Traitement temps réel des images en exploitant la reconfiguration dynamique : architecture et programmation

Real-time image processing using dynamic reconfiguration paradigm : architecture and programming

L. Kessal[*], N. Abel[*] et D. Demigny[**]

[*] ETIS, UMR CNRS 8051, 6 av. du Ponceau, 95014 Cergy Pontoise cedex, {kessal, abel}@ensea.fr
[**] R2D2-IRISA, 6 rue de Kérampont, BP 447, 22305 Lannion cedex, didier.demigny@univ-rennes1.fr

Manuscrit reçu le 8 avril 2005

Résumé et mots clés

Le projet ARDOISE[1] vise à démontrer l'intérêt des architectures matérielles à reconfiguration dynamique (RD) dans les systèmes associant contraintes temps réel fortes et versatilité des traitements. À l'heure actuelle, ARDOISE est la seule architecture développée en France, dont un prototype a effectivement été réalisé et testé en situation réelle, qui mette en œuvre ce concept. Cet article présente les principaux résultats obtenus : définition d'une architecture de test, définition de métriques génériques guidant le paramétrage de mise en œuvre de la RD, stratégies de partitionnement des algorithmes sur le matériel, outils de compilation et de gestion temps réel des configurations, test sur une gestion dynamique de traitements d'images temps réel.

Reconfiguration Dynamique, FPGA, traitement de l'image, outils de développement, séquencement de tâches.

Abstract and key words

The ARDOISE[1] project's aim is demonstrating the interest of the dynamic reconfiguration paradigm in systems associating strong real-time constraints and inconstancy of treatments. At this time, ARDOISE is the only dynamically reconfigurable architecture developed in France. A prototype of ARDOISE has been realized and tested in real situation which implements this concept. This article presents the final results of the ARDOISE project : the test architecture, generic metrics driving the development of applications, software suite for compilation and real-time management of the configurations. An example of an image processing application that uses our methodology is given.

Dynamic Reconfiguration, FPGA, image processing, development tools, tasks scheduling.

1. ARDOISE : Architecture Reconfigurable Dynamiquement Orientée Image et Signal Embarquable.

1. Introduction

Pendant les dix dernières années, plusieurs architectures combinant des processeurs et/ou des circuits reconfigurables (FPGA) ont été proposées pour accélérer l'exécution d'applications de plus en plus complexes. Les systèmes industriels dédiés au traitement du signal et de l'image utilisent actuellement soit des processeurs à usage général ou dédiés, soit des solutions câblées intégrant des circuits spécifiques de type ASIC ou la combinaison des deux. Cependant, la conjugaison de la complexité croissante des algorithmes et d'un grand volume de données à traiter, le tout avec des contraintes temps réel fortes, réclame des performances que les processeurs ne peuvent fournir. La mise en parallèle de processeurs peut apporter la rapidité et la souplesse de programmation, mais au détriment du coût et de la complexité de mise en œuvre. De plus, l'utilisation des ASICs qui offrent une plus grande vitesse de traitement souffre d'une certaine rigidité et d'un cycle de développement onéreux. En effet, ajouter une nouvelle fonctionnalité à un système câblé nécessitera probablement une nouvelle conception d'un ou plusieurs ASICs faisant accroître les coûts. Les architectures reconfigurables représentent une réponse intermédiaire appropriée, offrant de meilleures performances par rapport aux architectures programmables et plus de flexibilité par rapport aux solutions câblées. Elles utilisent des composants logiques reconfigurables qui permettent à l'utilisateur de modifier l'architecture après fabrication de façon logicielle, contrairement aux ASICs dont les algorithmes sont câblés dans le silicium. Le terme reconfiguration désigne l'opération qui vise à implanter dans le ou les composants une nouvelle fonctionnalité sans modification de l'architecture matérielle du système.

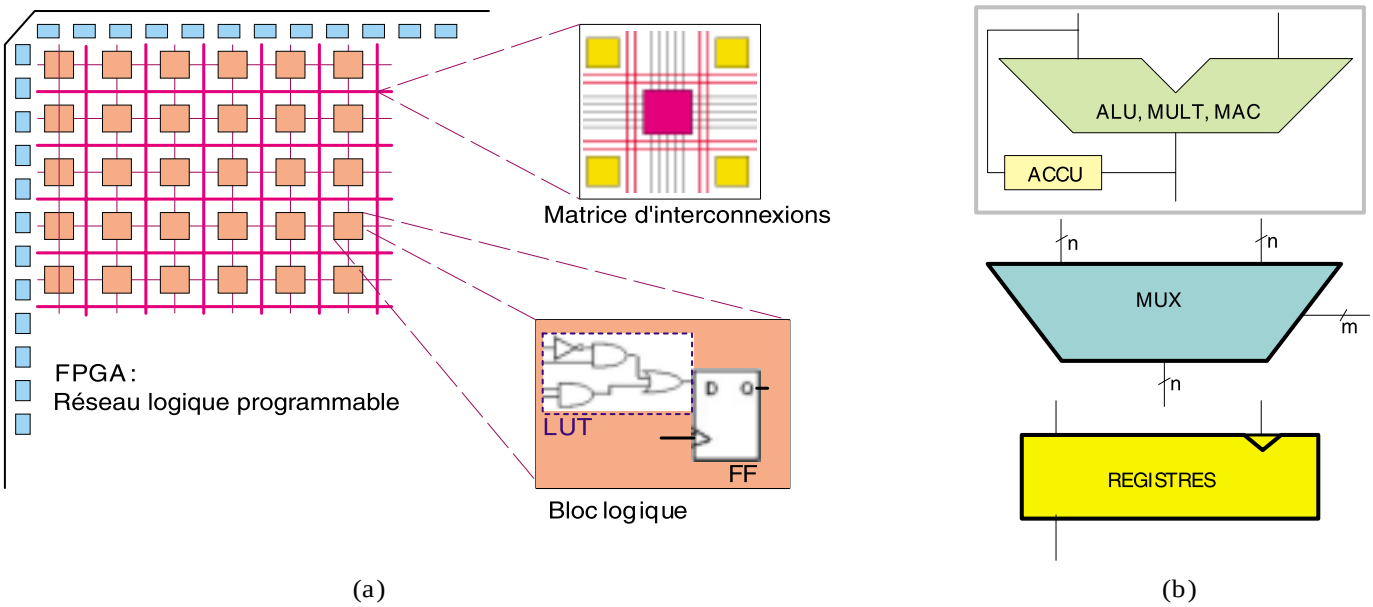
Granularité des systèmes reconfigurables

Ces dernières années, deux tendances fortes se confirment autour du niveau d'abstraction que doivent permettre les ressources matérielles des circuits reconfigurables. Les deux catégories se distinguent principalement par leur **granularité** qui représente la taille des cellules logiques élémentaires que l'utilisateur peut manipuler. On parle de technologie à **grain fin** pour l'une, et à **grain épais** pour l'autre.

Les circuits FPGA sont classés dans la première catégorie. Ils sont constitués d'une structure matricielle de blocs logiques et d'un réseau d'interconnexions qui peuvent être exploités pour construire des unités de traitement à l'échelle du bit. La majorité des projets de recherche, exemple de Splash-2 [1] et PeRle-2 [2], utilisent des composants FPGA du commerce pour concevoir des systèmes numériques en manipulant des portes logiques et des registres (bascules). Les FPGAs sont constitués en général de tables de codage ou LUT (Look-Up Table) connectées entre elles par un réseau de routage hiérarchisé et reconfigurable (figure 1a). Chaque LUT, associée à un ou plusieurs éléments de mémorisation ou FF (Flip-Flop), permet la réalisation d'une à deux fonctions logiques avec un nombre d'entrées variable, de quatre à six selon les familles et les fabricants. Les FPGAs reconfigurables les plus répandus utilisent un modèle de configuration basé sur des SRAM programmables.

La deuxième catégorie, les architectures reconfigurables à grain épais, est plus récente. Elles intègrent des unités de calcul complètes comme des multiplieurs et des cœurs de DSP[2] (figure 1b),

2. DSP : Digital Signal Processing.



(a) (b)

Figure 1. Organisation d'un composant reconfigurable à grain fin (a) et opérateurs des architectures à granularité épaisse (b).

de taille plus grande que les blocs logiques des FPGAs. Parmi les projets les plus importants on peut citer : Chimaera [3], RaPiD [4], XCC/PACT [5], PipeRench [6] et RAW [7]. La largeur des mots que les opérateurs manipulent est souvent de plusieurs bits. La reconfiguration de ce type d'architectures peut se réduire à définir les communications entre les unités en programmant les chemins de données. En France, les projets les plus avancés sont : Systolic Ring [8] et DART [9] en cours de développement respectivement au LIRMM (Montpellier) et à R2D2 (Lannion). Systolic Ring est basée sur des cellules de calcul, appelées Dnode, composées d'une ALU incluant un multiplieur. DART est organisée en *clusters* pouvant exécuter des tâches de manière autonome. Chaque *cluster* de DART intègre plusieurs unités de calcul et un cœur FPGA.

Chacune des deux technologies présente des avantages et des inconvénients, en voici quelques uns. En raison de la distribution spatiale des ressources de calcul, les FPGAs offrent une bonne densité fonctionnelle par unité de temps. Ainsi, ils autorisent une puissance de calcul élevée en exploitant un parallélisme massif inhérent à leur structure. Le concepteur a la possibilité de contrôler les opérations à l'échelle du bit, ce qui permet une meilleure optimisation de l'architecture pour une application donnée. Un autre avantage important est la maturité atteinte par les outils de conception qui, conjuguée avec la caractéristique intrinsèque des FPGAs à être reconfigurés, ont fait leur popularité notamment dans le prototypage rapide.

Les FPGAs par la régularité de leur structure et les ressources logiques intégrées nécessaires à la reconfiguration accroissent la complexité des composants, ainsi que la consommation. De plus, la finesse des cellules élémentaires pénalise la rapidité des opérateurs élaborés en exploitant des ressources réparties sur plusieurs blocs. Certaines familles de FPGA limitent l'effet de cette contrainte en câblant dans le silicium des multiplieurs (Virtex de Xilinx) et des cœurs de DSP (Stratix d'ALTERA) [10]. La démarche qui consiste à intégrer toujours plus d'opérateurs spécialisés aboutit à une consommation élevée et à une complexité accrue des méthodes de conception conduisant à une augmentation du temps de placement/routage (ou P&R) indispensable pour mapper les ressources dans le circuit cible.

Les architectures reconfigurables à grain épais sont destinées à des applications manipulant des données à l'échelle d'un mot (8, 16 ou 32 bits). Du fait que les cellules logiques élémentaires sont optimisées pour effectuer des opérations sur des données de grande largeur, leurs performances sur ce type de traitement dépassent celles des structures à grain fin. Les opérations essentielles telles que les additions, les multiplications et les opérations d'accumulations étant pour l'essentiel optimisées dans le silicium, la quantité de données nécessaires à la reconfiguration est réduite. Un cycle de compilation allégé et un temps de reconfiguration réduit font de ces circuits un choix judicieux pour construire des architectures reconfigurables dynamiquement.

Toutefois, la taille des blocs logiques étant définie à la fabrication, la structure du circuit ne peut optimiser la réalisation d'opérateurs ayant des tailles autres que celles qui sont prévues. Il est notamment très coûteux d'implanter des algorithmes manipulant des bits ou des machines à états finis.

Dans la suite de cet article, nous nous intéressons plus particulièrement aux FPGA-SRAM. Le choix de cette technologie est dicté par plusieurs impératifs : (i) simplicité, facilité de mise au point et rapidité de prototypage, (ii) disponibilité des composants dans le commerce qui, malgré leurs limitations, suffisent pour valider un concept, (iii) disponibilité d'outils logiciels performants et de langages de description évolués.

Les types de reconfiguration : statique et dynamique

La reconfiguration d'un FPGA-SRAM est réalisée par l'écriture des données de configuration destinées aux blocs logiques et aux interconnexions. Dans beaucoup d'applications industrielles les FPGAs sont préférés aux ASICs pour leur souplesse et leur faible coût. L'augmentation de leur capacité d'intégration les rend pratiquement incontournables pendant la phase de validation d'une application qui passe souvent par la fabrication d'un prototype. Dans la version définitive, le FPGA est utilisé dans les mêmes conditions que l'aurait été un ASIC avec toutefois la possibilité d'adapter la fonctionnalité à de nouveaux besoins. L'architecture implantée dans le FPGA n'est modifiée qu'à la suite d'une mise à jour. En fonctionnement normal, le FPGA n'est configuré qu'une seule fois au lancement de l'application (figure 2a), on parle de *reconfiguration statique*. Par conséquent, la capacité de reconfiguration offerte par les FPGAs n'est exploitée que partiellement.

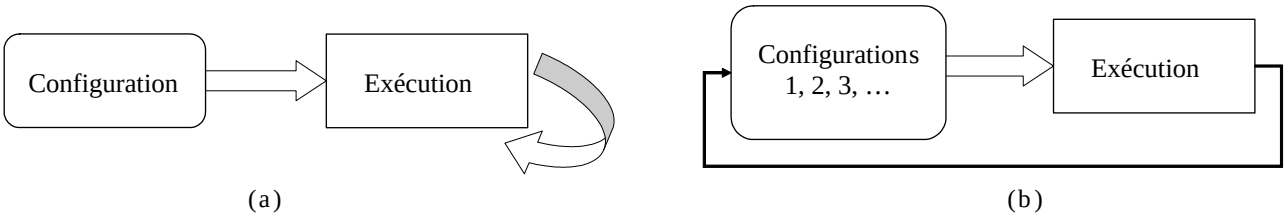


Figure 2. Configuration statique, le système est configuré une seule fois (a), et reconfiguration dynamique avec une succession des phases de reconfiguration et exécution (b).

L'idée qui consiste à reconfigurer le FPGA plusieurs fois (figure 2b) pour exécuter une ou plusieurs applications, est au centre d'un grand nombre de projets de recherche dans le monde, on parle de *reconfiguration dynamique*. La reconfiguration dynamique permet d'augmenter l'efficacité du FPGA en allouant ses ressources logiques à plusieurs tâches. Une application complexe peut être découpée en partitions exécutées de façon séquentielle sur une quantité de ressources logiques réduite par rapport à une implantation statique. Ainsi, par analogie avec la notion de *mémoire virtuelle* qui permet entre autres d'exécuter un programme de taille plus grande que la mémoire physique disponible, un FPGA reconfigurable dynamiquement peut offrir une quantité importante de *ressources logiques virtuelles* et les mapper au moment de l'exécution sur une quantité réduite de ressources physiquement disponibles.

Un autre avantage qui suscite un intérêt croissant chez les chercheurs, est la flexibilité introduite par la reconfiguration dynamique et qui se manifeste à deux niveaux. La première est la flexibilité au niveau système qui permet au concepteur d'adapter son application pour faire face à de nouvelles contraintes ou pour remplacer un algorithme par un autre plus efficace. Cette flexibilité assure une certaine évolution du système. La deuxième flexibilité se situe au niveau du choix des tâches à exécuter, qui peut se faire en dynamique, en fonction par exemple des données à traiter [12]. On introduit ainsi le mécanisme de rupture de séquence dans le cycle répétitif, configuration/exécution, où l'ordre des configurations à séquencer n'est plus fixé au moment de la compilation (statique) mais décidé au moment de l'exécution (dynamique). On peut donc tout à fait imaginer que les données d'entrée puissent influer en temps réel sur l'enchaînement des algorithmes. Le matériel devient extrêmement malléable et contrôlable par logiciel, il peut être utilisé à des instants différents pour exécuter des opérations différentes.

Théoriquement, la reconfiguration dynamique n'est pas réservée à une catégorie particulière de FPGA. Cependant, certains sont plus adaptés que d'autres pour satisfaire les contraintes temps réel. Le FPGA idéal pour construire un système reconfigurable dynamiquement doit posséder une vitesse d'exécution élevée et une durée de reconfiguration très faible. Malheureusement, les FPGAs commercialisés à l'heure actuelle ne réunissent que partiellement ces deux caractéristiques. Cependant, un FPGA offrant une seule des deux caractéristiques, une rapidité de reconfiguration à l'exemple de la famille AT40K d'Atmel [13] ou une vitesse de calcul à l'exemple des Virtex de Xilinx [14][15], sans que la deuxième caractéristique ne soit excessivement pénalisante, peut être utilisé efficacement pour construire un système dynamique.

Une des solutions permettant de diminuer la durée de reconfiguration est la possibilité offerte par certaines familles (AT40K, Virtex, ...) d'être reconfigurées partiellement. On distingue alors : la *reconfiguration dynamique totale*, consistant à changer la fonctionnalité de tout le circuit ; de la *reconfiguration dynamique partielle*, où seule une partie du circuit est modifiée. Actuellement, grâce aux évolutions technologiques, la densité des circuits FPGA est de plus en plus élevée augmentant mécaniquement la durée de reconfiguration totale. Cette dernière peut être limitée en exploitant la reconfiguration partielle. On peut dès lors penser que la taille optimale d'un FPGA pour construire un système dynamique sera le fruit d'un compromis entre la durée de reconfiguration et la puissance de calcul recherchée. La démonstration analytique sera apportée dans la suite de cet article. Il est clair que la durée de reconfiguration a un impact sur les performances d'un système dynamique, particulièrement si la modification de la fonctionnalité intervient fréquemment. La meilleure utilisation d'un FPGA dans un système dynamique est de servir de coprocesseur à une machine hôte pour accélérer des traitements répétitifs sur un volume de données important. La « rémanence » R, un critère discriminant introduit par Didier Demigny, donne une indication du volume de données minimum qui doit être traité entre deux configurations pour qu'une architecture soit efficace. La valeur élevée de R dans le cas d'ARDOISE [11], 16450 cycles, confirme que l'architecture proposée est bien adaptée au traitement des images. La reconfiguration dynamique en général et la reconfiguration partielle en particulier, s'inscrivant dans le cadre du concept émergeant de réutilisation de blocs déjà conçus et validés (IP ou Intellectual Properties), vont probablement conduire à sortir les FPGAs de leur utilisation classique : le prototypage rapide.

La suite de l'article est structurée comme suit. La section 2 présente l'architecture système d'ARDOISE et la stratégie de séquencement des algorithmes d'une application. La définition de nouvelles métriques pour l'évaluation et la comparaison des performances de différentes architectures dynamiques est détaillée dans la section 3. Un exemple de mise en œuvre de la reconfiguration dynamique et partielle est illustré dans la section 4. La section 5 est dédiée à la description des outils que nous avons développés pour mieux exploiter la plate-forme ARDOISE et aider au portage d'applications. Enfin, la section 6 conclut l'article.

2. ARDOISE : une plate-forme matérielle pour la RD

Le projet ARDOISE a réuni une dizaine d'équipes de recherche[3], appartenant aux communautés GDR-ISIS/ARP, ayant une expérience reconnue en architecture des systèmes temps réel pour le traitement d'images qui ont coopéré à la spécification d'une architecture à reconfiguration dynamique [16, 17].

3. Les équipes ayant travaillé sur les aspects architecturaux sont : LE2I (Dijon), LIEN (Nancy) et ETIS (Cergy).

2.1. Organisation système

L'idée principale d'ARDOISE est d'allouer les mêmes ressources matérielles pour exécuter des algorithmes de traitement d'images à des instants différents, en reconfigurant dynamiquement le système plusieurs fois pendant la durée d'une image (figure 3). En utilisant ce concept et en exploitant le parallélisme intrinsèque des FPGAs, il est possible d'augmenter de manière significative les performances du système.

L'architecture du système ARDOISE est composée d'au moins trois modules identiques (figure 4a). Chaque module (figure 4b) intègre un FPGA reconfigurable dynamiquement, AT40K40 d'Atmel (environ 40K portes équivalentes), connecté à deux mémoires statiques servant de mémoires locales pour la sauvegarde des résultats intermédiaires. Les modules voisins peuvent communiquer entre eux par un bus banalisé de 48 bits, servant indifféremment pour des données et/ou des adresses.

Un autre module d'une organisation similaire, appelé *carte mère*, comporte : des mémoires pour le stockage des configurations dans un format compressé, des ressources logiques (un FPGA AT40K40) pour gérer et séquencer les configurations, et enfin des générateurs d'horloges pour permettre aux différentes tâches de s'exécuter à leur fréquence maximale admissible. La *carte mère* peut fonctionner selon deux modes : mode autonome (stand-alone) et mode coprocesseur. Dans le mode autonome, le FPGA de la *carte mère* se configure à la mise sous tension et exécute une séquence de configurations pré-définie. Dans le deuxième mode, ARDOISE est pilotée par un processeur hôte, un DSP SHARC 21061 d'Analog Devices.

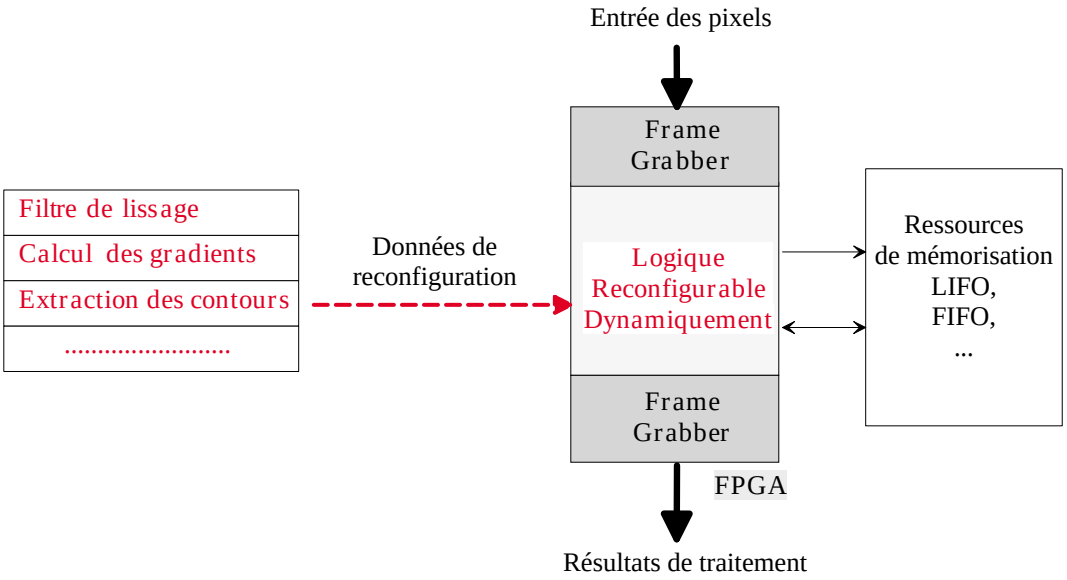


Figure 3. Architecture avec reconfiguration dynamique et partielle.

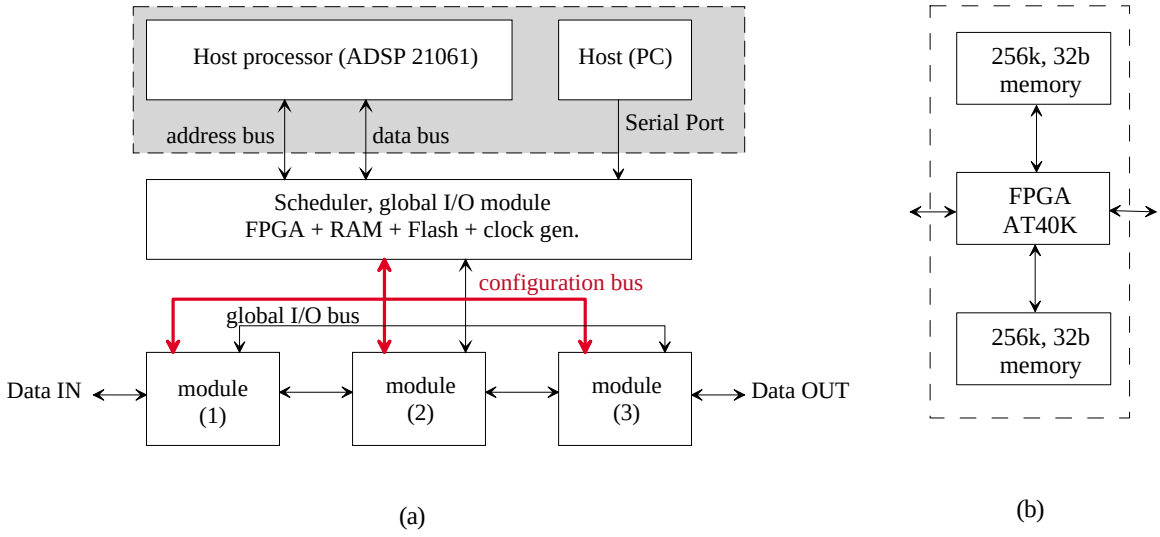


Figure 4. Organisation modulaire du système ARDOISE (a), constitution d'un module d'ARDOISE (b).

Deux modules du système ARDOISE, baptisés $GTI_{1,2}$ pour Gestionnaire de Tampons d'Images, font l'interface entre le système d'acquisition des images et le ou les modules de traitement, appelés UTGV, pour Unité(s) de Traitement à Grande Vitesse.

2.2. Séquencement des partitions d'une application sur ARDOISE

Pour qu'une application, initialement réalisée sur un système statique, puisse s'exécuter sur ARDOISE, elle doit être découpée en plusieurs partitions (figure 5). Pour chaque image d'entrée, ces partitions s'exécuteront par multiplexage temporel sur les mêmes ressources logiques du FPGA de l'unité de traitement (UTGV).

La figure 6 montre un des multiples modes d'utilisation d'ARDOISE. Les deux modules GTI servent de tampons d'images, en désynchronisant la fréquence d'acquisition de la fréquence de travail. Pendant le traitement de l'image n par l'UTGV , GTI1 s'occupe de l'acquisition de l'image $n+1$ en stockant ses pixels dans la mémoire A, et GTI2 envoie vers la sortie le résultat du traitement de l'image $n-1$ stocké au préalable dans la mémoire C. La première configuration est chargée

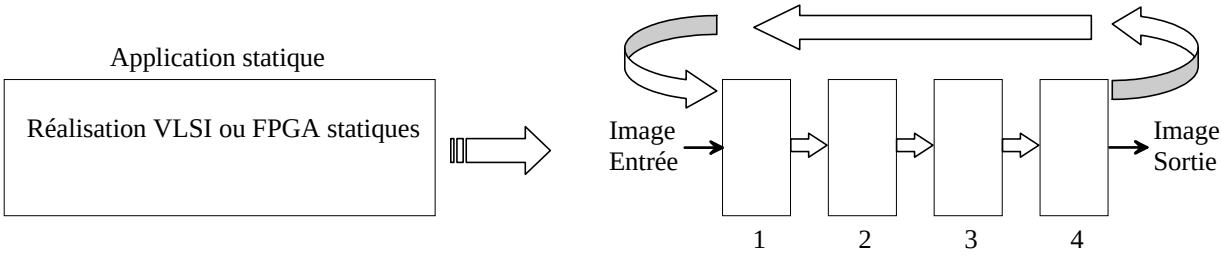


Figure 5. Réalisation dynamique d'une application découpée en 4 partitions.

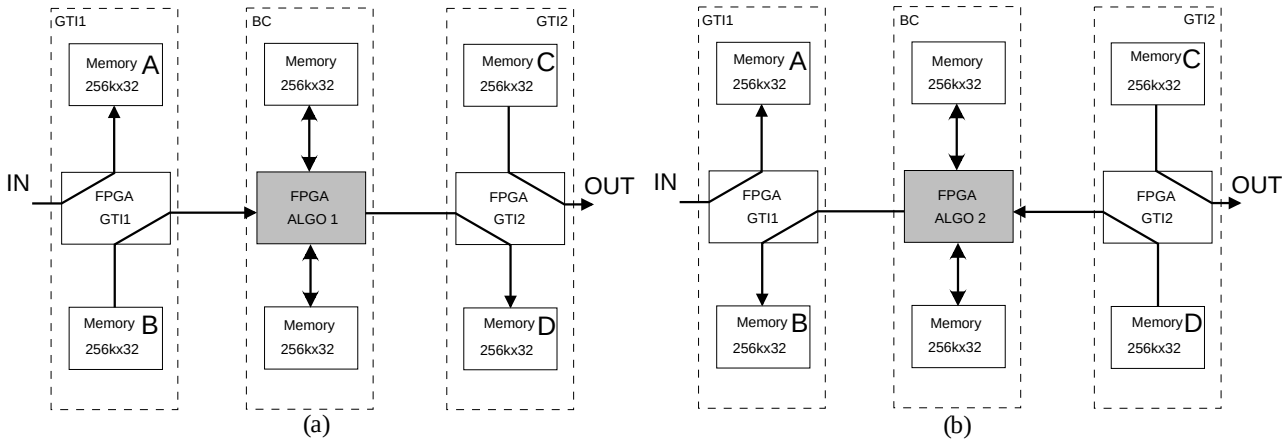


Figure 6. Exemple d'exécution de deux algorithmes en séquence sur ARDOISE.

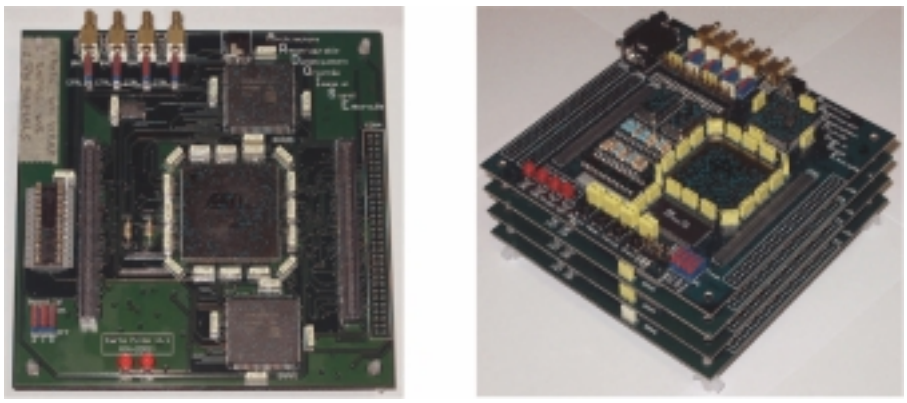


Figure 7. Photos d'ARDOISE : à gauche un module, à droite la plate-forme opérationnelle composée de 2 cartes GTI, 1 carte UTGV et de la carte mère (à droite).

dans le FPGA de l'UTGV (gris). L'image *n* étant précédemment stockée dans la mémoire B, le premier algorithme lui est appliqué (figure 6a), et le résultat est sauvegardé dans la mémoire D. Un second algorithme est appliqué (figure 6b) grâce à une reconfiguration dynamique du FPGA de l'UTGV. Pour cet algorithme, la mémoire D contient les données à traiter et B reçoit les résultats. Ce cycle de configuration/exécution continue à être appliqué sur les algorithmes suivants. Ainsi, pendant la durée d'une image, plusieurs traitements sont exécutés en partageant les mêmes ressources logiques (UTGV). À chaque nouvelle image, les rôles des mémoires A et B d'un côté et C et D de l'autre sont permutés. Grâce à la modularité de l'architecture, l'UTGV peut être composée de plusieurs modules pour faire face aux éventuelles exigences d'une application complexe.

3. Évaluation de la reconfiguration dynamique

Le rendement des systèmes statiques est souvent médiocre, notamment dans le cas des architectures flot de données où la vitesse de traitement est liée à la fréquence d'acquisition des échantillons. Dans le cas du traitement d'images, les algorithmes opérant sur des blocs de données, par exemple les pixels d'une image, le système est incapable d'exploiter les nombreux temps morts pour accélérer le traitement ou en effectuer d'autres. Avec les avancées technologiques, les FPGAs sont de plus en plus rapides et complexes. Cependant, si un FPGA est cadencé à la fréquence d'acquisition des échantillons, le rendement sera encore plus faible puisqu'il n'est alors possible d'exploiter ni le pipeline, ni le parallélisme afin de réduire la durée des traitements. Autrement dit, si les contraintes temporelles sont respectées, le choix d'un circuit plus récent n'améliorera pas les performances, hormis de disposer de plus de ressources logiques pour permettre la mise en cascade de plusieurs traitements. La reconfiguration statique ne permet pas d'utiliser tout le potentiel des FPGAs. En revanche, en plus de la flexibilité présentée précédemment, la reconfiguration dynamique a pour objectif d'optimiser l'utilisation de la puissance offerte par cette technologie. L'exécution de l'application dans le temps sur une surface beaucoup moins importante que l'implantation statique, va logiquement augmenter le rendement du matériel. De plus, l'utilisation du pipeline et/ou du parallélisme ajoutée à une stratégie de traitement de données par bloc qui réduit la durée globale de reconfiguration peuvent être mis à profit pour réduire le temps d'exécution de l'application.

3.1. Puissance de traitement et rendement matériel

Pour évaluer l'apport de la reconfiguration dynamique, nous avons développé de nouvelles métriques qui prennent en compte la vitesse de traitement, la surface utilisée et la durée de reconfiguration qui ne sont pas intégrées dans certaines métriques, comme la densité fonctionnelle [18] ou la contrainte temporelle surfacique [19]. Bertin dans [20] a suggéré une expression pour calculer la puissance utile d'une architecture statique Pu_s réalisant une application, par le produit du nombre de portes logiques G_s utilisées et de la fréquence de fonctionnement, correspondant le plus souvent à la fréquence d'échantillonnage F_e. Par analogie, nous avons étendu cette définition au cas d'une architecture dynamique en proposant de formuler la puissance utile d'une architecture dynamique Pu_d par le produit du nombre de portes par la fréquence de traitement. Une implantation statique occupant une surface totale G_s, sera découpée en C partitions en vue d'exploiter la reconfiguration dynamique sur une surface unique G_d. On désigne par V_c la vitesse de configuration donnée en nombre de portes configurées par seconde.
Pour simplifier les expressions qui vont suivre, nous allons considérer la fréquence d'exécution moyenne F_t. Si N est le nombre d'échantillons dans un bloc de données acquis pendant la durée T et β_p le nombre de données traitées en parallèle dans chacune des partitions, on peut écrire les expressions suivantes :

$$\begin{aligned} F_e &= \frac{N}{T} \\ Pu_s &= G_s \times F_e \\ G_s &= C \times G_d = \sum_{p=1}^{C} \frac{G_d}{\beta_p} \end{aligned} \tag{1}$$

la durée D_p pour exécuter un traitement p est donnée par l'expression :

$$\begin{aligned} D_p &= \frac{N}{F_t \beta_p} + \frac{G_d}{V_c} \\ \Longrightarrow \sum_{p=1}^{C} D_p &= \sum_{p=1}^{C} \frac{N}{F_t \beta_p} + \frac{G_d}{V_c} \leq T \end{aligned} \tag{2}$$

En utilisant les équations 1 et 2, on déduit l'expression de la puissance utile d'une architecture exploitant la reconfiguration dynamique :

$$Pu_d = F_t G_d \left(1 - C \frac{G_d}{V_c T}\right) \tag{3}$$

Pour un nombre de configurations donné, quand G_d augmente, la puissance utile Pu_d passe par un maximum donné par :

$$Pu_{max} = \frac{F_t G_d}{2} \quad \text{avec } Gd_{optimale} = \frac{V_c T}{2C} \tag{4}$$

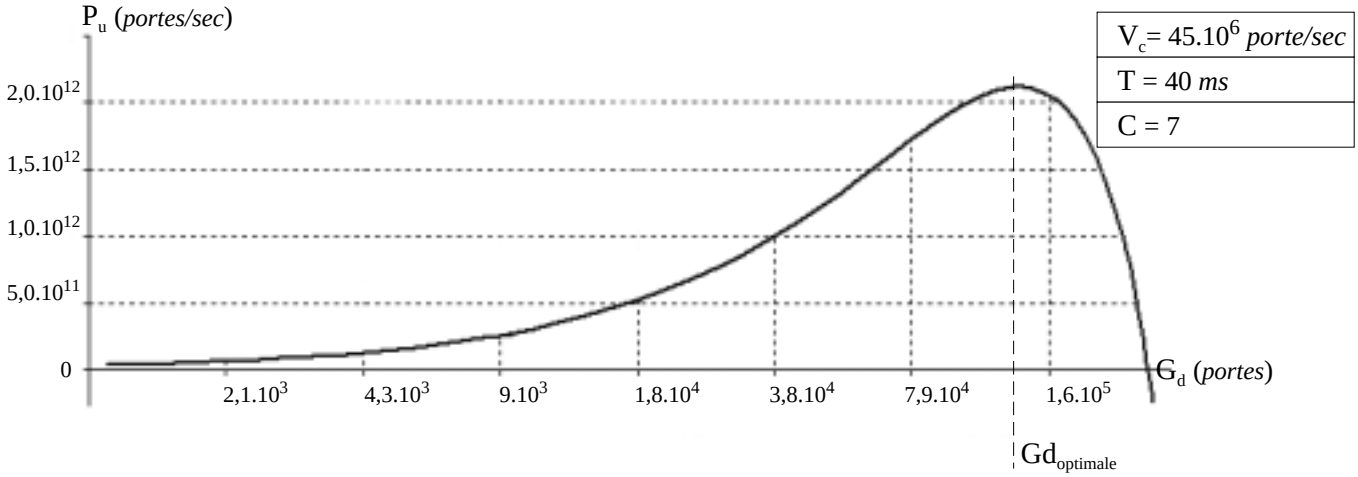


Figure 8. Variation de la puissance utile P_u en fonction de la granularité de partitionnement G_d.

Dans la figure 8, on distingue clairement deux zones délimitées par $Gd_{optimale}$. Une première, dans laquelle l'augmentation de la taille de Gd conduit à une augmentation de la puissance utile P_u. Puis une seconde située au delà de $Gd_{optimale}$, montre qu'augmenter Gd fait chuter la puissance utile. Cette courbe peut être exploitée pour dimensionner la taille du FPGA en fonction de la puissance de calcul que nécessite une application. En substituant la dernière expression de G_d (équation 4) dans l'équation 1, on peut déduire la limite supérieure de la complexité matérielle qu'une architecture à reconfiguration dynamique puisse réaliser :

$$Gs_{max} = \frac{V_c T}{2C} \sum_{p=1}^{C} \frac{1}{\beta_p} \leq \frac{V_c T}{2} \qquad (5)$$

L'équation 5 montre qu'il existe une limite haute à la complexité matérielle des applications à implanter en utilisant la reconfiguration dynamique. Celle-ci s'exprime exclusivement en fonction de la vitesse de configuration V_c autorisée par la technologie et éventuellement par la durée du bloc de donnée T sur lesquelles les différentes configurations sont appliquées. Par exemple, la faible vitesse de configuration V_c de la famille Xilinx 4000 par rapport à la famille AT40K d'Atmel, 100 fois plus petite, fait que la complexité maximale de l'application est 100 fois plus faible. Pour réduire la durée de reconfiguration, des solutions exploitant le mécanisme de *cache logic* de reconfiguration interne au composant FPGA ont été proposées, exemple du circuit DPGA [21]. Le terme de *cache logic*, par analogie au concept de *cache memory*, consiste à rapprocher les données de configuration le plus près possible de la logique active du FPGA. Le chargement d'une configuration est effectué dans un plan, pendant que le second plan sert à configurer les ressources logiques pour installer un algorithme nouveau. À la fin du traitement, les rôles des deux plans sont permutés réduisant la durée de reconfiguration au minimum.

En plus de l'importance de la vitesse de configuration, l'équation 5 rappelle que la taille du bloc de données à traiter et sa durée sont déterminants pour dimensionner l'architecture (choix du nombre de portes). Aussi, elle met en évidence le fait que le parallélisme de données réduit la complexité maximale. La limite supérieure $\frac{V_c T}{2}$ est obtenue quand le même parallélisme est utilisé pour toutes les configurations ($\beta_p = 1, \forall p$). Si un parallélisme de données est exploité dans le cas d'une application, sachant que le nombre de portes est figé, peu d'algorithmes peuvent être chargés en une seule configuration. Plusieurs configurations sont alors nécessaires ; il en résulte un temps de reconfiguration plus important, rendant la réduction de G_s indispensable pour respecter la contrainte temps réel.

La reconfiguration dynamique n'a aucun intérêt si le nombre de configurations C est inférieur à 2. Dans le cas où $C = 2$, d'après l'équation 5 la complexité de l'architecture reconfigurable dynamiquement est $G_d = \frac{V_c T}{4}$ et la puissance $Pu_{\max} = \frac{F_t V_c T}{8}$. Si l'application nécessite $Gs_{\max}$ portes, alors la cadence des données est $F_e = \frac{F_t}{4}$.

Par exemple, pour les FPGAs de la famille AT40K d'Atmel, $V_c = 45.10^6$ portes/seconde et $T = 40\ ms$, on obtient : $G_s \max = 900\ Kportes$, $G_d \max = 450\ Kportes$, $P_u \max = 7.4.10^{12}\ portes/\sec$ avec une fréquence de travail $F_t = 33\ MHz$. Les algorithmes habituellement utilisés en segmentation d'images nécessitent environ 160 $Kportes$. Ce qui montre que la technologie AT40K d'Atmel que nous avons choisie est bien adaptée à notre architecture.

3.2. Quelle surface dynamique choisir pour une application donnée ?

Il existe une dépendance entre la puissance utile, le nombre de configurations C et le taux de parallélisme β. Plaçons-nous du côté du concepteur qui doit concevoir une architecture matérielle à reconfiguration dynamique en vue d'implanter une application définie par F_e, T et G_s. Le système devra fournir la puissance de calcul nécessaire afin de respecter la contrainte temps réel, en utilisant un nombre minimum de ressources logiques.
En utilisant l'équation 3, nous pouvons écrire :

$$G_s F_e \leq F_t G_d \left(1 - C\frac{G_d}{V_c T}\right) \tag{6}$$

On peut déduire l'expression de $Gd_{\min}$:

$$Gd_{min} = \frac{G_s F_e}{F_t\left(1 - \frac{G_s}{V_c T}\right)} \tag{7}$$

On constate qu'il existe une limite inférieure de $Gd_{\min}$, obtenue dans le cas où G_s est très faible devant $Gs_{\max}$:

$$Gd_{\min} \simeq \frac{G_s F_e}{F_t} \quad \Longrightarrow \quad \frac{G_s}{Gd_{\min}} \simeq \frac{F_t}{F_e}$$

Ce résultat montre que plus la fréquence d'acquisition des données entrantes est faible, plus la réduction de la surface est importante.

3.3. Amélioration du rendement matériel

Dans le cas d'une architecture reconfigurable dynamiquement, les partitions seront exécutées avec des fréquences optimales afin de se donner la possibilité d'entrelacer le maximum de configurations pendant la durée d'un bloc de données. La puissance maximale Pm_s que peut fournir une architecture statique fonctionnant à sa fréquence maximale $F_{\max}$ est :

$$P_{\max} = G_s \times F_{\max}$$

Le rapport entre Pu_s et $P_{\max}$, donne le rendement matériel :

$$\eta_s = \frac{Pu_s}{P_{\max}} = \frac{F_e}{F_{\max}}$$
$$\Longrightarrow \quad \eta_s = 25\% \ (F_{\max} = 40\text{MHz}, F_e = 10\text{MHz})$$

Avec des fréquences de travail des FPGAs, qui atteignent les 400 MHz (Virtex-II), on constate à quel point le rendement matériel peut être médiocre, et ce, même si la vitesse du flot de données passe à 40 MHz pour des images de taille 1024×1024.
En exploitant les équations précédentes, nous pouvons déduire les expressions donnant le rendement et le nombre de configurations d'une architecture exploitant la reconfiguration dynamique :

$$\eta = \frac{1}{1 + \frac{\beta G_d F_t}{N V_c}} \tag{8}$$

$$C = \left(\frac{F_e}{\beta F_t} + \frac{\beta G_d}{V_c T}\right)^{-1} \tag{9}$$

L'étude de C en fonction de G_d et de β (figure 9a) montre que la granularité de découpage de l'application a une incidence directe sur le nombre de configurations possibles. En effet, la durée de reconfiguration s'allonge proportionnellement à G_d réduisant le temps disponible pour d'autres phases de reconfiguration/traitement. De plus, le parallélisme de données β n'est pas toujours un facteur favorable. À partir d'une certaine taille de découpage, la composante de C due aux durées des reconfigurations prend le pas sur celle due à la durée des traitements. Dès lors, le parallélisme fait passer le nombre de reconfigurations possibles en dessous du cas où il n'est pas exploité (β=1). Pour un taux de parallélisme donné, il existe donc deux zones où l'influence de β est inverse.
La figure 9b montre le gain relatif en silicium apporté par la reconfiguration dynamique en fonction de la puissance P_u que nécessite l'application. Notons pour finir, l'importance de la vitesse de configuration V_c sur la limite de complexité G_s pour

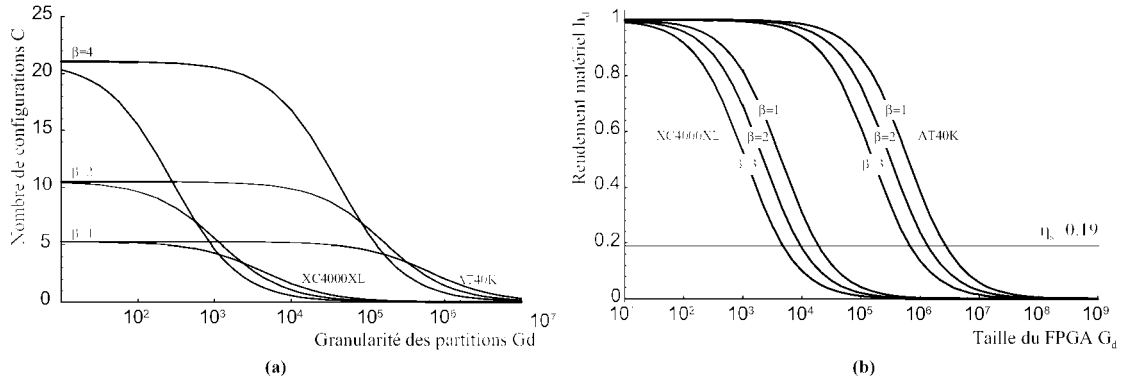


Figure 9. Nombre de configurations en fonction du nombre de portes logiques pour deux technologies FPGA (a), rendement matériel d'une architecture reconfigurable dynamiquement (b).

deux technologies de même génération, AT40K d'Atmel et XC4000 de Xilinx.

3.4. Autres techniques de réduction de la durée de reconfiguration

Nous avons déjà souligné l'impact de la technologie du FPGA sur les performances globales pour l'exploitation de la reconfiguration dynamique. Malheureusement, les composants commercialisés offrent très peu de support pour cette technique. Aussi, des chercheurs ont mené des travaux pour améliorer le rendement en agissant sur l'organisation au niveau système [19]. Deux solutions architecturales reposant sur l'utilisation de deux composants au lieu d'un seul ont été imaginées. Nous avons évalué et comparé les rendements de trois types de réalisations : (i) sans masquage des durées de reconfiguration, (ii) avec masquage et (iii) avec doublement de la vitesse de configuration.

Une architecture sans masquage des durées de reconfiguration, à l'image d'ARDOISE, enchaîne des phases de reconfiguration/exécution pour implanter une application. Dans le cas d'une architecture reconfigurable dynamiquement (ARD) avec masquage (figure 10), deux FPGAs fonctionnant en alternance sont utilisés, quand l'un est en cours de reconfiguration, le second exécute un traitement et réciproquement. La troisième solution consiste à utiliser deux FPGAs contrôlés de façon identique, en les reconfigurant en parallèle et les faire travailler simultanément. Cette technique (figure 11) permet de doubler la vitesse de configuration des FPGAs.

La comparaison s'effectue sur le critère de gain en surface G_g, puisqu'il tient compte à la fois du nombre de reconfigurations, donc de la souplesse, et de la surface, donc du rendement. La table 1 résume son expression pour les trois solutions.

La figure 12 représente le tracé de ce critère en fonction de la taille normalisée du FPGA, pour un rapport F_t/F_e de 10. On distingue clairement deux zones : une première, dans laquelle le

Tableau 1. Expressions du gain en surface.

Architecture normale	Architecture avec masquage	Architecture avec vitesse double
$G_{g1} = \frac{F_t}{F_e}\left(1 - \frac{G_s}{V_c T}\right)$	$G_{g2} = \frac{F_t}{2 \cdot F_e}$	$G_{g3} = \frac{F_t}{F_e}\left(1 - \frac{G_s}{2 \cdot V_c T}\right)$

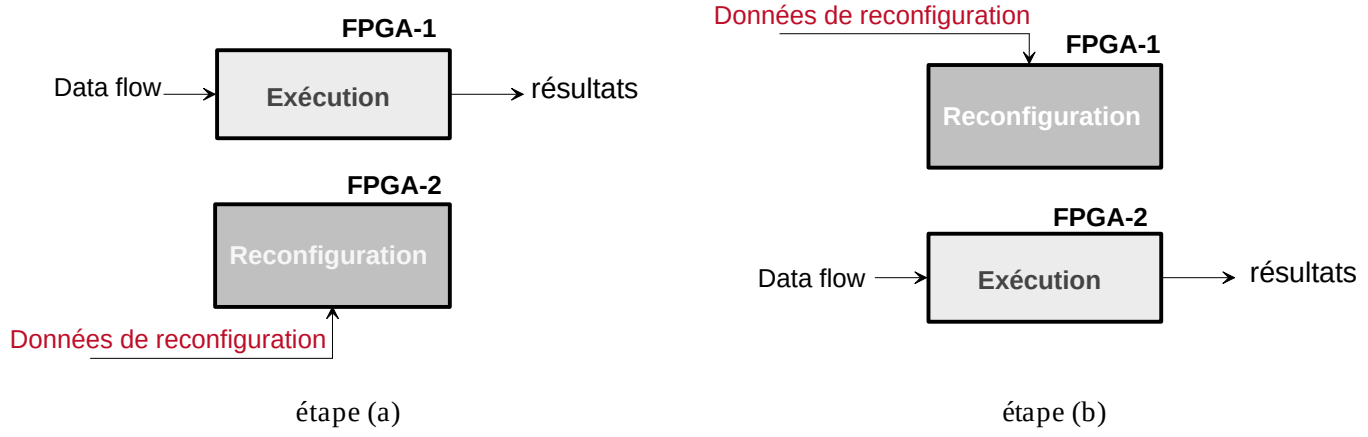


Figure 10. Technique de masquage du temps de configuration.

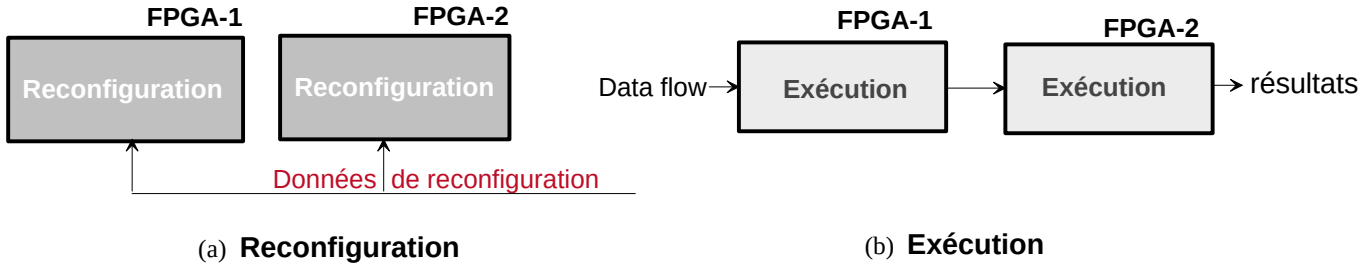


Figure 11. Technique de doublement de la vitesse de configuration.

masquage des délais de reconfiguration est déconseillé, puisque le gain de cette solution est inférieur à la solution simple. Puis, une seconde dans laquelle la tendance s'inverse : le masquage prend le dessus sur la solution classique. Le choix de l'une ou l'autre dépend donc de la technologie FPGA utilisée. En ce qui concerne les circuits AT40K d'Atmel, la durée de reconfiguration n'occupe en réalité que 10 à 20 % de la période T, nous sommes donc nettement dans la première zone. En revanche, ce n'est pas le cas pour des FPGAs de la famille XC4000 de Xilinx, pour lequel la reconfiguration peut occuper les trois quarts de la période T et nous placer dans la seconde zone.

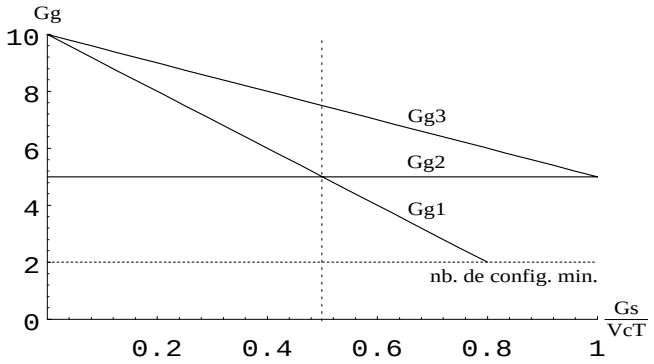


Figure 12. Gain en surface des différentes structures.

Si l'on considère maintenant la troisième solution, on constate qu'elle est meilleure en toutes circonstances. Dès lors, il est difficile de justifier l'intérêt du masquage des délais de reconfiguration, étant donné qu'il est plus facile de reconfigurer deux FPGAs en parallèle qu'en alternance.

3.5. Stratégie de gestion des partitions exploitant les critères

On se place maintenant dans le cas d'une ARD existante. Le nombre de portes étant figé, aucune réduction du silicium n'est recherchée. L'objectif est de trouver une stratégie de séquencement des partitions, en utilisant les équations précédentes, pour implanter une application définie par F_e, T et G_s. L'équation 3 peut servir à vérifier la faisabilité d'une réalisation. Si la réponse est positive, alors on peut trouver plusieurs implantations possibles en jouant sur le parallélisme de données et le nombre de reconfigurations. Le coût de chaque partition et la durée globale peuvent être évalués en utilisant l'équation 2 et en prenant le soin de vérifier le respect de la contrainte temporelle.
Prenons l'exemple d'une chaîne de segmentation pour expliciter cette stratégie. La chaîne de segmentation étudiée est composée du filtre de lissage de Deriche (D), d'une détection de contours (C), d'une fermeture de contours (F) et d'un étiquetage de régions (E). Les implantations de la plupart de ces algorithmes ont été grossièrement évaluées sur un simulateur d'ARDOISE et les outils de P&R d'Atmel. Nous résumons dans les tableaux 2 et 3 les résultats obtenus en prenant comme fréquence de travail $F_t = 33$ MHz et une taille d'image de 1024×1024 pixels. À noter que l'étiquetage des régions nécessite deux passages, E1 et E2. Ces deux partitions ne peuvent être exécutées en parallèle puisque les résultats de E1 sont exploités dans E2.

Tableau 2. Complexités des algorithmes en équivalent portes logiques.

Algorithme	D	C	F	E
nb. de portes (K)	45	22	45	45

Tableau 3. Nombre de configurations et durée d'exécution des algorithmes.

Algorithme	D	C	F	E1	E2
nb. de configurations	2	1	2	1	1
// de données (β)	2	2	8	1	2
temps de traitement (ms) par passe	7,9	3,9	2	7,9	4

En appliquant les équations précédentes sur les différentes stratégies possibles de partitionnement nous obtenons les résultats suivants résumés dans le tableau 4, la barre verticale correspond au chargement de configuration.

Tableau 4. 3 exemples de scénarii pour séquencer les configurations.

Stratégie	Durée totale normalisée	Nb. de configurations
$D_1 \mid D_2 \mid CF_1 \mid F_2\ E_1 \mid E_2$	0.999	5
$D_1 \mid D_2 \mid C \mid F_1 \mid F_2\ E_1 \mid E_2$	0.948	6
$D_1 \mid D_2 \mid C \mid F_1 \mid F_2 \mid E_1 \mid E_2$	0.995	7

On constate qu'il n'y a pas de lien direct entre la rapidité et le nombre de configurations. Si la contrainte temporelle est satisfaite, nous pensons qu'il est préférable de favoriser la modularité des implantations en vue de leur réutilisation.

4. Exemples d'implantations : détection de contours

L'utilisation d'ARDOISE est basée sur la construction d'une bibliothèque de blocs IP câblés. Les premières IPs que nous avons développées sont des filtres de lissage couramment utilisés comme pré-traitements : Nagao, Deriche et le lisseur de Sobel. La qualité du résultat de ces filtres sur des images est

variable. Le choix de l'un ou l'autre de ces filtres peut être dicté à la fois par la qualité des images (degrés de bruit additionnel) et la durée de calcul dont on dispose. Ce choix est le résultat d'un compromis entre la performance recherchée, la contrainte de temps et la quantité des ressources logiques disponibles. La flexibilité offerte par une architecture à reconfiguration dynamique doit permettre de choisir à tout moment le filtre le plus adapté. L'ensemble des algorithmes d'une application sont exécutés sur les mêmes ressources matérielles (module UTGV). Pour réduire les délais de reconfiguration les données de configuration des IPs sont stockées dans la mémoire de la *carte mère* qui joue ainsi le rôle de cache de configurations (« cache logic »). La gestion dynamique des configurations par la *carte mère* est semblable à la gestion d'une mémoire cache ; le but étant de rendre disponibles les configurations les plus utilisées afin d'optimiser l'occupation de la mémoire de configurations.

Le premier test visant à valider le concept de la reconfiguration dynamique sur ARDOISE a été l'implantation temps réel sur un seul module, de l'algorithme de Sobel pour le calcul de gradients [22]. La réalisation matérielle du filtre de Sobel est basée sur la décomposition de l'algorithme en une étape de lissage et une autre de dérivation. Pour chaque image les deux configurations sont successivement chargées et exécutées sur la même surface. La contrainte temps réel impose de ne pas interrompre le flux de données entrant et sortant. Il a donc fallu développer une implantation sous forme de configuration statique sur la périphérie du composant pour prendre en charge l'acquisition et la restitution des données images. Les configurations de lissage et de dérivation sont chargées dynamiquement sur la zone centrale en effectuant une reconfiguration partielle du FPGA (figure 13).

L'implantation du filtre de Sobel sous forme de trois configurations a permis de révéler la non adéquation des outils classiques avec les systèmes exploitant la reconfiguration dynamique et partielle. Aujourd'hui, la plate-forme ARDOISE, composée d'une *carte mère*, de deux cartes GTI et d'une carte de calcul UTGV, est opérationnelle (figure 14). Trois autres laboratoire Français[4] menant des travaux sur la reconfiguration dynamique ont reçu une plate-forme. Pour faciliter le développement d'applications temps réel un certain nombre de choix architecturaux ont été faits, tant au niveau de l'organisation système qu'au niveau du flot de conception. Les plus significatifs sont :

- Les fonctions assurées par le FPGA de la *carte mère* sont figées par une configuration statique. Un mini-processeur avec un jeu d'instructions très réduit y est implanté. Le processeur hôte, le SHARC 21061 d'Analog Devices, contrôle la *carte mère* en lui faisant exécuter un programme écrit avec des instructions simples. Ainsi, il peut exécuter une ou plusieurs instructions, avec possibilité de boucler sur une séquence d'instructions, ce qui permet de gérer l'ordonnancement des configurations des *cartes filles* de deux façons : soit par le processeur hôte en envoyant les instructions une par une et boucler si nécessaire, soit par la *carte mère*, qui reçoit dans ce cas le programme complet.
- Comme pour la *carte mère*, les GTIs sont configurés de manière statique. Ils implantent une structure câblée permettant de gérer les flux de données des différentes IPs qui sont mappées dans l'unité de traitement, UTGV. Le rôle des GTIs s'apparente à celui d'un contrôleur DMA, s'adaptant automatiquement aux exigences de l'IP selon la largeur des mots, le cadrage des données ou encore le sens de la circulation des données (de GTI1 vers GTI2 ou l'inverse).
- La mémoire statique de la *carte mère* contient les configurations des algorithmes les plus utilisés, jouant le rôle de *cache* de configurations. De plus, pour réduire l'espace mémoire et pour accélérer la reconfiguration, les données de configuration sont stockées dans un format compressé. C'est la *carte mère* qui se charge de décompresser les données à la volée au moment de la reconfiguration de l'UTGV. Toujours dans le but de réduire la taille des configurations, nous avons choisi de faire appel à la

4. Il s'agit des laboratoires : LE2I (Dijon), LIEN (Nancy) et R2D2 (Lannion).

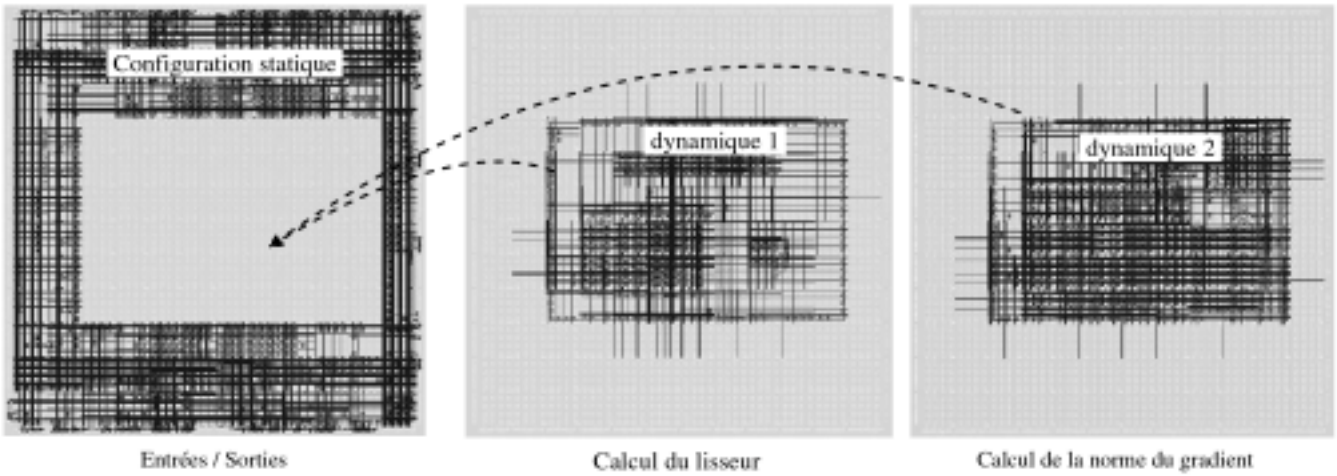


Figure 13. Configuration partielle : exemple du filtre de Sobel.

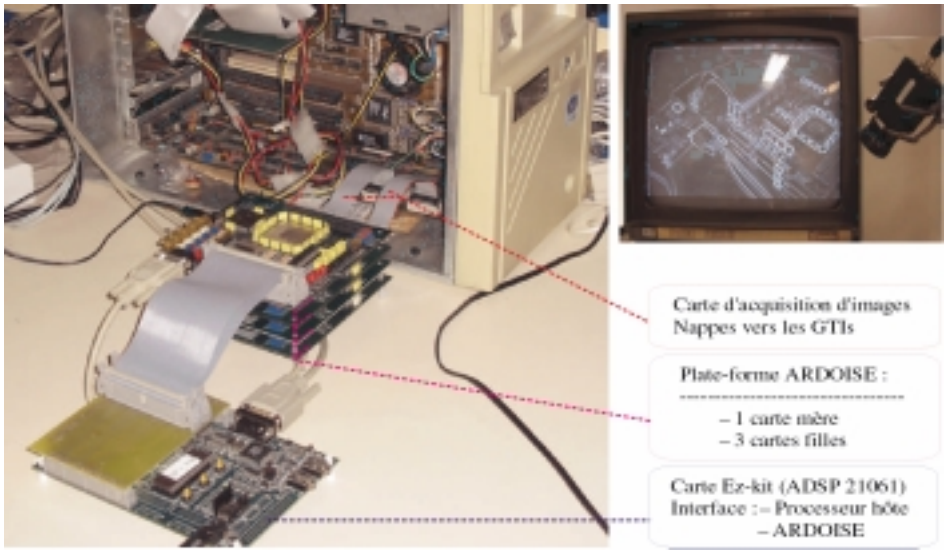


Figure 14. Photo de la plate-forme ARDOISE reliée à un système d'acquisition d'images.

reconfiguration partielle pour fixer un ou plusieurs paramètres d'un algorithme en ne prévoyant qu'une seule implantation câblée, plutôt qu'une implantation pour chaque valeur du paramètre (exemple du paramètre γ du filtre de Deriche).
- Le développement des scénarii de gestion des configurations et le contrôle de la plate-forme sont réalisés et exécutés sur la machine hôte (station PC). Ceci permet de décharger le processeur hôte de la plate-forme d'ARDOISE pour effectuer des tâches pour lesquelles il est plus adapté.

5. Outils pour la reconfiguration dynamique

Le mécanisme de la reconfiguration dynamique n'est que partiellement pris en compte par les outils et les flots de conception traditionnels. Par conséquent, il est indispensable d'élaborer des méthodes et des outils pouvant servir, dans un premier temps, à simuler et à valider fonctionnellement le portage d'applications. Dans un deuxième temps, ces méthodes doivent être intégrées dans la chaîne de conception utilisée pour cibler le circuit reconfigurable. Aujourd'hui encore, beaucoup de travail reste à faire pour disposer d'un environnement complet permettant la génération des données de configurations successives à partir d'une description de haut niveau.
Le premier objectif du projet ARDOISE est de fournir une plate-forme matérielle pour explorer et tester les possibilités et les apports de la reconfiguration dynamique. Proposer des méthodes de développement et des stratégies de partitionnement d'applications pour des systèmes dynamiques est d'un intérêt essentiel pour les outils du futur. Ils doivent permettre la validation de séquences d'applications aussi bien pendant la simulation que pendant l'exécution sur la plate-forme avec suffisamment de souplesse et de flexibilité.

5.1. Environnement d'aide au portage d'applications

Le test des modules d'ARDOISE est mené dans notre laboratoire en parallèle avec le portage d'algorithmes couramment utilisés pour la segmentation d'images. Le découpage d'une application en séquences de traitement (partitions) est effectué de façon intuitive et manuelle. La validation fonctionnelle des partitions est réalisée grâce à un simulateur d'ARDOISE dans lequel la structure matérielle est modélisée en VHDL [23, 24]. La synthèse logique puis la phase de P&R sont utilisées pour évaluer la complexité de l'application et la durée d'exécution de chaque configuration.
La plupart des outils de simulation VHDL offrent des passerelles logicielles autorisant le co-design avec l'interfaçage du noyau de simulation avec d'autres outils. Il devient par exemple possible de simuler un module écrit en VHDL en générant les stimuli dans un langage tel que C. Cette solution peut apporter un certain confort d'utilisation en simplifiant la manipulation des données et l'interprétation des résultats. Cependant, le temps de développement et de simulation dû à l'utilisation de deux langages s'en trouve souvent allongé.
C'est pour améliorer la souplesse de la conception et de la simulation que nous avons développé un environnement écrit entièrement en langage C++, basé sur l'approche SystemC [25]. À cet effet, l'architecture d'ARDOISE a été entièrement décrite en SystemC. L'ensemble forme un environnement convivial offrant une grande flexibilité et intégrant des fonctionnalités nouvelles faciles à maintenir et à faire évoluer (figures 15).

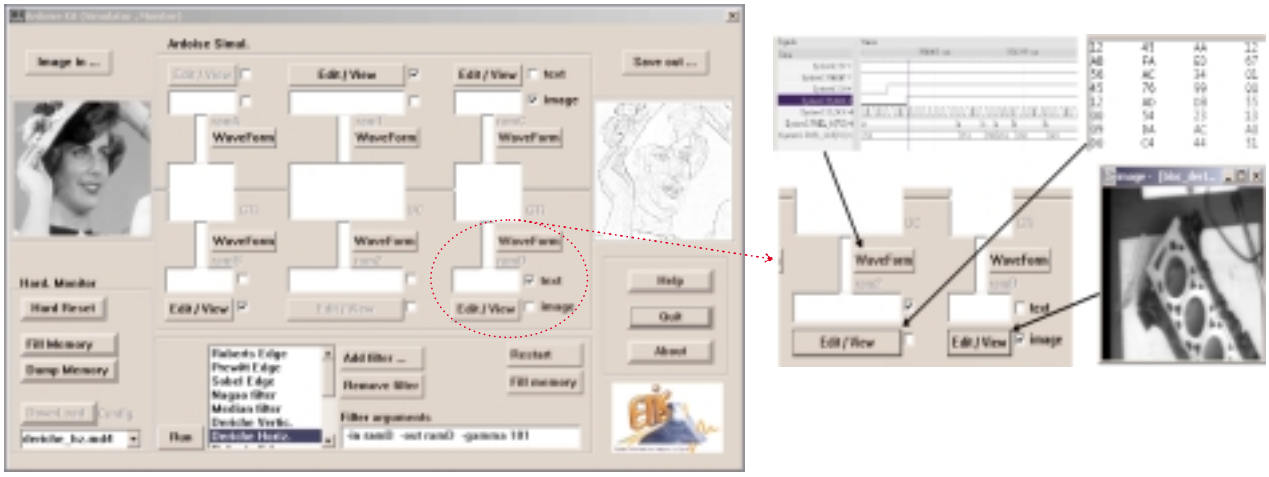
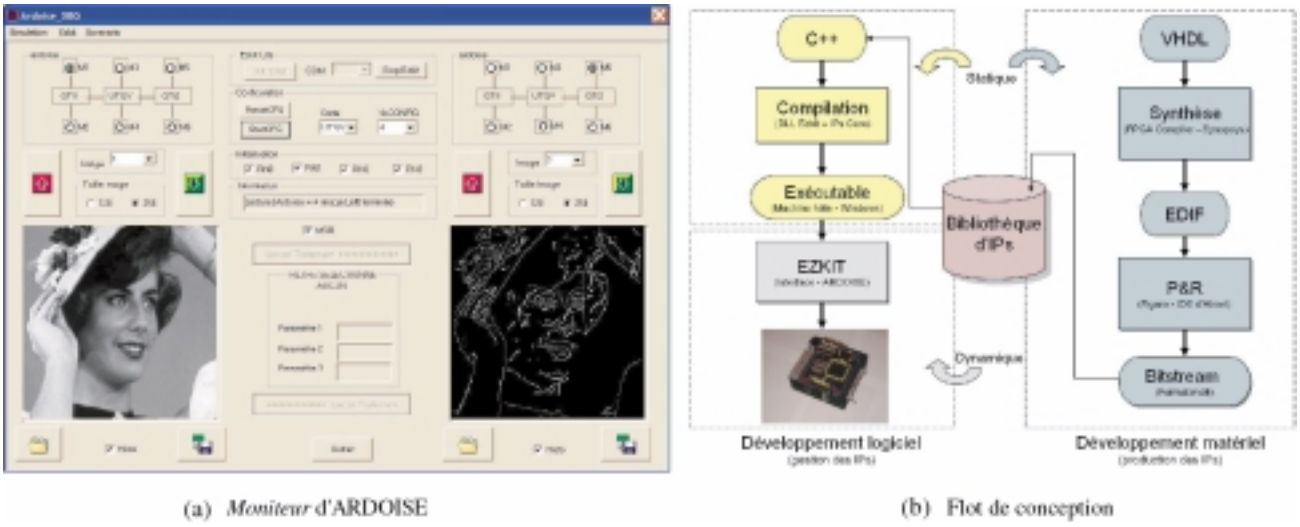


Figure 15. Environnement de simulation et de portage d'applications sur ARDOISE.



(a) *Moniteur* d'ARDOISE (b) Flot de conception

Figure 16. Méthodologie de développement et environnement de test pour ARDOISE.

Aujourd'hui, le développement de l'environnement d'ARDOISE est dans une phase très avancée. Dans sa version finale, l'outil fonctionnera selon 2 sessions : une session de *simulation* et une session de *monitoring*. La session *simulation* est utilisée pour valider fonctionnellement les partitions, alors que la session *monitoring* sert à piloter la plate-forme lors de la mise au point d'applications (*cf.* 5.2).

La partie *simulation* de l'environnement offre des fonctionnalités essentielles pour simuler un traitement, visualiser le contenu des mémoires après une configuration, choisir une image de test, afficher les résultats, visualiser les chronogrammes, créer des scripts pour automatiser les étapes de synthèse et de P&R. La synthèse logique vers une description RTL (Register-Transfer Level) des designs écrits en SystemC est basée sur le flot de conception Synopsys^{TM} CoCentric Compiler [26]. En résumé, les objectifs essentiels de cet environnement sont :

1. la validation des partitions par l'utilisation d'images de test,
2. la réduction du temps de simulation par rapport aux simulateurs HDL (VHDL, Verilog),
3. la validation et la mesure des performances réelles de la RD,
4. la mise en œuvre d'applications avec la possibilité d'exécuter séquentiellement les configurations pendant la phase de simulation (session *simulation*),
5. la co-simulation matériel/logiciel. Dans le mode *monitoring*, l'environnement peut être utilisé pour contrôler les modules d'ARDOISE et enchaîner des cycles de configuration/exécution afin de valider les séquences de traitement. Entre l'exécution de deux séquences, les données intermédiaires stockées dans les mémoires locales de chaque module peuvent être chargées en mémoire du système hôte pour être analysées.

5.2. Outils et stratégie de gestion des configurations

Les premiers tests matériels que nous avons menés sur le prototype ARDOISE, ont nécessité un nombre limité de développement logiciel. En effet, à l'exception de quelques utilitaires que nous avons développés, l'utilisation du flot de conception clas-

sique d'une part, et l'adaptation de certaines fonctionnalités des outils d'Atmel (gestion de projets, P&R, création de fichiers de configurations ou bitstreams), d'autre part, ont suffi pour tester le fonctionnement d'un module en statique, puis valider concrètement le concept de reconfiguration dynamique en enchaînant plusieurs configurations. En revanche, le test de la plate-forme complète, ne pouvait se faire sans développement d'outils efficaces et adaptés. La première priorité était de pouvoir effectuer des tests en utilisant des fichiers images, le but étant de s'affranchir du système d'acquisition des images (PC, carte d'acquisition, caméra et moniteur pour la visualisation). L'idée de base est de pouvoir charger une image de test dans une mémoire d'un GTI, puis configurer l'UTGV pour exécuter un traitement et enfin récupérer l'image de sortie dans une mémoire du second GTI pour la comparer avec un résultat obtenu par simulation. Pour ce faire, nous avons développé en C++ un outil qui nous a permis de contrôler entièrement les ressources des différentes cartes, à partir d'une IHM développée spécifiquement pour l'architecture d'ARDOISE (figure 16a).

5.2.1 Environnement de test et de validation d'IPs

La stratégie que nous avons adoptée pour le test et le développement d'applications pour ARDOISE (figure 16b) peut se résumer en deux points : (i) production des configurations en statique (« off-line ») et (ii) gestion et contrôle en dynamique (« on-line »). La conception et la fabrication d'IPs câblées se fait en utilisant le flot traditionnel : description de l'IP en VHDL/SYSTEMC, synthèse logique assurée par *FPGA Compiler* de Synopsys et enfin l'implantation physique avec les outils de P&R d'Atmel. Le *moniteur* permet l'exécution de l'IP produite en téléchargeant dans l'UTGV les données de configuration. Il faut remarquer que, grâce à la désynchronisation du flux vidéo d'entrée avec celui du traitement, le test des IPs se fait dans des conditions qui sont celles du temps réel.
Les principales fonctionnalités offertes par le *moniteur* sont :
- l'écriture des données d'entrée, images de test, dans une des mémoires d'ARDOISE,
- la lecture d'une image résultat suivie de son affichage,
- le chargement et exécution d'une configuration, partielle ou totale, dans un des FPGAs,
- l'initialisation de la mémoire statique de la *carte mère* avec une ou plusieurs configurations.

5.2.2. Environnement de développement d'applications temps réel

Le *moniteur* présenté ci-dessus peut être utilisé pour tester un scénario, les opérations étant effectuées manuellement. Pour programmer une application temps réel, nous avons choisi de développer une extension au langage C++ sous forme d'objets. Normalement, la gestion des configurations dynamiques est l'une des charges que devrait assurer un système d'exploitation. En l'absence de ce dernier, nous avons intégré un certain nombre de mécanismes dans des classes C++ pour prendre en charge la gestion des ressources utilisées, et ce de manière transparente pour le programmeur. Les structures de données les plus importantes que nous avons implantées sont :
Classe IP : permet la déclaration d'une IP en précisant les paramètres associés tels que le fichier contenant les données de configuration, la fréquence d'exécution, un ou plusieurs coefficients propres à l'algorithme, etc. La ligne suivante :

```
IP Lisseur_Deriche ("LD", "./configs
        /LD.ar2", 15, 0.625);
```

déclare une instance de la classe *IP* appelée *Lisseur_Deriche* en précisant le fichier bitstream correspondant, la fréquence d'exécution 15 MHz et le paramètre choisi pour le lissage $\gamma = 0.625$. Au moment de l'exécution de l'IP *Lisseur_Deriche*, la fréquence d'exécution est réglée à 15 MHz par la *carte mère* qui assure l'enchaînement de deux configurations, configuration totale pour la première et partielle pour la seconde afin de fixer le paramètre γ à 0.625.
- **Classe IMAGE :** permet la déclaration des mémoires images que le programme manipule. Les mémoires des GTIs sont organisées en segments, chacun pouvant stocker une image de 512×512 pixels. L'allocation d'une instance *IMAGE* peut être différée au moment de l'exécution de l'IP. Par exemple, la ligne suivante :

```
IMAGE im0("./images/lena.gdr"), im1;
```

déclare deux instances de type *IMAGE*, *im0* sera allouée et initialisée avec l'image « *lena.gdr* », alors que l'allocation de *im1* ne sera faite qu'au moment de l'exécution suivant la direction du flux de données de l'IP, GTI1 $\curvearrowright$ GTI2 ou GTI1 $\curvearrowleft$ GTI2.

5.3. Exemples d'utilisation

La programmation d'une application, faisant appel aux deux précédentes classes, peut cibler différents contextes d'utilisation. Pour mieux fixer les idées, nous allons présenter trois exemples d'utilisation.

5.3.1.Test d'une IP câblée

Le but étant de valider une IP, on se limite à l'utilisation d'une image de test.

```
IP Lisseur_Deriche ("LD", "./Configs/der1.ar2", 15, 0.625);
IMAGE im0 ("./images/lena.gdr"), im1;
Lisseur_Deriche.Run (im0, im1); /*
            Exécution de Lisseur_Deriche */
```

La méthode *Run()* lance l'exécution de *Lisseur_Deriche* sur des données stockées dans *im0*. Dans ce cas, la variable *im0* sera allouée par défaut dans GTI1, l'image *"lena.gdr"* y sera chargée au lancement de l'application. En revanche, *im1* sera allouée

dynamiquement dans GTI2 au moment de l'exécution de l'IP *Lisseur_Deriche* pour y ranger le résultat du traitement.

5.3.2. Application temps réel

Dans ce cas, il s'agit de définir un scénario, sous forme d'un programme C++, pour séquencer l'exécution des IPs en temps réel. Dans l'exemple suivant, trois configurations sont exécutées en temps réel sur chaque image. La fonction *Attente_Debut_Image* permet de se synchroniser avec le flux vidéo.

```
IP Lisseur_Deriche ("LD", "./Configs/der1.ar2", 15, 0.625);
IP gradient ("GRAD1", "./Configs/grad1.ar2", 10);
IMAGE im0, im1, im2, im3;
while(true)
{
        Attente_Debut_Image();
        Lisseur_Deriche.Run(im0, im1);
        gradient.Run(im1, im2);
}
```

5.3.3. Flexibilité : choix en dynamique de l'IP à exécuter

Dans ce cas, on peut envisager de prévoir un séquencement conditionnel des configurations.

```
while(true)
{
        Attente_Debut_Image();
        if(condition_sur_image)
            Lisseur_Deriche.Run (im0, im1);
        else
            Nagao.Run(im0, im1);
         gradient.Run(im1, im2);
}
```

La figure 17 montre le résultat d'une application temps réel avec un enchaînement sur une image d'une séquence de 7 configurations : 4 pour exécuter le lissage 2D de Deriche, 1 pour le calcul des gradients, 1 pour effectuer un seuillage et enfin la dernière pour faire une inversion vidéo. Le changement des paramètres des filtres est assuré en effectuant une reconfiguration partielle. Le résultat des calculs intermédiaires est affiché sur l'écran de sortie vidéo, décomposé pour l'occasion en quatre zones. Le contenu d'une mémoire image peut être redirigé pour être affiché sur un quadrant en utilisant la méthode *SetImageOut(int)* de la *classe IMAGE*. Cette opération est possible grâce à un module supplémentaire, ce qui démontre la possibilité d'augmenter la puissance de calcul offerte par la modularité de l'architecture.

Tableau 5. Durée de configuration et d'exécution des traitements

Algorithme	Durée d'exécution ms	Durée de reconfiguration ms
Lisseur de Deriche 2D	4 × 5,20	4 × 0,33
Calcul du gradient	5,20	0,45
Seuillage simple	5,20	0,28
Inversion vidéo	5,20	0,38
Bilan (38,83 ms) :	36,40	2,43

Le tableau 5 récapitule les temps de configuration et d'exécution de chaque algorithme. Les implantations matérielles sont toutes réalisées sans parallélisme de données et s'exécutent à des fréquences identiques de 15 MHz. Ces résultats, obtenus sans effort d'optimisation, nous permettent d'estimer le temps moyen de changement de configuration à environ 5ms, ce qui représente une moyenne de 200 IPs exécutées par seconde. Les performances seront certainement meilleures si les possibilités du système sont pleinement exploitées : traiter plusieurs pixels en parallèle, cascader plusieurs opérateurs, choisir pour chaque traitement la fréquence maximale admissible, etc.

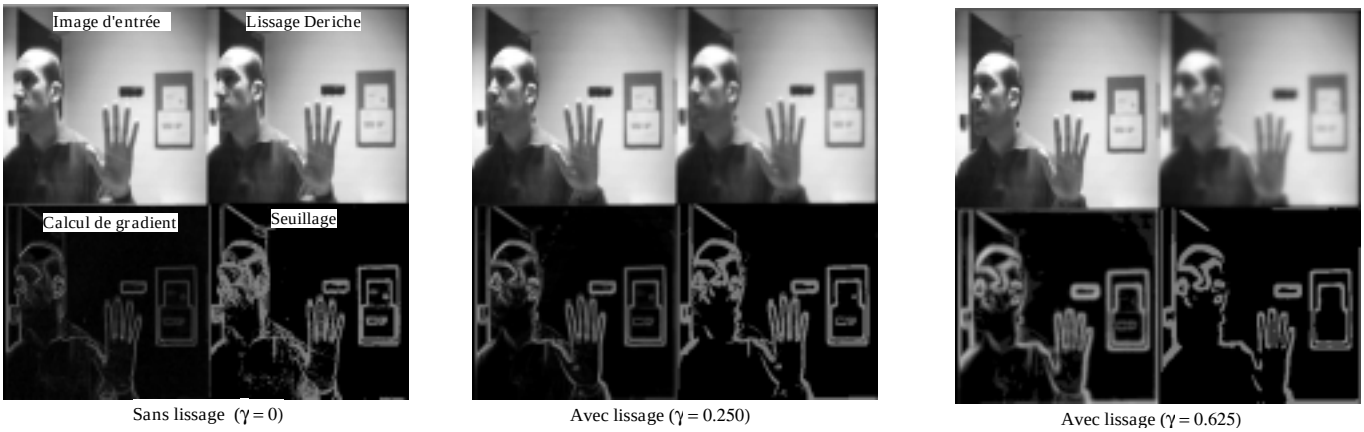


Figure 17. Segmentation d'images avec une séquence de 4 algorithmes : affichage en temps réel des résultats intermédiaires (7 configurations au total).

6. Conclusion et perspectives

Dans cet article, une architecture mettant en œuvre la reconfiguration dynamique de circuits FPGA est présentée. L'objectif principal du projet ARDOISE est de construire un démonstrateur matériel afin de susciter de nouvelles perspectives de recherches couvrant plusieurs aspects : l'architecture, les méthodes de développement et les applications. Les travaux à mener avaient pour but d'identifier les verrous technologiques et de révéler les points durs afin de proposer des solutions pour les résoudre. Il est important de souligner que la performance pure n'était pas la préoccupation majeure car la technologie de la famille AT40K n'était pas au point. En revanche, l'architecture générique d'ARDOISE représente un excellent support matériel pour évaluer les apports de la reconfiguration dynamique, élaborer des méthodes de gestion de l'ARD et aider à l'avènement d'outils de développement et de synthèse adaptés. De tels travaux auront naturellement des retombées importantes sur la conception des systèmes émergeants tels que les systèmes mono-puces (SoC : System on Chip) reconfigurables.

Les résultats issus de cette thématique de recherche centrée sur le concept de reconfiguration dynamique sont multiples. En premier lieu, l'aspect architecture, consacré par la réalisation et le test d'un prototype, a montré l'adéquation de l'organisation spécifiée à intégrer le mécanisme de reconfiguration dynamique. Ensuite, la souplesse de programmation de ce type d'architecture conditionne leur réussite. grâce à des choix architecturaux, qui nous ont permis d'implanter une gestion intelligente des flux de données entre IPs, nous avons proposé un environnement de développement utilisable à plusieurs niveaux : validation des blocs IP, ordonnancement temps réel des configurations, test sur une gestion dynamique de traitements d'images temps réel, etc. Enfin, une façon de caractériser les systèmes reconfigurables dynamiquement est la définition de métriques. Nous avons élaboré de nouveaux critères d'évaluation des ARD qui prennent en compte à la fois des paramètres technologiques, des spécificités de l'application et des choix architecturaux effectués pour implanter les algorithmes. Ces critères peuvent être exploités pour établir et valider des stratégies de partitionnement d'algorithmes sur le matériel.

Nous avons intégré dans l'environnement de développement des mécanismes logiciels indispensables pour la gestion des ressources logiques et le cache de configurations. Pour le concepteur, ces mécanismes se retrouvent indirectement inclus dans le programme de l'application. En guise de perspectives à ces travaux, le développement d'un OS temps réel prenant en compte la RD permettra une gestion centralisée des ressources. En effet, dans le cas d'un système multi-tâches la duplication des précédents mécanismes dans les applications sera évitée et conduira à une optimisation de l'utilisation des ressources matérielles.

La demande croissante en systèmes embarqués pousse les fabricants à intégrer une variété de composants dans un SoC, un ou plusieurs processeurs, des cœurs DSP, des mémoires locales et/ou globales, des blocs dédiés, etc. L'intégration d'une quantité de logique reconfigurable dynamiquement (LRD) à côté d'unités programmables apportera plus de flexibilité aux architectures SoC du futur. Pour augmenter les performances, une partie des applications mises en œuvre sous forme logicielle pourrait être implantée dans la LRD, offrant des possibilités d'évolution à l'application. Dans ce cadre, plusieurs axes de recherches stimulants sont à explorer : organisation et rôle de la LRD, type de granularité (épaisse ou fine) dont le débat est loin d'être tranché, OS temps réel prenant en compte la RD, gestion de tâches mixtes (logicielles et câblées), environnement de développement, etc.

Références

[1] J. ARNOLD, D. BUELL, E. DAVIS, Splash II, In *Proceedings of the Fourth ACM Symposium of Parallel Algorithms and Architectures*, San Diego, CA 1992. p. 316-322.

[2] J. VUILLEMIN, P. BERTIN, D. RONCIN, M. SHAND, H. TOUATI, P. BUCARD, *Programmable Active Memories: reconfigurable Systems Come of Age*, IEEE Transactions on VLSI Systems, 1996.

[3] ALEX ZHI YE, ANDREAS MOSHOVOS, SCOTT HAUCK, and PRITHVIRAJ BANERJEE, CHIMAERA : *A high-performance architecture with a tightly-coupled reconfigurable unit*, In International Symposium on Computer Architecture (ISCA), ACM Computer Architecture News. ACM Press, 2000.

[4] CARL EBELING, DARREN C. CRONQUIST, PAUL FRANKLIN, JASON SECOSKY and STEFAN G. BERG, *Mapping applications to the RaPiD configurable architecture*, In IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), 1997.

[5] JOO M. P. CARDOSO and MARKUS WEINHARDT, *PXPP-VC : A C compiler with temporal partitioning for the PACT-XPP architecture*, In International Conference on Field Programmable Logic and Applications (FPL), Montpellier (La Grande-Motte), France, 2002.

[6] SETH COPEN GOLDSTEIN, HERMAN SCHMIT, MATTHEW MOE, MIHAI BUDIU, SRIHARI CADAMBI, R. REED TAYLOR and RONALD LAUFER, *PipeRench : a coprocessor for streaming multimedia acceleration*, In International Symposium on Computer Architecture (ISCA), pages 28-39, Atlanta, GA, 1999.

[7] MICHAEL BEDFORD TAYLOR, *Comprehensive specification for the Raw processor*, Cambridge, MA, 1997.

[8] G. SASSATELLI, L. TORRES, P. BENOIT, G. CAMBON, M. ROBERT and J. GALY, Dynamically reconfigurable architectures for digital signal processing applications. SoC design methodology. Kluwer 2002. pages 63-74.

[9] R. DAVID, *Architecture reconfigurable dynamiquement pour applications mobiles*, Thèse de doctorat de l'université de Rennes I, 2003.

[10] ALTERA CORPORATION, *Stratix FPGA Family*, Data Sheet v3.0, 2002.

[11] D. DEMIGNY, N. BOUDOUANI, N. ABEL et L. KESSAL, La rémanence des architectures reconfigurables : un critère significatif de classification des architectures, JFAAA, Monastir, Tunis, Dec. 2002.

[12] N. ABEL, L. KESSAL et D. DEMIGNY, Design flexibility using FPGA dynamical reconfiguration, In ICIP 2004, Singapour, October 2004.

[13] ATMEL CORPORATION, AT40K FPGA with FreeRAM^{TM}, Data Sheet, 1999.

[14] P. BUTEL, G. HABAY, A. RACHET, *Managing Partial Dynamic Reconfiguration in Virtex-II Pro FPGAs*, Xcell Journal and http://www.reconf.org, 2004.

[15] BJÖRN GRIESE, ERIK VONNAHME, MARIO PORRMANN, and ULRICH RÜCKERT, *Hardware Support for Dynamic Reconfiguration in Reconfigurable SoC Architectures*, In Proc. of FPL'04, Belgium 2003.

[16] D. DEMIGNY, M. PAINDAVOINE, S. WEBER, Architecture à reconfiguration dynamique pour le traitement temps réel des images, *Technique et Science de l'Information Numéro Spécial Architectures Reconfigurables*, vol. 18, n° 4, pp.1087-1112, 1999.

[17] R. BOURGUIBA, *Conception d'architectures matérielles reconfigurables dynamiquement dédiées au traitement d'images temps réel*, Thèse de l'université de Cergy Pontoise, 2000.

[18] MICHAEL J. WIRTHLIN, Improving functional density through run-time circuit reconfigurable. PhD thesis, Brigham Young University, november 1997.

[19] H. GUERMOUD, *Architectures reconfigurables dynamiquement dédiées aus traitements temps réel des signaux vidéo*, Thèse de doctorat de l'université Henri Poincaré (Nancy), décembre 1997.

[20] P. BERTIN, D. RONCIN and J. VUILLEMIN, Programmable active memories : a performance assessment, In F. Meyer auf der Heide, B. Monien, and A. L. Rosenberg, editors, Parallel Architectures and their efficient use, Lecture notes in Computer Science, Springer-Verlag, 1992, pages 119-130.

[21] A. DEHON, PDPGA Utilization and Application, In ACM/SIGDA Fourth Internationnal Symposium on FPGAs, Monterey, Canada, February 1996.

[22] N. ABEL, D. DEMIGNY, L. KESSAL et N. BOUDOUANI, Mise en œuvre de la reconfiguration partielle sur l'architecture reconfigurable *ARDOISE*, In *Proceedings of the JFAAA*, pages 45-48, Monastir, Tunisie, 2002.

[23] N. BOUDOUANI, *Architectures reconfigurables dynamiquement : synthèse matérielle d'opérateurs de détection et d'estimation de mouvement temps réel*, Thèse de l'université de Cergy Pontoise, 2004.

[24] L. KESSAL, N. ABEL et D. DEMIGNY, Real-time Image Processing With Dynamically Recongurable Architecture, *Real Time Imaging, Elsevier*, 9 pp.297-313, 2003.

[25] SYSTEMC INITIATIVE, http://www.systemc.org

[26] SYNOPSYS CORPORATION, INC, *CoCentric SystemC Compiler*, http://www.synopsys.com

Lounis **Kessal**

Lounis Kessal est Maître de conférences à l'ENSEA. Il a reçu son Habilitation à Diriger des Recherches de l'Université de Cergy Pontoise, UCP, en 2004. Ses recherches portent essentiellement sur l'Adéquation Algorithme Architecture en traitement temps réel des images. Ses travaux de recherche l'ont conduit à étudier l'intérêt de la reconfiguration dynamique dans les architectures dédiées au traitement de l'image et du signal. Actuellement, il effectue ses recherches dans le domaine des systèmes sur puces reconfigurables (RSoC).

kessal@ensea.fr

Nicolas **Abel**

Nicolas Abel termine sa thèse, portant sur les méthodes et outils associés aux ARD, au sein d'ETIS. Lors de ce travail il s'est intéressé à la gestion des flux de données dans ce type d'architecture. En particulier, il étudie les implications de ce mode de gestion, tant sur les méthodologies de développement des modules traitements, que sur leur séquencement en temps réel.

abel@ensea.fr

Didier **Demigny**

Ses activités de recherche concernent les architectures dédiées au traitement de l'image et plus particulièrement les architectures reconfigurables à grain matériel fin ainsi que les métriques qui leur sont associées. Après 18 ans passé au sein du laboratoire « Équipes de traitement des images et du signal » commune à l'université de Cergy-Pontoise et à l'ENSEA, il a rejoint en 2004 l'IRISA et plus précisément l'équipe R2D2 « Exploration, estimation, prototypage pour la conception de systèmes matériels sur des plates-formes sur silicium reconfigurable » localisée à Lannion. Il est animateur du groupe « architecture reconfigurable » du GDR SOC-SIP.

didier.demigny@univ-rennes1.fr