Conception et intégration d'un correlateur systolique

Catherine Dezan, Eric Gautrin, Patrice Quinton

INRIA

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N° 1351

Programme 2
Structures Nouvelles d'Ordinateurs

CONCEPTION ET INTEGRATION D'UN CORRELATEUR SYSTOLIQUE

Catherine DEZAN
Eric GAUTRIN
Patrice QUINTON

Décembre 1990

RR_1351

IRISA INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTEMES ALEATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CEDEX
FRANCE
Téléphone : 99.36.20.00
Télex : UNIRISA 950 473F
Télécopie : 99.38.38.32

Conception et Intégration d'un corrélateur systolique*
Design and VLSI Implementation of a Systolic Correlator

Catherine Dezan[†]
Eric Gautrin[‡], Patrice Quinton[§]

Programme 2

Publication Interne n° 556 - Novembre 1990 - 16 Pages

Résumé: De nombreux algorithmes utilisés en traitement du signal peuvent être mis en œuvre sur des architectures parallèles dont la régularité simplifie l'intégration. Dans cet article, on présente un corrélateur systolique et on décrit son intégration à l'aide de trois techniques : circuit sur mesure, circuit précaractérisé, et circuit à logique configurable. On montre comment l'architecture de ce corrélateur peut être dérivée formellement en utilisant le langage ALPHA.

Abstract: Many signal processing algorithms can be implemented on parallel architectures, whose regularity simplifies VLSI integration. In this paper, we present a systolic correlator, and we describe its VLSI implementation using full-custom, standard cell, and logical configurable arrays. We show how the architecture of this chip is formally derived using the ALPHA language.

*Recherche particllement financée par la société SOREP (Chateaubourg, Ille et Vilaine) et par le projet Esprit BRA Action 3280

[†]IRISA et ENST Bretagne
[‡]IRISA-INRIA, Campus de Beaulieu, 35042 Rennes Cedex
[§]IRISA-CNRS, Campus de Beaulieu, 35042 Rennes Cedex

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
(U.R.A. 227)
UNIVERSITE DE RENNES I I.N.S.A. DE RENNES

INSTITUT NATIONAL DE RECHERCHE
EN INFORMATIQUE ET EN AUTOMATIQUE
(UNITE DE RECHERCHE DE RENNES)

1 Introduction

Avec l'accroissement de la densité des circuits intégrés, la réalisation de systèmes spécialisés sur circuit monolithique est de plus en plus souvent envisagée. Dans une telle perspective, le problème principal à surmonter est la réduction du temps de conception des circuits, source majeure du coût de mise en œuvre.

Cet objectif peut être atteint de plusieurs façons:

- en automatisant le plus possible les tâches de conception, grâce à l'utilisation de logiciels puissants. Les logiciels de CAO de circuits disponibles aujourd'hui permettent d'effectuer dans de bonnes conditions la conception, la simulation, voire même pour les plus avancés d'entre eux la validation des circuits, une fois l'architecture déterminée. Il existe par contre très peu de logiciels permettant la conception architecturale d'un système à partir des spécifications de l'algorithme à réaliser.

- en réduisant la conception d'un système complexe par une approche à base de structures régulières. C'est l'application du principe "diviser pour régner" : les algorithmes à réaliser sont mis en œuvre sur une architecture parallèle composée de processeurs identiques. L'effort de conception est donc concentré sur la recherche d'une structure parallèle adaptée, et sur la conception d'un seul processeur qui est ensuite répliqué.

Beaucoup d'algorithmes utilisés dans les systèmes spécialisés font appel à des calculs réguliers qui se prêtent bien à une réalisation sur architectures parallèles [4]. C'est le cas dans le domaine du traitement du signal, de l'image, et du calcul numérique par exemple.

L'objet de cet article est de présenter, à partir d'un exemple, les problèmes posés par la réalisation d'un circuit spécifique parallèle, et les solutions que l'on peut y apporter. L'exemple considéré est un corrélateur dont trois versions systoliques ont été réalisées en circuit sur mesure (full-custom), précaractérisé, et enfin, avec de la logique configurable.

La première partie de l'article présente l'architecture du corrélateur et son fonctionnement. La seconde partie décrit et compare les trois réalisations. La troisième partie est consacrée au problème du test du circuit. Enfin, la dernière partie présente une approche systématique de la conception architecturale du corrélateur, à l'aide de l'environnement ALPHA du CENTAUR développé à l'IRISA.

2 Le corrélateur

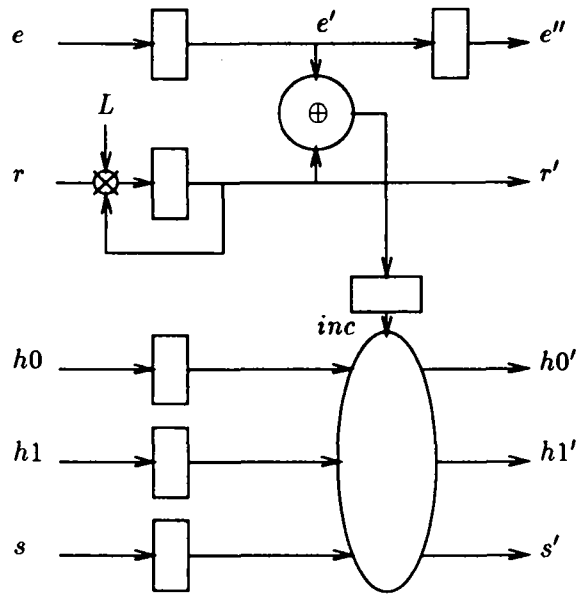
2.1 Spécifications

L'algorithme à réaliser consiste à reconnaître un mot binaire de référence r de N bits dans un signal binaire infini e_i, $i > 0$. On calcule ainsi la distance de Hamming, notée h_i, entre r et toute sous-suite e_i, ... , e_{i+N-1} de longueur N de e. Dans l'application que nous avons considérée, l'algorithme doit fournir un signal binaire de sortie s_i, égal à 1 si la distance est supérieure ou égale à 4, et égal à 0 sinon. Les calculs à effectuer peuvent donc être décrits par les équations suivantes:

$$h_i = \sum_{j=1}^{N} (r_j \oplus e_{i+j-1}) \quad (1)$$

$$s_i = (h_i \geq 4) \quad (2)$$

où $\oplus$ représente le ou exclusif de deux valeurs binaires.



$$
\begin{aligned}
e' &:= e \\
e'' &:= e' \\
inc &:= e \oplus r \\
s' &:= s \vee (inc \wedge h1 \wedge h0) \\
h1' &:= h1 \vee (inc \wedge h0) \\
h0' &:= inc \oplus h0 \\
r' &:= (L \wedge r) \vee (\neg L \wedge r')
\end{aligned}
$$

Figure 1 : Schéma et fonctionnement d'une cellule (Schematic and operation of one cell).

2.2 Réalisation systolique

Il existe un très grand nombre de réalisations parallèles pour un tel algorithme [8]. Celle qui est retenue ici est une version systolique linéaire, unidirectionnelle.

L'architecture est composée de N cellules identiques, cellules dont le schéma logique est donné par la figure 1. Le calcul de h_i se fait par accumulation successive, pour les valeurs décroissantes de j dans l'équation (1). Chacune des cellules est donc chargée de calculer le ou exclusif d'un bit de r et d'une valeur du signal e, et d'ajouter ce résultat à la valeur accumulée h qui lui est fournie par sa voisine de gauche. Le signal s est forcé à 1 par la cellule, lorsque la nouvelle valeur h accumulée dépasse 3. La valeur h est codée sur deux bits, $h0$ et $h1$. La valeur de r est contenue dans un registre, qui est chargé à partir d'une entrée de la cellule lorsque le signal de chargement L est actif.

La figure 2 donne le schéma du corrélateur pour $N = 4$. Les signaux e, $h0$, $h1$, et s circulent de

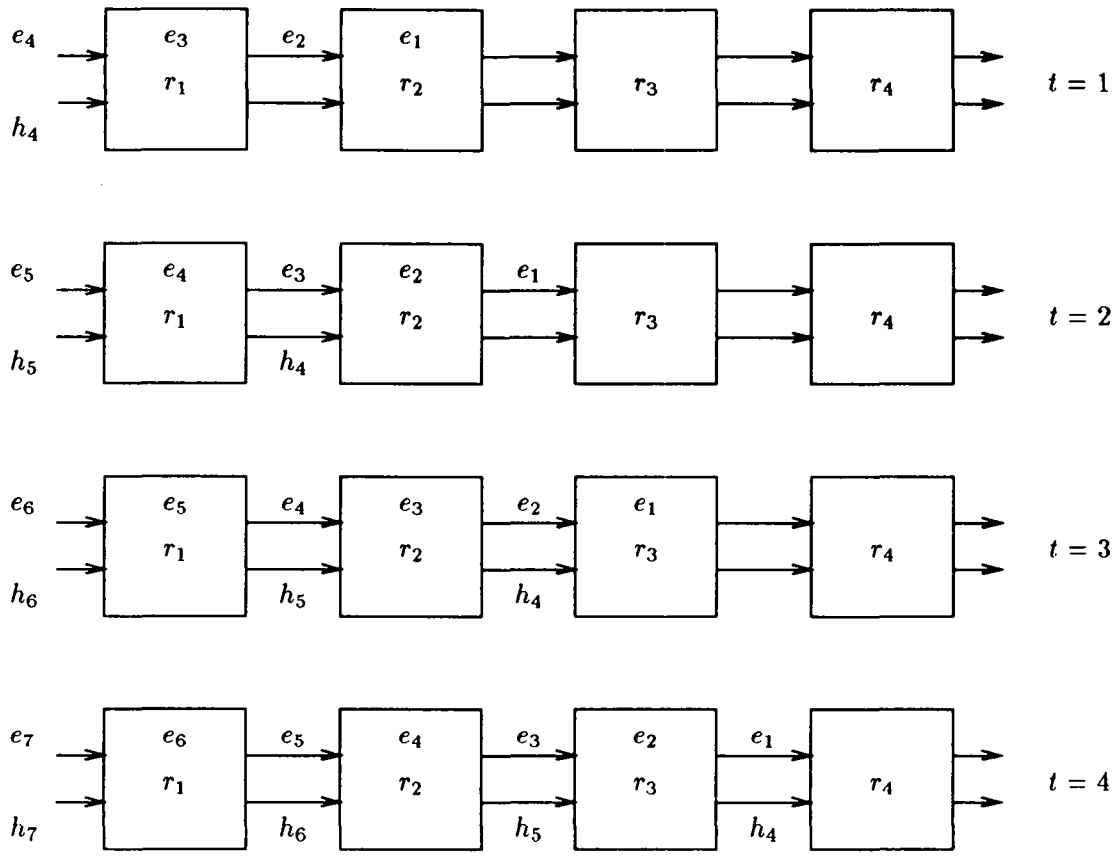


Figure 2 : Fonctionnement du corrélateur systolique (Operation of the systolic correlator).

gauche à droite. Le fonctionnement de l'architecture est illustré pour le calcul de

$$h_4 = r_1 \oplus e_4 + r_2 \oplus e_3 + r_3 \oplus e_2 + r_4 \oplus e_1.$$

Supposons que e_1, e_2 et e_3 aient été présentés sur la cellule de gauche, lors des cycles précédant l'instant initial de calcul. A l'instant $t = 1$, e_4 et $(h_4 = 0)$ sont présentés sur la cellule de gauche. Le calcul de $r_1 \oplus e_4$ est alors effectué. A $t = 2$, h_4 "rejoint" e_3, qui circule à vitesse moitié du fait de la présence de deux registres dans la cellule (voir figure 1). Après ce cycle, h_4 vaut donc $r_1 \oplus e_4 + r_2 \oplus e_3$. Et ainsi de suite. Noter que le calcul de h_5 est effectué de façon pipelinée, en commençant au cycle $t = 2$. L'architecture fournit donc en régime permanent un résultat à chaque cycle de calcul.

Le chargement des coefficients r est effectué, comme illustré par la figure 3, en chargeant sur la cellule de gauche les valeurs r_4 à r_1, et en activant le signal de chargement L. Notons que ce chargement est semi-systolique[1], puisque le signal de chargement est diffusé simultanément à l'ensemble des cellules. Une version systolique du chargement existe, mais elle n'est pas considérée ici pour des raisons de simplicité.

[1] Une architecture est dite *semi-systolique* si elle contient des bus de diffusion.

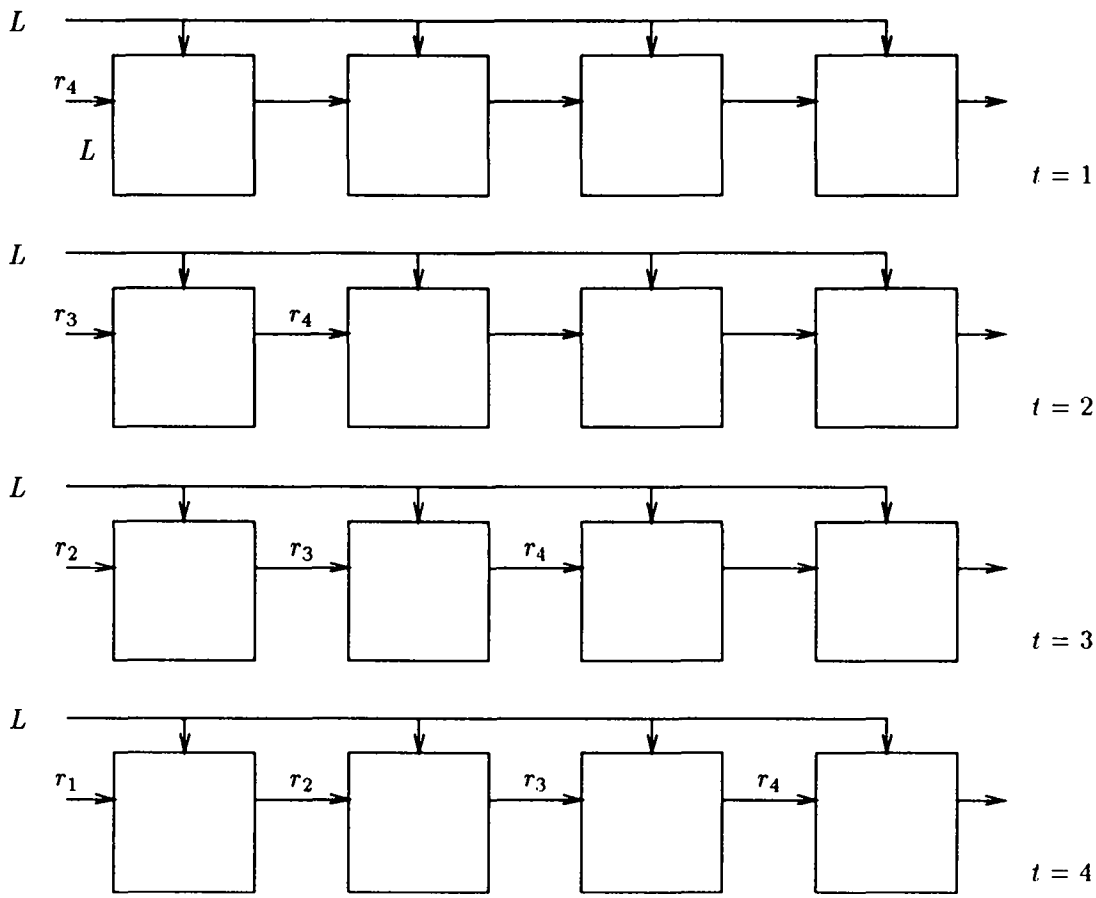


Figure 3 : Chargement des coefficients r (Loading of r coefficients).

3 Réalisation du corrélateur

Trois versions du corrélateur ont été réalisées. La première est un circuit sur mesure conçu avec le logiciel SOLO 2000 de CADENCE. La seconde mise en œuvre est un circuit précaractérisé réalisé avec le compilateur SOLO 1200 de ES2. Enfin, la dernière version a été conçue avec un circuit à logique configurable.

3.1 Circuit sur mesure

Cette version a été réalisée "manuellement", afin d'estimer les performances maximales qu'il est possible d'atteindre avec la technologie CMOS 2μ disponible. En d'autres termes, le logiciel SOLO 2000 a permis de simuler le circuit, logiquement et électriquement, et de comparer le dessin de masques avec le schéma électrique. La cellule est réalisée avec de la logique à précharge, et une horloge à deux phases non recouvrantes. La complexité d'une cellule, d'un point de vue conception, est de 46 transistors. La cellule finale, obtenue en répliquant certains blocs, comporte 96 transistors.

Les simulations électriques effectuées montrent que le circuit peut atteindre une fréquence maximale de 90MHz.

3.2 Circuit précaractérisé

La seconde version a été conçue avec le logiciel SOLO 1200 qui permet d'assembler des cellules précaractérisées d'une librairie en technologie CMOS 2μ. Cette librairie comporte des portes de base (inverseur, bascule, etc.) et des macro-cellules (additionneur, etc.). Les bascules et registres utilisent un système d'horloge sur front montant.

SOLO 1200 génère le circuit à partir de sa description en portes logiques. Le concepteur a en charge le choix des éléments de la librairie en fonction des facteurs vitesse et surface.

3.3 Comparaison des deux versions CMOS

Le tableau 1 présente différentes caractéristiques des deux versions d'un corrélateur de 16 cellules.

Mise en œuvre	Surface d'une cellule	Surface du corrélateur	Complexité	Fréquence (estimation)	Temps de conception
Précaractérisé	0,27 mm^2	8,88 mm^2	12 portes	50 MHz	2 semaines
Sur mesure	0,04 mm^2	4,70 mm^2	46 transistors	90 MHz	6 semaines

Table 1 : Comparaison des versions CMOS du corrélateur (Comparison of the two CMOS correlators).

Les remarques suivantes peuvent être faites :

Surface : Le résultat le plus significatif concerne la surface d'une cellule, qui varie de 1 à 7 entre les deux versions. Le rapport entre les surfaces totales est moins significatif, car il prend en compte les plots d'entrées/sorties dont la surface est indépendante du nombre de cellules. Pour un corrélateur de 128 cellules, ce rapport serait proche de 4.

Fréquence : Les deux versions du corrélateur ont une fréquence indépendante du nombre de cellules, si l'on ne tient pas compte du chargement du mot de référence r. Le rapport de vitesse est de 1 à 2.

Temps de conception : Les temps de conception indiqués ne tiennent pas compte de la formation aux outils de CAO, ni de la conception architecturale commune aux deux versions. Il est intéressant de noter que le temps de conception de la version sur mesure est court, ce qui s'explique par la faible complexité de la cellule. En outre, ce temps est pratiquement indépendant du nombre de cellules du corrélateur.

Il serait intéressant d'essayer de déterminer le nombre de pièces à partir duquel le circuit sur mesure devient moins onéreux que le circuit précaractérisé. Ce nombre dépend essentiellement des coûts de conception et de fabrication. Une estimation basée sur ces deux seuls coûts montre que la version sur mesure pourrait être moins chère à partir de 3000 pièces. Encore faut-il mentionner que cette estimation ne prend en compte ni le rendement de fabrication, qui avantage la version sur mesure plus petite, ni les performances en vitesse, qui peuvent se révéler des paramètres déterminants.

Cette comparaison montre l'un des effets bénéfiques du choix d'une structure parallèle : la modularité qui en découle permet d'atteindre des temps de conception très courts, et par conséquent, rend acceptable la conception du circuit sur mesure à partir d'un nombre d'exemplaires assez faible.

3.4 Mise en œuvre avec de la logique configurable

La troisième version du corrélateur a été réalisée avec de la logique configurable : LCA série 30XX de Xilinx [9]. Ces circuits sont composés de blocs configurables (CLB), chacun permettant de réaliser une fonction à 5 variables, ou deux fonctions à 4 variables. Les sorties des fonctions peuvent être verrouillées ou non. Les possibilités de routage entre CLB permettent des interconnexions locales ou globales.

Le logiciel Xact de Xilinx a été utilisé pour réaliser le corrélateur. Il s'agit d'un éditeur permettant de programmer chaque CLB, et de router les signaux entre les cellules. Une cellule du corrélateur nécessite 4 CLB. Par un système de macros, la cellule du corrélateur est automatiquement répliquée, et les signaux inter-cellules routés.

Les réseaux systoliques de niveau bit ont une structure qui s'adaptent bien aux LCA de Xilinx. Le routage local des LCA permet de mettre en œuvre efficacement les interconnexions locales des architectures systoliques. Lors de la conception du corrélateur, le routage a pu être effectué sans intervention du concepteur. Les seuls signaux globaux du corrélateur sont l'horloge et le signal de chargement L des coefficients r. Dans la mesure où les LCA offrent un système de routage sans délai de propagation pour deux signaux, la diffusion de ces signaux n'a pas posé de problème.

Une analyse temporelle statique montre que la fréquence maximale d'une cellule du corrélateur est de 43 Mhz. La fréquence estimée d'un corrélateur 16 cellules est de 33,3 Mhz. Cette différence est due au repliement du réseau nécessaire pour l'adapter à la topologie bidimensionnelle des LCA. Le temps de conception du circuit a été inférieure à une semaine.

Par rapport aux réalisations CMOS présentées ci-dessus, l'utilisation de réseaux à logique configurable présente l'intérêt évident de fournir un circuit dans des délais très brefs, aussi bien du point de vue de la conception que celui de la disponibilité du produit. Les performances du circuit sont bien sûr moins bonnes, mais restent néanmoins excellentes. La différence principale provient de la densité d'intégration atteinte. Dans le cas du corrélateur, une version comportant 128 cellules est aisément intégrable en circuit sur mesure, alors qu'il n'est pas possible de placer plus de 40 cellules sur un circuit à logique configurable à technologie équivalente[2].

[2]Une comparaison fine des densités est difficile à établir, dans la mesure où la technologie des circuits de la série 30XX de Xilinx est beaucoup plus avancée que celle qui était disponible pour les outils SOLO2000 et SOLO1200 au moment de la conception.

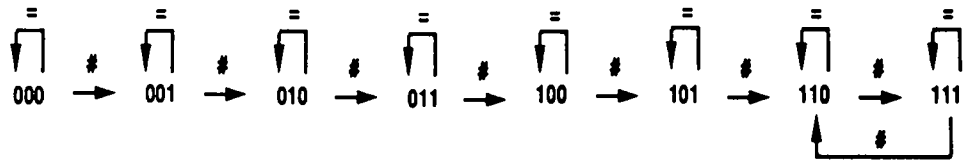


Figure 4 : Graphe de propagation des vecteurs du corrélateur (Vector propagation graph of the correlator).

4 Test de circuits systoliques

Le test d'un circuit consiste à détecter tous les défauts physiques dus au processus de fabrication [1]. Pour détecter un défaut physique, un vecteur de test est appliqué sur les entrées du circuit. Il contrôle, ou exhibe, l'effet du défaut, et propage son effet vers une sortie observable.

Un réseau systolique a peu d'entrées/sorties par rapport au volume de logique interne, et par conséquent, est difficilement contrôlable et observable. Des techniques classiques, comme le Scan Design, traditionnellement utilisées pour surmonter ce problème, n'apportent ici pas de solution satisfaisante. En effet, l'ajout de signaux pour contrôler et observer directement chaque cellule du circuit serait une opération très coûteuse, voire irréaliste pour des architectures de niveau bit comme le corrélateur.

En fait, le test d'une architecture systolique tire directement parti de sa structure régulière et hiérarchique, et de sa fonctionnalité. Les travaux décrits dans [7, 5] présentent la méthode dont nous nous sommes inspirés. Cette méthode suppose l'existence d'un seul défaut dans le circuit, ce qui est une hypothèse couramment admise.

Le principe de cette méthode est d'analyser par des méthodes classiques la testabilité d'une seule cellule, et de s'arranger ensuite pour propager les vecteurs de tests à l'ensemble des cellules, en tirant parti de la fonctionnalité de la cellule élémentaire. Dans un réseau systolique, certaines données circulent entre les processeurs sans être modifiées par ces derniers. C'est le cas des signaux e et r du corrélateur. Grâce à cette propriété, la logique dédiée à ces signaux est testée en appliquant en entrée du réseau une série de valeurs 0 et 1. La même série ressort inchangée du réseau en absence de défaut physique.

L'analyse de la propagation des vecteurs de test est faite par un graphe, appelé *graphe de propagation* des cellules faisant apparaître les transitions possibles entre les valeurs placées en entrée d'une cellule et les résultats obtenus en sortie.

A titre d'exemple, un graphe de propagation partiel du corrélateur est présenté figure 4. Les sommets du graphe sont ici les valeurs binaires des entrées $(s, h0, h1)$ d'une cellule. Les arcs sont étiquetés avec les relations $=$ et $\neq$. Un arc étiqueté avec $=$ représente une transition lorsque e et r ont la même valeur, et un arc avec $\neq$ une transition possible lorsque e et r sont différents.

L'analyse de ce graphe fait apparaître deux types de situations :

- lorsque e et r sont identiques et en l'absence de défaut, les vecteurs traversent le réseau sans être modifiés. Ils sont dits *invariants*. En outre, en présence d'un défaut dans une cellule, le vecteur $(s, h0, h1)$ est modifié, et sa valeur modifiée se propage telle quelle jusqu'à la sortie du réseau. En d'autres termes, un défaut lié à un tel vecteur de test est détecté par simple comparaison entre la sortie à un instant donné et la valeur appliquée en entrée N cycles systoliques auparavant, si N est la taille du réseau.

- lorsque e et r sont différents, un vecteur en entrée d'une cellule est modifié. Supposons le réseau configuré pour que le signal e soit différent du signal r seulement sur la cellule j. Un vecteur en entrée du réseau est alors propagé sans changement jusqu'à l'entrée de la cellule j, est modifié en sortie de celle-ci, puis propagé sans changement jusqu'en sortie du réseau. Un défaut dans la cellule j se détecte par comparaison entre la sortie réelle du réseau et la valeur attendue. En configurant successivement les cellules pour que e et r soient différents, il est possible d'appliquer tous les vecteurs de test à toutes les cellules. Ce type de propagation nécessite une séquence de test de taille en $O(N^2)$.

La séquence de test du corrélateur a été déterminée suivant ces principes. Dans un premier temps, les vecteurs nécessaires au test d'une cellule ont été déterminés, avec le logiciel HILO. Compte tenu de la simplicité de la cellule, cette opération a été très rapide. Pour la détection de tous les défauts de type collage ou court-circuit dans une cellule du corrélateur, 7 vecteurs de test sont nécessaires, dont 4 avec $e = r$, et 3 avec $e \neq r$. La séquence de test permettant la propagation de ces vecteurs à l'ensemble des cellules a ensuite été élaborée. Pour un réseau de N cellules, la séquence de test couvrant tous les défauts physiques a une longueur de $N^2 + 8N + 10$.

5 Vers la synthèse automatique de circuits réguliers

Il existe des techniques permettant la synthèse systématique d'architectures systoliques à partir de spécifications équationnelles. Le lecteur intéressé est renvoyé à [8] pour une bibliographie sur le sujet. La mise en œuvre de telles méthodes dans des logiciels utilisables est cependant loin d'être réalisée.

Dans cette partie, nous présentons brièvement un logiciel prototype, appelé ALPHA du CENTAUR, qui permet la dérivation d'architectures régulières. Le but de ce logiciel est essentiellement :

- d'offrir un langage permettant l'expression d'un algorithme régulier, depuis ses spécifications initiales jusqu'à sa réalisation architecturale;
- de fournir un ensemble de transformations interactives qui permettent à un architecte de passer pas à pas de la spécification à l'architecture, en préservant la sémantique de l'algorithme.

Dans ALPHA du CENTAUR, la dérivation d'une architecture est semi-automatique : le choix des transformations est laissé à l'utilisateur.

Nous illustrons ce processus sur l'exemple du corrélateur. Les équations qui définissent le calcul à effectuer sont :

$$h_i = \sum_{j=1}^{N} (r_j \oplus e_{i+j-1}) \tag{3}$$

$$s_i = (h_i \geq 4). \tag{4}$$

Remplaçons l'équation (3) par une récurrence décroissante sur l'indice j, en introduisant une nouvelle variable h' indexée par i et j. On obtient :

$$h'(i,j) = \begin{cases} \textbf{si } j = N+1 \textbf{ alors } 0 \\ \textbf{si } 1 \leq j \leq N \textbf{ alors } h'(i,j+1) + (r_j \oplus e_{i+j-1}) \end{cases} \tag{5}$$

$$h_i = h'(i,1). \tag{6}$$

```
-- Entrees et sorties du correlateur
system correlateur(e : {i | 1<=i} of boolean;
                   r : {j | 1<=j<=4} of boolean)
returns(s : {i | 1<=i} of boolean);
var
-- Variables locales
    h0,h1,S : {i,j | 1<=i; 1<=j<=5} of boolean;
    inc : {i,j | 1<=i; 1<=j<=4} of boolean;
let
-- Equations definissant les calculs
    inc = r.(i,j->j) <> e.(i,j->i+j-1);
    h0 = case
          {i,j | j=5} : false.(i,j->);
          {i,j | j<=4}: h0.(i,j->i,j+1) xor inc;
         esac;
    h1 = case
          {i,j | j=5} : false.(i,j->);
          {i,j | j<=4}: h1.(i,j->i,j+1) or (inc and h0.(i,j->i,j+1));
         esac;
    S  = case
          {i,j | j=5} : false.(i,j->);
          {i,j | j<=4}: S.(i,j->i,j+1) or
             (inc and h0.(i,j->i,j+1) and h1.(i,j->i,j+1));
         esac;
    s = S.(i->i,1);
tel;
```

Figure 5 : Spécification initiale en ALPHA du corrélateur (Initial ALPHA specification of the correlator).

La même transformation est appliquée à l'équation (4).

On obtient ainsi les spécifications initiales du corrélateur, dont la description dans le langage ALPHA est donnée par la figure 5, pour $N = 4$. Ce programme est un système d'équations, qui définit une fonction entre les entrées (`e` et `r`) et la sortie (`s`). Les objets utilisés dans le programme sont des variables indexées par les points entiers d'un domaine convexe. Ainsi, l'entrée `e`, dont la déclaration est :

```
e : {i | 1<=i} of boolean;
```

est une variable booléenne définie pour tout point i tel que $i \geq 1$. Les équations du programme peuvent utiliser des variables locales, définies après le mot-clé **var**. Dans l'exemple du corrélateur, ces variables locales sont `h0`, `h1`, `S` et `inc`, qui permettent le calcul itératif de la distance de Hamming `h` et du signal `s`.

Les équations, entre les mots-clés **let** et **tel**, définissent la valeur des variables locales et des sorties. Elle font appel aux constructions suivantes :

Opérateurs classiques : ils combinent des variables point à point. Par exemple, si `A` et `B` sont des variables définies sur le domaine `{i,j|i>=1, 1 <=j <=4}`, `A + B` désigne l'expression définie sur le même domaine, dont la valeur en chaque point est la somme des valeurs de `A` et `B`;

Restriction : une expression peut être précédée de la définition d'un domaine convexe qui restreint son domaine de définition. Par exemple, `{i,j|j=1}: A` désigne la restriction de la variable `A` au cas $j = 1$, soit le domaine de définition `{i,j|i>=1,j=1}`;

Dépendance : le domaine d'une expression peut être changé par utilisation d'une fonction de dépendance. Si `A` est définie sur le domaine `{i,j|i>=1, 1 <=j <=4}`, l'expression `A.(i,j -> i,j+1)` désigne la variable `A` translatée de $(0, 1)$. Une autre façon de lire cette expression est de remarquer que `A` est une fonction entre le domaine de `A` et l'ensemble de ses valeurs, et que l'expression `A.(i,j -> i,j+1)` n'est autre que la composition des fonctions `A` et de l'application affine `(i,j -> i,j+1)`;

Case : cet opérateur permet la définition d'une expression cas par cas. A titre d'exemple, l'équation définissant `h0` dans le programme ALPHA de la figure 5 est :

```
h0 = case
       {i,j | j=5} : false.(i,j->);
       {i,j | j<=4}: h0.(i,j->i,j+1) xor inc;
     esac;
```

Elle indique que `h0`, qui est défini en tout point (i, j) tel que $i \geq 1, 1 \leq j \leq 4$, vaut :

- `false` lorsque $j = 5$;
- `h0(i,j+1) xor inc(i,j)` sinon.

Le programme ALPHA initial est en fait la définition de la fonction qui lie les résultats `h` et `s` de l'algorithme en fonction des entrées. La dérivation de l'architecture systolique est effectuée à partir de ce programme ALPHA en effectuant une série de transformations qui préservent la définition de cette fonction. Dans le cas présent, ces transformations sont de deux types :

Changement de base : les variables du programme initial sont indexées par `i` et `j`; le but du changement de base est de remplacer ces indices par deux nouveaux indices `t` et `p` interprétés comme le temps auquel est effectué le calcul d'une équation, et le processeur sur lequel ce calcul est effectué. Il faut bien sûr que ce changement de base soit tel que le temps associé aux variables en partie droite d'une équation soit antérieur à celui de la partie gauche (causalité). A titre d'exemple, considérons à nouveau l'équation :

```
h0 = case
      {i,j | j=5} : false.(i,j->);
      {i,j | j<=4}: h0.(i,j->i,j+1) xor inc;
     esac;
```

En effectuant le changement de base

$$\begin{pmatrix} t \\ p \end{pmatrix} = \begin{pmatrix} i-j \\ j \end{pmatrix}$$

on obtient une nouvelle équation :

```
h0 = case
     {t,p|p=5} : false.(t,p->);
     {t,p|4>=p} : h0.(t,p->t-1,p+1) xor inc;
     esac;
```

qui exprime le fait que la valeur `h0(t,p)` calculée sur le processeur `p` à l'instant `t` est obtenue à partir de `inc(t,p)` et de la valeur `h0(t-1,p+1)` calculée à l'instant `t-1` sur le processeur `p+1`.

Des méthodes pour déterminer automatiquement le changement de base existent et sont décrites dans [8].

Pipeline : certaines équations font apparaître des variables "diffusées", dont les valeurs sont utilisées pour plusieurs calculs. La transformation de pipeline permet d'éliminer ces diffusions, anti-systoliques par nature, et de les remplacer par la circulation de valeurs de processeur à processeur. Ce phénomène apparaît par exemple dans l'équation :

```
inc = r.(i,j->j) <> e.(i,j->i+j-1);
```

où la valeur `e(i+j-1)` est commune aux calculs associés aux points (i,j) pour lesquels $i+j$ est constant. Par pipeline, cette équation est remplacée par

```
E = case
    {i,j|i>=2;j=4} , {i,j|4>=j;1=i;j>=1} : e.(i,j->i+j-1);
    {i,j|i>=2;3>=j;j>=1} : E.(i,j->i-1,j+1);
    esac;
inc = r.(i,j->j) <> E;
```

où une nouvelle variable `E` permet de transmettre `e(i+j-1)` de calcul en calcul.

```
system correlateur (e : {i|i>=1} of boolean;r : {j|4>=j;j>=1} of boolean)
       returns (s : {i|i>=1} of boolean);
  var
    R : {t,p|p>=1;4>=p;t+p>=6} of boolean;
    E : {t,p|p>=1;4>=p;t+p>=6} of boolean;
    h0 : {t,p|p>=1;5>=p;t+p>=6} of boolean;
    h1 : {t,p|p>=1;5>=p;t+p>=6} of boolean;
    S : {t,p|p>=1;5>=p;t+p>=6} of boolean;
    inc : {t,p|p>=1;4>=p;t+p>=6} of boolean;
  let
    R = case
        {t,p|t>=2;5>=t;t+p=6} : r.(t,p->p);
        {t,p|p>=1;4>=p;t+p>=7} : R.(t,p->t-1,p);
        esac;
    E = case
        {t,p|t>=2;5>=t;t+p=6} , {t,p|t>=3;4=p} : e.(t,p->t+2p-6);
        {t,p|p>=1;3>=p;t+p>=7} : E.(t,p->t-2,p+1);
        esac;
    inc = R.(t,p->t,p) <> E.(t,p->t,p);
    h0 = case
         {t,p|5=p} : false.(t,p->);
         {t,p|4>=p} : h0.(t,p->t-1,p+1) xor inc.(t,p->t,p);
         esac;
    h1 = case
         {t,p|5=p} : false.(t,p->);
         {t,p|4>=p} :
         h1.(t,p->t-1,p+1) or (inc.(t,p->t,p) and h0.(t,p->t-1,p+1));
         esac;
    S = case
        {t,p|5=p} : false.(t,p->);
        {t,p|4>=p} :
        S.(t,p->t-1,p+1) or
          inc.(t,p->t,p) and h0.(t,p->t-1,p+1) and h1.(t,p->t-1,p+1);
        esac;
    s = S.(i->i+4,1);
  tel;
```

Figure 6 : Description finale en ALPHA du corrélateur (Final ALPHA description of the correlator).

La version finale du corrélateur est donnée par la figure 6. En interprétant systématiquement l'indice `t` comme le temps, et l'indice `p` comme le numéro des processeurs, ce programme représente bien le fonctionnement du corrélateur, incluant en particulier tous les détails d'initialisation souvent difficiles à préciser dans une telle architecture.

Un environnement de programmation prototype construit sous le logiciel CENTAUR de l'INRIA [2] est actuellement opérationnel [3, 6]. Une interface avec le logiciel SOLO1400 est actuellement en cours de réalisation.

6 Conclusion

Différentes réalisations d'un corrélateur systolique ont été présentées et comparées. Nous avons montré comment l'architecture de ce corrélateur peut être construite de façon systématique par transformation d'une spécification équationnelle de l'algorithme, très proche de sa spécification mathématique naturelle.

Au fur et à mesure que les logiciels de conception de circuits progressent, les recherches s'orientent vers des méthodes et outils permettant la dérivation automatique et la vérification d'architectures. Posé dans toute sa généralité, le problème est extrêmement complexe, et conduit à une situation similaire à celle de la génération automatique de programmes, limitée pour l'instant à des exemples très simples. Pour des algorithmes possédant une structure régulière, une approche telle que celle qui a été présentée ici réduit considérablement les difficultés, permettant d'envisager à court terme une véritable compilation de silicium.

Bibliographie

[1] R. G. Bennetts. *Design of Testable Logic Circuits*, Addison-Wesley Publishing Company, 1984.

[2] P. Borras, D. Clément, Th. Despeyroux, J. Incerpi, G. Kahn, B. Lang, and V. Pascual. *CENTAUR: the System*, Technical Report 777, INRIA, Décembre 1987.

[3] P. Gachet, C. Mauras, P. Quinton, and Y. Saouter. *ALPHA du CENTAUR: an environment for the design of regular algorithms*, 1989 International Conference on Supercomputing, Crête, Grèce, Juin 1989.

[4] S.Y. Kung. *VLSI Array Processors*, Prentice Hall, 1988.

[5] W.P. Marnane, W.R. Moore, H.M. Yacine, E. Gautrin, and N. Burgess. *Testing bit-level systolic arrays*, International Test Conference 1987, Washington, Septembre 1987.

[6] C. Mauras. *Alpha: un langage équationnel pour la conception et la programmation d'architectures parallèles synchrones*, Thèse de l'Université de Rennes 1, IFSIC, Décembre 1989.

[7] W.R. Moore and V. Bawa. *Testability and diagnosability of a VLSI systolic array*, ESSCIRC, Toulouse, France, pages 271–276, Septembre 1985.

[8] P. Quinton and Y. Robert. *Algorithmes et architectures systoliques.* Masson, 1989.

[9] XILINX. *Programmable gate arrays data book*, XILINX, San Jose, CA, 1989-1990.

LISTE DES DERNIERES PUBLICATIONS INTERNES IRISA

PI 546 **LIMIT THEOREMS FOR MIXING PROCESSES**
Bernard DELYON
Septembre 1990, 22 Pages.

PI 547 **PERFORMANCES DES COMMUNICATIONS SUR LE T-NODE**
Frédéric GUIDEC
Septembre 1990, 38 Pages.

PI 548 **LES PREDICATS COLLECTIFS : UN MOYEN D'EXPRESSION DU CONTROLE DU PARALLELISME "OU" EN PROLOG**
René QUINIOU, Laurent TRILLING
Septembre 1990, 34 Pages.

PI 549 **NORMALISATION SOUS HYPOTHESE D'ABSENCE DE LIEN APPLICATION AU CAS NOMINAL**
François DAUDE
Septembre 1990, 42 Pages.

PI 550 **MULTISCALE SIGNAL PROCESSING : FROM QMF TO WAVELETS**
Albert BENVENISTE
Septembre 1990, 28 Pages.

PI 551 **ON THE TRANSITION GRAPHS OF AUTOMATA AND GRAMMARS**
Didier CAUCAL, Roland MONFORT
Septembre 1990, 46 Pages.

PI 552 **ERREURS DE CALCUL DES ORDINATEURS**
Jocelyne ERHEL
Septembre 1990, 58 Pages.

PI 553 **SEQUENTIAL FUNCTIONS**
Boubakar GAMATIE, Octobre 1990, 16 Pages.

PI 554 **ANALYSE DE LA FORME D'UN COEFFICIENT D'ASSOCIATION ENTRE VARIABLES QUALITATIVES**
Mohamed OUALI ALLAH
Octobre 1990, 26 Pages.

PI 555 **APPROXIMATION BY NONLINEAR WAVELET NETWORKS**
Qinghua ZHANG, Albert BENVENISTE
Octobre 1990, 16 Pages.

PI 556 **CONCEPTION ET INTEGRATION D'UN CORRELATEUR SYSTOLIQUE**
Catherine DEZAN, Eric GAUTRIN, Patrice QUINTON
Novembre 1990, 16 Pages.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique.

ISSN 0249-6399