VERS LA SYNTHÈSE AUTOMATIQUE DE CIRCUITS À PARTIR DE GRAPHES ALGORITHMIQUES FACTORISÉS

Ailton F. DIAS[1,2], Mohamed AKIL[2], Christophe LAVARENNE[3], Yves SOREL[3]

[1] *CNEN/CDTN–Divisão de Computação e Informação, CP 941 - 30123-970 Belo Horizonte, MG, Brésil*
E-mail: diasaf@urano.cdtn.br
[2] *Groupe ESIEE–Laboratoire A2SI, BP 99 - 93162 Noisy-le-Grand, France*
E-mail: diasa@esiee.fr, akilm@esiee.fr
[3] *INRIA Rocquencourt–Projet SOSSO, BP 105 - 78153 Le Chesnay Cedex, France*
E-mail: christophe.lavarenne@inria.fr, yves.sorel@inria.fr

5ème Workshop AAA sur l'Adéquation Algorithme Architecture– Rocquencourt, Jan. 2000

RESUME. On présente dans cet article les principes permettant de synthétiser le circuit correspondant à une spécification algorithmique faite à l'aide d'un modèle de graphe de dépendances factorisé. Ce dernier est transformé en un graphe matériel comprenant les chemins de données et de contrôle, en suivant des règles simples de synthèse basées sur le modèle RTL et des mécanismes de transferts de données synchronisés. Les règles définies ici serviront par la suite de base pour réaliser automatiquement la synthèse de circuits.

ABSTRACT. *In this paper we present the procedures to synthesize the circuit corresponding to an algorithmic specification following a model of factorized dependence graph. This algorithmic graph is transformed in a hardware graph containing data and control paths. We followed simple rules based on the RTL model and on mechanisms of synchronized data transfers. The rules defined here will be used in the future to allow the automatic synthesis of circuits.*

1 Introduction

Dans l'article [1], on présentait un modèle unifié pour la conception conjointe logiciel-matériel, basé sur la méthodologie AAA (Adéquation Algorithme Architecture) [2] définie dans le projet SOSSO de l'INRIA. Ce modèle permet, à partir d'une spécification algorithmique faite à l'aide d'un modèle de graphe factorisé de dépendances de données entre opérations, de générer soit des exécutifs pour la partie programmable (réseau de processeurs) de l'architecture, soit du VHDL structurel synthétisable pour la partie non programmable (réseau de circuits spécifiques). Dans l'article [3], on s'intéressait uniquement au deuxième aspect concernant la synthèse de circuits. On y présentait les principes permettant de passer d'une spécification algorithmique, faite à l'aide d'un modèle de graphes factorisés de dépendances de données entre opérations, à l'implantation de cet algorithme sous la forme d'un circuit représenté par un graphe matériel d'opérateurs. En particulier, on définissait les sommets de factorisation utilisés pour spécifier le contrôle qui actuellement est uniquement du type tableau d'opérations (pas de prise en compte ici du conditionnement). On y présentait aussi les principes permettant d'optimiser par défactorisation cette implantation pour respecter les contraintes temps-réel tout en minimisant la consommation de ressources matérielles.

Dans le présent article, nous donnons des règles simples permettant de synthétiser les chemins de données et de contrôle du circuit correspondant à l'algorithme spécifié à l'aide d'un modèle de graphe de dépendances factorisé. Il est habituellement admis que la synthèse du chemin de contrôle est la plus difficile à réaliser [4]. Nous montrons qu'il est possible de synthétiser de manière simple et systématique le chemin de contrôle à l'aide d'une technique de synchronisation de transferts de données entre registres. Cette approche devrait permettre par la suite de réaliser simplement un générateur automatique de VHDL structurel synthétisable, réalisant ainsi la synthèse automatique de circuits.

2 Spécification algorithmique

La spécification algorithmique est le point de départ du processus d'implantation matérielle d'une application sur une architecture. Dans cette spécification, les opérations n'ont pour relations d'ordre que celles impliquées par leurs dépendances de données, établissant un ordre partiel qui met en évidence un parallélisme potentiel. Pour représenter cette spécification, nous avons choisi un modèle de graphes factorisés.

2.1 Modèle de graphes factorisés

Comme décrit en [1], la spécification de l'algorithme d'une application sous la forme d'un graphe de dépendances comprend des parties régulières (motifs répéti-

tifs périodiques) et non régulières. Cette spécification doit être indépendante de toutes contraintes liées à l'implantation matérielle et nécessite la mise en œuvre d'un processus de décomposition de l'algorithme en opérations implantables. Cependant, ce processus génère souvent des répétitions de motifs d'opérations (opérations identiques mais opérant sur des données différentes). Pour réduire la taille de la spécification et mettre en évidence ses parties régulières, on applique un processus de factorisation, lequel fait apparaître des sommets spéciaux (sommets frontières de factorisation), F (partition d'un tableau en autant d'éléments que de répétitions du motif), J (composition d'un tableau à partir des résultats de chaque répétition du motif), D (diffusion d'une donnée à toutes les répétitions du motif) et I (dépendance de donnée inter-itération du motif), qui spécifient chacun l'une des différentes manières de factoriser les données et les opérations en traversant une frontière de factorisation. Une opération, située du côté d'une frontière qui présente un groupe d'opérations identiques opérant sur un groupe factorisé de données différentes, sera réalisée au moyen d'un seul opérateur (ou de plusieurs opérateurs en parallèle) utilisé(s) itérativement autant de fois qu'il existe d'opérations dans le groupe factorisé. Ainsi, l'intérêt de la factorisation ne se restreint pas uniquement à réduire la taille de la spécification algorithmique, sans en modifier la sémantique opératoire, elle permet aussi de décrire en intention plusieurs implantations plus ou moins séquentielles ou parallèles, chacune avec des caractéristiques (surface, temps de réponse) différentes.

2.2 Relations entre frontières de factorisation

Une frontière de factorisation sépare deux zones, l'une "rapide" étant répétée par rapport à l'autre "lente". D'un point de vue comportemental ou opératoire, une frontière peut être consommatrice (située en aval) ou/et productrice (située en amont) par rapport à une autre frontière, en fonction des dépendances de données entre elles. Deux frontières sont voisines s'il existe entre elles au moins une relation de dépendance de donnée directe qui ne passe pas par l'intermédiaire d'une troisième frontière.

Les relations de voisinage entre les frontières de factorisation d'un graphe algorithmique peuvent être représentées sous la forme d'un graphe de voisinage. Les sommets de ce graphe représentent les frontières de factorisation et ils sont sous-divisés en quatre régions, comme le montre la Fig. 1 :

- lent-amont : côté "lent" de FF, consommatrice ;
- rapide-aval : côté "rapide" de FF, productrice ;
- rapide-amont : côté "rapide" de FF, consommatrice ;
- lent-aval : côté "lent" de FF, productrice.

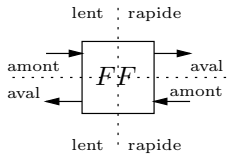

Figure 1: Sommet d'un graphe de voisinage représentant la frontière de factorisation FF

Les arcs orientés entrant et sortant des sommets représentent les transferts de données et les éventuelles opérations (par exemple, de calcul, *Explode*, *Implode*, etc.) situées entre les frontières. L'orientation de l'arc indique les relations de production-consommation : l'arc part d'un producteur vers un consommateur. Ce graphe de relations de voisinage, qui peut être obtenu automatiquement à partir d'un graphe algorithmique, nous permettra d'établir les relations de contrôle entre les frontières, comme nous le verrons par la suite.

2.3 Exemple : Spécification du PMV (Produit Matrice-Vecteur)

Le produit d'une matrice $A \in \mathbb{R}^m \times \mathbb{R}^n$ par un vecteur $B \in \mathbb{R}^n$ donne un vecteur $C \in \mathbb{R}^m$, et peut s'écrire, sous forme factorisée :

$$C = \left[\sum_{j=1}^{n} a_{ij} b_j \right]_{i=1}^{m} \quad (1)$$

où
m : nombre de lignes de la matrice A,
n : nombre de colonnes de A, dimension de B,
a_{ij} : ijème élément de la matrice A,
b_j : $j$ème élément du vecteur B.

On part de l'éq. 1, qui décrit le PMV sous forme factorisée, pour obtenir le graphe correspondant à la spécification algorithmique factorisée du PMV (Fig. 2). L'interface avec l'environnement est délimitée par les entrées (F_A^∞ et F_B^∞) et par la sortie (J_C^∞), correspondant à une frontière de factorisation de motifs de graphes répétitifs infinis (FF_1). Le crochet $[\ ]_{i=1}^{m}$ correspond à une deuxième frontière (FF_2), délimitée par des sommets de factorisation de motifs de graphes répétitifs finis, effectuant la sélection des m lignes de la matrice A (F_{21}), la diffusion du vecteur B (D_{21}) et la collecte du vecteur-résultat C (J_{21}). Le foncteur $\sum_{j=1}^{n}$ correspond à une troisième frontière (FF_3), délimitée aussi par des sommets de factorisation de motifs répétitifs finis, effectuant la sélection des éléments a_{ij} de la $i$ème ligne de la matrice A (F_{31}) et des éléments b_j du vecteur B (F_{32}) et fournissant le résultat de la somme des produits entre a_{ij} et b_j pour chaque ligne de la matrice A (I_{31}). Les côtés "lent"

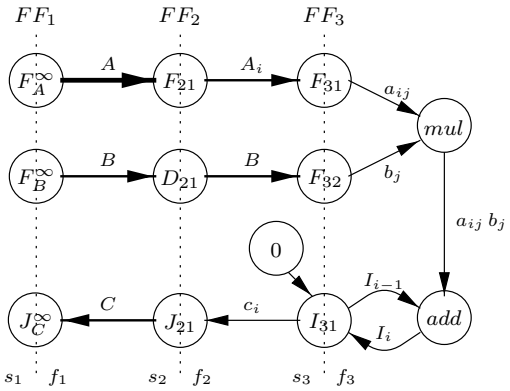

Figure 2: Spécification algorithmique factorisée du PMV

et rapide de chaque frontière sont etiquetés respectivement "s" (*slow*) et "f" (*fast*).

Le graphe de relations entre frontières obtenu à partir du graphe algorithmique du PMV factorisé est montré par la Fig. 3. La frontière FF_1 est une frontière infinie. Ainsi, elle n'a pas de voisines de son côté "lent" (car celui-ci correspond à l'environnement). FF_1 est à la fois productrice (arcs A et B) et consommatrice (arc C) par rapport à FF_2. FF_2 est à son tour aussi productrice (arcs A_i et B) et consommatrice (arc c_i) par rapport à FF_3. FF_3 est productrice et consommatrice par rapport à elle-même à travers les opérations de calcul *mul* et *add*.

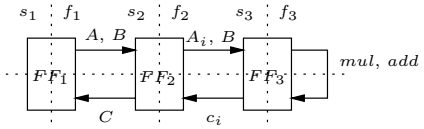

Figure 3: Relations entre frontières du PMV factorisé

Une spécification factorisée peut avoir plusieurs défactorisations : pour n frontières de factorisation, il y a au minimum 2^n implantations défactorisées de la spécification. De plus, chaque frontière peut n'être défactorisée que partiellement : une factorisation de r répétitions d'un motif peut se décomposer en f factorisations de r/f répétitions du motif [1]. En partant de l'éq. 1, la défactorisation partielle en deux factorisations de trois répétitions de la frontière FF_2 de la spécification factorisée du PMV d'une matrice 6×6 par un vecteur de 6 éléments peut s'écrire :

$$C = \begin{bmatrix} \left[\sum_{j=1}^{6} a_{ij} b_j \right]_{i=1}^{3} \\ \left[\sum_{j=1}^{6} a_{ij} b_j \right]_{i=4}^{6} \end{bmatrix} \qquad (2)$$

À partir de l'éq. 2, qui décrit le PMV partiellement défactorisé par sa frontière correspondant au crochet $[\ ]_{i=1}^{m}$ (FF_2) dans l'éq. 1, on obtient le graphe algorithmique montré par la Fig. 4. L'interface avec l'environnement (FF_1) n'a pas été modifiée, mais la frontière FF_2 a été divisée en deux sous-ensembles (FF_{2a} et FF_{2b}) à cause de sa défactorisation partielle. La frontière FF_3 n'a pas été modifiée mais elle apparaît deux fois (FF_{3a} et FF_{3b}) à cause de la défactorisation partielle de FF_2.

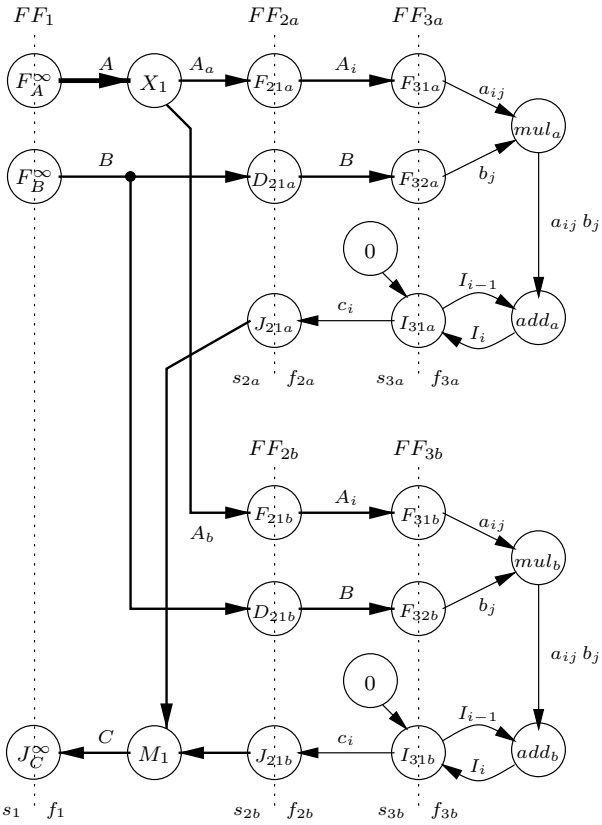

Figure 4: Spécification algorithmique partiellement défactorisée du PMV

Le graphe de relations entre frontières obtenu à partir du graphe algorithmique du PMV partiellement défactorisé est montré par la Fig. 5. La frontière FF_1 est une frontière infinie, n'ayant pas de voisines de son côté "lent". FF_1 est à la fois productrice (arcs A et B) et consommatrice (arc C) par rapport aux frontières FF_{2a} et FF_{2b}. FF_{2a} (FF_{2b}) est à son tour aussi productrice (arcs A_i et B) et consommatrice (arc c_i) par rapport à FF_{3a} (FF_{3b}). FF_{3a} (FF_{3b}) est productrice et consommatrice par rapport à elle-même à travers les opérations de calcul mul_a et add_a (mul_b et add_b).

3 Synthèse de circuits

La synthèse automatique de circuits consiste à construire automatiquement une structure physique à partir d'une représentation comportementale de plus haut niveau [5] [6], qui est pour nous le graphe de dépendances factorisé. La synthèse réduit consid-

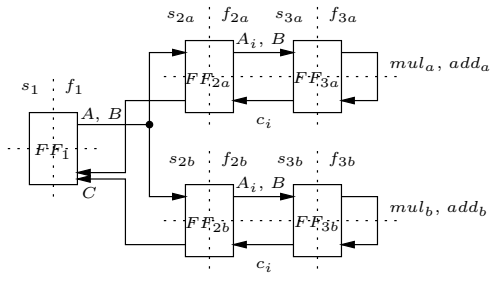

Figure 5: Relations entre frontières du PMV partiellement défactorisé

érablement le cycle de développement d'un circuit et permet d'exploiter différentes implantations matérielles pour obtenir un compromis surface/performance temporelle idéal pour l'application.

3.1 Règles de synthèse du chemin de données

L'implantation matérielle des opérations consiste à faire correspondre un opérateur à chaque sommet du graphe factorisé d'opérations (il s'agit du graphe algorithmique transformé, obtenu après l'optimisation par défactorisation telle que décrite dans [3]). Cet opérateur est combinatoire pour un sommet opération, ou est composé d'un multiplexeur et/ou de registres pour un sommet frontière (cf. [3] pour une description détaillée des opérateurs frontières F, J, D et I). L'implantation matérielle des dépendances de données entre opérations consiste à faire correspondre à chaque arc du graphe une connexion physique entre les opérateurs correspondant aux opérations. Les opérateurs et leurs interconnexions constituent le chemin de données du circuit.

3.2 Règles de synthèse du chemin de contrôle

Le chemin de contrôle correspond à ce qu'il faut ajouter au chemin de données pour contrôler les multiplexeurs et les transitions des registres des opérateurs frontières. Il est obtenu en synchronisant les transferts entre registres. Dans le cas d'un circuit composé par exemple d'un registre d'entrée Reg_E, d'un opérateur combinatoire pur X et d'un registre de sortie Reg_S, comme le montre la Fig. 6, les registres Reg_E et Reg_S sont pilotés par un signal d'horloge clk et ils transitent à chaque front montant du signal clk. Au premier front montant clk, les données T_E seront stockées par Reg_E. Ces données vont se propager à travers l'opérateur combinatoire et le résultat sera stocké par Reg_S lors du prochain front montant de clk. Ainsi, la période de l'horloge doit être supérieure à la latence de l'opérateur combinatoire X, pour que le circuit produise des résultats corrects. Si le circuit X comporte de parties qui se décomposent en sous parties identiques, répétées régulièrement, ces dernières peuvent être factorisées en une seule sous-partie, entourée d'opérateurs frontière composés de multiplexeurs et de registres, contrôlant l'utilisation itérative de la sous-partie. Ces nouveaux registres doivent donc transiter plusieurs fois à chaque transition des registres Reg_E et Reg_S, lesquels ne devront donc plus transiter à chaque cycle d'horloge, mais seulement tous les d cycles d'horloge si la sous-partie est itérée d fois. Pour qu'un registre puisse transiter, deux conditions sont nécessaires : les nouvelles données en amont du registre doivent être stables, et les consommateurs en aval du registre doivent avoir fini de consommer les données précédentes [7].

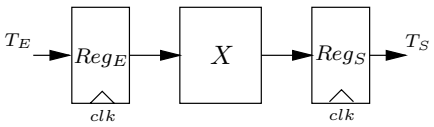

Figure 6: Transferts de données entre registres

Si les données en amont d'un circuit viennent de différentes sources avec des durées de propagation différentes, il est nécessaire d'avoir un circuit synchronisé. La synchronisation d'un circuit X est possible à travers l'utilisation d'un protocole de communication de type requête/acquittement [7], comme le montre la Fig. 7. Les producteurs des données en amont de X (Td_1 et Td_2 sur la Fig. 7, le suffixe d signifie aval - *downstream*) indiquent la disponibilité de ces données en provoquant une transition des signaux de requête (rd_1 et rd_2). Le circuit X utilise ces données seulement quand son entrée de requête (ru) est active. Le circuit X indique à ses producteurs en amont qu'il a fini de consommer leurs données en activant sa sortie d'acquittement (au, le suffixe u signifie amont - *upstream*). Il en va de même symétriquement du côté aval de X. Dans le cas où X est un circuit acyclique purement combinatoire, comme c'est le cas pour toutes les opérations de l'algorithme (autres que les sommets de factorisation), les entrées de requête ru et d'acquittement ad sont respectivement connectées aux sorties de requête rd et d'acquittement au. On notera que la composition, par interconnexion acyclique, de plusieurs opérateurs combinatoires synchronisés est aussi un circuit combinatoire synchronisé [8].

3.2.1 Unité de contrôle

Dans le cas où X est un circuit séquentiel, c'est-à-dire comprenant des registres, comme c'est le cas pour un circuit comprenant des parties factorisées et donc des opérateurs de factorisation, chaque frontière de factorisation a des relations amont et aval des deux côtés, lent et rapide. Les relations entre les signaux de requête et d'aquittement amont et aval des deux côtés, constituent "l'unité de contrôle" de la frontière de factorisation (Fig. 8). Cette unité de contrôle

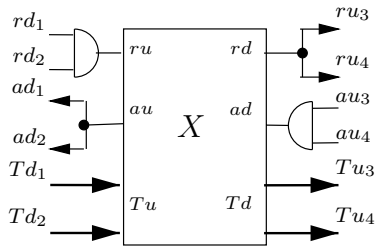

Figure 7: Circuit synchronisé

est composée d'un compteur C à d états et d'une logique supplémentaire afin de générer le protocole de communication entre frontières (signaux de requête et d'acquittement lent et rapide, en amont et en aval), la valeur du compteur (*cpt*) et le signal de validation (*en*) qui commandent les opérateurs frontières. Le signal *init* remet le compteur à l'état 0. Le signal fin indique que le compteur est à son dernier état $(d-1)$. La valeur du compteur *cpt* commande les multiplexeurs des opérateurs F, J et I de la frontière. Le signal de validation *en* détermine les cycles d'horloges où les registres des opérateurs de la frontière (F^∞, J^∞, J et I) devront transiter. Si le signal *en*, qui transite comme les autres signaux pendant un cycle d'horloge, est vrai au moment du front montant de l'horloge, alors le registre transite aussi, sinon la sortie du registre n'est pas affectée par son entrée. Au niveau de l'interface avec l'environnement (capteurs et actionneurs), c'est l'activation du signal de validation associé aux registres internes des opérateurs F^∞ et J^∞ qui permettra de lire des nouvelles données en entrée et d'écrire des résultats en sortie.

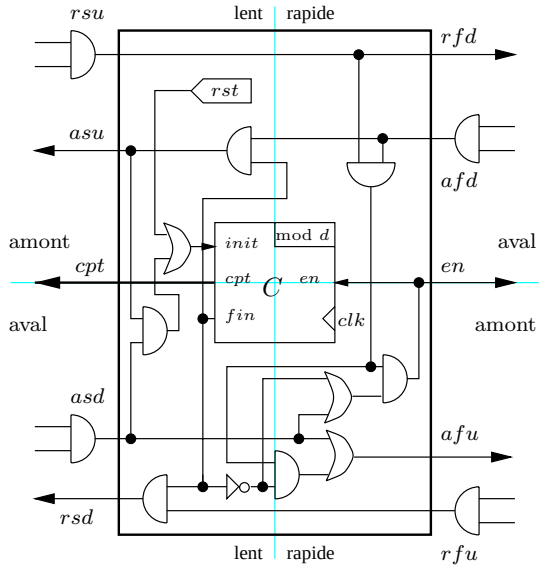

Figure 8: Unité de contrôle

Ces signaux de gestion correspondent aux signaux de requête et d'acquittement générés par les frontières en amont ou diffusés aux frontières en aval et ils sont séparés en deux groupes : ceux qui concernent les frontières situées du côté "lent" et ceux qui concernent les frontières situées du côté "rapide", correspondant aux quatre régions de l'unité de contrôle : lent-amont, lent-aval, rapide-amont, rapide-aval.

Pour simplifier la description des relations entre les signaux de gestion des frontières, on propose la notation suivante :

?/! : entrée/sortie,
s/f : *slow*/*fast* (lent/rapide),
u/d : *upstream*/*downstream* (amont/aval),
r/a : requête/acquittement,
p/c : productrice/consommatrice,
&/|/¬ : AND/OR/NOT.

- rsu : signal de *requête-lent-en-amont* reçu des frontières productrices situées du côté "lent" de FF quand tous les résultats sont stables en leurs sorties. rsu est la conjonction de tous les signaux de requête en aval des frontières productrices situées du côté "lent" de FF :

$$?\,rsu = !\,rsd_{pi}\ \&\ \ldots\ \&\ !\,rfd_{pj}\ \&\ \ldots \qquad (3)$$

où, *pi* et *pj* correspondent aux indices des frontières productrices situées du côté "lent" de FF.

- rfu : signal de *requête-rapide-en-amont* reçu des frontières productrices situées du côté "rapide" de FF quand tous les résultats sont stables en leurs sorties. rfu est la conjonction de tous les signaux de requête en aval des frontières productrices situées du côté "rapide" de FF :

$$?\,rfu = !\,rsd_{pi}\ \&\ \ldots\ \&\ !\,rfd_{pj}\ \&\ \ldots \qquad (4)$$

où, *pi* et *pj* correspondent aux indices des frontières productrices situées du côté "rapide" de FF.

- asd : signal d'*acquittement-lent-en-aval* reçu des frontières consommatrices situées du côté "lent" de FF quand ces frontières ont fini d'utiliser les données en leurs entrées. asd est la conjonction de tous les signaux d'acquittement en amont des frontières consommatrices situées du côté "lent" de FF :

$$?\,asd = !\,asu_{ci}\ \&\ \ldots\ \&\ !\,afu_{cj}\ \&\ \ldots \qquad (5)$$

où, *ci* et *cj* correspondent aux indices des frontières consommatrices situées du côté "lent" de FF.

- afd : signal d'*acquittement-rapide-en-aval* reçu des frontières consommatrices situées du côté "rapide" de FF quand ces frontières ont fini d'utiliser les données en leurs entrées. afd est la conjonction de tous les signaux d'acquittement en amont des frontières consommatrices situées du côté "rapide" de FF :

$$?\,afd = !\,asu_{ci}\ \&\ \ldots\ \&\ !\,afu_{cj}\ \&\ \ldots \qquad (6)$$

où, ci et cj correspondent aux indices des frontières consommatrices situées du côté "rapide" de FF.

- rfd : signal de *requête-rapide-en-aval* diffusé aux frontières consommatrices situées du côté "rapide" de FF quand les données sont stables dans les entrées du côté "lent" de FF :

$$!\,rfd = ?\,rsu \qquad (7)$$

- rsd : signal de *requête-lent-en-aval* diffusé aux frontières consommatrices situées du côté "lent" de FF quand les résultats sont stables en sortie des frontières productrices situées du côté "rapide" de FF et la frontière FF a fini de consommer les données en ses entrées :

$$!\,rsd = ?\,rfu\ \&\ fin \qquad (8)$$

- afu : signal d'*acquittement-rapide-en-amont* diffusé aux frontières productrices situées du côté "rapide" de FF quand les frontières consommatrices du côté "lent" de FF ont fini d'utiliser les données en leurs entrées ou, quand les données sont stables en sortie des frontières productrices situées du côté "lent" de FF et les frontières consommatrices situées du côté "rapide" de FF ont fini d'utiliser les données en leurs entrées et la frontière FF n'a pas fini de consommer les données en ses entrées :

$$!\,afu = ?\,asd \mid (?\,rsu\ \&\ ?\,afd\ \&\ \neg fin) \qquad (9)$$

- asu : signal d'*acquittement-lent-en-amont* diffusé aux frontières productrices situées du côté "lent" de FF quand les frontières consommatrices situées du côté "rapide" de FF ont fini d'utiliser les données en leurs entrées et la frontière FF a fini de consommer les données en ses entrées :

$$!\,asu = ?\,afd\ \&\ fin \qquad (10)$$

- en : signal de *validation* du compteur, utilisé pour autoriser les changements d'état des registres des opérateurs J et I appartenant à la frontière FF, est actif quand les frontières consommatrices situées du côté "rapide" de FF ont fini d'utiliser les données en leurs entrées, les résultats sont stables en sortie des frontières productrices situées du côté "lent" de FF et, soit les frontières consommatrices situées du côté "lent" de FF ont fini d'utiliser les données en leurs entrées, soit la frontière FF n'a pas fini de consommer les données en ses entrées :

$$!\,en = ?\,afd\ \&\ ?\,rsu\ \&\ (\neg fin \mid ?\,asd) \qquad (11)$$

- $init$: signal d'*initialisation* du compteur, il est actif soit quand le signal de rémise à zéro global (rst) est actif, soit quand les frontières consommatrices situées du côté "rapide" de FF, les frontières consommatrices situées du côté "lent" de FF et la frontière FF ont fini de consommer les données en leurs entrées :

$$init = ?\,rst \mid (?\,afd\ \&\ ?\,asd\ \&\ fin) \qquad (12)$$

Dans le cas des frontières dites "infinies" représentant l'interface avec l'environnement, l'unité de contrôle devient un sous-ensemble de celle montrée Fig. 8. Cela parce qu'il n'y a évidemment pas de compteurs associés à ces frontières. La suppression du compteur implique la suppression des signaux d'initialisation ($init$), d'horloge (clk) et de remise à zéro (rst). La sortie valeur de comptage (cpt) a été également supprimée. Cette unité de contrôle simplifiée est représentée Fig. 9.

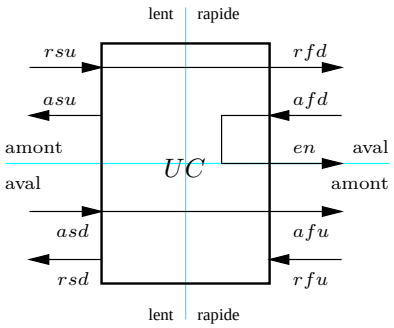

Figure 9: Unité de contrôle "infinie"

Les entrées rsu et asd sont toujours actives ($rsu = 1$ et $asd = 1$) parce que les données sont produites et consommées en continu par l'environnement. Les sorties asu et rsd ne sont jamais connectées parce qu'il n'y a pas de frontiéres du côté "lent" d'une frontière"infini".

3.2.2 Interconnexion des unités de contrôle

Les unités de contrôle peuvent être interconnectées de façon automatique à partir du graphe de relations de voisinage entre les frontières correspondant

à l'application. Dans ce graphe, les sommets correspondent aux unités de contrôle et les arcs correspondent aux signaux de requête transmis entre les unités de contrôle. Les signaux d'acquittement associés aux signaux de requête sont transmis, en sens inverse des signaux de requête, entre les mêmes unités de contrôle. Quand plusieurs signaux arrivent à une même entrée d'une unité de contrôle, on en prend la conjonction par une porte logique ET. Dans la section 3.3, nous verrons deux exemples de synthèse des chemins de données et de contrôle.

3.3 Exemple : Synthèse de l'implantation du PMV

La Fig. 10 représente l'implantation matérielle du PMV factorisé correspondant à la spécification algorithmique représentée Fig. 2. Le chemin de données est constitué des opérateurs frontière de factorisation (F, D, J et I) et des opérateurs combinatoires délimités par les boîtes en pointillés : la boîte à droite de FF_3 est composée des opérateurs combinatoires *mul* et *add*, les autres boîtes sont composées trivialement des interconnexions entre les opérateurs frontière de factorisation. Le chemin de contrôle est constitué par les unités de contrôle UC_1, UC_2 et UC_3, et par leurs signaux de contrôle r, a, cpt, en. L'interconnexion des signaux de requête et d'acquittement est effectuée à travers l'analyse des relations de voisinage entre les frontières :

1. le point de départ est le graphe de relations de voisinage (Fig. 3) ;
2. pour la frontière FF_2, par exemple, on obtient :
   - productrice côté "lent" : FF_1
   $$rsu_2 = rfd_1$$
   - consommatrice côté "lent" : FF_1
   $$asd_2 = afu_1$$
   - productrice côté "rapide" : FF_3
   $$rfu_2 = rsd_3$$
   - consommatrice côté "rapide" : FF_3
   $$afd_2 = asu_3$$
3. en généralisant le principe ci-dessus, on obtient la Fig. 10.

La Fig. 11 représente l'implantation du PMV partiellement défactorisé correspondant à la spécification représentée Fig. 4. Le chemin de données est constitué des opérateurs frontières de factorisation et des opérateurs combinatoires, y compris les

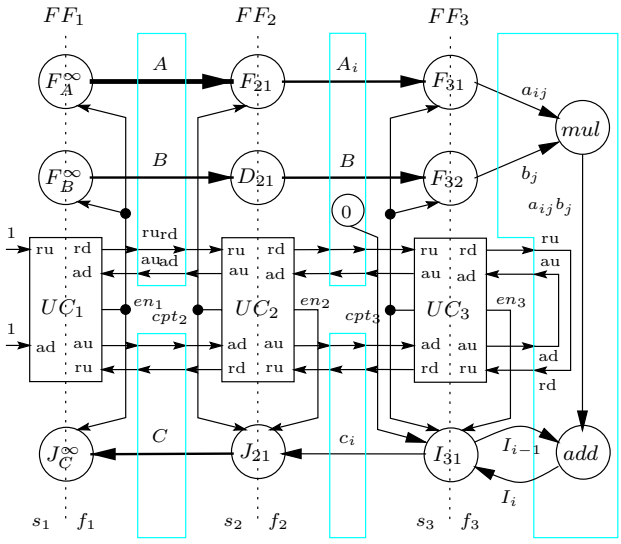

Figure 10: Implantation matérielle du PMV factorisé

opérateurs X_1 et M_1 qui implantent la défactorisation partielle des données. Le chemin de contrôle est constitué par les unités de contrôle UC_1, UC_{2a}, UC_{2b}, UC_{3a} et UC_{3b}, et par les signaux de contrôle. La synchronisation des frontières FF_{2a} et FF_{2b} est assurée par les portes logiques ET en aval de UC_1.

Le Tab. 1 présente les résultats des implantations matérielles factorisée et partiellement défactorisée (FF_2 défactorisée par un facteur $k = 2$) sur un FPGA *Xilinx* XC4052XL du PMV entre une matrice 6×6 et un vecteur de 6 éléments codés sur 3 bits, en utilisant les outils de CAO ci-dessous :

- compilateur VHDL : *QuickVHDL qvcom v8.4-4.3e HP-UX A.09.05*, développé par *Mentor Graphics Co.* ;
- simulateur VHDL : *QuickVHDL qvsim v8.4-4.3e HP-UX A.09.05*, développé par *Mentor Graphics Co.* ;
- synthétiseur VHDL : *Leonardo Spectrum Level 2, v. 1998.2e*, développé par *Exemplar Logic, Inc.*

Table 1: Implantations matérielles du PMV

Implantation	*Surf.* (*CLB*)	*Nb. cycl.*	*Freq.* (*MHz*)	*Lat.* (*ns*)
Factorisée	66	36	20	1728
Défac. part. de FF_2	104	18	16	1116

4 Conclusion et perspectives

Nous avons montré ici qu'à partir d'une spécification algorithmique basée sur un modèle de graphe de

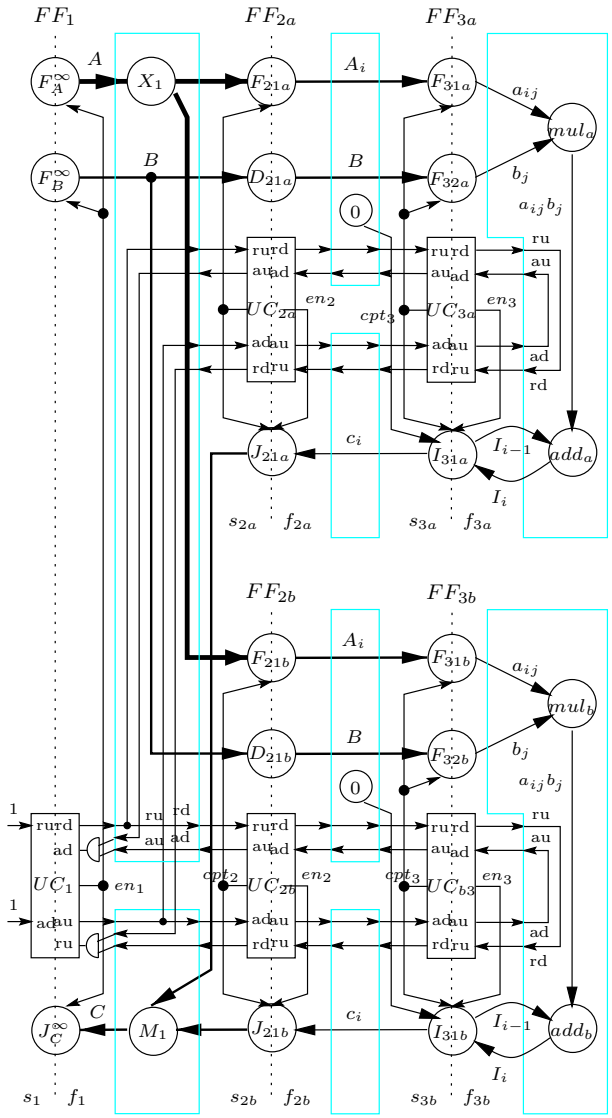

Figure 11: Implantation matérielle du PMV défactorisé partiellement

dépendances factorisé on peut obtenir, en utilisant des règles simples de synthèse des chemins de données et de contrôle, une implantation matérielle sous la forme d'un circuit.

La logique de contrôle "délocalisée" présentée dans cet article permet aux outils de CAO utilisés pour la synthèse de placer les unités de contrôle plus proches des opérateurs à contrôler qu'une logique de contrôle "centralisée". Dans l'exemple de la Fig. 11, le contrôle, en utilisant notre stratégie "délocalisée" correspond à deux compteurs de trois bits et une vingtaine de portes logiques (*ET*, *OU* et *NON*). En revanche, l'implantation de ce contrôle en utilisant une stratégie "centralisée" exigerait une machine à état de 36 états, dont la surface est plus importante que la solution préconisée.

Ce processus de génération d'implantation matérielle pourra être automatisé pour générer un code VHDL structurel synthétisable pour les outils de CAO. Dans un premier temps, nous envisageons de développer ce processus de génération pour une technologie FPGA (*Xilinx*) et des architectures mono-circuit. On envisage aussi de prendre en compte dans la spécification algorithmique, en plus du contrôle des tableaux d'opération, le contrôle du conditionnement et d'en faire aussi la synthèse automatique.

5 Remerciements

Cette étude a été soutenue par la *Fundação Coordenação de Aperfeiçoamento de Pessoal de Nível Superior*–CAPES, Brésil (dossier 0100/95-13) et par la *Comissão Nacional de Energia Nuclear*–CNEN, Brésil (dossier 01030.001317/95).

References

[1] C. Lavarenne, Y. Sorel. Modèle unifié pour la conception conjointe logiciel-matériel. *Traitement du Signal*, vol. 14, n. 6, 1997, p. 569-577.

[2] Y. Sorel. *Massively Parallel Computing Systems with Real-Time Constraints: the "Algorithm Architecture Adequation" methodology.* Proc. of Massively Parallel Computing Systems, Ischia, Italy, May 1994.

[3] A. F. Dias, M. Akil, C. Lavarenne, Y. Sorel. *Adéquation Algorithme-Architecture appliquée aux circuits reconfigurables.* 4èmes Journées Adéquation Algorithmes Architectures en Traitement du Signal et Images, Saclay, 1998, p. 35-42.

[4] R. Bergamaschi, R. Camposano, M. Payer. *Data path synthesis using path analysis.* Yorktown Heights: IBM Research Division, 1990. 13p. (Report N. RC 16274).

[5] D. Gajski, F. Vahid. Specification and design of embedded hardware-software systems. *IEEE Design & Test of Computers*, Spring 1995, p. 53-67.

[6] M. Aichouchi, P. Kission, A. A. Jerraya. Lien entre la synthèse architecturale et les outils de conception au niveau transfert de registres. *Technique et Science Informatiques*, v. 15, n. 2/1996, p. 179-199.

[7] C. A. Mead, L. A. Conway. *Introduction to VLSI systems.* s.l.: Ed. Addison-Wesley, 1980.

[8] L. Dutrieux, D. Demigny. *Logique programmable* : architecture des FPGA et CPLD, méthodes de conception, le langage VHDL. Paris : Éd. Eyrolles, 1997. 234p.