Partitionnement automatique optimisant les ressources FPGA pour l'aide à la conception de SoC reconfigurable

P. BRUNET, C. TANOUGAST, Y. BERVILLER, S. WEBER

Laboratoire d'Instrumentation Électronique de Nancy, Université Henri Poincaré Nancy I, Faculté des Sciences,

BP 239, 54506 Vandoeuvre-lès-Nancy cedex, France

Philippe.brunet@lien.uhp-nancy.fr , Camel.tanougast@lien.uhp-nancy.fr

Yves.berviller@lien.uhp-nancy.fr , Serge.weber@lien.uhp-nancy.fr

Résumé – Nous proposons un outil permettant d'aider les concepteurs lors de la mise en place d'algorithmes utilisant la reconfiguration dynamique. Il permet de trouver la meilleure adéquation entre une application et son implantation sur un SOC reconfigurable. Reposant sur un partitionnement temporel original, il vise différents objectifs : minimisation des ressources logiques et minimisation de la bande passante mémoire.

Abstract *– In the dynamically reconfigurable architectures field, the lack of software support is now well known. To answer this problem, we propose tools that helps designer to implement algorithms using dynamic reconfiguration. This tool allows to find the best adequacy between an application and its implementation on a SoC. Based on an original temporal partitioning methodology, its aims are: minimizing logic resources and minimizing the memory bandwidth.*

1. Introduction

Nous proposons un outil prenant en compte l'aspect reconfiguration dynamique des FPGAs et permettant de spécifier les caractéristiques d'une architecture SoC reconfigurable. Les avantages de la RD ont été plusieurs fois démontrés [1, 2, 3]. L'objectif est de trouver la meilleure utilisation des ressources nécessaire à l'application en mettant en œuvre la Reconfiguration Dynamique (RD). Nous ne cherchons pas ici à faire un choix entre différentes architectures reconfigurable [4, 5] ni à modifier un algorithme pour l'adapter à une cible. Les méthodes permettant de prendre en compte la reconfiguration dynamique sont proposées [6, 7, 8]. Cependant les outils qui permettent d'automatiser ces méthodes ne sont pas nombreux [9]. Notre outil qui repose sur un partitionnement temporel original [10] vise différents objectifs: minimisation des ressources logiques nécessaires à une implantation sous contrainte de temps et minimisation de la bande passante lors de l'utilisation de la RD. En effet, utiliser la RD peut conduire à une augmentation des besoins en bande passante, en consommation d'énergie, en mémoire temporaire. Notre outil appelé DAGARD (Découpage Automatique de Graphes pour Architectures Reconfigurable Dynamiquement) permet de réaliser le découpage d'une application complète représentée sous forme d'un graphe flot de données (GFD) en plusieurs sous applications successivement exécutées par la surface logique visée grâce à la RD. Il s'insère dans le flot de conception d'une application (Fig:1).

La prochaine section présente le fonctionnement de la méthode de partitionnement temporel. La section 3 illustre l'utilisation de l'outil pour un exemple d'implantation.

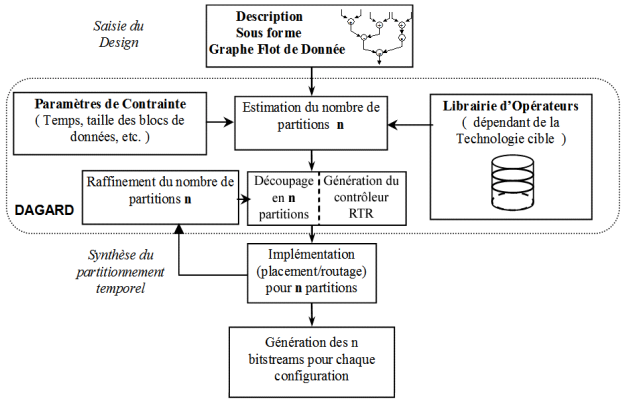


Figure 1 : Intégration de DAGARD dans le flot de conception.

2. Fonctionnement

Le fonctionnement de DAGARD repose sur une méthode de partitionnement [10] qui recherche le découpage d'une application en un maximum de sous applications de tailles homogènes. Il délivre un nombre de partitions **n** réalisable sous une contrainte de temps donnée. Connaissant **n**, deux stratégies sont possibles. La première minimise la surface nécessaire à l'application en créant des partitions homogènes et tente ensuite de minimiser les besoins en bande passante induit par le découpage. La seconde a le fonctionnement inverse : minimisation de la bande passante mémoire, puis, un raffinement tente d'homogénéiser les surfaces des différentes partitions.

2.1 Fonctionnement de base

Le pseudo algorithme ci-dessous présente la structure de l'automatisation du partitionnement.

```
V, N, T = constantes     //contraintes diverses
G <= DFG                 //saisie du GFD
C <= 0                   //Surface totale nécessaire
TO <= 0                  //Temps de l'opérateur le plus lent
POUR chaque noeud Ni de G
        TO <= max(TO,Ni.ti)//temps d'exécution maximum
        C <= C + Ni.Surface //calcul de la surface totale
FIN POUR
n   <= T / [(N.TO) + C/V]        // calcul de n et Cn
Cn <= C / n
```

La vitesse V de reconfiguration du FPGA, la taille N des blocs de données à traiter et le temps total T accordé au traitement permettent de connaître **n** (nombre de reconfigurations que nous somme certains de pouvoir implémenter en satisfaisant les contraintes de temps), ainsi que Cn, la surface FPGA minimale nécessaire à l'application partitionnée en n étapes. Toutes les informations technologiques (le temps d'exécution de chaque opérateur (ti), la surface nécessaire à sont implantation (Ni)) sont issues d'une librairie caractéristique de la technologie FPGA visée par l'application (Altera, Atmel, Xilinx, etc.).

2.2 Optimisation de la surface logique et/ou de la bande passante.

L'accumulation des opérateurs successifs du GFD en partant de son sommet jusqu'à l'obtention d'une surface supérieure ou égale à Cn définit la première partition. Les suivantes sont réalisées de la même façon en réinitialisant le cumul de surface, ceci jusqu'à la fin du GFD. L'accumulation s'effectue de façon à répondre aux dépendances de données du GFD. Un raffinement du découpage permet dans la mesure du possible de diminuer les besoins en bande passante vers l'extérieur du FPGA. La bande passante mémoire nécessaire en sortie d'une partition (écriture) est sensiblement égale à la bande passante mémoire nécessaire en entrée pour la partition suivante (lecture). En fait, seul le changement de fréquence de travail modifie cette bande passante. Le pseudo algorithme de ce raffinement est donné ici. Ce code, par une légère variation du lieu de découpage du graphe, permet de diminuer la bande passante nécessaire tout en conservant une homogénéité des partitions acceptable. L'écart autorisé sur la surface des partitions est contrôlé par α. Nous considérons la fonction FirstLeave() (respectivement PreviousFirstLeave() et NextFirstLeave()) qui prend le GFD en argument et renvoie un noeud (opérateur et ces données propres) respectant la dépendance de donnée. Cette fonction fait appel à la librairie d'opérateurs de la technologie ciblée.

```
Pnode = PreviousFirstLeave(G)
Cnode = FirstLeave(G)
Nnode = NextFirstLeave(G)
SI [Pnode.bit = Min (Pnode.bit, Cnode.bit, Nnode.bit)]
        ET [(C -Pnode.area) > Cn ( 1 – α )]
ALORS
        Supprime (Pi, Pnode)
SI [Nnode.bit = Min (Pnode.bit, Cnode.bit, Nnode.bit)]
        ET [(C + Nnode.area) < Cn (1 + α )]
ALORS
        Ajoute (Pi, Nnode)
```

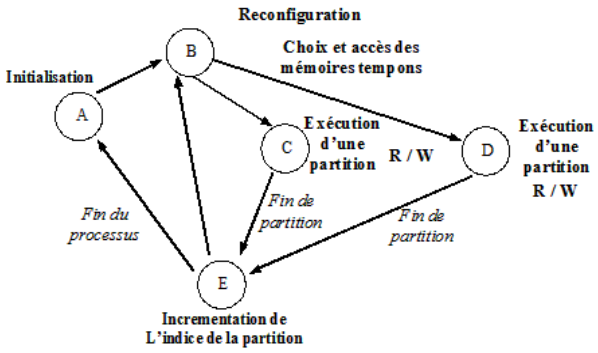
La seconde stratégie proposée est une recherche de tous les niveaux du GFD pour lesquels la bande passante est minimale. Cette information est obtenue lors du parcourt du GFD en accumulant à chaque niveau les informations de taille de données contenues dans chacun des arcs qui relient les opérateurs du graphe. Une fois ces minimums référencés, on peut alors utiliser ceux qui réalisent le découpage le plus homogène (en terme de surface) de l'application.

L'ensemble de ces informations est représenté sur DAGARD. Le pseudo algorithme de création des partitions est présenté ci-dessous.

```
P1, P2, ..., Pn <= partition vide
POUR i DE 1 A n
        C <= 0
        Tant que  C ≤ Cn
                Ajoute(Pi, FirstLeave(G))
                C <= C + FirstLeave(G).Area
                Supprime(G, FirstLeave(G))
        Fin Tantque
FIN POUR
```

2.3 Génération du contrôleur du partitionnement

Connaissant le nombre de partitions **n** final, la génération d'un contrôleur exécutant l'ordonnancement des partitions en RD et la gestion des mémoires (stockage des données intermédiaires) est implicite (Fig :2). Ce contrôleur est implanté avec la première partition de façon statique. Il va alors pouvoir gérer les échanges de données entres les différentes mémoires temporaire. A la fin de l'exécution d'une partition, il sera chargé de reconfigurer la surface logique de façon dynamique et partiel.



Machine d'état du contrôleur

Figure 2 : Machine d'état du contrôleur de reconfiguration.

3. Exemple d'implantation

3.1 Présentation de l'algorithme

Notre domaine de prédilection est le traitement d'image. La régularité et la taille des données à traiter dans ce domaine en fait un champ d'application privilégié pour la reconfiguration dynamique. En effet, des blocs de données important permettent de minimiser le « coût relatif » des temps de reconfiguration par rapport aux temps de traitement. L'algorithme utilisé comme test est un détecteur de contours temps réel. Les contraintes de temps sont donc celle d'une période image standard (40 ms), et les blocs de données correspondent à la taille de l'image (512² pixels x 8 bits).

La plate-forme utilisée pour implanter l'algorithme est ARDOISE [11]. Par conséquent la surface logique disponible est celle d'un FPGA AT40K40 de chez ATMEL (40.000 portes). Pour élaborer le découpage, une librairie spécifique a donc été réalisée, comprenant surface logique, temps de traversé des opérateurs, vitesse de reconfiguration etc. Ces informations sont issues de [12].

Le découpage par DAGARD propose un découpage en « 3,273 » configurations. Cela signifie que nous sommes certains de pouvoir réaliser un découpage en trois étapes. Nous prenons en compte les temps de routage et temps d'exécution « pire cas » dans notre évaluation. Fort de cela, nous avons également demandé une estimation pour un découpage en quatre partitions du même algorithme dont l'implantation à été réalisée par la suite et dont les résultats sont présentés en section suivante. Le découpage proposé en trois étapes a d'abord été réalisé. Il montrait un temps de traitement permettant la réalisation d'une quatrième étape, et donc la possibilité de réduire d'avantage la surface logique nécessaire à la réalisation de cette application.

3.2 Résultats données par DAGARD

DAGARD donne une estimation de surface de 991 cellules au total et la répartition pour un découpage en quatre partitions est présentée par le tableau 1.

TAB 1 : Résultat d'estimation de DAGARD

Partition	Surface	Fréquence
1	220 cells	36,37 MHz
2	241 cells	22,56 MHz
3	243 cells	22,56 MHz
4	277 cells	22,56 MHz

L'estimation du temps de calcul de chaque partition est représenté par le tableau 2.

TAB 2 : Temps d'exécution de chaque partition.

Partition	Reconfiguration	Traitement	Total
1	161 µs	7,2 ms	7,361 ms
2	176 µs	11,62 ms	11,796ms
3	178 µs	11,62 ms	11,798 ms
4	203 µs	11,62 ms	11,865 ms
		Total :	42,82 ms

3.3 Résultats d'implantation

Nous avons réalisé l'implantation de l'algorithme en suivant le découpage recommandé par DAGARD. La surface totale des partitions obtenue est alors de 1008 cellules logiques. La répartition et les fréquences de travail sont présentées par le tableau 3.

TAB 3 : Résultats d'implantation

Partition	Surface	Fréquence
1	225 cells	36,9 MHz
2	241 cells	25,84 MHz
3	248 cells	25,84 MHz
4	294 cells	26,45 MHz

L'estimation du temps de calcul de chaque partition est représenté par le tableau 4.

TAB 4 : Temps d'exécution de chaque partition.

Partition	Reconfiguration	Traitement	Total
1	173 µs	7,1 ms	7,273 ms
2	180 µs	10,15 ms	10,33 ms
3	180 µs	10,15 ms	10,33 ms
4	190 µs	9,91 ms	10,10 ms
		Total :	38,033 ms

3.4 Bilan/discussion

La réalisation montre que les résultats de l'implantation s'écartent peu de l'estimation (voir tableau 5).

TAB 5 : Comparaison Estimation/Réalisation

Partition	Estimation	Implantation	Écart
1	220	225	2,3%
2	241	241	0,0%
3	243	248	2,1%
4	277	294	6,1%
Total :	991	1008	1 ,7%

Néanmoins, il faut ajouter à chacune des partitions la surface logique occupée par le contrôleur. Cette surface n'a pas été prise en compte ici puisqu'elle est fixe et pratiquement indépendante du partitionnement temporel. Dans le cas de notre exemple, le contrôleur se compose essentiellement de deux compteurs 18 bits adressant deux mémoires tampon travaillant en mode « ping-pong » (images de 512² pixels), d'une machine d'état simple et de quelques registres pour contrôler le déroulement de l'application (partition en cours, nombre de partitions pour l'application, mémoire d'entrée et mémoire de sortie actuelle, etc.)

La surface nécessaire à la réalisation d'un contrôleur est de 49 cellules logiques pour la technologie AT40K. Par conséquent, à partir de la vitesse de reconfiguration de cette technologie, on déduit une durée supplémentaire de 35,9 µs lors de la première configuration pour implanter ce contrôleur.

De plus, l'ajout de quelques registres est parfois nécessaire en sortie de l'algorithme pour stabiliser la sortie des données

vers les mémoires tampon, ce qui explique une légère augmentation des ressources nécessaires lors de la réalisation.

4. Conclusions et perspectives

Nous avons réalisé l'automatisation d'une méthode de partitionnement pour la reconfiguration dynamique temps réel sur FPGA. Ce partitionnement automatique permet une utilisation de la surface logique du FPGA/SoC beaucoup plus intense (à fréquence plus élevée) et donc une réduction des besoins en surface logique par rapport à une application statique. Ce mode de fonctionnement se révèle très utile pour des applications sur systèmes embarqués où la surface logique peut s'avérer un élément critique. En effet, notre outil fournit les caractéristiques matérielles nécessaires à l'implantation optimale d'une application (taille de la matrice FPGA, besoins en mémoire temporaire, bande passante mémoire). Nous avons validé notre outil avec les différentes approches décrites. Dans le cas où le but est d'exploiter au maximum les ressources disponibles d'une plateforme, l'outil a été validé sur plusieurs algorithmes de traitement d'images ayant pour cible technologique l'architecture Ardoise [11]. Il nous reste maintenant à augmenter le champ d'action de DAGARD, notamment en prenant en compte le critère de consommation électrique [13, 14]. Actuellement, nous réalisons la génération automatique du contrôleur de reconfigurations et du code VHDL de chaque partition. Les opérateurs sont instanciés sous forme de composants génériques. Le code VHDL des opérateurs fait partie intégrante de la librairie utilisée pour le GFD. Les signaux intermédiaires sont également créés et reliés. Pratiquement, seules les entités sont à compléter, ainsi que les signaux associés. A terme, cette génération automatique devrait parvenir à réaliser plus de 95% du code VHDL total de l'application, ce qui accélérera fortement son développement. Néanmoins, l'exemple présenté dans cet article n'est pas issu de cette génération automatique de code VHDL. L'évaluation des performances du code généré par DAGARD pour l'implantation d'un démonstrateur fait évidemment partie des projets à court terme.

Références

[1] M. J. Wirthlin and B.L. Hutchings, "Improving functional density using run-time circuit reconfiguration," *IEEE trans. VLSI Syst.*, vol. 6, no. 2, pp. 247-256, June 1998.

[2] S. A. Guccione and D. Levi. *The advantages of run-time reconfiguration.* Reconfigurable Technology : FPGAs for Computing and Applications, SPIE - The International Society for Optical Engineering, (Proc. SPIE 3844) :87–92, September 1999.

[3] D. Demigny, L. Kessal, R. Bourguiba and N. Boudouani, *How to use high speed reconfigurable fpga for real time image processing ?,* In Proc. IEEE Conf. on Computer Architecture for Machine Perception, IEEE Circuit and Systems, Padova, september 2000, pp. 240-246.

[4] Dehon, "Comparing computing machines," *In Proc of the SPIE vol 3526*, 1998, pp. 124-133.

[5] P. Benoit, G. Sassatelli, L. Torres, T. Gil, G. Cambon, M. Robert, *Caractérisation et Comparaison d'Architectures Reconfigurables Dynamiquement, Un Exemple : Le Systolic Ring*, JFAAA'02: Journées Francophones sur l'Adéquation Algorithme Architecture 2002, pp. 30-34, Monastir (Tunisie), 16-18 décembre 2002

[6] M. Kaul and R. Vemuri, *Optimal temporal partitioning and synthesis for reconfigurable architectures*, Int. Symposium on Field-Programmable Custom Computing Machines, April 1998, pages 312-313.

[7] W. Luk, N. Shirazi and P.Y.K. Cheung, *Modeling and Optimizing Run-Time Reconfiguration Systems,* Proc. IEEE Symposium on FPGAs for Custom Computing Machines, K.L. Pocek and J. Arnold (editors), IEEE Computer Society Press, 1996, pp. 167-176.

[8] L. Bossuet, G. Gogniat, J.P. Diguet and J.L. Philippe. *A Modeling Method for Reconfigurable Architectures*, http://lester.univ-ubs.fr:8080

[9] M. Valsiko, "*DYNASTY : A temporal floorplanning based CAD framework for dynamically reconfigurable logic systems*," Lecture notes in computer science, vol 1673, pp. 124-133

[10] C. Tanougast, Y. Berviller, S. Weber, and P. Brunet. *A partitioning methodology that optimises the area on reconfigurable real-time embedded systems*. EURASIP Journal on Applied Signal Processing, Special Issue on Rapid Prototyping of DSP Systems, A paraître 2003.

[11] D. Demigny, M. Paindavoine and S. Weber, *Architecture à reconfiguration dynamique-Application au traitement temps réel des images,* TSI Vol 18 n°10/1999, pages 1087 à 1112.

[12] Atmel AT40k datasheet.

[13] N. Togawa, M. Ienaga, M. Yanagisawa and T. Ohtsuki, "An area/time optimizing algorithm in high-level synthesis for control-base hardwares," *in Proc. Of the ASP-DAC 2000*, IEEE circuits and systems, Yokohama, Japan, 2000, pp. 309-312.

[14] K. K. W. Poon, A. Yan and S. J. E. Wilton, "A flexible power model for FPGA," *Lecture notes in computer science,* vol 2438, pp. 312-321, M Glesner, P. Zipf, M. Renovell, Eds., Montpellier, France, 2002.