPM sur une mémoire moins rapide comme une DRAM.

Le circuit de gestion de SPM est implémenté sous une architecture embarqué multiprocesseur. Nous avons un système sur puce (SoC) détient des processeurs multiples avec une SRAM_on_chip, l'architecture est composée aussi des SRAM off-chip et de DRAM ainsi d'autre type de mémoires (ROM, Flash, ….etc.). La figure ci-dessous montre l'architecture de notre circuit.

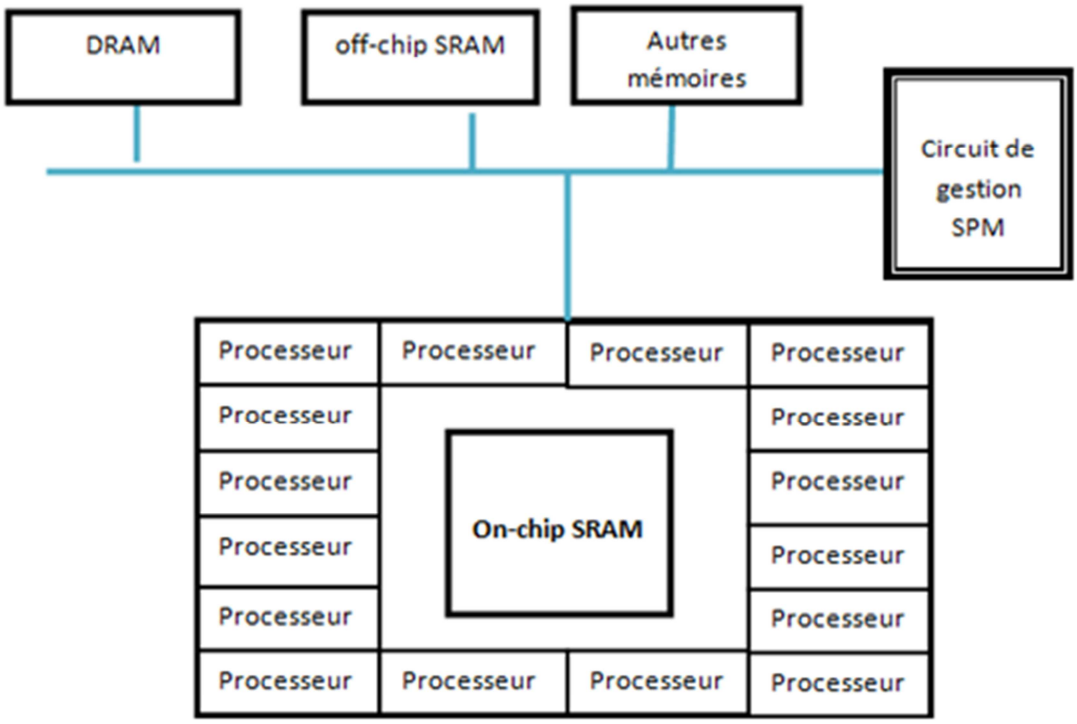


Figure III.4 Architecture du système embarqué.

Le dialogue avec le circuit est réalisé à travers un ensemble d'API[2], tout d'abord, quand une tâche a besoin de crée un espace SPM, elle le fait à travers un appel à une méthode d'allocation, l'appel se fait sans préciser l'adresse de l'espace SPM car cette dernière n'a pas besoin de connaitre l'état de la SPM et en retour elle reçoit un pointeur vers l'adresse de début de l'espace a alloué par le circuit en fonction de la disponibilité.

Donc pour la tâche, il est possible d'allouer un espace SPM à tout moment. En effet dans un système en entier, l'espace SPM est mappé effectivement dans les SRAMs les plus rapides si y a suffisamment d'espace libre sinon dans la DRAM si aucun espace libre n'est disponible.

Enfin si une tâche à besoin de libérer l'espace SPM elle fait appelle à une autre méthode spécifiée dans l'ensemble d'API.

La figure ci-dessous illustre la procédure de réservation de l'espace mémoire avant l'entrée d'une nouvelle tâche i au système .La tâche i initie une demande d'allocation de 3KO dans l'espace SPM. Le circuit de gestion de la SPM cherche d'abord un espace libre dans la RAM on-chip comme il ne reste que 2 Ko libre, il examine l'emplacement suivant qui est la SRAM_off_chip. Cette dernière dispose de suffisamment d'espace libre, il lui alloue donc l'espace dans cette zone et retourne un pointeur vers le début de l'espace.

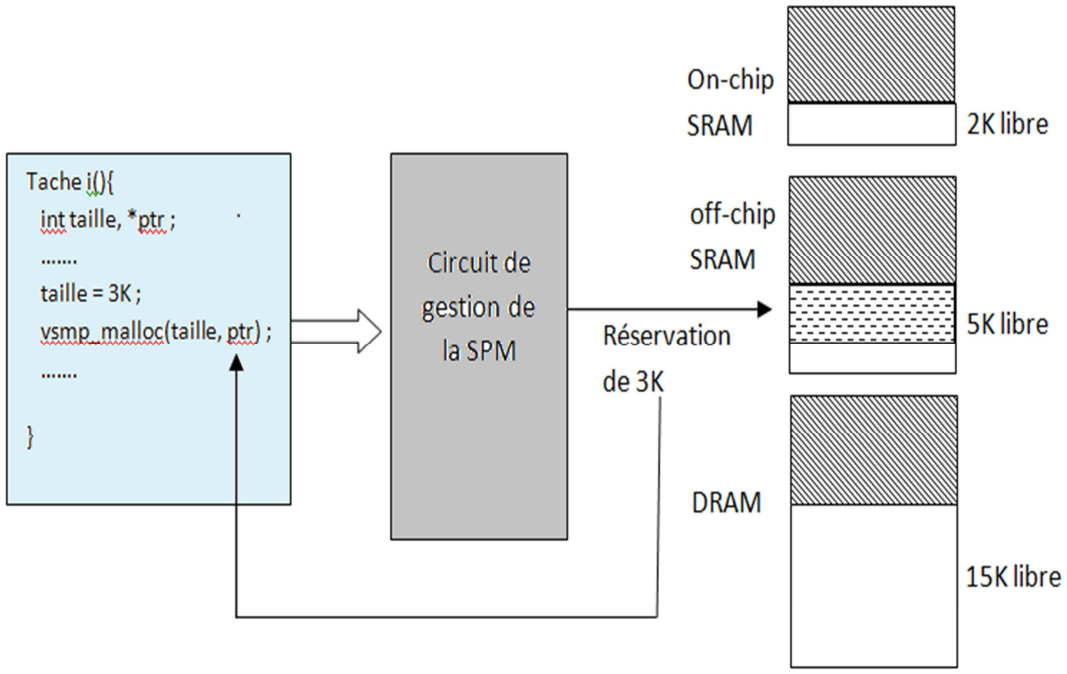


Figure III.5 Processus de réservation de la SPM.

[2]API :application Programming Interface

III.3.3 Structure de circuit de gestion de SPM

1. interface externe du circuit

Le circuit de la gestion SPM forme de trois lignes d'adresses A0, A1 et E. Les lignes A0, A1 permettent d'adresser les registres internes. La ligne E permet d'activer le circuit pour exécuter la fonction demandée (réservation, libération, …etc.). La ligne Err est activée quand il n y a plus d'espace libre.

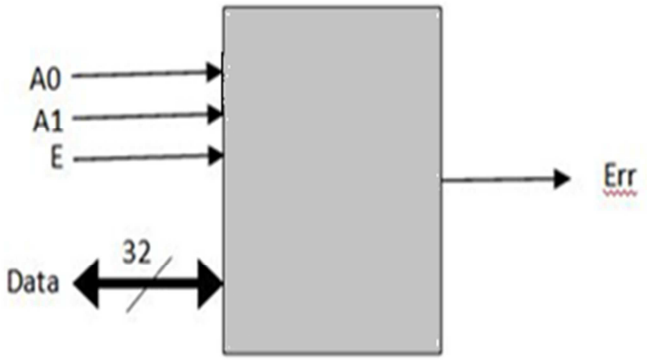


Figure III.6 Signaux d'interface du circuit.

2. Structure interne du circuit

La figure III.8 donne la structure interne du circuit SPM. Il est composé de trois registres d'interfaces : commande, base et état, d'un contrôleur et de trois bancs mémoires.

Notre circuit dialogue avec le processeur à travers les registres commande, état et de base. Le registre commande permettant de demander une réservation ou une libération d'espace, le registre d'état indiquant si la réservation ou la libération est effectuée et enfin, le registre de base fournissant soit le pointeur vers le début de l'espace réservé par le circuit , soit la taille a réservé ou bien l'adresse de base .

Le contrôleur est l'organe responsable des opérations du circuit. Il est composé de quatre éléments : l'interpréteur permettant d'interpréter la commande et de générer les signaux nécessaires. Le module de réservation permettant de réserver l'espace demandé, le module de libération permettant de libérer un espace et enfin le module de positionnement du banc mémoire.

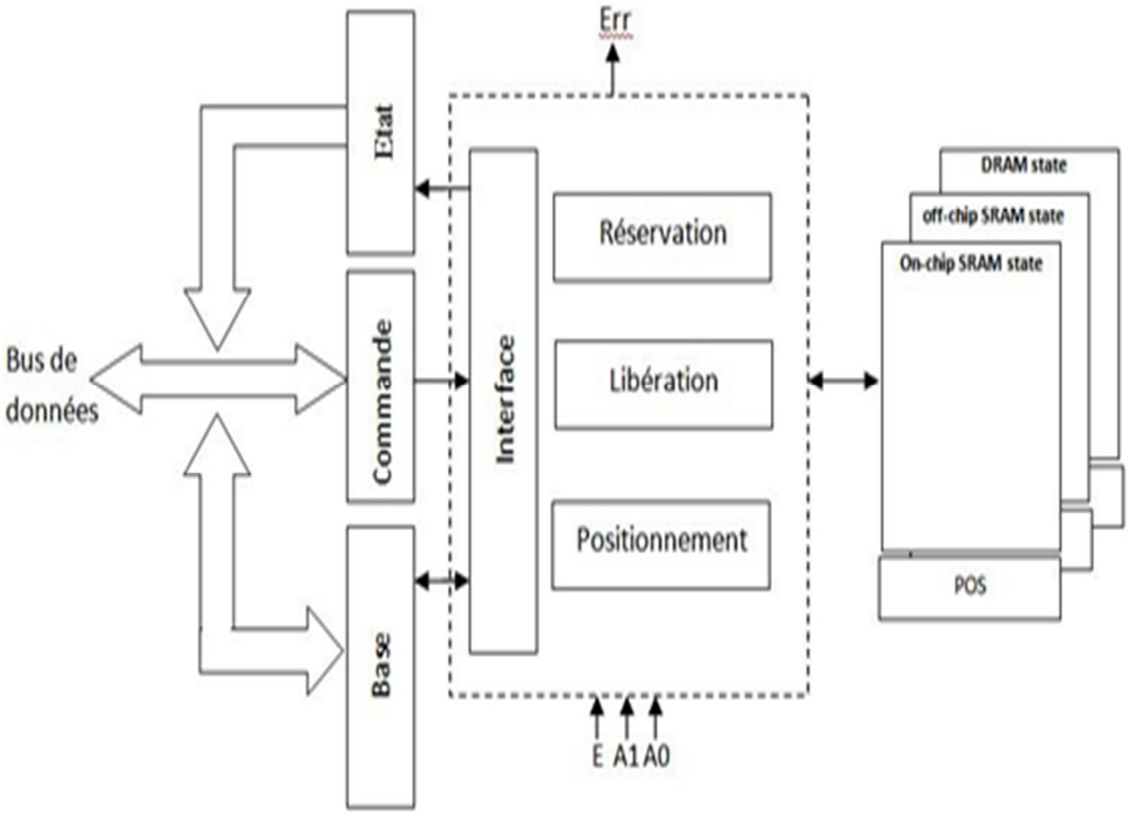


Figure III.7 Structure interne du circuit de gestion SPM.

Les bancs mémoires contenant l'état des blocs mémoires on-chip, of-chip et DRAM.

Chaque banc mémoire est organisé sous forme de lignes où chaque ligne correspond à un bloc mémoire (on-chip, off-chip ou DRAM). Une ligne est composée d'un champ état sur 1 bit indique l'état du bloc (0 libre, 1 réservé). La taille de chaque ligne de banc est 1KO.

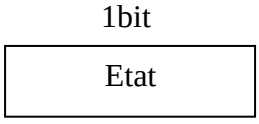
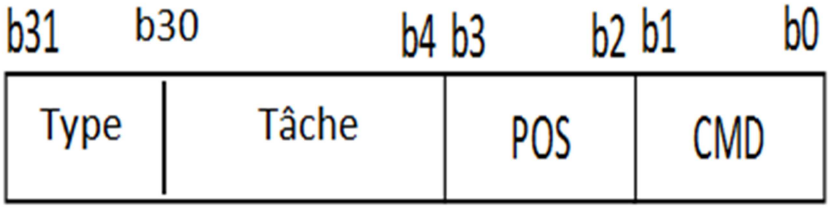


Figure III.8 Structure d'une ligne du banc d'état de blocs.

La structure du registre commande est donnée dans la figure III.9. Le champ **CMD** sur deux bits est utilisé pour spécifier la commande. Le champ **POS** sur deux bits également est utilisé pour spécifier le banc, le champ **Tâche** sur **27** bits pour spécifier et identifier la tâche donc on peut avoir $\mathbf{2^{27}}$ tâches et le champ **TYPE** sur un bit est utilisé pour déterminer le type de réservation (méthode vspm-malloc ou bien méthode malloc). Les autres bits ne sont pas utilisés.



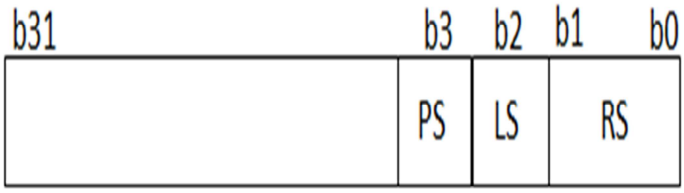
CMD	Signification
00	Positionnement d'un banc
01	Réservation mémoire
10	Liberation memoire

POS	Signification
00	Banc On-chip RAM state
01	Banc Off-chip RAM state
10	Banc DRAM state

TYPE	Signification
0	Méthode malloc()
1	Méthode vspm_malloc()

Figure III.9 Structure du registre commande.

La structure du registre état est indiquée dans la figure III.10. Trois champs indiquent l'état de l'opération effectuée. Le champ RS (Reservation state) indique si la réservation est effectuée, pas encore ou impossible. Le champ LS (libération state) indique si la libération est finalisée et enfin le champ PS (positionnement state) indique si le positionnement d'un banc est finalisée.



RS	Signification
00	Réservation en attente
01	Réservation effectuée
10	Erreur de réservation (espace insuffisant)

LS	Signification
0	Libération en attente
1	Libération effectuée

PS	Signification
0	Positionnement en attente
1	Positionnement effectué

Figure III.10 Structure du registre état.

III.4 Conclusion

Dans ce chapitre, nous avons introduit le concept de dispositif hardware, afin de faciliter l'utilisation de SPM de manière efficace, pour la réduction de temps d'exécution des tâches. Nous avons présentés un contrôleur, prend en charge les listes de contrôle d'accès au niveau des SPMs on-chip et hors chip ainsi la DRAM. Enfin, afin de gérer efficacement les transactions, notre circuit prend en charge les stratégies d'allocation dynamique.

Pour montrer que notre approche améliore les performances en terme de la réduction de temps d'exécution par rapport aux systèmes traditionnels, nous allons simuler notre circuit afin de montré l'avantage d'utilisation des SPM (virtuel) a tous moment dans un environnement varie (un environnement multitâches).

En effet nous allons explorer le mappage des instructions et des données sur les SPMs pour montrer que le partage de l'espace on chip réduit à la fois le temps d'exécution ainsi que l'énergie.

Dans le chapitre suivant nous sommes prêts à la mise en œuvre de notre projet, la prochaine étape sera la plus importante, nous procéderont l'implémentation d'une simulation a événement discret et nous ferons les différent analyse des résultats qui seront produit par les différentes simulations.

Chapitre IV

Réalisation

IV.1 Introduction

Les plates-formes cibles sur lesquelles porte notre étude sont des environnements embarqués multi-cores qui peuvent être situées sur de nombreux domaines temps réels. Par conséquent, il peut être relativement difficile de conduire des expériences de longue durée dans de telles architectures. Nous avons donc choisi de faire usage d'un simulateur afin de tester l'efficacité de notre approche. Un autre avantage de l'utilisation d'un simulateur est la possibilité de pouvoir maitriser l'ensemble des paramètres de la plate-forme simulée, ce qui est peut être difficile, voire impossible dans un environnement réel.

Dans ce chapitre, nous allons débuter par la description générale de notre système, puis présentations des différents outils de développement, enfin l'évaluation des résultats obtenu après la simulation de notre système

IV.2 Motivations et objectifs

L'objectif de ce travail consiste à observer le comportement de notre modèle de gestion de SPM par simulation et collecter des informations statistiques concernant le taux d'utilisation de chaque mémoires afin d'obtenir le temps d'exécution totale des tâches. Pour réaliser ce travail un outil qui permet de tester et simuler notre gestionnaire de SPM est utilisé. Nous allons commencer par une présentation rapide du type de simulation envisagée ainsi que les paramètres ayant été considérés lors de ces expérimentations. Nous exposerons ensuite les résultats obtenus.

IV.3 Description du système étudie

IV.3.1 Modélisation du système

Le modèle est un schéma, une représentation simplifiée du système qu'on veut simuler, un modèle peut se limiter à la représentation d'une partie ou d'une fonctionnalité du système. Un modèle est donc une représentation d'un système (réel ou imaginaire) dont le but est d'expliquer et de prédire certains aspects du comportement de ce système.

Cette représentation est plus ou moins fidèle car d'une part, le modèle devra être assez complet afin de pouvoir répondre aux diverses questions qu'on peut se poser sur le système qu'il représente, et d'autre part, il ne doit pas être trop complexe pour pouvoir être facilement manipulé. Ceci implique qu'il y a un intérêt à bien définir les limites ou les frontières du modèle qui est censé représenter le système

IV.3.1.1 Modélisation du fonctionnement globale du système

Nous pouvons résumer le fonctionnement global du système dans le schéma de la figure suivante :

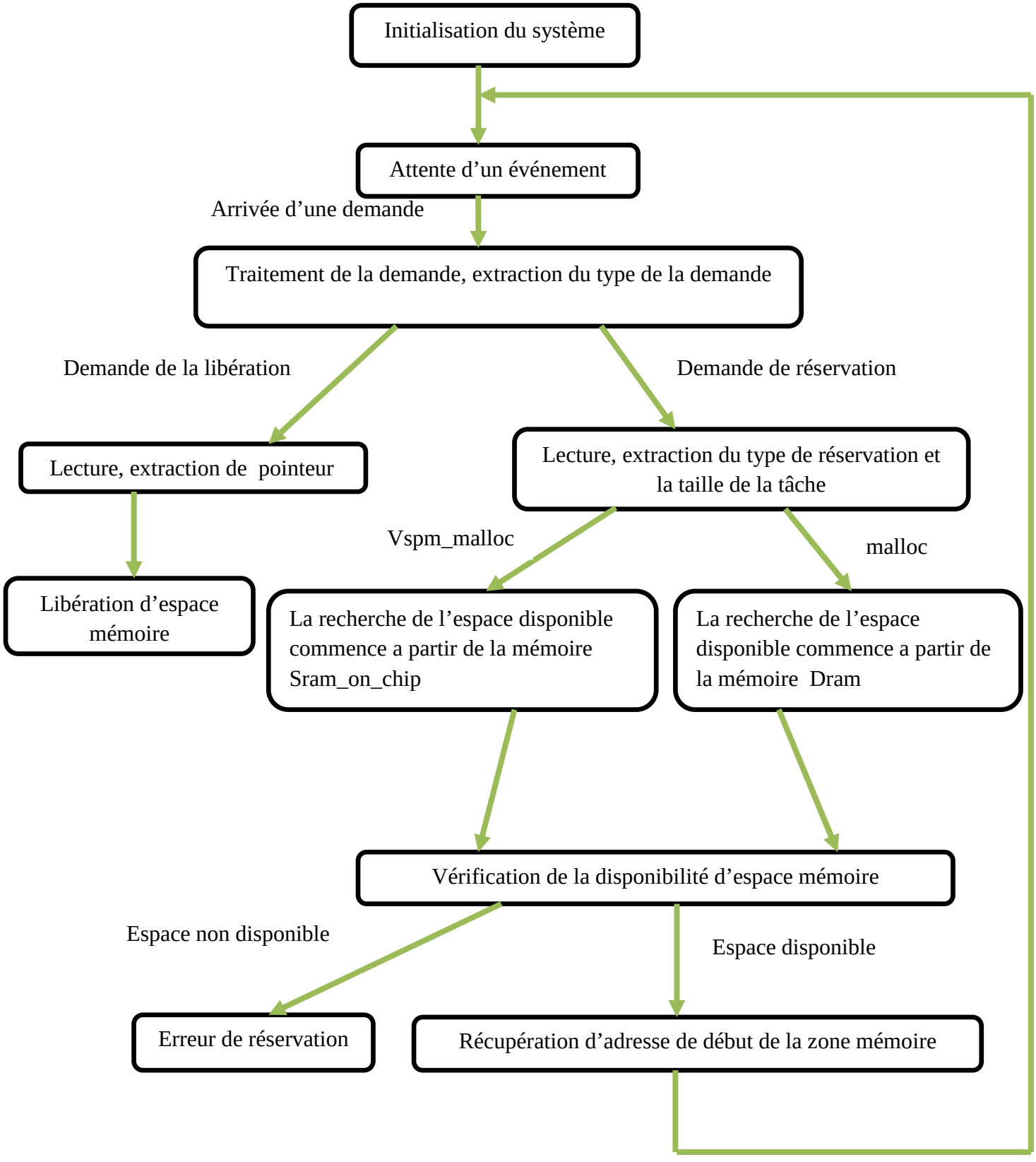


Figure IV.1 Schéma de modélisation du système.

IV.3.1.2 modélisation par le diagramme de classe

Dans cette phase une nouvelle vue du modèle fait son apparition. Cette vue exprime les modules et les exécutables physiques sans aller à la réalisation concrète du système.

Après avoir décrit schématiquement notre système nous allons utiliser le diagramme de classes.

- **Présentation de l'UML (*UnifiedModelingLanguage*)**

  Le langage de modélisation unifié (UML), est un langage de modélisation graphique conçu pour fournir une méthode normalisée pour visualiser la conception d'un système. Il est couramment utilisé en développement logiciel et en conception orientée objet **[35].**

  UML permet d'exprimer les modèles objets à travers un ensemble de diagrammes. Ces derniers sont des moyens de description des objets ainsi que des liens qui les relient **[36].**

  Le diagramme de classe qui résume l'ensemble de classe de notre système ce présente ci-dessous

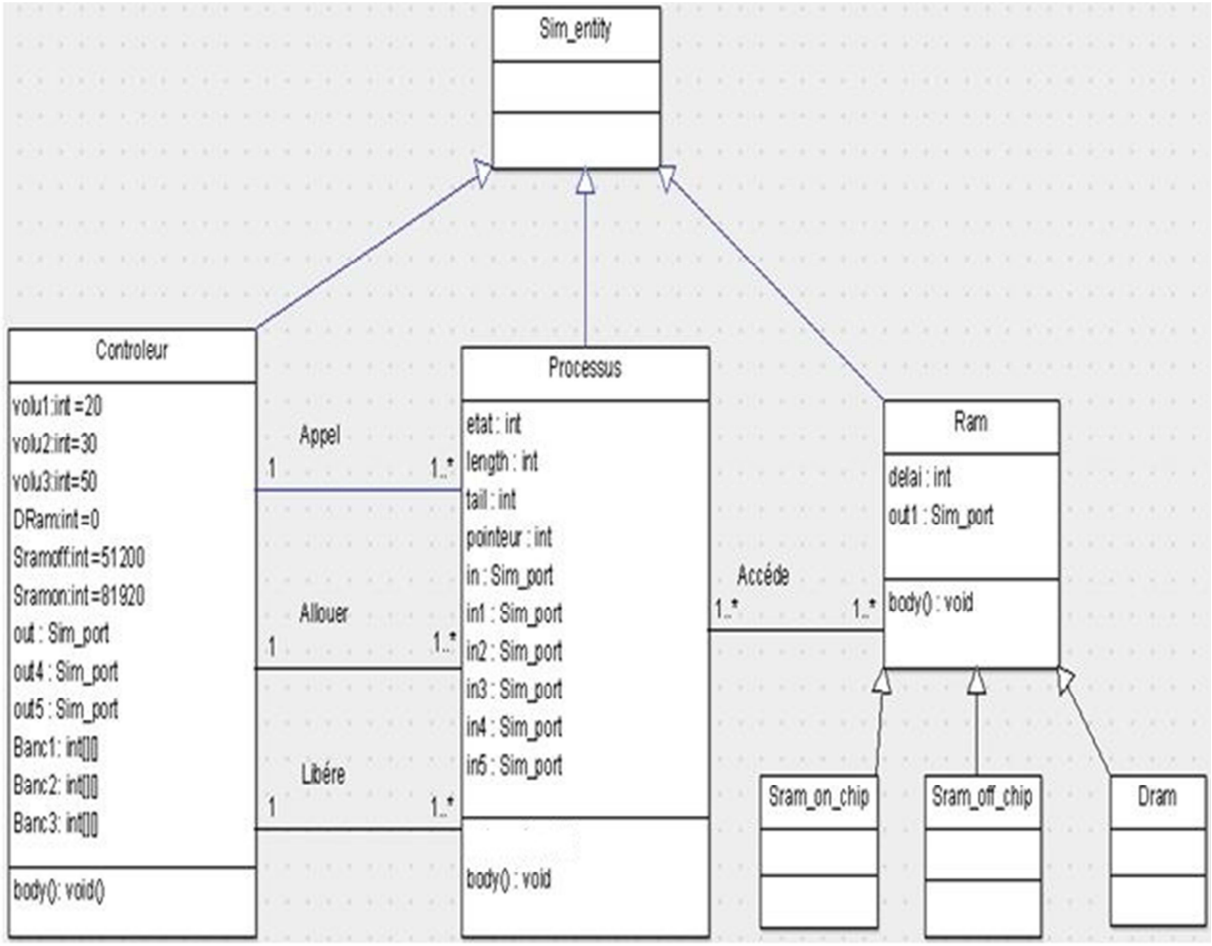


Figure IV.2 Le diagramme de classe.

IV.3.2 Les paramètre du système

Plusieurs paramètres influencent directement sur les performances de l'architecture proposée par notre approche. Un des paramètres les plus importants est le temps d'accès aux différentes mémoires. En effet, le temps d'accès est un critère qui détermine les performances d'une mémoire ainsi que les performances de l'architecture globale du système par conséquent plus le temps d'accès est faible, plus la vitesse est grande.

Un autre paramètre est la taille de chaque mémoire, l'avantage de définir une taille est de pouvoir satisfaire la demande d'allocation d'espace mémoire pour chaque tâche.

	Nom du paramètre	
Type de mémoire	Temps d'accès	La taille
SRAM_On_chip	**10(ns)**	**20 KO**
SRAM_Off_chip	**20 (ns)**	**30 KO**
DRAM	**60 (ns)**	**50 KO**

Tableau IV.1 exemple de valeurs possible pour les différentes mémoires.

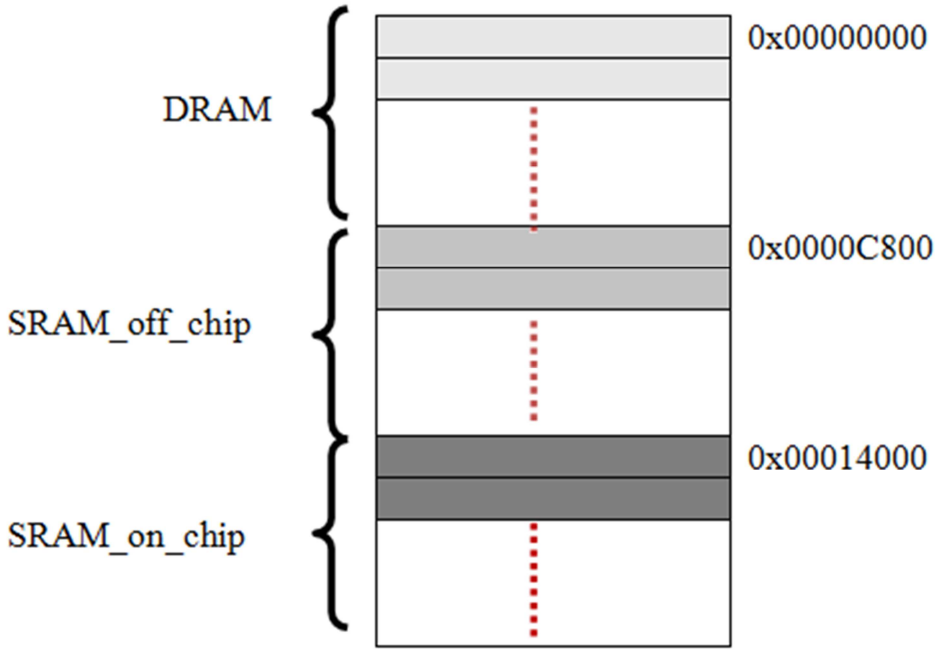


Figure IV.3 Organisation de la mémoire.

IV.3.3 présentation de l'algorithme « gestion de contrôleur »

Dans cette partie, nous allons présenter l'algorithme que nous avons conçu pour l'accomplissement de notre projet.

- **Algorithme « gestion de contrôleur »**

/* définition des paramètres de configuration */

taille1 : entier initialisé à 20 /* la taille de Sram_on_chip (20 lignes)*/

taille2 : entier initialisé à 30 /* la taille de Sram_hors_chip (30 lignes)*/

taille3 : entier initialisé à 50 /* la taille Dram (50 lignes)*/

DRam : entier initialisé à 0 /* l'adresse de base de Dram*/

Sramoff: entier initialisé à 51200 /* l'adresse de base Sram_off_chip*/

/* 51200 correspond à 0x0000C800 en hexadécimal */

Sramon: entier initialisé à 81920 /* l'adresse de base Sram_on_chip*/

/* le 81920 correspond à 0x00014000 en hexadécimal */

/* déclaration des bancs mémoires */

Banc1 : tableau [**taille1**] de **entier** /* déclaration de banc mémoire Sram_on_chip */

Banc2 :tableau [**taille2**] de **entier** /* déclaration de banc mémoire Sram_off_chip */

Banc3 :tableau [**taille3**] de **entier** /* déclaration de banc mémoire Dram */

Com: tableau [**4**] de **entier** /* registre de commande*/

etat: tableau [**3**] de **entier** /* registre d'état*/

R_base :**entier** **/***registre de base***/**

libere : tableau [length][4] de **entier**

Pos :entier **/*** positionnement du banc mémoire*/

non_trouve1, non_trouve2, non_trouve3:**boolean**

Debut

- Lecture, extraire les valeurs qui se trouvent dans le registre de commande
- Vérifier le type de la commande reçu (commande de réservation ou libération)

Si (**Com**[0]=1) alors /* champ **Com[0]** spécifier la commande si Com[0]=1 alors une commande de réservation */

```
/*la taille de la tache  se trouve dans le R_base  */

 /* le type de réservation (vspm_malloc ou malloc)  dans le champ Com[3]  */

Si (Com[3]=0) alors /* vérifier s'il y'a l'espace dans DRAM*/

    priority← 3
    Pos←2

Sinon /* vérifier s'il y'a l'espace dans Sram_on_chip*/

    priority← 1
    Pos←0

Finsi

    etat[0]←0      /* la réservation en attente*/
    non_trouve1←faux
    non_trouve 2←faux
    non_trouve 3←faux
    tant que( etat[0]=0)

        /* banc mémoire  Dram*/

        Si (priority=3) alors
            k←0
            tant que (k< taille3)et (etat[0]=0)faire
                si ( Banc3[k]=0) alors
                    j←k
                    l←1

                    tant que((l<= R_base) et(k< taille3)) faire

                        si (Banc3[l]=0)
                            l←l+1
                            k←k+1
                        finsi
                    fait

                    si (l-1= R_base ) alors

                        index ←j
```

```
                                    tant que( j< (R_base + index) )faire

                                            Banc3[j]←1
                                            j←j+1
                                    fait
                                    etat[0]←1  /* la réservation est effectuée*/
                                   taille← R_base
                            finsi
                      finsi
                  i←i+1
                fait

                si(etat[0]=0) alors     /* la réservation en attente*/
                      priority←2
                      non_trouve3←vrai
                     Pos←1   /*positionnement dans le Banc Off-chip RAM */

              finsi

  finsi

 /* banc mémoire Sram_off_chip*/

Si (priority=2) alors

    k←0

    tant que (k<taille2)et (etat[0]=0)faire

        si ( Banc2[k]=0) alors
            j←k
            l←1

          tant que((l<= R_base )et(k<taille2))faire
              si (Banc2[l]=0)

                  l←l+1
                  k←k+1

              finsi

          fait
```

```
                    si (l-1= R_base )alors

                        index ←j
                        tant que( j< (R_base + index) )faire

                            Banc2[j]=1
                            j←j+1
                        fait
                          etat[0]←1      /* la réservation est effectuée*/
                          taille← R_base
                finsi
            finsi
             i←i+1
      fait
          si (etat[0]=0) alors   /* la réservation en attente*/

                       non_trouve2←vrai
                       si (Com[3]=0) alors
                          /* donner la priorité au banc Sram_on_chip*/
                          Pos←0    /*positionnement dans le Banc SRAM _on_chip*/

                      Sinon si (Com[3]=1) alors
                        /* donner la priorité au banc DRAM*/
                          priority←3
                          Pos←2   /*positionnement dans le Banc DRAM */
                     Finsi

            finsi

  finsi

/* banc mémoire   Sram_on_chip */

  Si (priority=1) alors
       k←0
      tant que (k<taille1) et (etat[0]=0) faire

              si ( Banc1[k]=0) alors
                  j=k
                  l=1
                  tant que((l<= R_base) et(k<taille1))faire
                      si (Banc1[l]=0)
                            l←l+1
                            k←k+1
```

```
                    finsi
                fait
                si (l-1= R_base) alors

                        index ←j

                        tant que( j< (R_base + index) )faire
                            Banc1[j]←1
                            j←j+1
                        fait
                        etat[0]←1   /* la réservation est effectuée*/
                        taille← R_base

                    finsi
            finsi
            i←i+1
        fait
            si(etat[0]=0) alors
                    priority←2
                    non_trouve1← vrai
                    Pos←1   /*positionnement dans le Banc Off-chip RAM */
                finsi
    finsi


     si ((non_trouve1=vrai) et (non_trouve2=vrai) et (non_trouve3=vrai)) alors

        etat[0] ←2
        Envoyer un événement au processus  lorsque l'espace est insuffisant (erreur de
     réservation)
     finsi

fait

si (etat[0]=1) alors

    si(Pos=0)
        R_base  ←index*1024+Sramon

    Sinon   si (Pos=1) alors

        R_base ← index *1024+Sramoff
```

Sinon si (**Pos**=2) alors

R_base ← index *1024+DRam

Finsi

q←0
tant que (q<length) **faire**
si ((libere [q][1]=0) et (libere [q][3]=0))alors
libere[q][0]←index
libere[q][1]← R_base
libere[q][2]←taille
libere[q][3]←**Pos**

sinon
q←q+1
finsi
fait

- Envoyer un événement au processus lorsque la réservation est effectuée (etat[0]=1)

finsi

/ * effectuer la libération */

Si (**Com**[0]=2) alors /* demande de la liberation*/

/*l'adresse début de l'espace réservé se trouve dans le registre de base **R_base** */
Indice←0
etat[1]←0 /* la libération en attente*/

tant que(indice<length) et (etat[1]=0)**faire**
si (R_base =libere[indice][1]) alors

si(libere[indice][3]=0) **alors**
Pos←0
r←libere[indice][0]
tant que (r<(libere[indice][0]+libere[indice][2])) **faire**
Banc1[r]←0
r←r+1
fait
etat[1]=1
finsi

```
                    si(libere[indice][3]=1) alors
                        Pos ←1
                        r←libere[indice][0]
                        tant que (r<(libere[indice][0]+libere[indice][2])) faire
                            Banc2[r]←0
                            r←r+1
                        fait
                        etat[1]=1
                    finsi
                    si(libere[indice][3]=2) alors
                        Pos←2
                        r←libere[indice][0]
                        tant que (r<(libere[indice][0]+libere[indice][2])) faire
                            Banc3[r]←0
                            r←r+1
                        fait
                        etat[1]=1
                    finsi
                finsi
                indice←indice+1
            fait
            si (etat[1]=1) alors

                Envoyer un événement au processus lorsque la libération est effectuée

            finsi

    finsi

fin
```

IV.4 Implémentation de la simulation

IV.4.1 Outils de développement utilisés

IV.4.1.1 Langage de programmation (java)

Java est un langage de programmation à usage général, évolué et orienté objet dont la syntaxe est proche du C. Il est destiné pour permettre aux développeurs d'applications "écrire une fois, exécuter partout (WORA : Write Once, Run Anywhere), ce qui signifie que le code qui s'exécute sur une plate-forme n'a pas besoin d'être recompilés pour fonctionner sur un autre. Le code source est compilé en bytecode puis exécuté par une machine virtuelle Java (JVM) quelle que soit l'architecture informatique **[37].**

IV.4.1.2 Environnements de travail

❖ **La plateforme NetBeans**

NetBeans est un environnement de développement intégré (EDI), NetBeans permet la prise en charge native de divers langages tels le C, le C++,..ect. Il offre toutes les facilités d'un IDE moderne (éditeur en couleurs,éditeur graphique d'interfaces).NetBeans constitue par ailleurs une plate-forme qui permet le développement d'applications spécifiques (bibliothèque Swing (Java)) **[38].**

L'environnement de base comprend les fonctions générales suivantes:

- configuration et gestion de l'interface graphique des utilisateurs,
- support de différents langages de programmation,
- traitement du code source (édition,..).
- fonctions d'import/export depuis et vers d'autres IDE, tels qu'Eclipse.
- accès et gestion de bases de données.

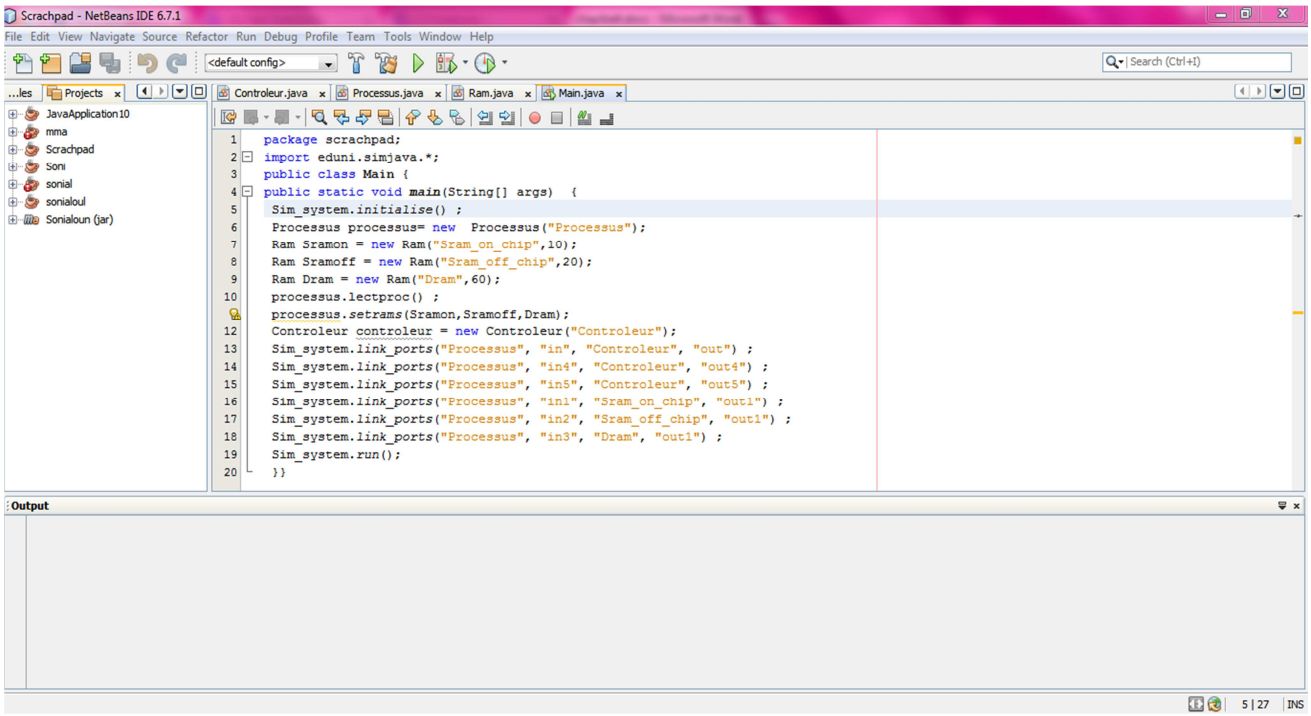


Figure IV.4 La plateforme NetBeans.

IV.4.2 Modèle de simulation

La simulation est actuellement le seul outil envisageable lorsqu'on désire appréhender le fonctionnement d'un système complexe que l'on ne parvient pas à formaliser pour utiliser des méthodes analytiques .les applications de la simulation sont très nombreuses : étude d'un système de production alors que celui-ci est en phase de conception, évaluation des performances d'un système existant. La simulation réalisée dans ce travail est une simulation à événements discrets. En d'autres termes, le système est modélisé comme une suite d'évènements.

❖ **Vue d'ensemble de SimJava**

a) Qu'est-ce que SimJava?

SimJava est un package implémenté en java dédié à la simulation des événements discret. SimJava contient un certain nombre d'entités dont le comportement d'une entité est programmé en java en utilisant la méthode body () **[36].** Chaque système est considéré comme un ensemble d'interaction des processus ou des entités comme ils sont appelés dans SimJava. Ces entités communiquent entre eux par des événements qui passent. Le temps de simulation progresse sur la base de ces événements.

Le but de la conception était que SimJava soit un ensemble de «cours de fondations de simulation» utiles pour la construction de modèles de simulation d'événements discrets **[39].**

b) Construire une simulation sous SimJava

1) Définition des entités

Avant d'écrire un code, le modélisateur doit séparer le système à l'étude en entités qui présentent les éléments constitutifs d'une simulation. En SimJava, les entités sont représentées par la classe **Sim_entity**. Cette classe encapsule toutes les fonctionnalités qui devraient être disponibles à des entités dans la simulation. Afin de définir une entité, ou plutôt un type d'entité, le modélisateur doit créer une sous – classe dérivée de **Sim_entity**. La sous - classe sera ensuite mis en œuvre pour contenir le comportement souhaité de l'entité. Ce comportement est fourni au moyen de la méthode **body ()** qui doit être remplacée dans la sous-classe.

Le constructeur de la sous-classe doit également être fourni dans lesquelles toutes initialisations aura lieu. C'est l'endroit où la superclasse Sim_entity est appelée et aussi où les ports **[39].**

2) Définition des ports

Les ports sont les moyens par lesquels les entités communiquent entre eux par des événements de planification. Ils sont toujours considérés par paires car un port dans une entité aura toujours un port correspondant dans une autre. Chaque port ne peut avoir qu'un autre port comme destination, mais un port à la fin du destinataire peut être lié à n'importe quel nombre de ports source.

Les ports sont représentés par des instances de la classe **Sim_port** et sont ajoutés à leur entités en utilisant la méthode **add_port()** et seront liés à l'aide de **Sim_system**, le noyau de la simulation **[39].**

3) Spécification du comportement d'une entité

Les entités disposent d'une sélection de méthodes d'exécution pour implémenter leur comportement. Les principales familles de méthodes d'exécution sont les suivantes **[39]:**

- **Méthodes sim_schedule() :** Ces méthodes sont utilisées pour planifier des événements sur d'autres entités dans la simulation.

- **Méthodes sim_wait ()** : Ces méthodes sont utilisées pour attendre un événement entrant. Alternativement, les méthodes **sim_select ()** ou **sim_get_next ()** peuvent être utilisées pour vérifier également les événements arrivés alors que l'entité était occupée.
- **Méthodes sim_pause()** : Ces méthodes sont utilisées lorsque l'entité est inactive. Des exemples d'un tel comportement sont les délais et les intervalles de délai entre la planification des événements.

- **Méthodes sim_process()**:Ces méthodes sont utilisées lorsque l'entité doit être considérée comme active. Une méthode sim_process doit être utilisée lorsque l'entité est considérée comme occupée.

- **Méthode sim_completed()** :Est utilisée pour signaler lorsqu'un événement est considéré comme ayant terminé tous les services d'une entité.

Ces fonctionnalités aident à construire un réseau d'entités actives qui communique par l'envoi et la réception d'objets événement d'une manière efficace.

IV.4.3 Utilisation de SimJava pour la simulation de notre système

Le système présente une sous-unité de contrôleur-Ram simple d'un système informatique. Le système actuel se compose d'un contrôleur dont la tâche est de traiter les demandes provenant de processus (demande réservation ou libérations) afin d'accéder à la zone mémoire réservée ou libérer l'espace mémoire (Sram_on_chip, Sram_off_chip et Dram). La Ram sélectionnée est défini par son adresse de base retournée par le contrôleur.

Les motivations possibles pour simuler un tel système est d'examiner les performances de la mémoire Scrachpad en exploitant les APIs.

Nous allons simuler le système avec l'outil SimJava. Le contrôleur et Ram sont considérés comme des entités évidentes du système. Nous avons besoin d'une autre entité, le processus qui conduit à l'activation du contrôleur.

Trois sous-classe impérativement hériter de la classe prédéfinie Sim_entity qui offre les fonctionnalités nécessaires.

-la classe Processus modélise l'entité processus.

-la classe Contrôleur modélise l'entité contrôleur.

-la classe Ram modélise l'entité Ram.

Le but à atteindre dans cette étape est de construire un modèle valide qui soit le plus simple possible, tout en restant cohérent avec les objectifs de l'étude. Il faut donc tout d'abord formuler explicitement ces objectifs, et les divers scénarios à étudier, en effet nous allons décrire deux méthodes d'allocation afin de pouvoir comparer les résultats de simulation obtenus et confirmé la performance de notre approche.

IV.4.4 La description des deux méthodes d'allocation vspm_malloc() et malloc()

1. La méthode vspm_malloc()

Dans notre approche le dialogue avec le circuit de gestion SPM pour demander une allocation mémoire est réalisé à travers la méthode vspm_malloc(taille, *pointeur)

Le seul paramètre à passer à vspm_malloc est le nombre d'octets à allouer. La valeur retournée est l'adresse du premier octet de la zone mémoire alloué.

Tout d'abord, afin d'effectuer une réservation, une tâche/processus essaye de demander un espace SPM via la méthode vspm_malloc, qui permet à la tâche de spécifier la taille de l'espace à réservé afin que le contrôleur puisse choisir dynamiquement un espace libre dans la RAM on-chip partagé par tous les processeurs en premier lieu et si ne trouve pas de l'espace libre dans cette dernière, il examine l'emplacement suivant qui est la off-chip SRAM ou dans la DRAM s'il n y a plus de place dans la off-chip SRAM.

Donc avec cette méthode l'espace SPM est mappé effectivement dans les SRAMs les plus rapides en premier lieu plutôt que de le mapper dans la DRAM.

2. La méthode malloc()

Cette méthode est similaire à la méthode vspm_malloc la seul différence est l'emplacement mémoire, en effet le contrôleur avec la méthode malloc () ne dispose pas d'une logique de réservation comme dans le cas avec vspm_malloc().Lorsque une tâche demande une allocation le contrôleur n'est pas censé de commencer la recherche de l'espace disponible dans l'espace mémoire plus rapide il commence directement la réservation de puis les adresses mémoire les plus basses ce qui implique la réservation dans la DRAM et si y a plus d'espace disponible dans cette dernière, va examiner l'emplacement suivant qui est la off-chip SRAM et si ne trouve pas de l'espace, enfin va examiner la mémoire SRAM on-chip si aucun espace n'est libre dans la off-chip SRAM et DRAM.

Donc avec cette méthode l'espace réservé est mappé effectivement dans la DRAM plutôt que de le mapper dans les SRAMs les plus rapides.

IV.4.5 Expérimentation des résultats

IV.4.5.1 Objectifs expérimentaux

Notre objectif est de montrer les avantages de notre approche de la gestion de SPM. Tout d'abord, nous montrons le temps d'exécution des tâches après avoir effectué une demande d'allocation avec la méthode malloc (). Deuxièmement, nous nous montrons le temps d'exécution des tâches après avoir effectué une demande d'allocation avec la méthode vspm_malloc ( ).

enfin nous montrons les avantages d'utilisation de la méthode vspm_malloc( ) par rapport à la méthode malloc() afin de valider davantage les avantage du dispositif de gestion de la SPM.

IV.4.5.2 Résultats expérimentaux

- **Expérience 1**

Cas1 : la simulation du système en utilisant seulement la méthode malloc

Nous avons procédé à l'essai de notre système avec dix (10) tâches qui utilisent la méthode traditionnelle **malloc**

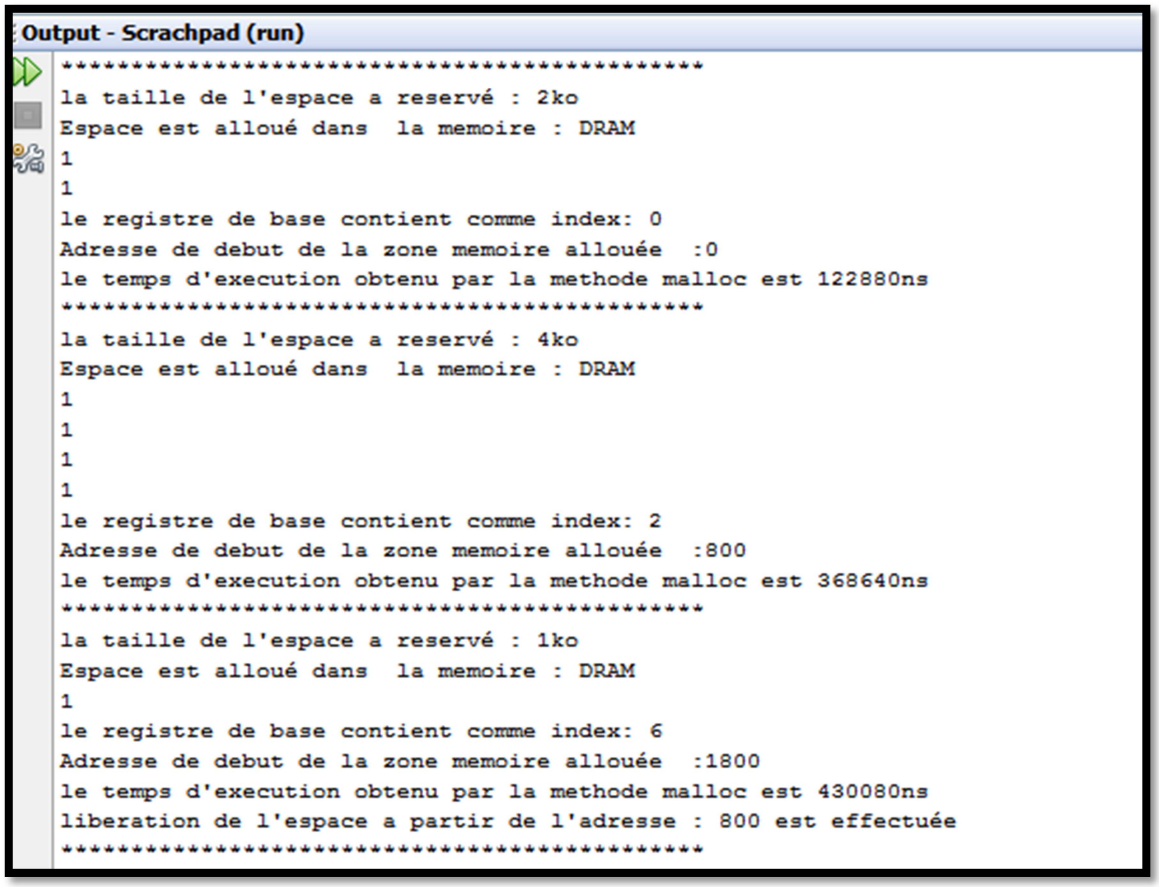


Figure IV.5 Résultats obtenus en utilisant la méthode traditionnelle malloc.

<u>Cas2</u> : la simulation du système en utilisant seulement la méthode vspm_malloc

Nous avons procédé à l'essai de notre système avec dix (10) tâches qui utilisent la méthode **vspm_malloc**

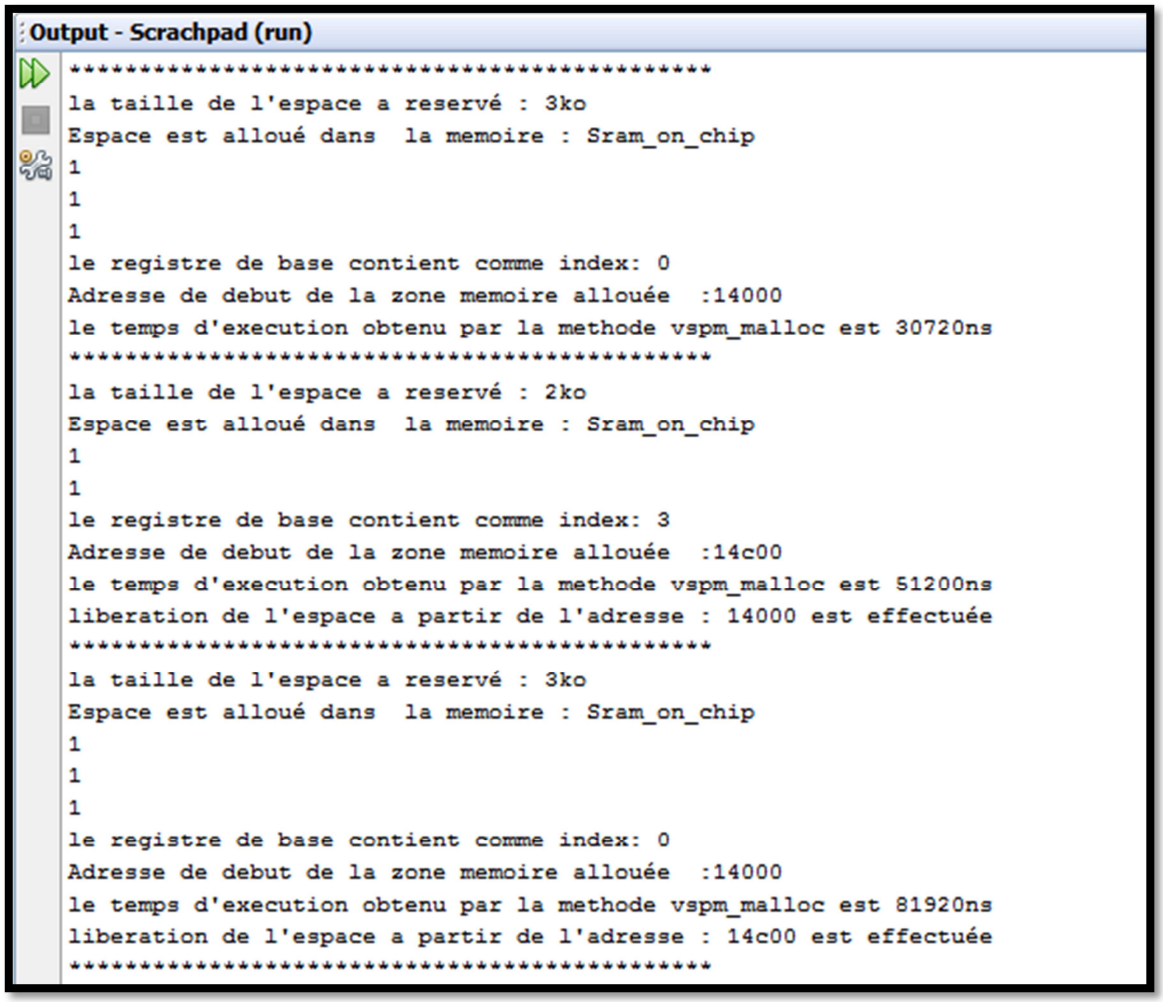


Figure IV.6 Résultats obtenus en utilisant la méthode vspm_malloc.

a) Analyse des résultats :

Le tableau suivant résume les résultats de simulation obtenus où nous avons utilisé la méthode **malloc** dans le cas1 et la méthode **vspm_ malloc** le cas 2.

nombres de tâches	malloc temps d'execution (ns)	vspm_malloc temps d'execution (ns)
1	245760	40960
2	430080	71680
3	675840	112640
4	737280	143360
5	921600	174080
6	1228800	194560
7	1351680	225280
8	1413120	266240
9	1536000	296960
10	1781760	307200

Tableau IV.2 Résulats obtenus dans les deux cas

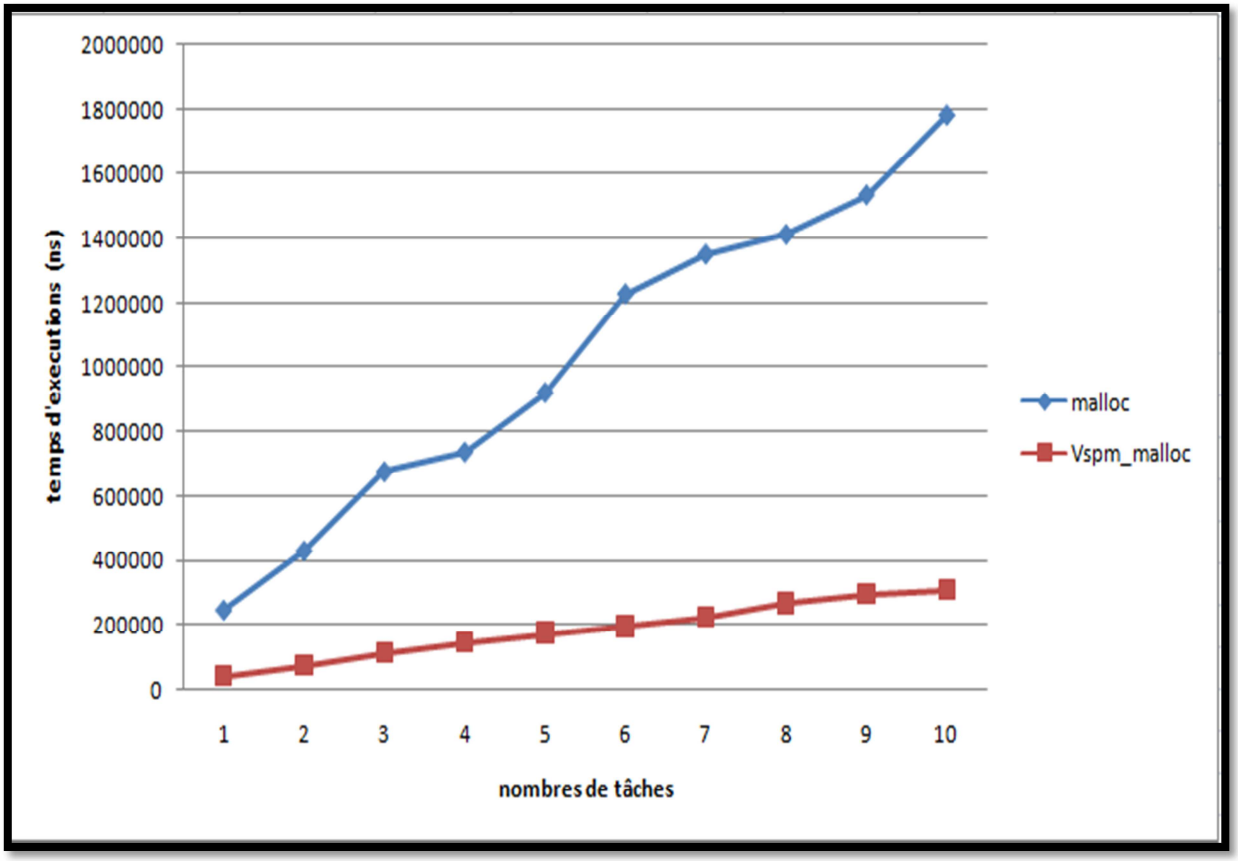


Figure IV.7 Evolution de temps d'exécution totale en fonction de nombre de tâches.

Nous remarquons dans la figure IV.7 que plus nous augmentons le nombre de tâches, plus le temps d'exécution total augmente, cette augmentations est expliquée par le nombre d'accès aux différentes mémoires.

En outre le temps d'exécution retenu dans le graphe présenté en bleu est plus grand que celui du graphe présenté en rouge, en effet le nombre d'accès à la sram_on_chip avec la méthode malloc est moins par rapport à la méthode vspm_malloc ce qui implique que notre approche génère les meilleurs résultats.

- **<u>Expérience 2</u> : La simulation du système en utilisant la méthode malloc et vspm_malloc**
  Nous avons procédé à l'essai de notre système avec vingt (**20**) tâches qui utilisent les deux méthodes **malloc** et **vspm_malloc** dont **9** tâches demandent la réservation d'espace en la méthode **malloc** et **11** tâches demandent la réservation d'espace en utilisant la méthode **vspm_malloc.**

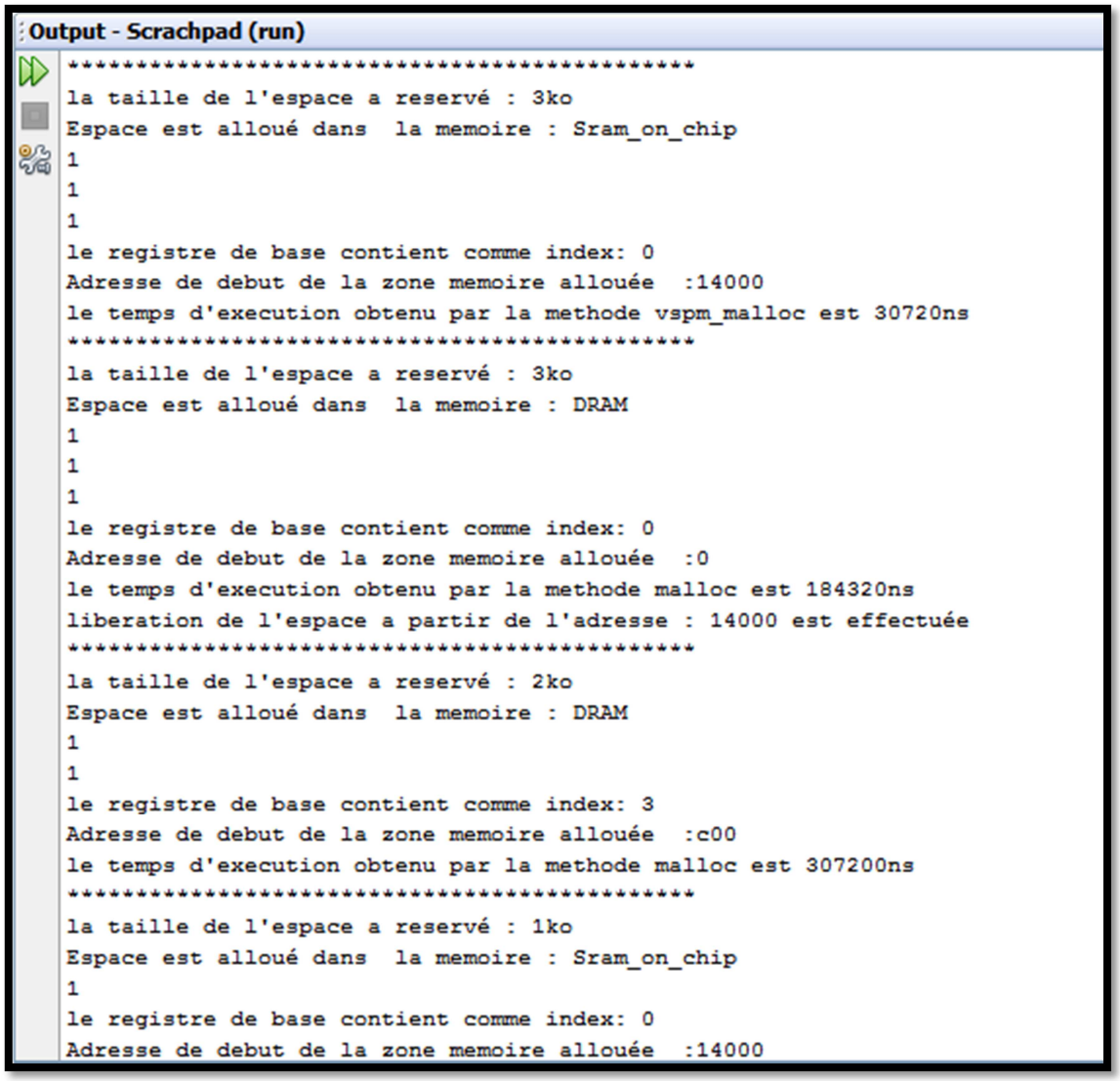


Figure IV.8 Résultats obtenus en utilisant la méthode malloc et vspm_malloc.

a) Analyse des résultats :

Le tableau suivant résume les résultats de simulation obtenus dans le cas où nous avons utilisé les deux méthodes **malloc** et **vspm_ malloc.**

nombres de tâches	temps d'execution (ns)	
	malloc	vspm_malloc
1	61440	40960
2	122880	61440
3	430080	92160
4	614400	133120
5	798720	174080
6	921600	225280
7	1105920	245760
8	1228800	266240
9	1290240	317440
10		368640
11		409600

Tableau IV.3 Resulats obtenus dans la simulation du système en utilisant la méthode malloc et vspm_malloc

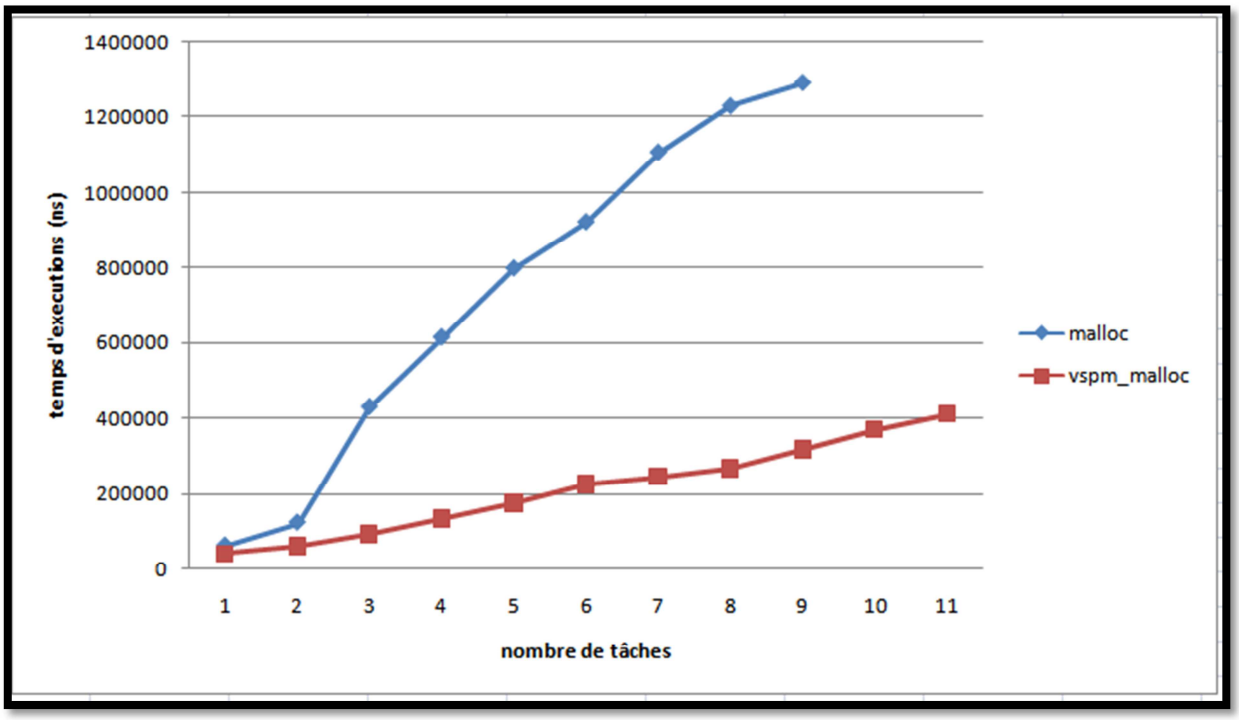


Figure IV.9 Evolution de temps d'exécution totale en fonction de nombre de tâches.

Nous remarquons dans la figure IV.9 que plus nous augmentons le nombre de tâches, plus le temps d'exécution total augmente, cette augmentations est expliquer par le nombre d'accès aux différentes mémoires.

En outre le temps d'exécution retenu dans le graphe présenté en bleu est plus grand que celui du graphe présenté en rouge, malgré que le nombre de tâches qui appellent à la méthode **vspm_malloc** est plus grand aux nombres de tâches qui appellent la méthode **malloc**.

En fin nous quantifions les avantages de notre approche qui permet de minimiser le temps d'exécution totale.

IV.5 Conclusion

Lors de la phase de la réalisation, nous avons implémenté notre circuit de gestion de SPM à l'aide de l'environnement de développement NetBeans, en utilisant le langage java et plus exactement la bibliothèque SimJava afin d'évaluer les performances de notre système. Suite à cela nous avons testé l'intégralité du système en comparant les deux méthodes d'allocations, En outre nous avons présenté les résultats d'évaluations sous forme des tableaux et des graphes.
les résultats obtenus montrent que la méthode d'allocation vspm_malloc implémenté par notre contrôleur a une influence importante sur le temps d'exécution total ,en effet il réduit le temps d'exécution .

Conclusion Générale

Conclusion générale

Dans ce mémoire une problématique a été traitée, à savoir : la réduction du temps d'exécution des processus dans les systèmes embarqués. La prévalence des systèmes embarqués ainsi que leur très forte dépendance en énergie est un problème récurrent et primordial. La mémoire est considérée comme étant un principal consommateur d'énergie dans ce type de systèmes et les efforts se retournent vers elle. L'optimisation globale des accès mémoires par le bon choix d'une méthode d'allocation conduit à la réduction de temps d'exécution ce qui influence ainsi sur la consommation d'énergie.

L'objectif du présent travail étant d'évaluer à grande échelle d'un système permettant la gestion efficace d'une mémoire Scratchpad dans une architecture embarquée multi-cores.
La méthode d'évaluation de performance qu'on a utilisée est la simulation. L'utilisation de la simulation a événements discrets est choisi car on s'est s'intéressés à un système discret où les changements se produisent à des instants particuliers.

Afin de mieux cerner et comprendre les caractéristiques des systèmes embarqué ou se situ notre domaine d'étude nous avons présenté dans le premier chapitre quelques généralités sur les systèmes embarqués, leur architecture, leurs types et leurs utilisations.
Les différentes mémoires qui présentent les composants les plus critiques des systèmes embarqués ont été introduites au deuxième chapitre. Dans cette partie, nous avons abordé les différent type de mémoires afin de mieux connaitre la différence entre eux plus particulièrement la différence entre la mémoire cache et Scratchpad.
Ainsi, dans le troisième chapitre nous avons proposé la nouvelle approche de gestion SPM qui est l'intégration d'un dispositif matériel doté d'une logique d'allocation lui permettant de gérer efficacement la mémoire SRAM-on-chip.
Le quatrième chapitre a donc porté sur la phase d'implémentation du système proposé dans le troisième chapitre, Les outils de développements utilisés ont été mis en avant à fin de valider les avantages de notre approche où nous avons effectué une comparaison entre deux méthodes d'allocation. Nous avons pu constater que l'allocation avec la méthode vspm_malloc pilotant le circuit de gestion de la SPM était capable de réduire le temps d'exécution au mieux par rapport à la politique d'allocation traditionnelle.

L'objectif fixé au début du travail est atteint, néanmoins des perspectives peuvent être envisagées afin d'améliorer la gestion de la mémoire scratchpad. Le principal point concerne à améliorer les performances de la méthode d'allocation utilisée. Actuellement, elle effectue une exploration un peu limitée, en confondant les applications prioritaires et moins prioritaires. Plus exactement, des applications temps réel sont traitées de la même façon que les applications normales. Dans ce cadre, il serait intéressant d'ajuster la méthode en introduisant la notion de priorité, tout en gardant le principe d'allocation dynamique déjà utilisé. Par exemple si la SPM est pleine, une tache critique ne doit pas être mappée vers une mémoire off-chip car les accès hors-puce pourraient le conduire à manquer ses délais. Pour cela il est préférable d'exclure les données d'une tâche qui a été déjà mappé dans la SPM avec une priorité moins que la tache critique et accorder cet espace à cette dernières.

Références bibliographiques

Références bibliographiques

[1]: M. Verma, L. Wehmeyer and P.Marwedel, Cache-Aware Scratchpad Allocation Algorithms for Energy-Constrained Embedded Systems, IEEE trans. Computer-Aided Design of Integrated Circuits and Systems, Vol. 25, no 10, PP2035-2051, Oct 2006.

[2]: R.Banakar, S.Steinker, B.Lee, M.Balakrishnan, et P.Marwedel. Scratchpad memory: design alternative for cache on-chip memory in embedded systems. In CODES, page 73-78, New York, NY, USA, 2002. ACM Press

[3] : Ramzi BOULKROUNE, Les Systèmes Embarqués, thèse d'Ingéniorat, université d'Annaba, 2009.

[4] : Corine Alonso et Bruno Estibals, Conception et commande de Systèmes Electriques Embarqué, université Paul Sabatier, Toulouse, 2002

[6] : TILIOUINE NOR EL HOUDA, Observation d'exécution de tâches dans MPSOC a base de NOC, Mémoire de Fin d'Etudes Pour l'Obtention du Diplôme de Master en Informatique, Juin 2015

[8]:J. STANKOVIC. Strategic directions in real-time and embedded systems. In « ACM Computing Surveys», volume 28(4):751-763, 1996.

[10]:L. R. CLAUSEN, L. P. SCHULTZ, C. CONSEL, G. MULLER. *Java bytecode compression for low-end embedded systems.* In « ACM Transactions on Programming Languages and Systems », volume 22(3):471-489, 2000.

[12]: Aimen BOUCHHIMA, Modélisation du logiciel embarque à différents niveaux d'abstraction en vue de la validation et la synthèse des systèmes monopuces, thèse de doctorat, Institut National Polytechnique de Grenoble, Mai 2006.

[13]: A.Jerraya et W.Wolf, "Multiprocessor Systems-on-Chip", Elsevier Morgan Kaufmann, San Francisco, Californie, 2005

[13]: Youssef ATAT: « Conception de haut niveau des MPSoCs à partir d'une spécification Simulink : Passerelle entre la conception au niveau Système et la génération d'architecture », Mai 2007

[22] : Mohamed Salah SOUAHI, Optimisation des Ressources dans les Systèmes Embarqués, memoires_magister, Université de Tebessa, Novembre 2010.

[23]: Maha Idrissi Aouad, Conception d'Algorithmes Hybrides pour l'Optimisation de l'Energie Mémoire dans les Systèmes Embarqués et de Fonctions Multimodales, Thèse doctorat, université Henri Poincaré – Nancy 1, 4 Juillet 2011

[25]: Alexis ARNAUD, Utilisation prévisible de caches d'instructions dans les systèmes temps-réel strict, Thèse doctorat, Université de Rennes 1, 13 avril 2006.

[27]:M. Lalam, Architectures parallèles / les mémoire caches, Support du cours, 2015

[29] : L. D. Bathen, D. Shin, S. S. Lim, Virtualising on-chip distributed scratchpad memories for low power and trusted application execution, Design Automation for Embedded Systems, Springer link, 2012.

[30]: Panda PR, Dutt ND, Nicolau A (1997) Efficient utilization of scratch-pad memory in embedded processor applications. In: Proc of the European conf on design and test, EDTC ’97

[31]: Verma M, Steinke S, Marwedel P (2003) Data partitioning for maximal scratchpad usage. In: Proc of the 2003 Asia and South Pacific design automation conf, ASP-DAC ’03

[32]: Francesco P, Marchal P, Atienza D, Benini L, Catthoor F, Mendias JM (2004) An integrated hardware/-software approach for run-time scratchpad management. In: Proc of the 41st annual design automation conf, DAC ’04

[33]: Verma M, Petzold K, Wehmeyer L, Falk H, Marwedel P (2005) Scratchpad sharing strategies for multiprocess embedded systems: a first approach. In: 3rd workshop on embedded systems for real-time multimedia, 2005, pp 115–120.

[34]: Mahmut Kandemir , J. Ramanujam, A. Choudhary, Exploiting Shared Scratch Pad Memory Space in Embedded Multiprocessor Systems.June 10-14, 2002.

[36]: F. Howell and R. McNab. Simjava : A discrete event simulation package for java with applications in computer systems modelling. In First International Conference on Web based Modelling and Simulation. Society for Computer Simulation, 1998.

webliographie

[5]:http://www-igm.univ-mlv.fr/~dr/XPOSE2002/SE/architecture.html

[7]:http://reds.heig-vd.ch/share/cours/IFS/IFS_SysEmb_1_intro.pdf

[9]:http://raweb.inria.fr/rapportsactivite/RA2002/aces/aces.pdf

[11]: https://tel.archives-ouvertes.fr

[15]: https://fr.wikipedia.org/wiki/Mémoire_(informatique)

[16]: http://www.teluq.ca/expl_inf1130/pdf/chap2.pdf

[17]: http://amrouche.esi.dz/doc/ch7_memoires.pdf

[18]: http://www.info.univ-angers.fr/~richer/ensl3i_crs4.php

[19]:https://fr.wikipedia.org/wiki/Mémoire_flash

[20]: http://www.memoireonline.com/01/12/5117/m_volution-sur-la-memoire-vive4.html

[21] http://schema-montage-electronique.blogspot.com/2012/02/architecture-des ordinateurs.html

[24] : http://michel-vause.infographie-heaj.eu/E3/html/page2.html

[26]:https://fr.wikibooks.org/wiki/Fonctionnement_d'un_ordinateur/Les_mémoires_cache

[28] : https://tel.archives-ouvertes.fr/tel-01402354/document

[35]: https://fr.wikipedia.org/wiki/UML_(informatique).

[37]: http://jmdoudoux.developpez.com/cours/developpons/java/chap-presentation.php

[38]: https://fr.wikipedia.org/wiki/NetBeans

[39] http://www.dcs.ed.ac.uk/home/simjava/tutorial/#1.1