

Implantation VLSI de l'échantillonnage d'un contour à l'aide d'une spécification flot de données conditionné

VLSI Implementation of the Edge Sampling Using a Conditionned Data Flow Specification

par J.B. CHOQUEL*, J.P. DUBUS**, Y. SOREL***

* LISIR Université du Littoral – BP 649 F-62228 Calais cedex

** LMA Université de Lille 1 – Bât. P3 F-59655 Villeneuve d'Ascq cedex

*** INRIA Rocquencourt F-78153 Le Chesnay cedex

résumé et mots clés

Cet article présente une nouvelle approche permettant de synthétiser automatiquement le circuit VLSI implantant un algorithme spécifié et vérifié avec un graphe flot de données conditionné. Un algorithme d'échantillonnage de contour, classique en traitement d'images, est utilisé comme exemple pour illustrer l'approche. On le spécifie et on le vérifie avec le langage synchrone flot de données conditionné SIGNAL afin de produire directement, en utilisant des règles simples, le schéma logique correspondant. Ce dernier servira d'entrée à un logiciel de CAO de synthèse automatique de circuit, pour produire un circuit VLSI.

Traitement d'images, Echantillonnage de contour, Graphe flot de données conditionné, Langages synchrones, SIGNAL, Architectures parallèles, Circuit VLSI.

abstract and key words

The paper presents a novel approach to automatically synthesize a VLSI circuit implementing an algorithm specified and verified with a conditioned data flow graph. An edge sampling algorithm, classically used in image processing, is taken to experiment the approach. It is specified and verified with the conditioned data flow language SIGNAL. This allows to produce easily, using straightforward rules, the digital logic diagram, which will be exploited by an automatic synthesis CAD software to produce a VLSI circuit.

Image processing, Edge sampling, Conditionned data flow graph, Synchronous languages, SIGNAL, Parallel architectures, VLSI circuit.

1. introduction

Ce travail fait suite à une étude en simulation [Holle91] [Decom92] de l'algorithme d'échantillonnage de contour, il consiste à réaliser une implantation matérielle fonctionnant en temps réel de cet algorithme. Pour cela, une première étape a con-

sisté à spécifier l'algorithme sous la forme d'un graphe flot de données afin de mettre en évidence le parallélisme inhérent à l'algorithme (parallélisme potentiel) qui sera exploité en fonction des ressources matérielles disponibles (parallélisme effectif). Plusieurs approches sont alors possibles.

La première consiste à transformer le graphe flot de données en un programme qui s'exécutera sur un ordinateur parallèle commandé par les données (statique ou dynamique). Cette approche

est très peu utilisée car le calculateur flot de données reste une machine expérimentale, et par conséquent difficilement disponible.

La deuxième consiste à transformer le graphe flot de données en un programme qui s'exécutera sur un calculateur parallèle à flot de contrôle émulant un calculateur parallèle commandé par les données. Le problème consiste alors à distribuer et à ordonnancer les opérations (tâches séquentielles : sommets du graphe flot de données) sur les processeurs, et les transferts de données entre opérations (arcs du graphe) sur les moyens de communication. Pour simplifier ce processus on peut utiliser un logiciel d'aide à l'implantation comme SynDEX qui permet l'implantation optimisée d'un algorithme spécifié avec les langages flot de données Synchrones, tout en respectant des contraintes temps réel [Sorel92]. Il produit automatiquement des exécutifs optimisés [Lavar93] pour des multiprocesseurs construits avec des Transputers T800 et des processeurs de traitement du signal TMS320C40.

La troisième approche consiste à implanter la spécification flot de données sur un circuit VLSI (VHSIC-LSI : Very High Speed Integrated Circuit—Large Scale Integrated). Cette solution bien que donnant les meilleures performances temps réel, est en général la plus difficile et la plus coûteuse à mettre en œuvre. En effet, la conception du schéma logique correspondant à l'algorithme est long et les erreurs de conception (dues à la difficulté de gérer les registres et le contrôle) sont en général détectées lors des tests en simulation du circuit. Lorsque il est difficile de générer des vecteurs de test représentatifs, comme cela est souvent le cas en traitement d'images, c'est lors des tests en temps réel sur le circuit VLSI lui-même que ces erreurs sont détectées. Il faut alors concevoir et produire un nouveau circuit, ce qui augmente d'autant le coût.

Nous montrons ici que cette troisième approche peut être simplifiée en appliquant des règles de traduction simples permettant de passer directement d'une spécification sous forme d'un graphe flot de données conditionné respectant la sémantique des langages synchrones, à un schéma logique dont la gestion des registres et la partie contrôle a été vérifiée, limitant ainsi les erreurs découvertes lors des premiers tests en temps réel.

L'article est organisé comme suit. On donne tout d'abord les principes du flot de données conditionné respectant la sémantique du langage synchrone SIGNAL. On donne ensuite les règles permettant d'en déduire directement le schéma logique correspondant qui a été vérifié. Après avoir décrit la méthode d'échantillonnage d'un contour, on propose un algorithme utilisant la technique des secteurs, et on applique l'approche présentée aux chapitres précédents pour produire le schéma logique correspondant. Ce schéma logique utilisé en entrée d'un logiciel de synthèse permettra de produire un circuit VLSI. Ce dernier exécutera en temps réel l'algorithme d'échantillonnage de contour, plus rapidement que la version programmée sur un microprocesseur.

2. spécification flot de données avec le langage synchrone SIGNAL

Un algorithme peut être spécifié par un graphe décrivant des relations entre des opérations (opérations algébriques et transformations de données) à réaliser sur des données, en vue de produire des résultats.

Dans la version organigramme d'un graphe flot de contrôle, les sommets du graphe sont des opérations qui prennent leurs opérands dans des variables et produisent leurs résultats dans des variables. Les arcs traduisent une relation d'ordre d'exécution entre les opérations qu'ils relient. Dans la version automate, les sommets du graphe sont les états et les arcs définissent les transitions d'état exécutant les opérations qui elles aussi manipulent des variables. Dans les deux cas un ordre total d'exécution a été imposé sur l'ensemble des opérations du graphe. En mettant en parallèle ces graphes de contrôle et en établissant des communications selon le modèle CSP (Communicating Sequential Processes) de HOARE [HOARE 85], on obtient un ordre partiel sur les opérations à réaliser.

Dans un graphe de dépendances les sommets sont les opérations et les arcs sont des transferts de données entre opérations. On ne représente de cette manière que les relations d'ordre d'exécution rendues nécessaires par les dépendances de données. Elles sont indispensables quand une opération a besoin d'une donnée produite par une autre opération. Les relations d'ordre d'exécution qui ne sont pas nécessaires ne sont pas explicitées. Il est cependant possible de forcer des relations d'ordre d'exécution entre opérations en ajoutant des arcs pour lesquels les données transférées ne sont pas utilisées. Un graphe de dépendances induit un ordre partiel sur l'exécution des sommets du graphe. Le graphe flot de données engendré par un graphe de dépendances est une suite infinie de ces graphes de dépendances, chaque élément de la suite correspondant à l'arrivée d'un élément du flot de données venant de l'environnement (entrée de chaque graphe de dépendances). Le graphe flot de données correspond à la factorisation de la suite des graphes de dépendances. Chacun des arcs du graphe flot de données est une suite de transferts de données. Il faut souligner ici que la notion de variable utilisée dans les graphes flot de contrôle, qui est à l'origine de nombreuses erreurs dues par exemple à des réutilisations mal maîtrisées, est ici remplacée par celle de flot de données qui évite ce type d'erreur. La gestion des registres associés aux transferts de données est ainsi plus sûre. De plus, le graphe flot de données possède un sommet particulier pour localiser la mémoire d'état de l'algorithme, contrairement au graphe flot de contrôle où la mémoire d'état se trouve délocalisée parmi les autres variables. L'intérêt principal d'un graphe flot de données, auquel est associé un ordre partiel définissant le parallélisme potentiel de

l'algorithme, réside dans le fait qu'il offre le maximum de degrés de liberté lors de son implantation. En effet, les opérations qui ne sont pas en relation d'ordre d'exécution pourront, si les ressources le permettent, être exécutées en parallèle.

Les langages flot de données synchrones LUSTRE et SIGNAL, comme les autres langages synchrones, possèdent une sémantique basée sur l'hypothèse que les données produites par une opération (sommet du graphe flot de données) apparaissent simultanément avec les données qui ont déclenché l'opération. Les calculs et les communications sont instantanés, leurs durées physiques ne sont pas considérées. Par transitivité appliquée à tous les sommets du graphe, les données produites par le graphe apparaissent simultanément avec les données y entrant. Ces dernières venant de l'environnement définissent des événements d'entrée ou stimuli. De même les données produites par le graphe définissent des événements de sortie ou réactions. Cela permet de définir un temps logique dont les instants correspondent à l'entrelacement des événements. La notion de durée (logique) n'existe alors qu'au travers du comptage des événements.

De plus dans le langage SIGNAL, le modèle flot de données classique tel que défini par Dennis [Denni75] est étendu pour permettre, lors d'instantanés logiques, de ne pas exécuter certaines parties d'un graphe de dépendances. Ce type de graphe flot de données est dit conditionné, car il existe un sommet particulier dit de conditionnement, ne produisant pas de donnée bien que ses deux entrées soient présentes. Pour ce sommet, si l'entrée normale de type quelconque et l'entrée de conditionnement obligatoirement de type booléen sont présentes, mais que l'entrée de conditionnement porte la valeur faux, le sommet ne produit pas de données sur sa sortie. Là encore par transitivité, tous les sommets qui en dépendent ne seront pas exécutés lors de l'instant logique considéré. On peut ainsi associer à chaque entrée et à chaque sortie d'une opération et donc à chaque arc du graphe, un signal constitué de l'ensemble des événements portant une valeur et de son horloge. Cette dernière est définie par la présence de ces valeurs, relativement à d'autres signaux. Une horloge est une classe d'équivalence, car on peut insérer autant d'événements absents que l'on veut dans toutes les horloges équivalentes d'une classe, c'est-à-dire que seuls les événements présents sont significatifs. Deux signaux sont synchrones si leurs événements sont deux à deux simultanés.

Ce cadre formel permet d'effectuer des vérifications sur la spécification en terme d'ordre d'événements. Ainsi, le compilateur du langage synchrone SIGNAL vérifie si l'ordre des réactions est cohérent avec l'ordre des stimuli qui les déclenche. Il vérifie la cohérence des signaux, de rythmes différents, que l'on peut créer avec le mécanisme associé aux sommets de conditionnement. Il est important de noter qu'à ce niveau, on ne s'intéresse pas aux durées d'exécution et de transfert liées au temps physique qui s'écoule entre événements, mais uniquement à l'ordre entre ces événements.

Ces vérifications jouent un rôle important dans l'approche proposée. En effet, elles permettent de limiter les erreurs de concep-

tion dues à la difficulté de gérer les registres et le contrôle. Elles sont ici détectées le plus tôt possible, avant les tests en temps réel sur le circuit VLSI.

Le langage SIGNAL comporte quatre instructions de base donc quatre types de sommets (figure 1).

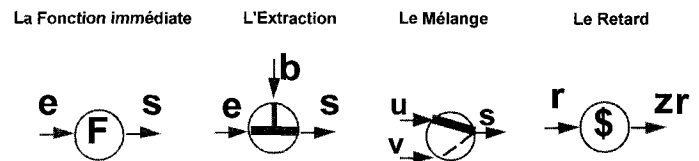


Figure 1. - Quatre instructions de base du langage SIGNAL.

L'horloge d'un signal de type quelconque e est notée $P(e)$, l'horloge d'un signal de type booléen b toujours vrai est notée $T(b)$. On associe à chaque instruction SIGNAL une équation d'horloge établissant une relation entre les horloges des sorties et des entrées. Comme dans tous les langages on associe à chaque instruction SIGNAL une relation entre les valeurs des entrées et des sorties. Les relations sur les valeurs, les équations d'horloges et les dépendances instantanées (dépendances entrées-sorties d'une instruction lors de l'instant considéré) sont décrites ci-dessous pour chacune des instructions :

- la fonction immédiate $s := F(e)$ (figure 2) met en relation fonctionnelle la sortie s avec l'entrée e . Le rythme des signaux de sortie est identique à celui des signaux d'entrée, l'horloge de s est égale à l'horloge de e .

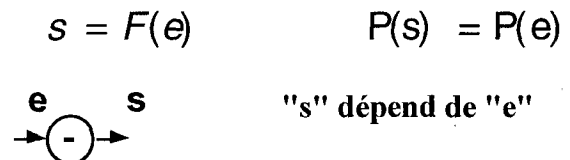


Figure 2. - La fonction immédiate.

- l'extraction $s := e$ when b (figure 3) fournit en sortie s la valeur présente sur l'entrée e , lorsque la condition b est présente et vraie, b doit être de type booléen. L'horloge de sortie est égale à l'intersection de l'horloge du signal d'entrée e et de l'horloge du signal b vrai.

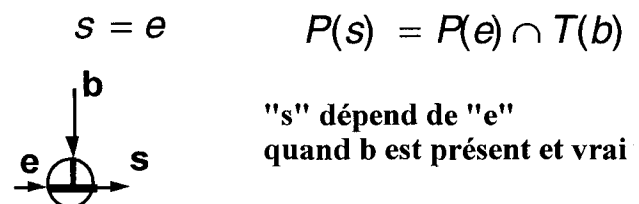
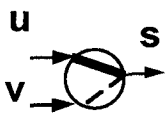


Figure 3. - L'extraction.

– le mélange prioritaire $s : = u \text{ défaut } v$ (figure 4) fournit en sortie s la valeur de celle des deux entrées u et v qui est présente. L'entrée u est prioritaire lorsque u et v sont présentes en même temps. L'horloge de la sortie est égale à l'union des horloges des signaux d'entrée.

$$s = u \text{ OU } s = v \quad P(s) = P(u) \cup P(v)$$



"s" dépend de "u" si "u" présent, ou de "v" si "u" absent et "v" présent

Figure 4. – Le mélange prioritaire.

– le retard $zr : = r\$1$ (figure 5) fournit en sortie zr la valeur précédente du signal d'entrée r . Cela permet de mémoriser un signal d'un instant logique à l'autre. Une valeur initiale doit être définie pour la sortie zr afin qu'elle soit déterminée au premier instant. Le retard établit une dépendance entre instants logiques et non pas à l'intérieur d'un instant logique.

$$zr_n = r_{n-1} \quad P(zr) = P(r)$$



"zr" ne dépend pas de "r" de façon instantanée

Figure 5. – Le retard.

La figure 6 illustre le concept de présence et d'absence de signal conduisant à la notion d'horloge dans le langage SIGNAL appliqué à l'instruction **when**. Le signal d'entrée e de type quelconque prend les valeurs $V1, V2, V3$ etc. Le signal d'entrée b de type booléen prend les valeurs vrai (T) ou fausse (F). La sortie s prend la valeur du signal e lorsque e est présent et que b est présent et vrai. Si e est présent et que b est présent et faux la sortie s est absente. Si e et/ou b sont absents s est absent. Lorsque des signaux sont absents ils sont représentés par le symbole (\perp). Les horloges des signaux peuvent être décrites

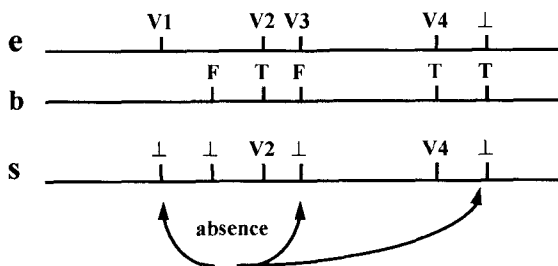


Figure 6. – Présence et absence de signal.

comme des signaux booléens toujours vrais, on ne s'intéresse qu'à leur présence. On parle ainsi de la classe d'équivalence d'une horloge qui est l'ensemble des horloges qui ont des instants de présence identiques.

3. règles de traduction des instructions signal en schéma logique

Pour satisfaire les règles de présence et d'absence de signal définies dans le langage SIGNAL, nous imaginons, dans un premier temps, des signaux réels transportant l'information sous trois états (figure 7).

$$\left. \begin{matrix} +V & 1 \\ 0V & 0 \end{matrix} \right\} \text{Présence}$$

$$+V/2 \text{ Absence notée } Hz$$

Figure 7. – Signaux trois états.

La présence est ici représentée par deux niveaux logiques $+V$ et $0V$. L'absence est représentée par $V/2$ et sera notée par la suite Hz .

La détection de la présence d'un signal e est effectuée par le dispositif décrit sur le schéma de la figure 8.

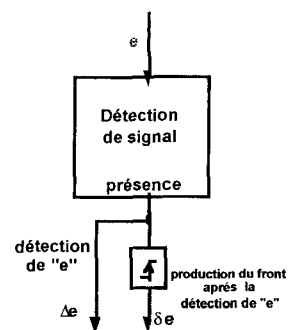


Figure 8. – Détecteur de Présence.

– Ce dispositif permet de produire un signal de type logique (0 ou 1) Δe qui sera utilisé comme signal de commande d'un multiplexeur, pendant la présence du signal d'entrée e de type quelconque.

– Ce dispositif permet de produire également une impulsion δe dont le front sera utilisé comme signal de commande des registres.

Le diagramme temporel de la figure 9 montre un exemple de détection d'un signal e possédant la valeur V_1 pendant une certaine durée.

La valeur V_1 du signal e est codée sur quatre bits. Le signal logique Δe , produit par la détection de la présence du signal e , sert de base à la commutation d'un multiplexeur analogique. Le multiplexeur est analogique de manière à transmettre les trois états du signal e [Choquel94].

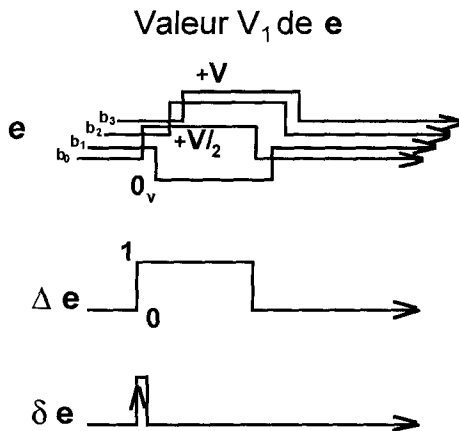


Figure 9. – Diagramme temporel.

Une réalisation électronique simplifiée du détecteur de présence est montrée figure 10.

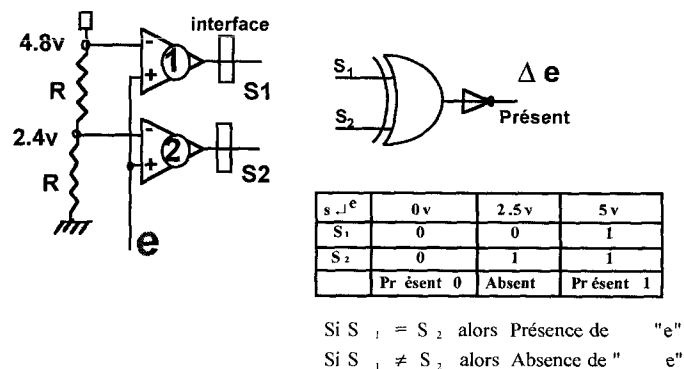


Figure 10. – Circuit électronique de détection.

Lorsque la tension d'entrée e est inférieure à la tension de seuil (2,4 v) du comparateur 2, la différence de tension ($V^+ - V^-$) du comparateur 2 est négative. En conséquence la tension de sortie du comparateur 2 est négative. L'interface fournit en sortie S_2 un état logique égal à 0.

Lorsque la tension d'entrée e est juste supérieure à la tension de seuil (2,4 v) du comparateur 2, la différence de tension ($V^+ - V^-$) du comparateur 2 est positive. En conséquence la tension de sortie du comparateur 2 est positive. L'interface fournit en sortie S_2 un état logique égal à 1.

Le fonctionnement est identique avec le comparateur 1 (associé à la tension de seuil de 4,8 v).

On remarque que la sortie S_1 détecte les états hauts des signaux (supérieurs à 4,8 v). La sortie S_2 détecte les états bas des signaux (inférieurs à 2,4 v).

L'état absent est matérialisé dans notre exemple par une tension égale à 2,5 v et cet état est détecté lorsque les sorties S_1 et S_2 sont différentes. En plaçant un OU Exclusif Complémenté connecté aux sorties S_1 et S_2 , on obtient en sortie le signal logique, noté Δe , qui indique la détection de la présence de e .

Ce montage fonctionne à condition que les signaux d'entrée évoluent toujours en passant par l'état absent.

Un circuit supplémentaire de production d'une impulsion δe peut être placé après la création du signal logique Δe afin d'être utilisé par les organes de mémorisation. Celui-ci n'est pas obligatoire si l'on utilise le front montant généré par le signal Δe pour autoriser l'activation d'un registre.

Le multiplexeur analogique figure 11 est l'élément de base utilisé lors de la traduction des instructions du langage SIGNAL. Il commute par l'intermédiaire de son entrée de sélection sel .

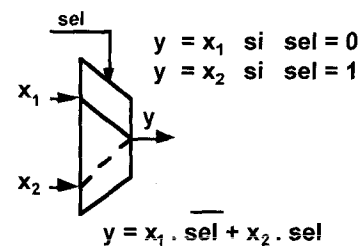


Figure 11. – Multiplexeur Analogique.

3.1. traduction de l'instruction Extraction (When)

Le symbole de l'instruction Extraction peut être traduit matériellement de deux façons différentes figure 12. Sur la figure 12(a), si l'entrée logique b est absente, on propage l'état absent H_z de l'entrée x_1 du multiplexeur. Si b est présent et e absent, on propage l'état H_z de e .

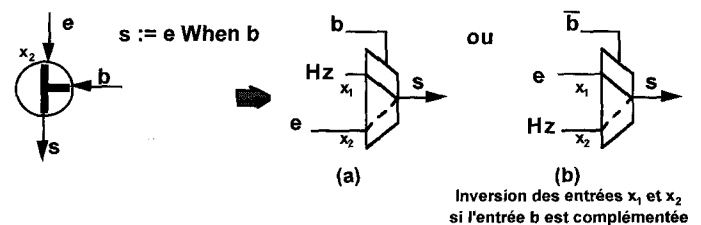


Figure 12. – Instruction Extraction (When).

Lorsque e est présent et lorsque b est présent et vrai, alors le signal e est placé en sortie s .

Sur la figure 12(b) on produit le même résultat en sortie s en plaçant le signal complémenté de b sur l'entrée sel et en inversant les entrées x_1 et x_2 .

En fonction du contexte, il peut être plus intéressant d'utiliser la forme (a) ou (b) afin de réaliser une optimisation matérielle.

3.2. traduction de l'instruction Mélange Prioritaire (Default)

Le symbole de l'instruction Mélange Prioritaire peut être traduit matériellement de deux façons différentes figure 13. Sur la figure 13(a), la détection de la présence de e_1 (notée Δe_1) avec le détecteur de présence (figure 8) provoque la commutation du multiplexeur analogique et la transmission de e_1 en sortie.

Lorsque les entrées e_1 et e_2 sont absentes, la sortie s prend la valeur absente de e_2 .

Sur la figure 13(b), on utilise la sortie complémentée après détection de e_1 pour provoquer la commutation du multiplexeur analogique.

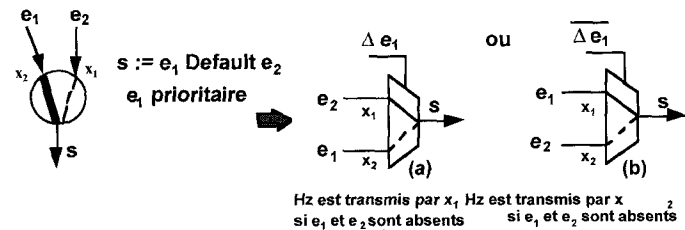


Figure 13. – Instruction Mélange prioritaire (Default).

3.3. traduction d'une fonction immédiate

La figure 14(b) montre le graphe d'une fonction immédiate possédant 4 entrées (exemple : additions multiples, calcul matriciel multiple ...). Toutes les entrées doivent être présentes en même temps pour que le résultat s soit valide.

Lorsque toutes les entrées d'une fonction immédiate ne sont pas présentes, la sortie doit rester à l'état absent.

Dans le graphe on introduit un nœud de synchronisation qui impose de prendre en compte la simultanéité de la présence de tous les signaux d'entrée. La figure 14(a) montre un graphe réalisant l'instruction de synchronisation en combinant les instructions d'Extraction (When) commandées par les signaux Δe_i issus des détecteurs de présence des signaux e_i .

L'instruction de synchronisation introduite dans le graphe de la figure 14(b) fait apparaître une sortie valide nommée s^* .

En appliquant la règle de traduction de l'instruction extraction du paragraphe 3.1 on aboutit au circuit logique de la figure 14(c). Si une des entrées e_i n'est pas détectée, alors le signal logique Δe_i est à zéro, et le multiplexeur correspondant propagera en sortie s^* l'état absent (matérialisé par le niveau Hz).

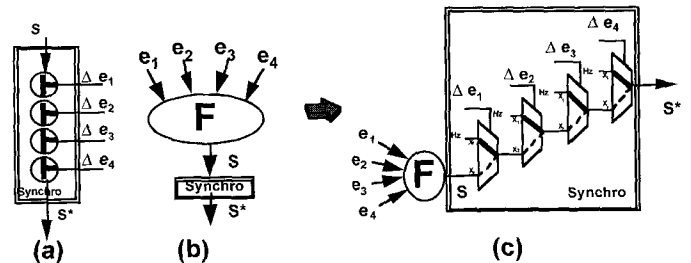


Figure 14. – La fonction immédiate

3.4. traduction de l'instruction retard

Le retard est matérialisé par un registre qui est activé à l'aide du front d'une impulsion produite après la détection de présence du signal d'entrée r (figure 15).

En sortie d'un registre le signal est toujours présent. Cette particularité est employée dans le cas où le signal logique de sortie du registre est utilisé pour commander un multiplexeur.

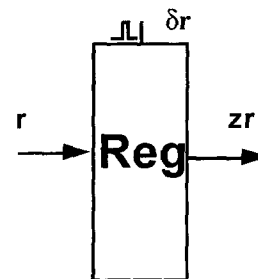


Figure 15. – Le retard.

4. algorithme d'échantillonnage de contour

4.1. introduction

Les contours des courbes considérées présentent des imperfections (coupures, épaissement, fourches) qui sont éliminées en utilisant des instructions morphologiques et des instructions d'ébarbage. La courbe résultat a une épaisseur d'un seul pixel comme sur la figure 16 suivante.

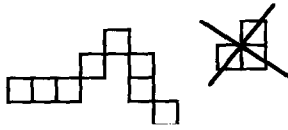


Figure 16. – Association de Pixels.

La méthode habituellement utilisée, de décomposition d'une courbe en segments de droites, [Pavli82] permet d'extraire les points significatifs en s'appuyant sur le principe de mesure de distance de la normale entre une corde, située entre deux points du contour et les points intermédiaires (figure 17).

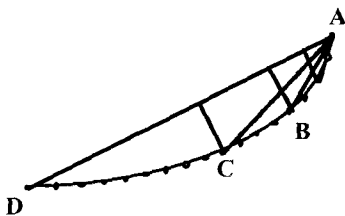


Figure 17. – Segmentation par mesure de distance.

Lorsque la distance est inférieure à un seuil fixé par l'utilisateur, une nouvelle mesure de distance est effectuée en utilisant le point situé à une longueur de corde double de la précédente. Lorsqu'une distance est supérieure au seuil, on conserve le point précédent. Cette approche a aussi été programmée dans [Degui92], mais en analysant tous les points intermédiaires.

Dans la méthode proposée ici, nous cherchons à supprimer le critère de mesure de distance. On cherche à décomposer la courbe originale en arcs de cercles de rayon de courbure variable et d'angle constant. Les points échantillonnés sont les extrémités de ces arcs. La courbe échantillonnée est faite de segments reliant les points échantillonnés. De cette façon, la courbe est définie par des secteurs juxtaposés, limités par les points échantillonnés, d'angle au centre constant et de rayon variable (figure 18).

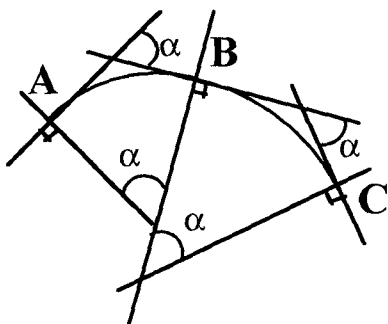


Figure 18. –Secteurs juxtaposés.

L'erreur résultante de la transformation est directement liée à la différence entre la surface initiale d'une part et la surface délimitée par les segments de droites d'autre part (figure 19).

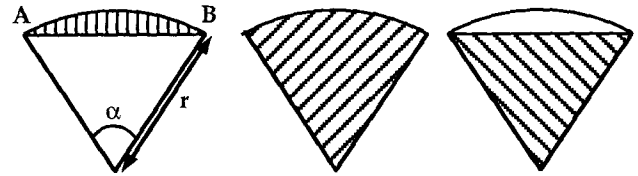


Figure 19. – Différence de surfaces.

L'erreur ε est donnée par la relation suivante :

$$\varepsilon = \frac{\text{Surface Réelle} - \text{Surface approximée}}{\text{Surface Réelle}} \quad (1)$$

d'où

$$\varepsilon = 1 - \frac{\sin \alpha}{\alpha} \quad (2)$$

avec α petit, le développement du sinus donne :

$$\varepsilon = \frac{\alpha^2}{3!} + \gamma(\alpha) \quad (3)$$

$\gamma(\alpha)$ représente l'erreur due à l'approximation de l'angle α qui donne une erreur relative de 1%, lorsque $\alpha = \pi/13$. Si l'on choisit $\alpha = \pi/16$, alors $\gamma(\alpha) = 2,11 \cdot 10^{-4}$.

Ainsi, tout angle α plus petit que $\pi/13$ donnera une erreur relative de surface ε inférieure à 1%, et le nombre de points conservés sera plus important. Pour déterminer les points échantillonnés, il suffit de détecter les tangentes à la courbe qui font un angle alpha entre elles. On a donc $\frac{2\pi}{\alpha}$ directions à étudier. On effectue la rotation d'un axe par pas d'angle α sur Π , avec ($\alpha = \pi/16$), et l'on détecte les tangentes perpendiculaires à cet axe (figure 20). Le nombre de rotations pour chaque point est donc de 16.

La rotation peut être limitée à $\Pi/2$, si les projections sur les deux axes orthogonaux sont considérées simultanément pour chaque position angulaire. Le nombre de rotations pour chaque point est alors de 8.

On utilise la particularité du stockage ordonné des points de la courbe. Chaque point de la courbe n'est lu qu'une seule fois et les rotations sont effectuées à chaque triplé. Comme on peut le remarquer en figure 20, cette méthode montre une densité de points plus importante dans les régions à faible rayon de courbure.

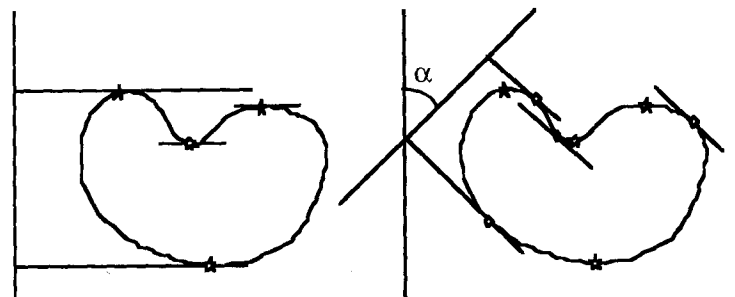


Figure 20. – Détection des tangentes.

Plusieurs techniques d'échantillonnage ont été simulées, nous retenons celle des secteurs qui élimine le calcul des n rotations.

4.2. méthode d'échantillonnage par la technique des secteurs

4.2.1. principe de base

Les points de la courbe initiale sont étudiés par triplets figure 21 [Taleb91] [Choquel94].

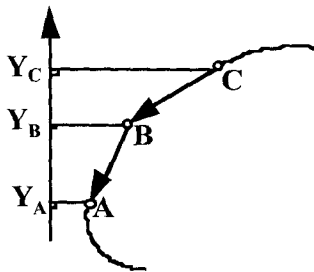


Figure 21. – Trois points à analyser.

Un point central B est éliminé si l'angle θ entre les vecteurs \overline{AB} et \overline{BC} est inférieur à une valeur donnée α (figure 22).

L'angle α , déterminé au chapitre 4.1, donne une erreur inférieure à 1% sur la surface du contour, et a pour valeur $\alpha = \pi/16$.

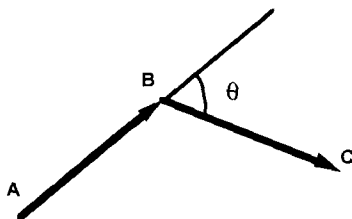


Figure 22. – Angle entre deux vecteurs consécutifs.

4.2.2. technique de détection des points échantillonnés

Les vecteurs \overline{AB} , \overline{BC} , \overline{CD} ... correspondent aux sécantes d'un contour. Ils sont représentés par leurs projections DX , DY . Si l'on reporte ces vecteurs dans un système de coordonnées DX , DY , où le point d'origine de chaque vecteur est confondu avec l'origine du système DX , DY ; alors $[DX, DY]$ peut être considéré comme un nouvel espace qui contient un ensemble fini de vecteurs. L'ensemble complet, représenté figure 23, est divisé en trente-deux secteurs d'angle $\alpha = \pi/16$. Chaque vecteur \overline{AB} est contenu dans un de ces secteurs. Le dimensionnement de DX et DY est volontairement limité à cinq bits pour restreindre la dimension du nouvel espace dans la figure. On suppose — ici par

exemple — qu'un écart entre deux pixels de ± 16 unités n'est pas atteint. Dans l'architecture, on pourra dimensionner DX et DY à la valeur de X et de Y (exemple 10 bits), pour autoriser des écarts maximaux.

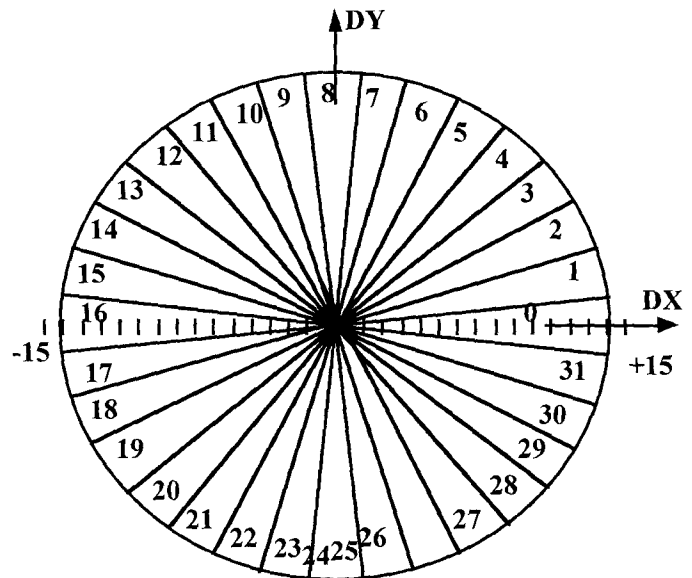


Figure 23. – Espace des différences de coordonnées.

A chaque triplet étudié correspondent deux vecteurs dans le même repère.

– Si ces vecteurs appartiennent au même secteur, alors les points de la courbe initiale A , B , C sont considérés comme alignés ou presque alignés, au sens de la segmentation envisagée. Dans ce cas, le point B est éliminé (figure 24).

– Si les vecteurs n'appartiennent pas au même secteur, alors le point B est échantillonné (conservé).

En fait comme l'algorithme montre une lecture séquentielle unique obligatoire du contour, on peut travailler par vecteurs en partant de AB et en recherchant le point C qui produira un vecteur BC n'appartenant pas au même secteur que le vecteur AB .

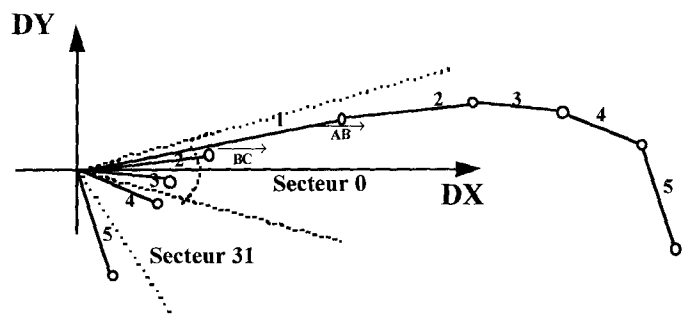


Figure 24. – Situation de trois vecteurs dans un même secteur.

4.2.3. algorithme de la technique des secteurs

Les 2 premiers points A et B , dans le sens de circulation, forment le premier vecteur et la situation de ce vecteur dans le nouvel espace des différences de coordonnées $DX - DY$, donne le numéro du secteur AB qui est mémorisé. L'algorithme se poursuit alors de la façon suivante :

Scruter l'ensemble du contour par groupes de 2 points à analyser
Rechercher le point C suivant

Déterminer la situation du nouveau vecteur BC dans l'espace des secteurs (N° du secteur BC)

si les numéros de secteur sont identiques :

alors le point C est éliminé et l'on conserve le N° du secteur AB
sinon le point C est conservé, et l'on mémorise le nouveau numéro de secteur à la place de l'ancien (Ce dernier devient N° secteur AB pour la prochaine comparaison)

Rechercher le point C suivant.

4.2.4. graphe flot de données conditionné appliqué à la technique des secteurs

La traduction de l'algorithme précédent conduit au graphe flot de données conditionné de la figure 25.

On trouve le vecteur courant \vec{BC} à l'aide du calcul des écarts de coordonnées $DX = X_{BC} = X_c - X_b$ et $DY = Y_{BC} = Y_c - Y_b$. Ce calcul est pratiqué en parallèle à l'aide des fonctions immédiates $F1$ et $F2$.

Le changement d'espace est défini par la fonction immédiate $F3$.

La mémorisation d'un événement sur un autre est schématisée par le symbole « \$ » qui indique le retard (figure 25).

La comparaison des deux événements est effectuée par la fonction immédiate $F4$.

Enfin si les deux vecteurs appartiennent au même secteur, alors la comparaison à 0 par la fonction immédiate $F5$ est vérifiée et le point C est à supprimer.

Lorsque le résultat de la comparaison n'est pas vrai (signal « Conservé » = faux), on réinjecte dans le retard la valeur précédente du N° de secteur AB , par l'intermédiaire du WHEN et du DEFAULT, dans le graphe.

Lorsque le résultat de la comparaison est vrai (signal « Conservé » = vrai), la valeur courante du N° de secteur BC prend la place de la valeur précédente du N° de secteur AB , dans la mémoire, pour devenir le signal $SectAB$, utile lors de la comparaison suivante.

Le chronogramme de la figure 26 correspond à l'évolution des données dans le graphe flot de données conditionné de la figure 25 en reprenant l'exemple de la figure 24.

La présence de la différence de coordonnées ($N^\circ 1$) sur le signal nommé $SectBC$, provoque la traversée de l'opérateur extraction (when) jusqu'au signal $SectAB$ (à l'entrée de l'opérateur retard), en passant par l'entrée (u) prioritaire de l'opérateur mélange

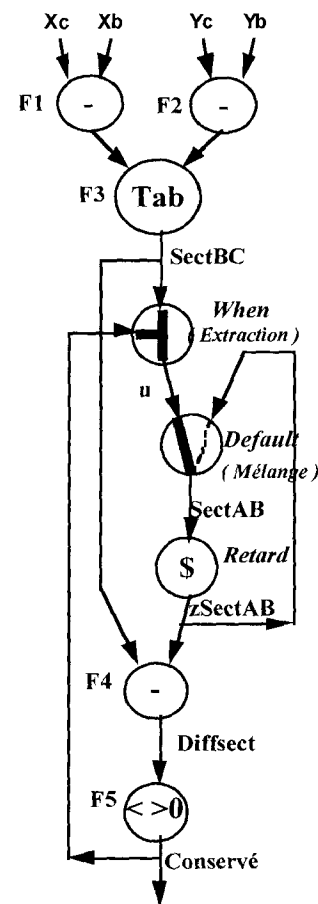


Figure 25. – Graphe flot de données.

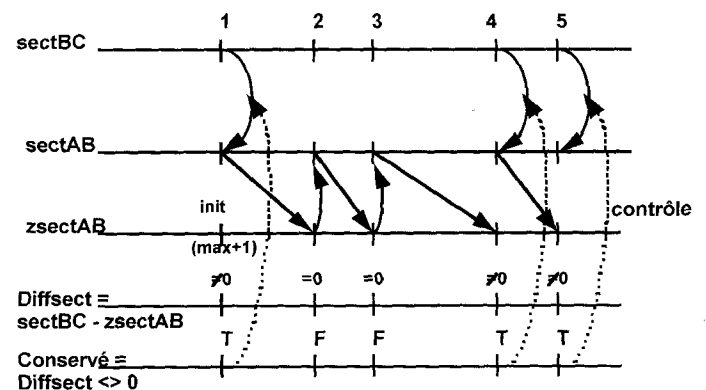


Figure 26. – Chronogramme du flot de données.

(default), car la fonction de comparaison à zéro a produit un signal booléen « conservé » à la valeur vraie, conditionnant le flot de la donnée $N^\circ 1$. Notons que la valeur initiale du signal de l'opérateur retard ($zSectAB$) a été placée à $max + 1$ (c'est-à-dire au nombre maximum de secteurs possibles plus une unité).

Puis il y a absence d'événement sur le signal $SectBC$.

Ensuite l'événement $N^\circ 2$ est présent. Il ne conduit pas au même déroulement car les deux vecteurs sont dans le même secteur

(le booléen « conservé » est faux). L'entrée « u » de l'opérateur mélange (default) est absente, l'entrée par défaut présente au signal $SectAB$ le contenu du signal $zSectAB$.

La traduction du graphe flot de données, en se limitant à la partie du graphe située après la fonction tabulée, en langage SIGNAL est reproduite ci-dessous :

```

Process SECTEURS=
  { ? integer SectBC           %variable d'entrée%
    ! logical Conservé }      %variable de sortie%
  %Equations Entrées - Sorties %Equations des Horloges%
  |(SectAB :=(SectBC when Conservé) default zSectAB
    %P(SectAB)=[P(SectBC) T(Conservé)]+P(zSectAB)%
  |zSectAB :=Sect AB $1       %P(zSectAB)= P(SectAB)%
  |Diffsect :=SectBC - zSectAB
    %P(Diffsect)= P(SectBC)= P(zSectAB)%
  |Conservé :=(Diffsect/=0)   %P(Conservé)= P(Diffsect)%
  |)
Where
  integer SectAB, Diffsect, zSectAB init max+1
end
    
```

La réduction des équations d'horloges — qui fonctionne sur les associations d'ensembles de présence des événements — permet de vérifier qu'il n'y a pas de contraintes à ajouter pour synchroniser des signaux éventuellement indépendants :

$$\begin{aligned}
 \%P(\text{Conservé}) &= P(zSectAB) \rightarrow = P(SectAB) \% \\
 \%P(SectBC) T(\text{Conservé}) &\rightarrow = T(\text{Conservé}) \\
 \%T(\text{Conservé}) + P(\text{sectAB}) &\rightarrow = P(SectAB) \%
 \end{aligned}$$

Toutes les équations sont vérifiées, l'entrée $SectBC$ est en cohérence avec la sortie $Conservé$.

5. architecture de la technique des secteurs

La lecture du graphe flot de données conditionné donne l'architecture représentée figure 27. Celle-ci est mise sous la forme d'un circuit logique en faisant référence aux règles de traduction du chapitre 3.

Nous admettons l'hypothèse d'un système extérieur à notre échantillonneur de contours, fournissant les données qui deviennent maîtresses du graphe flot de données. Les coordonnées des deux points d'entrées Xb, Yb et Xc, Yc sont mémorisées dans un registre à décalage (pipeline d'entrée) au rythme propre de la présence de la donnée Xc, Yc . Les coordonnées du point courant (Xc, Yc) sont présentées aux entrées des deux soustracteurs. Les différences de coordonnées (Xbc, Ybc) sont présentées aux entrées d'une mémoire comportant les numéros des secteurs.

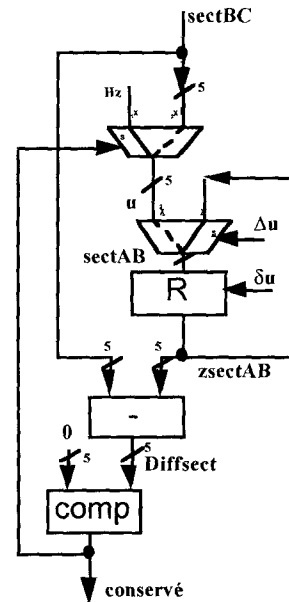


Figure 27. - Architecture de la technique des secteurs.

Le signal de lecture (Rd) de la mémoire Tab est activé par la détection de la présence des coordonnées du point courant ($Rd = \Delta Xc$).

La détection de la présence d'une nouvelle donnée en entrée provoque la production d'un événement sur le signal $SectBC$.

Nous n'aborderons pas dans cet exemple les simplifications que l'on peut effectuer dans le graphe, notamment en ajoutant des contraintes de synchronisation pour satisfaire les vérifications effectuées sur les équations d'horloge, et qui peuvent provoquer des réductions dans l'architecture.

Le numéro de secteur courant $SectBC$ est présenté avec le numéro de secteur précédent ($zSectAB$) à un soustracteur qui produit la différence entre les numéros de secteurs ($Diffsect$).

La différence entre les numéros de secteurs ($Diffsect$) est comparée à zéro par l'intermédiaire d'un comparateur ($Comp$).

Ce dernier produit un signal logique (**conservé** sur la figure 27). Le signal **conservé** provoque la commutation du premier multiplexeur ($when$) pour conserver la nouvelle valeur du numéro de secteur ($SectBC$) dans le registre R .

Le signal **conservé** est également utilisé par un système extérieur en sortie pour ne retenir que les points Xc, Yc élus par l'échantillonneur de points de contours.

La détection d'une nouvelle valeur à mémoriser s'effectue sur l'entrée du « default » notée « u » sur la figure 27. Cette détection produit un signal état (Δu) pour la commutation du deuxième multiplexeur (default) et une impulsion (δu) afin de mémoriser la nouvelle valeur de u ($SectBC$) dans le registre R .

Le registre de donnée R est initialisé à $max+1$ (c'est-à-dire $31+1$ dans notre exemple) par un mécanisme à multiplexeur

qu'il n'est pas nécessaire de présenter. Cette initialisation permet de mémoriser le premier numéro de secteur dans le registre R .

Nous n'avons pas représenté les multiplexeurs qu'il faudrait ajouter en sortie de chaque fonction immédiate (le soustracteur et le comparateur), auxquels sont associés des détecteurs de présence sur chacune de leurs entrées. Ces multiplexeurs propagent les états absents (H_z) comme indiqué en figure 14(c) pour ne pas que les fonctions immédiates produisent des résultats erronés.

6. conclusion

Dans cet article, nous avons choisi un algorithme d'échantillonnage d'un contour caractéristique en traitement d'images, pour montrer qu'il est possible de produire en suivant des règles simples un circuit VLSI à partir d'une spécification flot de données conditionné de cet algorithme. Le langage synchrone flot de données conditionné SIGNAL, permet de spécifier l'algorithme étudié en mettant en évidence son parallélisme potentiel. Son compilateur permet de vérifier si la spécification est correcte en termes d'ordre des événements qui vont être produits en réaction aux événements d'entrée. Ces vérifications à la compilation de la partie contrôle du schéma logique, conduisent à une conception de circuit plus sûre et diminuent la phase de tests en temps réel qui est très coûteuse. On en déduit directement le schéma logique complet de l'architecture matérielle correspondante par une traduction des instructions SIGNAL.

Cette méthode de traduction d'un graphe flot de données conditionné en un circuit logique introduit un accroissement de la complexité des circuits car pour chaque instruction SIGNAL il faut ajouter un mécanisme détectant la présence des données. En revanche, cela évite l'emploi d'une horloge globale et permet de bénéficier des avantages des techniques asynchrones.

L'approche matérielle présentée ici est basée sur une logique trois états. Elle est à classer dans les architectures asynchrones [Mérigot95] auto-séquentées [Renaudin96], bien que n'étant pas complètement insensible aux délais. En effet le signal de fin traitement (ici *conservé*) doit être estimé en terme de délai au niveau du détecteur de présence du signal (u) sur l'opérateur aiguillage (Default) du graphe flot de données, afin de présenter une impulsion (δu) retardée du délai nécessaire à la stabilisation des données [Choquel96]. Par ailleurs, d'autres travaux utilisant l'approche asynchrone pour traduire l'implantation d'une spécification synchrone sont en cours. Ils permettent de générer une description VHDL à partir d'un programme SIGNAL [Belhadj94], et semblent introduire une complexité plus importante due à l'utilisation des portes de Muller qui permettent d'être insensibles aux délais.

La prochaine étape de ce travail consistera à étudier et développer un traducteur automatique de programme SIGNAL en schéma logique selon un format acceptable directement par des outils de CAO de synthèse automatique ou bien en un sous-ensemble

synthésiable de VHDL. Pour cela on définira un fondement de règles basé sur celles définies dans cet article.

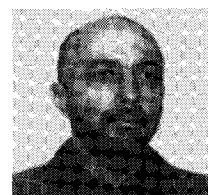
BIBLIOGRAPHIE

- [Belhadj94] M. Belhadj, « Conception d'architectures en utilisant SIGNAL et VHDL », *Thèse de doctorat en informatique*, Rennes, Décembre 1994.
- [Choquel94] J.B. Choquel, « Contribution à l'étude de la méthodologie de conception d'architectures parallèles par flot de données pour le traitement temps réel de la représentation d'objets en trois dimensions », *Thèse de doctorat en électronique*, Lille, Février 1994.
- [Choquel96] J.B. Choquel, J.P. Dubus, « Implantation VLSI à l'aide d'une spécification flot de données », *Journées Adéquation Algorithme Architecture en traitement du Signal et Images*, Toulouse, 17-18-19 Janvier 1996.
- [Decom92] D. Decomble, « Etude et réalisation d'un système de reconstruction et de visualisation 3D de l'arbre vasculaire cérébral à partir de coupes scanner IRM », *Mémoire CNAM*, Lille, Septembre 1992.
- [Degui92] D. Deguillemont, « Etude et réalisation d'un système de reconstruction 4D de chambres cardiaques à partir de vues échocardiographiques sous incidence apicale », *Mémoire CNAM*, Lille, Septembre 1992.
- [Denni75] J.B. Dennis, « First Version Data-flow Procedure Language », *Technical Memo MAC TM61*, MIT Laboratory for Computer Science, 1975.
- [HOARE 85] C.A.R. Hoare, « Communicating Sequential Processes. », *Version Data-flow Procedure Language*, Prentice Hall, 1985.
- [Holle91] A. Hollebecq, « Etude d'un système de reconstruction 3D à partir de clichés de coupes scanner X. Applications à la biopsie assistée par ordinateur », *Mémoire CNAM*, Lille, Janvier 1991.
- [Lavar93] C. Lavarenne, Y. Sorel, « Performance optimization of Multi-processor applications by graphs transformations », *Parallel Computing 93*, Septembre 93.
- [Mérigot95] A. Merigot, *Vers les architectures asynchrones*, Portovecchio, juillet 1995.
- [Pavli82] *Algorithms for graphics and Image processing*, Springer Verlag, Berlin, 1982.
- [Renaudin96] Marc Renaudin, « Asynchronisme et Adéquation Algorithme Architecture », *Journées Adéquation Algorithme Architecture en traitement du Signal et Images*, Toulouse, 17-18-19 Janvier 1996.
- [Sorel92] Y. Sorel, « Langages synchrones et exécutifs distribués optimisés », *GDR 134 — Adéquation Algorithme Architecture*, Lannion, 14/15 Septembre 1992.
- [Taleb91] A. Taleb-Ahmed, J.B. Choquel, F. Wauquier, J.P. Dubus, « Algorithme d'échantillonnage pour la reconstruction 3D d'objets définis par des coupes parallèles et implémentation par une architecture de traitement temps réel », *13ième congrès de GRETSI*, Juan les pins, Septembre 1991.

Manuscrit reçu le 16 mai 1994.

LES AUTEURS

Jean-Bernard CHOQUEL



Maître de conférence en électronique et en informatique industrielle à l'IUP du Littoral, Calais. Coordonnateur du Laboratoire d'Instrumentation du Signal de l'Image et des Réseaux (LISIR). A obtenu le grade de Docteur en électronique à Lille en 1994. Domaines d'intérêts : Traitement du Signal et des Images et Adéquation Algorithme Architecture.

Implantation VLSI

Jean-Paul DUBUS



Professeur d'électronique et Traitement du Signal à l'Université des Sciences et Technologies de LILLE. Directeur du Laboratoire de Mesures Automatiques (LMA). A obtenu le grade de Docteur es Sciences Physiques à LILLE en 1974. Domaines d'intérêts : Traitement du Signal et des Images appliqués à l'analyse des scènes dynamiques et reconstruction en trois dimensions. Télécommunications et codage de scènes.

Yves SOREL



Jean-Michel Directeur de Recherche à l'INRIA (Institut National de Recherche en Informatique et Automatique). Ses recherches concernent la formalisation et l'optimisation d'implantations d'algorithmes de commande, de traitement du signal et d'images s'exécutant en temps réel sur des architectures multi-composants. Il est responsable du groupe de travail : Adéquation Algorithme Architecture du GDR/PRC ISIS.