

UNE MÉTHODOLOGIE « FONCTIONNELLE » DE PROTOTYPAGE RAPIDE EN TRAITEMENT D'IMAGES

Ivan C. KRALJIC¹, Georges M. QUÉNOT² et Bertrand ZAVIDOVIQUE³

1. *Laboratoire Système de Perception*

DGA/Établissement Technique Central de l'Armement

16 bis avenue Prieur de la Côte d'Or, 94114 Arcueil

2. *LIMSI-CNRS, BP133, 91403 Orsay*

3. *Institut d'Électronique Fondamentale, Université de Paris XI, 91405 Orsay*

RÉSUMÉ - Une méthodologie complète de prototypage rapide de circuits VLSI de traitement d'images en temps réel est présentée. L'algorithme, décrit en langage fonctionnel, y est d'abord validé dans son environnement (temps réel et scènes réelles) grâce à un émulateur dédié: le Calculateur Fonctionnel. L'adéquation entre la description des algorithmes et l'architecture de l'émulateur apporte des avantages importants en terme de facilité d'émulation, de qualité de validation et d'intégrabilité.

ABSTRACT - *This paper presents a methodology for rapid prototyping of VLSI circuits dedicated to real-time image processing. The algorithm, written in a functional language, is firstly validated in the target environment of the circuit (real time and real life scenes) on a custom emulator: the Data-Flow Functional Computer. The adequation between the description of algorithms and the emulator's architecture yields important advantages in terms of ease of emulation, quality of validation and integrability.*

1 INTRODUCTION

La conception de circuits VLSI de Traitement d'Images en Temps Réel (TITR) requiert un soin particulier quant à la validation d'une part de l'algorithme, d'autre part du circuit lui-même. En effet, la spécification d'algorithmes de traitement d'images se fait à l'aide d'heuristiques et de beaucoup d'expérimentation. La méthode de validation de l'algorithme doit donc être suffisamment rapide et souple pour pouvoir supporter de nombreuses modifications de la procédure et de ses paramètres (noyaux de convolution, seuils, etc.). Quant à la validation du circuit (i.e. de sa description structurelle), elle est extrêmement gourmande en puissance de calcul du fait de la très grande quantité de données à traiter (une seconde d'images couleur en 512×512 contient 20 millions de pixels). À cela s'ajoute le fait que la validation devra se faire préférentiellement dans l'environnement dans lequel le circuit serait amené à évoluer.

Une approche efficace pour résoudre ces problèmes de validation (et qui s'avère aujourd'hui viable industriellement) est l'utilisation d'un émulateur [8]. Il s'agit d'une machine constituée de circuits reprogrammables : ces circuits sont programmés de façon à ce qu'ils reproduisent le plus fidèlement possible la structure du circuit en cours de conception. L'émulateur ainsi configuré constitue donc un prototype du circuit que l'on peut valider rapidement dans son environnement de destination. L'énorme majorité des émulateurs existants est basée sur des circuits FPGAs (*Field-Programmable Gate Arrays*). Citons notamment *Splash-2* [1], *Anyboard* [6], ainsi que les émulateurs commerciaux de Quickturn. Un FPGA est un circuit prédiffusé dont les connexions entre blocs logiques sont définies par des points mémoires programmables. La faible granularité des FPGAs (un bloc logique contient quelques portes et registres) rend ces émulateurs peu adaptés à l'*émulation*

systematique de circuits de traitement d'images en temps réel : le temps de configuration (placement et routage) d'un FPGA peut durer plusieurs heures, la fréquence de fonctionnement du FPGA dépend du circuit qu'il émule et de la qualité du placement/routage, l'émulateur est difficilement contrôlable et faiblement observable. De plus le passage du prototype au(x) circuit(s) est délicat (re-synthèse) ou sous-optimal (transfert de technologie).

La méthodologie de conception de circuits VLSI de traitement d'images présentée dans cet article est basée sur un émulateur dédié au TITR : le Calculateur Fonctionnel. La consécration de la méthode et de l'architecture de l'émulateur au TITR apporte des avantages en terme de facilité de spécification des algorithmes, de qualité de validation et de facilité de passage du prototype aux circuits VLSI. Le processus de conception contient deux étapes : 1) l'algorithme est émulé en temps réel sur le Calculateur Fonctionnel – l'émulation prouve l'existence d'une architecture implantant l'algorithme et valide son comportement en temps réel sur des scènes réelles ; 2) la configuration de l'émulateur validée implantant l'architecture est automatiquement analysée : les ressources réellement mobilisées sont compactées, leurs liens optimisés et l'automate finalement intégré sous forme de jeu de circuits VLSI.

La Section 2 présente les concepts principaux de la méthodologie, l'émulateur dédié ainsi que le flot d'émulation. La méthode d'intégration des circuits est décrite dans la Section 3. Un exemple d'application (détection de défauts sur images texturées) est présenté en Section 4. Finalement la Section 5 conclut l'article.

2 ÉMULATION EN TEMPS RÉEL

Le concept au cœur de la méthodologie est la décomposition fonctionnelle : l'algorithme de traitement d'images est décomposé en un ensemble d'opérateurs primitifs implantés et validés sur l'émulateur. Ainsi la description fonctionnelle de l'algorithme reflète simultanément la fonctionnalité de l'algorithme (i.e. sa description comportementale) et son implantation matérielle (i.e. sa description structurelle). Il s'agit alors de définir des bibliothèques d'opérateurs de traitement d'images de bas, moyen et haut niveau contenant les opérateurs primitifs susceptibles d'être utilisés. L'architecture de l'émulateur devra d'une part, pouvoir planter aisément les opérateurs de bibliothèques, d'autre part autoriser les associations entre ces opérateurs. Toutes ces considérations ont abouti à la conception du Calculateur Fonctionnel.

2.1 Le Calculateur Fonctionnel

L'architecture du Calculateur Fonctionnel est illustrée Fig. 1 [5]. Il s'agit principalement d'un réseau tridimensionnel de 1024 Processeurs Fonctionnels (PF). Un co-réseau de 12 Transputers T800 est connecté au réseau principal et permet d'implanter des traitements de haut niveau. Le modèle d'exécution de l'émulateur est flot de données. Plus précisément c'est de parallélisme fonctionnel qu'il s'agit : toutes les opérations correspondant à une itération de l'algorithme pour un pixel sont exécutées en parallèle à chaque cycle. Il y a ainsi une équivalence complète entre chaque opérateur de l'algorithme et un opérateur matériel.

La régularité de la structure de l'émulateur autorise uniquement des connexions locales, des connexions longue distance étant réalisées au moyen de PFs configurés en tant qu'éléments de routage. Chaque PF (voir Fig. 2) contient un chemin de données 8/16 bits à 3 niveaux de pipeline configuré par un contrôleur programmable (une RAM de 64×32 bits contient le graphe des transitions de la machine à états). Le PF dispose aussi de ressources de communication inter-PF (gérant les flots de données) : 3 piles FIFO d'entrée et 3 piles FIFO de sortie qui ont accès à 6 ports d'E/S 10 bits configurables. Le chemin de données consiste principalement en un multiplieur 8×8 bits, une unité arithmétique et logique 16 bits, deux compteurs 8 bits et une RAM 256×9 bits. Un ASIC contenant 2 PFs a été fabriqué en technologie CMOS 1 μ m.

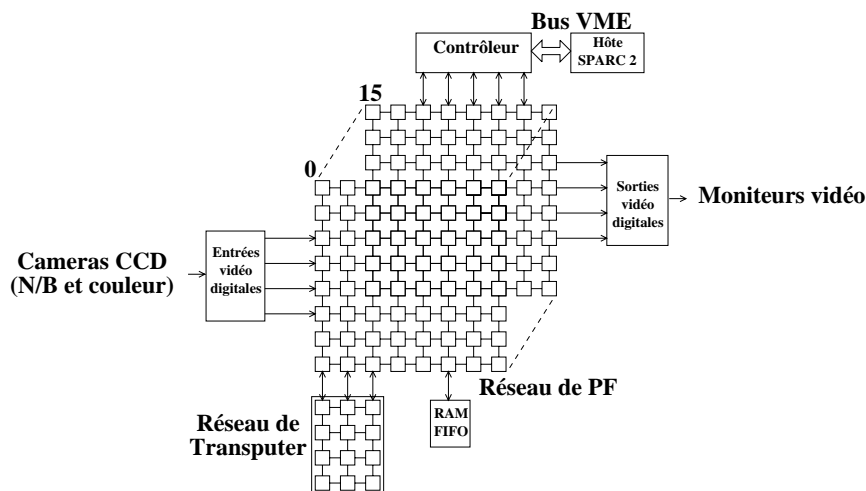


FIG. 1 - : Le Calculateur Fonctionnel.

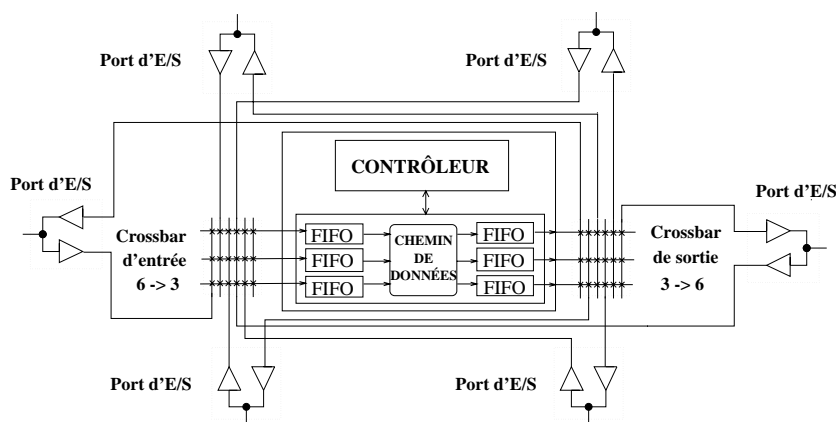


FIG. 2 - : Architecture du Processeur Fonctionnel.

Le PF a été conçu afin de pouvoir implanter une très large gamme d'opérateurs de TI de bas niveau tels que opérations arithmétiques et logiques, comparateurs, compteurs, retards lignes et pixels, dérivées et histogrammes. Des opérateurs plus complexes tels que la convolution sont facilement implantables sur un groupe de PF. Quelle que soit la complexité de l'opérateur implanté sur un Processeur Fonctionnel, il fonctionnera en temps réel à la fréquence de 25 MHz. Grâce à l'architecture pipeline du PF et donc du Calculateur Fonctionnel, ce dernier fonctionnera lui aussi à la fréquence de 25 MHz indépendamment de la complexité de l'algorithme qu'il émule.

2.2 Du langage fonctionnel au prototype

Le concept de décomposition fonctionnelle conduit naturellement à une description des algorithmes sous forme de graphes de flot de données. Bien que possédant un caractère intuitif et la capacité à supporter une décomposition hiérarchique des traitements, ils sont difficilement utilisables en tant que langage de programmation (interfaces graphiques complexes, manque de lisibilité pour de gros graphes). Le choix a donc été fait d'utiliser un langage fonctionnel de type Backus pour décrire les algorithmes [5].

L'algorithme, décrit en langage fonctionnel, est d'abord compilé en un graphe flot de données. Ce graphe est alors traduit en un graphe de Processeurs Fonctionnels utilisant les opérateurs de bibliothèque (150 opérateurs bas niveau sur un PF, 40 opérateurs moyen niveau sur plusieurs PFs, 10 opérateurs haut niveau sur Transputer). Le graphe de PFs doit finalement être « plaqué » sur le Calculateur Fonctionnel, c'est-à-dire placé et routé. Actuellement, ceci se fait manuellement pour des gros graphes, un outil de placement/routage automatique étant en cours de développement. La configuration à proprement parler du

Calculateur prend au maximum 15 s. En un sens, le Calculateur Fonctionnel peut être vu comme un émulateur à base de FPGA dont la granularité serait au niveau de l'opérateur flot de données plutôt qu'au niveau de la porte logique. Les graphes d'opérateurs flot de données spécifiés en langage fonctionnel remplacent les listes de portes. Les applications les plus significatives développées à ce jour incluent un étiquetage en composantes connexes, une poursuite couleur, une détection de défauts (présentée Section 4) ainsi qu'un outil interactif d'analyse d'images satellite [4].

Le prototype (i.e. émulateur configuré) est donc disponible très tôt dans le cycle de conception grâce au concept de décomposition fonctionnelle (ainsi que la granularité de l'émulateur au niveau transfert de registres). Il n'y a pas besoin de générer de description du circuit au niveau portes lors de la phase de prototypage. Le prototype est ensuite validé dans son environnement : en temps réel et sur des scènes réelles – les modifications de l'algorithme et de ses paramètres (seuils, filtres...) pouvant se faire à la volée. La robustesse des algorithmes émulés bénéficie grandement de ces propriétés. Le débogage se fait au niveau Processeur Fonctionnel ; l'utilisateur n'a pas à descendre plus bas dans les niveaux d'abstraction. Le prototype est néanmoins entièrement observable et contrôlable (au niveau transfert de registres).

Les limitations de la méthodologie concernent surtout le type d'algorithmes émulables : les opérateurs locaux sont plutôt du type convolution (invariance en translation), les globaux sont des listes ou courbes (histogramme). De plus les boucles doivent être « éclatées » et implantées de cette façon. En particulier la récursion en tant que telle est bannie. Ceci s'applique surtout aux traitements émulés sur le réseau de PFs et ne limite pas nécessairement le type d'applications émulables. Le choix de l'algorithme devra par contre être fait de façon à s'implanter efficacement sur l'émulateur, quitte à retailler la procédure correspondante. Les traitements de haut niveau doivent être émulés sur le réseau de Transputers.

3 DU PROTOTYPE AUX CIRCUITS INTÉGRÉS

Une fois que l'algorithme a été validé sur l'émulateur, il faut générer un jeu de circuits VLSI implantant l'algorithme. Un des moyens pour y arriver consiste à synthétiser les circuits à partir du graphe flots de données validé (synthèse comportementale [7]). Bien que produisant des circuits quasi-optimaux en termes de surface, cette approche n'utilise pas tous les résultats de l'émulation (car l'architecture de l'émulateur a aussi été validée) et est très coûteuse en temps de simulation des circuits synthétisés. L'autre méthode consiste à traduire la liste d'opérateurs représentant l'architecture de l'émulateur validée en une technologie VLSI, par exemple prédiffusée (transfert de technologie). Les circuits ainsi obtenus sont par contre très coûteux en surface de silicium.

Une méthode intermédiaire qui utilise la totalité des résultats de l'émulation est rendue possible. La méthode, appelée dérivation à partir de résultats d'émulation [3], consiste à utiliser tous les résultats de l'émulation : l'architecture qui a été validée sur l'émulateur est optimisée automatiquement en tenant compte du graphe flot de données validé. Le logiciel de dérivation construit une liste d'opérateurs au niveau transfert de registres à partir du graphe de PFs, la hiérarchie en termes de PF étant conservée.

3.1 Optimisations au niveau PF

Elles réduisent chaque Processeur Fonctionnel à son implantation minimale équivalente. Une analyse du graphe des transitions de la machine à états de chaque contrôleur PF identifie le chemin actif dans le chemin de données (configurable). Les opérateurs définissant le chemin actif sont les seuls réellement utilisés dans le PF en question ; tous les autres opérateurs ne sont dorénavant plus pris en compte. La Fig. 3 montre le chemin actif de l'opérateur addition. Les opérateurs du chemin actif sont ensuite optimisés : largeur de bus et nombre de niveaux de pipeline sont réduits au minimum.

Le deuxième élément à optimiser est le contrôleur de chaque PF. La méthode utilisée consiste à générer une description optimisée au niveau portes à partir des équations logiques définissant le graphe des transitions. Cette approche a l'avantage de la flexibilité :

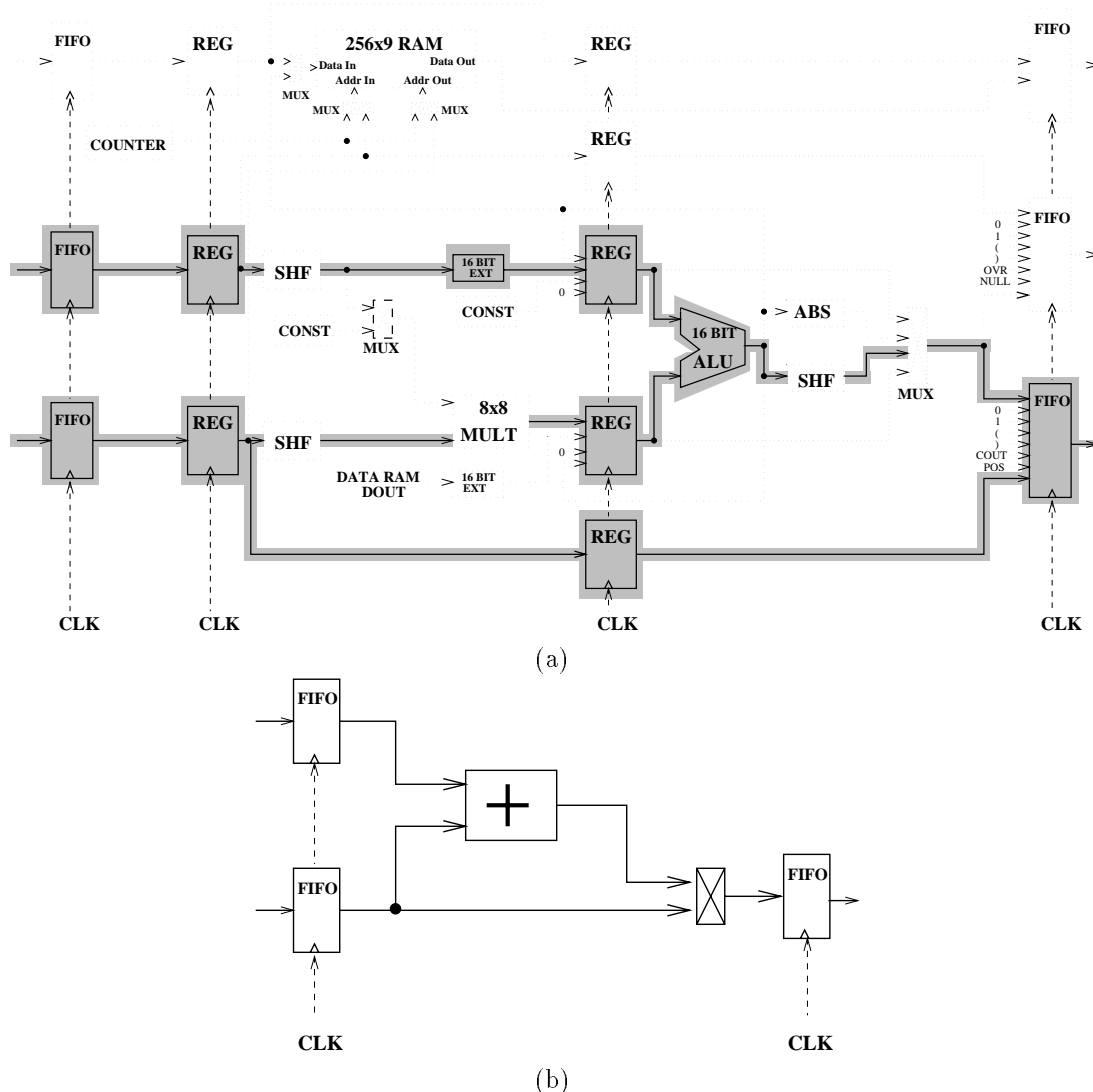


FIG. 3 - : Chemin actif (a) et architecture dérivée (b) de l'opérateur add.

le contrôleur dérivé peut être implanté sous forme de PLA, de cellules précaractérisées ou prédéfinies.

Finalement, la réduction des ports d'E/S consiste à remplacer les crossbars (entre les FIFOs d'E/S et les ports) par des connexions statiques implantant les liens de communication réellement utilisés.

À cet étape du processus de dérivation, le circuit est représenté par un graphe de « Processeurs Fonctionnels spécifiques », par exemple des additionneurs flots de données, des retards flots de données, des histogrammeurs flots de données... La table 1 présente les résultats de tels PFs dérivés. Chacun contient des ressources de gestion des flots (FIFOs et ports d'E/S) coûteuses en silicium.

3.2 Optimisations au niveau graphe de PFs

Les ressources de gestion des flots sont supprimées par fusion (automatique) de PFs adjacents ou par remplacement (manuel dans le fichier VHDL du circuit) de macro-PFs sous-optimaux. La fusion de PFs adjacents consiste à remplacer les piles FIFO par de simples registres pipeline là où les FIFOs n'effectuent pas de mémorisation explicite. Les FIFOs restantes effectuant une véritable mémorisation (comme par exemple dans l'opérateur retard pixel) voient quant à elles leur profondeur réduite au minimum.

Opérateur	PF complet	abs	add	max	Retard pixel	Retard ligne	Histo. 8-bits
Surface (mm ²)	35.64	1.12	1.39	1.91	1.44	1.50	3.91
FIFO/Op.	0.11	0.51	0.58	0.42	0.65	0.62	0.24

TAB. 1 - : Processeurs Fonctionnels (complet et dérivés) en technologie CMOS 1 μ m.

Un exemple de remplacement manuel est l'opérateur arctangente(y/x) du détecteur de défauts. Il nécessite une ROM de 1024×8 bits et a donc été émulé sur 9 PFs à cause des limitations du PF – nécessitant 15 FIFOs. Cet opérateur a été dérivé en un seul macro-opérateur contenant la ROM nécessaire grâce à la prise en compte du graphe flot de données – et ne nécessitant plus que 3 piles FIFO.

La méthode de dérivation constitue un compromis extrêmement intéressant entre la surface des circuits dérivés et le temps mis pour les générer (y compris le temps de simulation). D'autre part, les circuits dérivés ont le même modèle d'exécution flot de données que l'émulateur et contiennent les mêmes opérateurs (au niveau VLSI) que le graphe flot de données de l'algorithme. La méthode de dérivation constitue ainsi un prolongement naturel du concept de décomposition fonctionnelle.

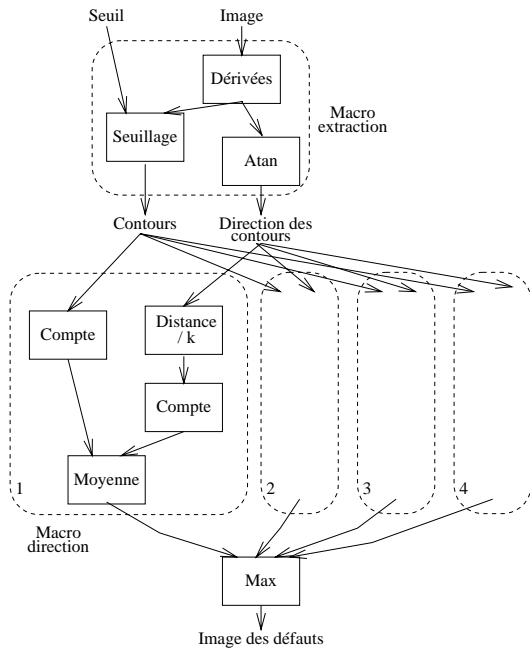
3.3 Implantation des circuits

Le logiciel de dérivation a été écrit en langage C. Il génère une description VHDL de type portes logiques (pour les contrôleurs) et transfert de registres (pour les chemins de données). L'outil COMPASS sert à placer et router les circuits. Les technologies précaractérisées (*standard cells*) et cellules de chemin de données compilées (*datapath compiler cells*) en CMOS 1 μ m sont actuellement utilisées. La surface moyenne d'un PF dérivé individuellement est de 1,5 mm². Les optimisations au niveau graphe de PF réduisent la surface des circuits d'environ 50 %. On peut donc dériver au maximum une centaine de PFs sur une seule puce. L'utilisation d'une technologie 0,6 μ m permettrait de dériver 200 PFs par puce. Ceci est l'ordre de grandeur pour des opérateurs comme détection de lignes droites horizontales ou verticales, ou poursuite couleur.

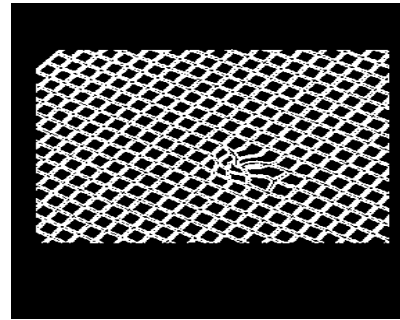
L'étape de placement/routage des circuits dérivés est la plus longue, et demande un certain effort à l'utilisateur. En effet, le placement des blocs dans le plan de masse (*floor-planning*) est manuel. La procédure est incrémentale : le plan est modifié et amélioré jusqu'à ce que l'utilisateur le trouve satisfaisant. On peut espérer un gain en surface de 10 % entre un plan bâclé et un plan soigneusement étudié. Les circuits implantés en cellules précaractérisées et en cellules de chemin de données compilées sont plus petits que ceux utilisant les seules cellules précaractérisées – au prix d'un effort accru pour générer le plan de masse. En pratique, le seul moyen pour y arriver est de placer et router chaque PF et macro-PF individuellement, puis d'assembler le circuit entier.

4 EXEMPLE : DÉTECTION DE DÉFAUTS

Le prototypage rapide d'un détecteur de défauts en temps réel est présenté. L'algorithme [2] consiste à identifier les régions défectueuses dans des images fortement texturées. Son graphe flot de données est présenté Fig. 4. La macro d'extraction détecte les contours et calcule leurs directions, chaque macro de direction compte ensuite les contours ayant une direction donnée. L'algorithme a été émulé en temps réel sur 108 PFs, la puissance de calcul est d'environ 800 MIPS (70 op./pixel). La macro d'extraction a été dérivée sur une puce de 36.24 mm² (précaractérisé et chemin de données), et la macro de direction sur une puce de 47.79 mm² (précaractérisé).



(a) Graphe flot de données.



(b) Images initiale et résultante.

FIG. 4 - : Détecteur de défauts - Résultats d'émulation.

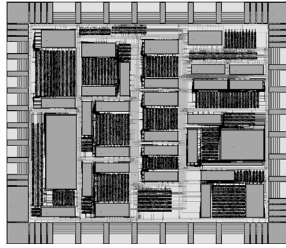
5 CONCLUSION

Un environnement complet de prototypage rapide de circuits VLSI de traitement d'images en temps réel a été présenté. La contribution principale réside en la méthodologie cohérente pour la spécification, la validation et la dérivation de circuits VLSI, basée sur un émulateur dédié. La performance de l'émulation peut être considérée comme maximale: traitement à la volée des capteurs (25 images/s.), images de taille 1000×1000 , séquences réelles de n'importe quelle durée.

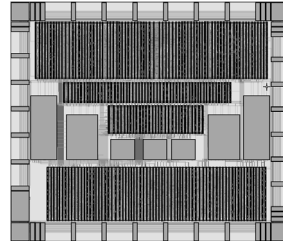
La méthode de dérivation à partir des résultats de l'émulation permet de réduire le temps de simulation des circuits intégrés. La dérivation constitue un prolongement naturel du concept de décomposition fonctionnelle, ce qui lui octroie une autre propriété intéressante. Les circuits dérivés peuvent être connectés tels quels à l'émulateur, jouant ainsi le rôle de co-Processeurs Fonctionnels d'un niveau sémantique plus élevé que le Processeur de l'émulateur. Si l'on considère une application trop large pour être émulée entièrement, cette propriété (alliée à la décomposition fonctionnelle exploitée dans sa totalité) permet d'émuler une partie de l'application, pendant que d'autres parties sont implantées dans des co-processeurs – qui sont en fait des circuits déjà émulés et dérivés – connectés à l'émulateur.

BIBLIOGRAPHIE

- [1] J.M. Arnold, D.A. Buell, and E.G. Davis. Splash 2. In *Proc. of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 316–22, 1992.
- [2] V. Brecher. New techniques for patterned wafer inspection based on a model of human preattentive vision. *SPIE Journal on Applications of Artificial Intelligence : Machine Vision and Robotics*, 1708:452–459, 1992.
- [3] I.C. Kraljić, G.M. Quénot, and B. Zavidovique. From Real-Time Emulation to ASIC Integration for Image Processing Applications. In *Proc. of the 8th Annual IEEE International ASIC Conference*, pages 31–4, Sept. 1995. Austin, TX, USA.



(a) Extraction



(b) Direction

FIG. 5 - : Circuits dérivés du détecteur de défauts.

- [4] S. Praud, P. Germain, and J. Plantier. Prototyping of Interactive Satellite Image Analysis Tools using a Real-Time Data-Flow Computer. In *Proc. of the International Conference on Image Analysis and Processing*, Sept. 1995. Sanremo, Italy.
- [5] J. Sérot, G. Quénot, and B. Zavidovique. Functional programming on a dataflow architecture: applications in real-time image processing. *Machine Vision and Applications*, 7(1):44–56, 1993.
- [6] D.E. Van den Bout, O. Kahn, and D. Thomae. The 1993 Anyboard Rapid-Prototyping Environment. In *Proc. of the 4th IEEE International Workshop on Rapid System Prototyping*, pages 31–40, 1994.
- [7] F.S. Verdier. *Conception de logiciels d'optimisation sous contraintes d'architectures VLSI pour le traitement d'images: le problème du contrôle*. Thèse de Doctorat, Université de Paris-Sud Centre d'Orsay, 1995.
- [8] N. Zafar. Using emulation to cut ASIC and system verification time. *ASIC Design*, Avr. 1994.