Séquenceur Mémoire pour Applications Multimédias Temps Réel Gérant les Séquences d'Accès Indéterministes

Bertrand Le Gal, Emmanuel Casseau, Sylvain Huet, Caaliph Andriamisaina, Eric Martin

Laboratoire LESTER CNRS-FRE 2734
Université de Bretagne Sud, 56000, Lorient, FRANCE
Bertrand.Legal@univ-ubs.fr, Emmanuel.Casseau@univ-ubs.fr, Sylvain.Huet@univ-ubs.fr,
Caaliph.Andriamisaina@univ-ubs.fr, Eric.Martin@univ-ubs.fr,

Résumé – Dans le domaine du traitement du signal et de l'image, les applications multimédias sont souvent caractérisées par un grand nombre d'accès aux données. Pour la plupart de ces applications, les accès aux données structurées (tableaux, vecteurs) sont réguliers et périodiques. Dans ces conditions, il est possible et efficace de générer des contrôleurs pipeline d'accès à la mémoire. Cette technique est utilisée afin d'améliorer les accès en mode pipeline autorisés par les mémoires actuelles. On utilise pour cela des composants matériels dédiés pour générer les adresses et pour packer/dépacker les données. Dans cet article nous présentons l'architecture d'un séquenceur mémoire qui permet de prendre en compte de manière efficace les accès prédictibles aussi bien que les séquences d'accès non prédictibles (calculs d'adresses dynamiques) de manière pipeline.

Abstract – Multimedia applications such as video and image processing are often characterized by a large number of data accesses. In many digital signal-processing applications, the array access patterns are regular and periodic. In these cases, it becomes feasible and efficient to generate optimized Pipelined Memory Access Controllers. This technique is used to improve the pipeline access mode to actual RAM by creating specialized hardware components for generating addresses and packing and unpacking data items. In this paper we focus on the design of a memory sequencer which can efficiently handle predictable address patterns as well as unpredictable (dynamic address computations) in a pipeline way.

1 Introduction

Les applications multimédias de type traitement du signal et de l'image (TDSI) sont caractérisées par un nombre important d'accès aux données. Dans de telles applications, les accès à la mémoire sont des facteurs limitant de la vitesse d'exécution des processeurs qui leurs sont dédiés. L'architecture mémoire (la hiérarchie, le nombre de bancs, le placement des données, etc ...) impacte sur les performances du système mais aussi sur la consommation d'énergie qui est critique pour les applications embarquées. Par ailleurs temps, le temps attribué à la conception personnalisée de circuits diminue face à la pression du "time to market".

D'un coté, les recherches actuelles dans le domaine des applications multimédias visent à réduire la complexité calculatoire des algorithmes en utilisant des solutions adhoc reposant sur des calculs conditionnels (par exemple, la transformation de la recherche exhaustive en recherche en 3 étapes dans le domaine de l'estimation de mouvement [5]). Ces transformations introduisent des aléas d'exécution dus aux calculs conditionnels (sélection du meilleur vecteur dans la recherche en 3 étapes).

D'un autre coté, les recherches effectuées sur les architectures d'implémentation de ces algorithmes sous contrainte temps réel, tentent d'exploiter le parallélisme entre les opérations. Dans ce cas, les architectures optimales sont obtenues pour des algorithmes réguliers, sans aléas d'exécution. Cela est aussi vrai dans le cadre des architectures mémoires qui sont optimales lorsque leurs séquences d'accès sont prédictibles.

Malheureusement dans une certaine partie des applications multimédias actuelles, les séquences d'accès à la mémoire ne sont pas entièrement connues a priori. Les aléas présents dans ces dernières empêchent les concepteurs et les outils de synthèse de haut niveau de pouvoir prendre en compte efficacement les séquences d'accès répétitives de l'application.

Dans cet article, nous apportons les contributions suivantes : nous présentons une nouvelle architecture de séquenceur mémoire qui permet de réaliser des accès pipelinés à la mémoire dans le cadre de séquences d'accès à la mémoire statiques mais aussi dynamiques. Cette technique sera ensuite étendue en transférant des calculs d'adresses du chemin de données vers des unités spécialisées au sein du séquenceur mémoire afin d'augmenter les performances et diminuer la consommation (le nombre de transferts entre ces deux entités est alors réduit). Les résultats présentés dans ce papier montrent qu'il est possible d'exploiter les informations spécifiques à l'application afin de générer un séquenceur d'accès à la mémoire capable de réduire les surcoûts associés aux accès aléatoires à la mémoire.

2 Travaux Connexes

La majorité des recherches ciblant des applications orientées contrôle intensif sont basées sur des accès directs à des RAM (mémoire à accès aléatoires). Ces applications uti-

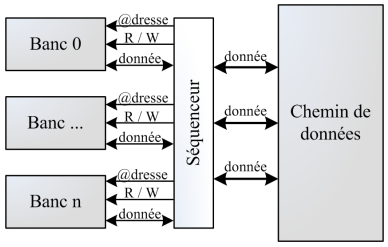


FIG. 1 – Architecture à Base de Séquenceur Mémoire

lisent une architecture mémoire générique qui prend une adresse binaire en entrée et la décode en ligne et colonne afin de réaliser l'accès souhaité. Cette solution est adaptée aux applications pour lesquelles les accès mémoire sont non prédictibles a priori.

Une bonne partie des applications TDSI les séquences d'accès aux éléments d'un vecteur ou d'un tableau sont régulières et périodiques. Dans ces dernières, il est possible de mémoriser/programmer directement au sein du séquenceur les séquences d'adresses des données manipulées par l'application (utilisant des compteurs dédiés [2], des registres à décalage, une machine à états finis ...). Le séquenceur peut ordonnancer de manière optimale l'ordre des accès afin d'exploiter les spécificités de la mémoire réalisant si besoin des accès décalés (lecture préemptives, écriture tardives).

Les circuits basés sur une architecture à séquenceur mémoire permettent au concepteur (ou à l'outil de synthèse) de synthétiser et optimiser de manière decorélée le chemin de données et l'architecture de la mémoire. Comme cela est montré par la figure 1, seules les données produites et consommées par le chemin de données sont transférées d'une unité à l'autre (le transfert des adresses n'étant pas nécessaire dans le cadre de séquences d'accès connues à priori). Ces approches permettent de réduire la consommation de l'architecture finale en réduisant le nombre de transferts inter-unités. Des recherches ont démontré que les séquenceurs d'accès à la mémoire générés à partir des séquences d'accès connues a priori peuvent être optimisés afin d'obtenir des architectures mémoire optimales [3] [1]. Des optimisations visant à augmenter les performances du circuit peuvent être aussi employées. Dans [6], l'auteur présente l'impact sur la surface et les performances (latence) de la simplification de l'architecture du séquenceur et de la mémoire.

L'utilisation d'un séquenceur permet aussi au concepteur de séparer les problèmes d'interfaçage de la mémoire de celui de l'ordonnancement des calculs du chemin de données [4]. Cela permet une meilleure exploitation a priori des techniques d'accès pipeline à la mémoire en tamponnant via des FIFOs les données en cours de transfert.

Dans les approches traditionnelles, le séquenceur d'accès à la mémoire *ne peut être crée que dans le cas d'accès à la mémoire purement déterministes (nommés aussi statiques)*. C'est pourquoi nous proposons une nouvelle architecture de séquenceur permettant la décorrélation de la partie traitement de données et de la partie mémorisation, prenant en compte les accès non déterministes nommés aussi dynamiques(adresses calculées à l'exécution et accès réalisés au sein des branches conditionnelles).

3 Architectures d'Implémentation

Notre objectif est de générer un séquenceur d'accès à la mémoire optimal supportant les adressages dynamiques pour les séquences d'accès *principalement déterministes*. Nous présentons dans un premier temps une architecture autorisant les adressages dynamiques, puis dans un deuxième temps une architecture étendue permettant de réaliser des calculs d'adresses dans le séquenceur mémoire afin de réduire encore le nombre de transferts d'adresses entre les différentes unités. Cela permet de réduire de manière significative le nombre de bus et induit une réduction du coût en surface tout en réduisant le nombre de commutations (réduction de la consommation).

3.1 Séquenceur avec Adressage Dynamique

Dans cette première architecture, nous supposons que tous les calculs d'adresses dynamiques sont réalisés au sein de l'unité de traitement et que les calculs résultats de ces calculs sont ensuite sont ensuite transférés au sein du séquenceur par l'intermédiaire des bus de données.

L'architecture du séquenceur présentée dans la figure 2 est composée de 4 unités différentes : un ordonnanceur d'accès à la mémoire, un contrôleur pour les accès dynamiques, un générateur d'adresses et une table de translation d'adresses.

Les bus reliant le chemin de données sont reliés à la mémoire par un *crossbar* qui les multiplexe. Ce *crossbar* est piloté par l'intermédiaire du séquenceur d'accès à la mémoire, qui connaît a priori les séquences d'accès à la mémoire.

L'ordonnanceur contrôle la progression du générateur d'adresses de manière synchrone par rapport à la progression des calculs dans le chemin de données. Dans le cas d'accès dynamiques à la mémoire, l'adresse utilisera la table de translation d'adresses afin de convertir l'adresse logique en un couple (banc mémoire, adresse physique). Cela a pour effet de permettre au concepteur d'assigner différents éléments d'un même vecteur au sein de différents bancs mémoire et cela de manière non contiguë afin d'exploiter au mieux le parallélisme de l'application.

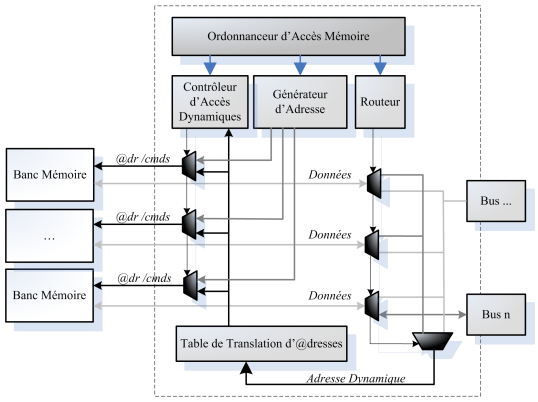


FIG. 2 – Séquenceur pour des Accès Dynamiques

Dans le cas d'un accès dynamique à la mémoire, en fonction du couple (banc mémoire, adresse physique), le contrôleur d'accès dynamique va générer les signaux de commande permettant d'accéder au banc mémoire ciblé. Durant cet accès dynamique, l'ensemble des bancs mémoire potentiellement accessibles sera bloqué.

Cette architecture de séquenceur permet de traiter des accès dynamiques à la mémoire dans une approche de type "séquence d'accès connue à priori" en s'assurant du respect des contraintes de latence/cadence.

3.2 Transfert des Calculs d'Adresse Dynamiques au Sein du Séquenceur

Nous avons précédemment présenté une architecture permettant de réaliser des accès dynamiques au sein de la mémoire à l'aide d'un séquenceur. Nous étendons maintenant notre approche en transférant les calculs d'adresses dynamiques de l'unité de traitement vers le séquenceur mémoire. Afin de permettre cette évolution, nous ajoutons au sein de l'architecture du séquenceur des unités de calculs dédiées. Le chemin de données interne au séquenceur est spécialisé dans les calculs d'adresses dynamiques. Ces unités sont donc dimensionnées de manière optimale par rapport à la largeur (nombre de bits) des adresses. Un autre intérêt de cette méthode est la localisation en mémoire des variables et des constantes exclusivement réservées aux calculs d'adresses. Leur utilisation dans les calculs d'adresses dynamique ne nécessite donc plus de transfert vers le chemin de données.

La nouvelle architecture du séquenceur est présentée en figure 3 ; un chemin de données dédié, composée d'opérateurs et de registres a été ajouté à coté de la table de translation d'adresses.

Cette évolution permet d'obtenir des gains intéressant en terme de performance dans le cas de circuits pipelines où le transfert des données d'une unité à l'autre peut prendre plusieurs cycles d'horloge (mémoires "lentes", bus pipelinés, etc.).

Localiser les calculs d'adresses dynamiques au sein du séquenceur fournit alors un gain en terme de latence. La réduction des transferts entre l'unité de calculs et le séquenceur entraîne également une réduction de la commutation des fils composant les bus (i.e. réduction de la consommation globale du circuit). De plus la réduction du trafic due à la déportation des calculs d'adresses entraîne une réduction des besoins filaire entre les unités (bus) ainsi que des mémorisations nécessaires pour ces transferts.

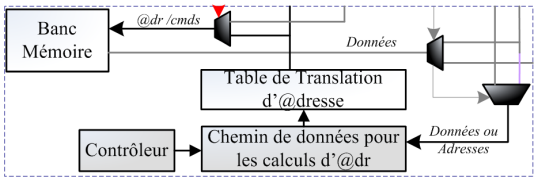


FIG. 3 – Évolution de l'Architecture du Séquenceur

4 Flot de Conception

Afin d'implémenter une application possédant des calculs d'adresses dynamiques sur l'architecture définie précédemment, nous avons élaboré un flot de conception basé sur un modèle de graphe qui permet de prendre en compte les contraintes temporelles dues aux transferts de données entre les unités ainsi que le temps d'accès aux RAM.

Le point d'entré de notre flot de conception (figure 4) est une description algorithmique de l'application accompagnée du mapping mémoire si le concepteur l'a déjà réalisé. Notre méthode peut être utilisée avec ou sans mapping à priori. La première étape consiste à annoter le graphe modélisant l'application avec les contraintes temporelles dues aux transferts de données et aux adressages dynamiques. Une partie des informations utilisées dans cette étape est fournie par le mapping mémoire. Le graphe de type flot de signaux (SDF) est ensuite annoté de manière à prendre en compte le transfert des données et des adresses entre les diverses unités composant le circuit.

Au départ, les calculs d'adresse dynamiques sont supposés être localisés au sein de l'unité de traitement(figure 5a). Ensuite en fonction du mapping mémoire et des informations sur le type des données, des noeuds modélisant les transferts de données et les adressages dynamiques sont insérés dans le graphe. Une fois cette étape réalisée, on obtient le graphe présenté en figure 5b. Ensuite à l'aide d'une métrique de décision, nous déportons les calculs dynamiques d'adresses de l'unité de traitement vers le séquenceur mémoire (figure 5c).

La métrique de décision prend en considération plusieurs critères : le nombre de transferts nécessaires entre les unités, l'allongement ou la réduction des chemins critiques, la réduction de la largeur des opérateurs dans le séquenceur mémoire, etc. La méthode de transfert des calculs d'adresses dynamiques de l'unité de traitement vers le séquenceur est appliquée statiquement sur le graphe. Cela génère un graphe optimal modélisant les calculs d'adresses et leur lieu d'implémentation privilégié.

Le concepteur va ainsi pouvoir synthétiser manuellement (handed coding) ou automatiquement (high-level syn-

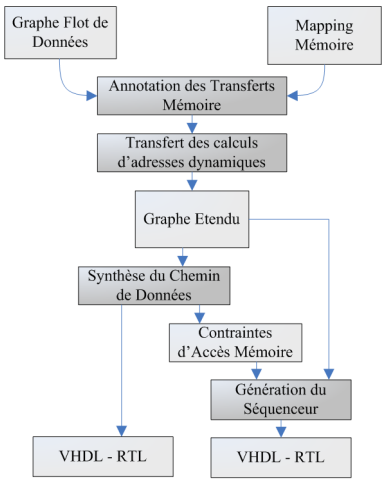
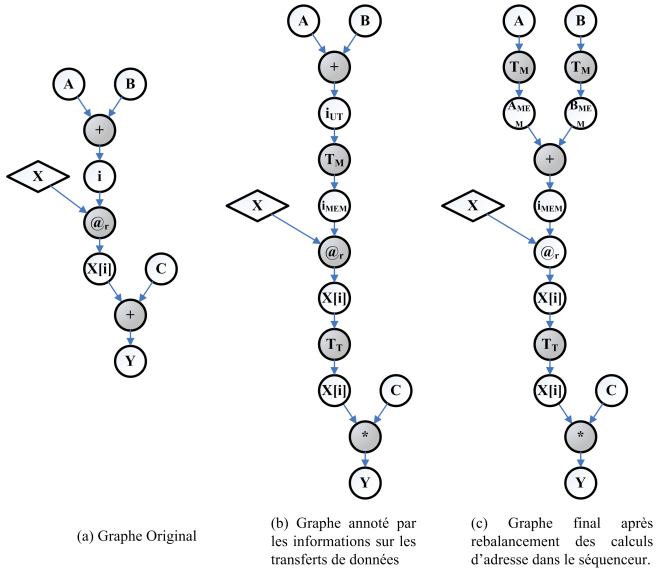


FIG. 4 – Flot de Conception



(a) Graphe Original

(b) Graphe annoté par les informations sur les transferts de données

(c) Graphe final après rebalancement des calculs d'adresse dans le séquenceur.

FIG. 5 – Exemple de Graphe Modélisant l'Application et les Transferts (algo : $y = X[i] + c$)

thesis tool) le chemin de données sans se préoccuper de l'implémentation du séquenceur mémoire car les contraintes temporelles sont directement intégrées dans le graphe modélisant l'application.

A l'issue de la synthèse des chemins de données, le concepteur doit fournir les contraintes liées au séquenceur mémoire (date des accès, type des accès, calculs à réaliser, etc.) afin de permettre la génération du séquenceur adéquate.

5 Expériences

Nous avons appliqué notre approche de conception sur une estimation de mouvement de type *Three Step Search* [5], qui possède une séquence d'accès à la mémoire indéterministe. Dans le cadre de l'architecture de référence (contrôle), nous considérons l'ensemble des transferts d'adresses (bloc de référence et macro-blocs dynamiquement sélectionnés). Dans la première solution d'architecture présentée dans cet article, le transfert des 5 premiers macro-blocs est connus de manière statique *à priori* de la même manière que le bloc de référence. Pour la seconde architecture proposée (avec unités de calculs dédiées dans le séquenceur), les seuls transferts qui sont réalisés sont les adresses de base des macro-blocs dynamiquement sélectionnés, les autres calculs étant réalisés au sein du séquenceur.

Les résultats présentés dans la figure 6 montre que les approches à base de séquenceur permettent de réduire le nombre de transferts entre le chemin de données et la mémoire. Il faut tout de même noter que le nombre d'accès à la mémoire est identique mais ces derniers peuvent être optimisés dans le cadre d'une architecture à base de séquenceur. Ces résultats montrent aussi l'intérêt de la méthode dans le cadre d'architecture pipelines où les communications entre les unités prennent plusieurs cycles d'horloge. Cela permet comme le souligne Park [4] d'utiliser au mieux les techniques d'accès à la mémoire afin de réaliser le cas échéant des lectures spéculatives.

	Macro-bloc	*Fenêtre de recherche*	*Transferts d'adresses*
Architecture Contrôle	*8x8*	*16x16*	*1664*
Architecture avec le 1er Séquenceur			*1024*
Architecture avec le 2nd Séquenceur			*16*
Architecture Contrôle	*16x16*	*48x48*	*6656*
Architecture avec le 1er Séquenceur			*4096*
Architecture avec le 2nd Séquenceur			*16*

FIG. 6 – Nombre de transferts nécessaires

6 Conclusion

Dans ce papier nous avons présenté une nouvelle architecture de séquenceur mémoire pour les applications TDSI possédant des séquences d'accès à la mémoire *en partie indéterministes*. Nous montrons de plus que le flot de synthèse proposé permet au concepteur d'optimiser librement chaque unité de son architecture (unité de traitement, unité de mémorisation). Notre travail futur consistera en l'intégration du processus de création du séquenceur dans un Flot de Synthèse de Haut Niveau afin d'automatiser toutes les étapes de création de l'architecture.

Références

[1] T. Kim C. Lyuh and K. Kim. Coupling-aware high-level interconnect synthesis. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 23(1) :157–164, January 2004.

[2] P.B Denyer D. Grant and I. Finlay. Synthesis of address generators. In *Proceedings of ICCAD 89*, pages 116–119, 1989.

[3] A. Dasgupta and R. Karri. High-reliability, low-energy microarchitecture synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(12) :1273–1280, 1998.

[4] J. Park and P.C. Diniz. Synthesis of pipelined memory access controllers for streamed data applications on fpga-based computing engines. In *Proceedings of ISSS '01*, pages 221–226. ACM Press, 2001.

[5] B.Zeng R.Li and M.L.Liou. A new three-step search algorithm for block motion estimation994. *IEEE Trans. on Circuits and Systems for Video Technology*, 4(4) :438–442, August 1994.

[6] P. Cheung S. Hettiaratchi and T. Clarke. Performance-area trade-off of address generators for address decoder-decoupled memory. In *Proceedings of DATE 2002*, page 902. IEEE Computer Society, 2002.