SympA'5
Rennes, 8 juin - 11 juin 1999
5$^{\text{ème}}$ Symposium sur les Architectures Nouvelles de Machines

Synthèse haut niveau d'un co-processeur pour le calcul de bases de Gröbner

Tanguy Risset
IRISA, Campus de Beaulieu,
35042, Rennes Cedex, France.
`risset@irisa.fr`

Yannick Saouter
IRIT, 118 route de Narbonne,
31062 Toulouse Cedex, France.
`saouter@irit.fr`

Mots clé : base de Gröbner, codes correcteurs d'erreurs, architectures systoliques, synthèses haut niveau.

Résumé

Les bases de Gröbner sont un outil puissant utilisé dans de nombreuses applications en matière de calcul symbolique. Cet article présente un réseau systolique linéaire pouvant être utilisé comme co-processeur pour le calcul de bases de Gröbner, dédié aux codes correcteurs d'erreurs. Il a été conçu à partir d'une spécification de haut niveau en langage Alpha, ce qui permet d'effectuer divers contrôles d'analyse statique. Une étude de cas révèle le gain de temps réalisable grâce à un tel co-processeur.

1 Introduction

Les bases de Gröbner ont été développées par Bruno Buchberger [1] en 1965 dans le cadre des calculs de classes de résidus d'anneaux polynomiaux. Ce concept permet de nombreuses applications en matière de calcul symbolique [2, 3], mais aussi de géométrie algorithmique non linéaire, de preuve automatique en géométrie et de théorie de la modélisation [4]. Cependant, ces domaines ne nécessitent pas d'application en temps réel et par conséquent il n'est pas indispensable de disposer d'un matériel spécifique. Les bases de Gröbner permettent également des applications en robotique [5, 6], mais la puissance de calcul requise pour répondre aux exigences du temps réel est loin d'être atteinte par la technologie actuelle. Au cours des dernières années, plusieurs applications des bases de Gröbner sont apparues dans le domaine du décodage des codes correcteurs d'erreurs (codes de Reed-Solomon [7], BCH et codes à résidus quadratiques [8], codes abéliens [9], codes cycliques en général). L'article suivant présente un réseau systolique linéaire qui implémente les algorithmes de calcul des bases de Gröbner en temps réel.

2 Définitions et théorèmes

Cette section présente des notions concernant les bases de Gröbner sur des anneaux de polynômes multivariables [10]. Soit $K[\mathrm{X}_1, \mathrm{X}_2, ..., \mathrm{X}_\mathrm{n}]$ l'anneau de polynômes à n inconnues et à coefficients sur le corps K.

Définition 2.1 (termes - ordre de terme) *Soit* $T^n = \{\mathrm{X}_1{}^{a_1} \ ... \ \mathrm{X}_\mathrm{n}{}^{a_n} \mid a_i \in I\!N\}$. *Les éléments de* T^n *sont appelés termes. Soit* σ *un ordre total strict sur l'ensemble de termes* T^n. *Cet ordre est appelé ordre de terme si : (i)* $1 <_\sigma t$, *pour tout terme* $t \in T^n - \{1\}$ *et (ii) si* $t_1 <_\sigma t_2$, *pour* t_1 *et* t_2 *dans* T^n, *alors* $t.t_1 <_\sigma t.t_2$, *pour tout* t *dans* T^n.

Définition 2.2 (terme principal - coefficient principal) *Soit* K *un corps et* σ *un ordre de terme sur* $A = K[\mathrm{X}_1, \mathrm{X}_2, ..., \mathrm{X}_\mathrm{n}]$. *Alors tout polynôme non nul* $f = f(\mathrm{X}_1, \mathrm{X}_2, ..., \mathrm{X}_\mathrm{n})$ *peut être écrit d'une seule façon :* $f = \sum_{i=1}^{r} c_i T_i$, *avec* $c_i \in K^*$ *et* $T_1 >_\sigma T_2 >_\sigma ... >_\sigma T_r$. *On note* $c(f, T_i) = c_i$, *et* $Lt_\sigma(f) = T_1$. T_1 *est appelé le terme principal de* f *par rapport à* σ *, et que* $Lc_\sigma(f) = c_1$ *est appelé le coefficient principal de* f *par rapport à* σ.

Définition 2.3 (réduction) *Soient* f *et* g *deux polynômes de* A, *et* $F = \{f_1, ..., f_t\} \subset A - \{0\}$. *On dit que* f *se réduit en un pas en* g *selon* F, *si il existe* $f_i \in F$ *et* $T \in T^n$ *tel que* $g = f - (c/Lc_\sigma(f_i)).T.f_i$, *ou* $c = c(f, T.Lt_\sigma(f_i))$. *Cette notion s'étend immédiatement par fermeture transitive et nous l'appelons* $\xrightarrow{F}$.

Une telle relation $\xrightarrow{\mathrm{F}}$ est confluente si, lorsque deux réductions sont possibles pour x: $x \xrightarrow{\mathrm{F}} y_1$ et $x \xrightarrow{\mathrm{F}} y_2$, alors il existe z tel que $y_1 \xrightarrow{\mathrm{F}} z$ et $y_2 \xrightarrow{\mathrm{F}} z$.

Définition 2.4 (base de Gröbner) *Soit* $F = \{f_1, ..., f_t\} \subset A - \{0\}$, *et* (F) *l'idéal généré par* F. *Si la relation* $\xrightarrow{F}$ *est confluente alors* F *représente une base de Gröbner de* (F).

On trouvera dans [3, p B23] la démonstration que tout idéal de polynôme multivariable possède une base de Gröbner. La propriété suivante est importante :

Théorème 2.1 *Soit* $F = \{f_1, ..., f_t\} \subset A - \{0\}$ *une base de Gröbner de l'idéal* (F). *Soit* $f \in A$, *On a alors* $f \in (F) \Leftrightarrow f \xrightarrow{F} 0$.

Définition 2.5 (S-polynôme) *Soient* f_1 *et* f_2 *deux polynômes de* $A = K[\mathrm{X}_1, \mathrm{X}_2, ..., \mathrm{X}_\mathrm{n}]$, *et* σ *un ordre de terme. Alors le S-polynôme de* f_1 *et* f_2 *est le polynôme :* $S(f_1, f_2) = Lc_\sigma(f_2).u_1.f_1 - Lc_\sigma(f_1).u_2.f_2$, *où* $u_1 = \mathrm{ppcm}(Lt_\sigma(f_1), Lt_\sigma(f_2)) / \ Lt_\sigma(f_1)$ *et* $u_2 = \mathrm{ppcm}(Lt_\sigma(f_1), Lt_\sigma(f_2)) / \ Lt_\sigma(f_2)$ *(*ppcm *représente le plus petit commun multiple de deux polynômes).*

Soit un ensemble de générateurs $F = \{f_1, ..., f_t\} \subset A - \{0\}$ de l'idéal (F), la base de Gröbner qui lui est associée est calculée par l'algorithme de Buchberger dont l'exactitude est démontrée en [1] :

Algorithme 1
Entrées: *F*, σ.
- **Initialisation**: *Soit* $G = F$ *et* $B = \{\{f_1, f_2\} \mid f_1, f_2 \in F, f_1 \neq f_2\}$.
- **Étape 1**: *Si* $B = \emptyset$, *alors* **Stop**.
- **Étape 2**: *Choisir* $\{f_1, f_2\} \in B$ *tel que* $\text{ppcm}(Lt_\sigma(f_1), Lt_\sigma(f_2))$ *soit minimal par rapport à* σ. $B = B - \{f_1, f_2\}$.
- **Étape 3**: *Calculer* $h = S(f_1, f_2)$ *et calculer* h' *tel que* $h \xrightarrow{G} h'$.
- **Étape 4**: *Si* $h' \neq 0$, *alors* $B = B \cup \{\{g, h'\} \mid g \in G\}$ *et* $G = G \cup \{h'\}$. *Aller à l'* **Étape 1**.

Sortie: *G*.

Le résultat final G est une base de Gröbner de l'idéal (F) par rapport à l'ordre de terme σ. Dans cet algorithme, l'ensemble G augmente constamment. Cet ensemble pourrait être réduit en vérifiant, pour chaque polynôme $f \in G$, si il se réduit à 0 selon l'ensemble $G - \{f\}$.

3 Utilisation des bases de Gröbner pour les codes correcteurs d'erreurs

Les bases de Gröbner sont utilisées dans les codes multivariables (codes semi simples, codes abéliens, voir [11]). Ces codes emploient des éléments de $K[\mathrm{X}_1, \ldots, \mathrm{X}_\mathrm{n}]/I$ où K est un corps fini (ou corps de Galois, $\mathrm{GF}(p^n)$ voir [12]), et I est un idéal de $K[\mathrm{X}_1, \ldots, \mathrm{X}_\mathrm{n}]$. L'opérateur de réduction et le calcul du S-polynôme sont très souvent utilisés pour le calcul dans $K[\mathrm{X}_1, \ldots, \mathrm{X}_\mathrm{n}]/I$.

Etant donné que de nombreuses applications des bases de Gröbner utilisent $K = \mathrm{GF}(2)$ (voir section 6), nous nous limiterons à l'étude de ce cas. Par conséquent, les opérateurs du corps de Galois sont très simples : le *Xor* binaire représente la somme et le *And* binaire représente la multiplication. La version de cette architecture étendue à $K = \mathrm{GF}(p^n)$ est étudiée dans [4], et les principes sont pratiquement similaires (voir les documents [11, 13, 14] pour les opérateurs utilisés dans les corps de Galois).

Dans ce qui suit, nous décrivons, une architecture pour $\mathrm{GF}(2)[\mathrm{X}, \mathrm{Y}]$. Les principes de l'architecture peuvent être facilement étendus à un plus grand nombre d'inconnues. Dans ce document l'ordre des termes est toujours l'ordre lexicographique.

3.1 Choix d'implémentation

Le modèle d'architecture utilisé est un réseau systolique linéaire. Ce modèle est synchrone, les communications avec l'hôte sont connues de façon statique [15]. Le réseau est unidirectionnel, l'algorithme peut ainsi être partitionné et accompli en plusieurs passages si nécessaire (partitionnement LPGS [16]).

Un polynôme est représenté par une liste de triplets (λ_i, x_i, y_i), où λ_i est un booléen et x_i, y_i sont des entiers. Le triplet (λ_i, x_i, y_i) code le monôme $\lambda_i X^{x_i} Y^{y_i}$. L'étape 1 ou l'étape 1' de l'algorithme ci dessous est appliquée au premier coefficient de la liste (voir section 3.2). Par conséquent, dans le cas du calcul d'un S-polynôme, ce coefficient doit être le terme principal (celui qui est le plus important lexicographiquement sur (x_i, y_i)), alors que dans le cas d'une réduction, ce coefficient doit être le terme par lequel la réduction est effectuée. En supposant que cette condition est respectée, le résultat obtenu sera une liste de termes dont l'ordre lexicographique sera décroissant sur (x_i, y_i). Pour l'implémentation, λ, x et y doivent être compris entre 0 et 255. Nous supposons également que λ_0 est non nul.

3.2 Algorithmes

Cette section présente les algorithmes de réduction et de calcul du S-polynôme sur L_1, L_2, listes chainées selon les conventions précédentes.

Algorithme 2 (algorithmes de réduction et S-polynôme)
Entrées: *deux listes* $L_1(\lambda_i^1, x_i^1, y_i^1)$, *et* $L_2(\lambda_i^2, x_i^2, y_i^2)$ *de coefficients triés lexicographiquement. Les premiers coefficients* ($i = 0$) *de chaque liste sont les termes au moyen desquels l'opération est effectuée.*
Sorties: *le résultat de l'opération est une liste* $L = (\lambda_i, x_i, y_i)$. *Cette liste est triée lexicographiquement sur* (x_i, y_i).

- **Étape 1 (Reduction)**: $\Lambda_1 = \lambda_0^2$, $\Lambda_2 = \lambda_0^1$, $u_x^1 = 0$, $u_y^1 = 0$, $u_x^2 = x_{01} - x_{02}$ *et* $u_y^2 = y_{01} - y_{02}$. *Si* $u_x^2 < 0$ *ou* $u_y^2 < 0$ *alors* **Stop**.
- **Étape 1' (S-polynôme)**: $m_x = Max(x_0^1, x_0^2)$, $m_y = Max(y_0^1, y_0^2)$, $u_x^1 = m_x - x_0^1$, $u_x^2 = m_x - x_0^2$, $u_y^1 = m_y - y_0^1$, $u_y^2 = m_y - y_0^2$, $\Lambda_1 = \lambda_0^2$, $\Lambda_2 = \lambda_0^1$.
- **Étape 2** l_1=*Longueur*(L_1), l_2=*Longueur*(L_2)
  - *Pour* $i = 1$ *à* l_1
    $(\lambda_i^3, x_i^3, y_i^3) = (\lambda_i^1 \times \Lambda_1, x_i^1 + u_x^1, y_i^1 + u_y^1)$
  - *Pour* $i = 1$ *à* l_2
    $(\lambda_{i+l_1}^3, x_{i+l_1}^3, y_{i+l_1}^3) = (-\lambda_i^2 \times \Lambda_2, x_i^2 + u_x^2, y_i^2 + u_y^2)$
- **Étape 3**: $(\lambda, x, y) = \mathbf{LexSort}(\lambda_i^3, x_i^3, y_i^3)$.

La procédure **LexSort** trie les éléments de L_3 lexicographiquement sur (x_i, y_i). Si deux éléments de même degré sur x et y existent, les λ correspondant sont ajoutés. L'algorithme 2 décrit ci-dessus peut calculer le S-polynôme de P_1 et P_2 ou la réduction de P_2 par P_1 en utilisant le premier terme de P_1. Cette réduction peut ne pas être possible, tandis que la réduction utilisant un autre terme de P_1 est possible. Ce cas est détecté au cours de l'**Étape 1**. Un signal de sortie $Valid$ sera alors activé. Lors de l'utilisation, l'hôte peut interrompre le calcul en cours et en relancer un nouveau, en commençant avec le coefficient suivant de P_1.

En fait, l'algorithme de réduction décrit ci-dessus est légèrement différent de celui de la définition 2.3 : le polynôme réduit est multiplié par $Lc_\sigma(P_2)$. L'algorithme de Buchberger se poursuit jusqu'à la fin et donne des polynômes qui sont proportionnels (multiplié par une constante) à ceux obtenus par l'algorithme d'origine. D'un point de vue de architectural, l'avantage est que l'on peut éviter la division et utiliser uniquement des multiplications qui peuvent être effectuées de manière séquentielle (d'où un partage des ressources possible).

4 Réseau systolique

4.1 Architecture

Le réseau systolique (voir figure 1) se compose de trois types de processeurs (ou cellules). La première et la dernière cellule sont des cellules particulières qui interviennent dans les

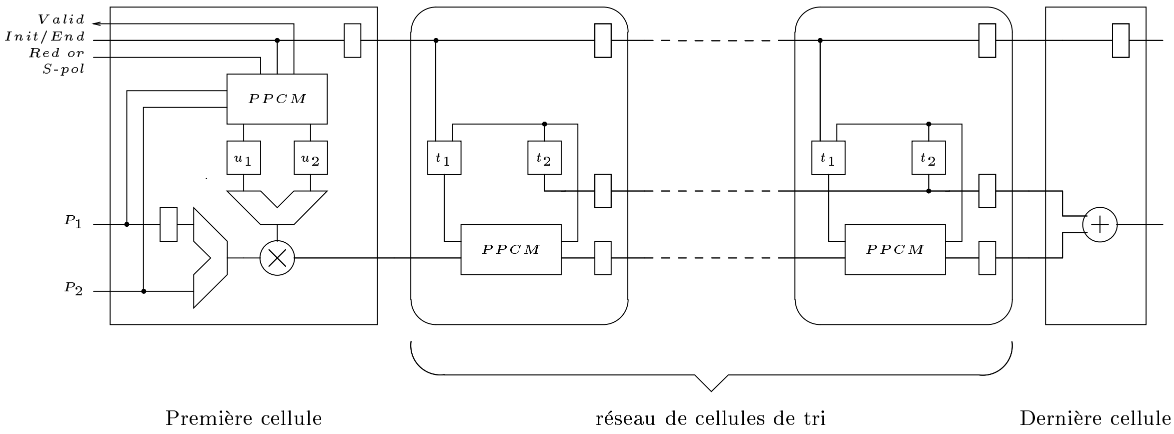

FIG. 1 – *Le réseau systolique : la première cellule calcule les monômes u_1 et u_2, puis multiplie P_1 et P_2 et entrelace les coefficients obtenus dans un seul train de bits. Les autres cellules trient les coefficients. La dernière cellule additionne les coefficients de même degré.*

étapes **1**, **1'** et **2** de l'algorithme 2. Toutes les autres sont identiques. Elles forment un réseau qui est presque un réseau systolique de tri et qui exécute la procédure **LexSort** dans l'algorithme 2. La première cellule détermine les valeurs des monômes u_1 et u_2 qui seront multipliés par les monômes de P_1 et P_2, exécute les multiplications et entrelace les coefficients des deux polynômes. À la sortie de la première cellule, nous obtenons donc un seul train de coefficients de polynômes non triés (par conséquent, les coefficients de P_1 et P_2 entrent dans la première cellule tous les deux cycles).

Les cellules intermédiaires (ou cellules de tri) trient les coefficients dans l'ordre décroissant de leur degré : chaque cellule stocke le premier coefficient qu'elle reçoit, puis elle compare le degré de chaque coefficient au degré stocké. S'il est inférieur au degré stocké, le coefficient est transmis à la cellule suivante. S'il est supérieur, le coefficient interne de la cellule est transmis et le nouveau est stocké. Si les degrés sont égaux, le nouveau coefficient est stocké dans un registre supplémentaire (ce cas ne peut se présenter qu'une fois pour chaque degré).

A la fin du processus, pour vider le réseau, un signal de contrôle spécial est envoyé au réseau et les coefficients sont acheminés vers l'extrémité du réseau. Dans le cas d'un degré identique pour deux coefficients, ceux-ci sont acheminés par deux canaux parallèles. Ces deux coefficients sont additionnés dans la dernière cellule du réseau qui est connectée à l'hôte. L'architecture de ce réseau est décrite en détail dans le document [4].

Notez que dans la figure 1, un canal de communication représente en fait trois canaux (λ, x et y). Par conséquent, le multiplieur de la première cellule représente deux additionneurs d'entiers (un par addition des puissances de x et y) et un multiplieur de corps de Galois qui est implémenté avec un *And*. Le comparateur utilisé par les cellules de tri est un comparateur lexicographique de degrés (qui sont entiers). Enfin, la dernière cellule additionne simplement les deux coefficients de Galois avec un*Xor*.

Le réseau est initialisé par un signal de contrôle *Init* qui marque le début du calcul de

u_1 et u_2. S'il n'est pas possible d'exécuter la réduction, le signal $Valid$ est envoyé à l'hôte. Seuls les calculs de u_1 et u_2 dépendent du fait que l'on calcule une réduction ou un S-polynôme. Les coefficients u_1 et u_2 ne seront plus modifiés jusqu'à ce que le signal *End* soit envoyé dans le réseau, ce qui déclenche l'acheminement des coefficients vers la dernière cellule. Les coefficients de degré supérieur sont extraits en premier.

4.2 Complexité du réseau

Supposons que nous ayons un réseau avec m cellules et des polynômes avec n coefficients. Si $m \geq 2n-2$, le laps de temps qui s'écoule entre l'entrée dans le réseau du premier coefficient et la sortie du dernier coefficient à la fin du processus est égal à $4n+m$. Cependant, on peut commencer un nouveau calcul dès que tous les coefficients ont été entrés. Le délai entre deux calculs successifs est donc égal à $2n$. Le nombre de registres d'entiers est de $2m+8$, 2 additionneurs d'entiers (dans la première cellule) sont nécessaires, ainsi que $2m$ comparateurs d'entiers pour le tri.

Si m est inférieur à $2n$, le calcul doit être partitionné. Supposons que nous ayons deux polynômes de n valeurs monômiales et un réseau de m cellules où $m < 2n-2$, l'hôte recevra donc $2n-2$ coefficients : les premiers $2n-2-m$ ne seront pas triés et les m derniers (tous supérieurs aux $2n-2-m$ premiers) seront triés par ordre décroissant. Pour terminer le processus, les premiers $2n-2-m$ coefficients devront être réintroduits dans le réseau (sans passer par la première cellule), tandis que seront extraits les m premiers résultats. Ce traitement est répété jusqu'à ce que tous les coefficients soient triés. $\lceil \frac{2n-2}{m} \rceil$ passages sont ainsi nécessaires. La phase k dure $2n-2-(k-1).m+1$ cycles pour le tri de m monômes. La dernière phase dure $((2n-2) \ mod \ m)+1$ cycles. Le temps d'attente entre deux calculs successifs est donc égal à :

$$T_l = \begin{cases} 2n & \textbf{si} \quad 2n-2 \leq m \\ a(2n-2) - \frac{a(a-1)}{2}.m + ((2n-2) \ mod \ m) + 1 & \textbf{si} \quad 2n-2 > m \end{cases}$$

où $a = \lfloor \frac{2n-2}{m} \rfloor$.

5 Synthèse de l'architecture

Une fois établie la complexité théorique du réseau, il est intéressant d'aller plus loin pour obtenir un évaluation précise de la complexité de l'implémentation VLSI. Cependant, il n'est pas évident de faire une synthèse logique à partir d'une description de haut niveau. Il est difficile d'obtenir une description valide de l'architecture en VHDL, étant donné que le langage n'est pas vraiment approprié aux contrôles statiques. Cette section explique comment nous avons utilisé les outils Alpha comme aide à la conception.

5.1 Méthodologie

Nous avons utilisé la méthodologie suivante : nous avons commencé par décrire le réseau avec le langage Alpha [17, 18], puis nous avons mis en place une simulation au niveau fonctionnel. Nous avons ensuite traduit automatiquement cette spécification en code VHDL, ce qui peut être synthétisé par des outils commercialisés : le logiciel Compass a été utilisé pour définir la taille du masque et l'évaluation du cycle d'horloge. La synthèse était automatique (sauf pour la procédure de placement, cf. section 5.2). Le cycle d'horloge était minimisé sans contrainte de ressource.

Le choix d'Alpha pour décrire l'architecture présente les avantages suivants :

- Alpha fournit une analyse sémantique approfondie du programme (et donc beaucoup de contrôles statiques) en même temps qu'un moyen adéquat d'exprimer une hiérarchie et des structures répétitives (particulièrement utile pour les réseaux systoliques). Le tableau 1 montre qu'une conception hiérarchisée est nécessaire pour obtenir un dessin de masque grand.

- Les programmes Alpha peuvent être paramétrés. Cette généricité est très importante pour la réutilisation de la conception. Par exemple, les différents réseaux testés dans la section 5.2 ont été obtenus en changeant simplement la valeur du paramètre correspondant à la taille du réseau.

- La simulation du fonctionnement et l'analyse statique sont à la fois plus aisées et plus efficaces qu'un code VHDL. Il a par exemple été très facile de simuler en Alpha l'interface de l'hôte et de vérifier que le protocole d'exception, lorsque la réduction n'est pas possible, était correct.

5.2 Résultats de la synthèse de l'architecture

Les résultats de la synthèse sont présentés dans le tableau 1. La technologie utilisée est $\lambda = 0.4\ \mu$. Le processus complet de conception consiste à générer la liste d'interconnexion, le placement des blocs et le routage des connexions. L'exécution des deux dernières phases peut être hiérarchisée ou non (conception non structurée, par défaut dans Compass). Dans les deux cas, nous avons testé une conception entièrement automatisée et une conception avec procédure de placement manuelle (très rapide en pratique). Le cycle d'horloge est de 26 ns (évaluation par Compass).

Si les évaluations de surface sont réalistes, les évaluations de temps ne sont que de simples estimations (faites par Compass) et ne changent pas d'une conception à l'autre puisqu'ils ne dépendent que de la liste d'interconnexion.

Les tests montrent que les conceptions non structurées sont les plus performantes en termes de surface. Cela peut supprendre, mais il faut avoir à l'esprit que les outils de synthèse sont très performants car optimisés depuis des années et la petite place perdue par la forme (nécessairement rectangulaire) d'une cellule explique cette différence. Par contre, ces outils ne peuvent traiter de gros circuits: si l'on a plus de 20 cellules, Compass est incapable de fournir des résultats corrects. En fait, pour synthétiser rapidement des descriptions importantes, il est indispensable d'avoir une description hiérarchisée avec un placement intelligent (les cellules voisines sont placées les unes à côté des autres).

Avec ce cycle d'horloge, et si $m = 2n = 50$, une réduction est réalisée en 7.8 μs et 1000 réductions prennent 1.3 ms (grâce au pipeline). La surface totale du réseau serait approximativement de 16 mm^2. Lorsque l'on a plus de cellules (pour des polynômes plus importants), les connections étant simples, il est facile de répartir les cellules sur plusieurs puces. La figure 2 montre la disposition d'un réseau de 5 cellules.

Nombre de cellules, type de conception	**surface** (λ^2, $\lambda = 0.4\ \mu m$)	**surface** (mm^2)	**remarques**
5 cellules, placement automatique non structuré	2042×3540	1.16	
5 cellules, placement automatique structuré	3021×3459	1.67	
5 cellules, placement manuel structuré	2194×4638	1.6	
20 cellules, placement automatique non structuré	3802×6591	4	erreur de routage
20 cellules, placement automatique structuré	6026×6856	6.61	
20 cellules, placement manuel structuré	11567×3180	5.88	
50 cellules, placement automatique non structuré	5810×10251	9.5	erreur de routage
50 cellules, placement automatique structuré	8906×10024	14.3	erreur de routage
50 cellules, placement manuel structuré	18420×5401	15.9	

TAB. 1 – *Résultats des expériences. Le cycle d'horloge est de 26 ns. Les tests ont été faits pour des réseaux de 5, 20 et 50 cellules ; la synthèse a été faite pour une description* VHDL *non structurée ou avec une structure donnée en Alpha, et avec placement automatique des blocs amélioré manuellement. Lorsque Compass n'a pas été en mesure de router les connections, l'erreur correspondante est reportée dans la colonne des remarques.*

6 Une étude de cas : le décodage par Chabanne du code Jensen

6.1 Principes du décodage

L'utilisation d'un circuit est ici illustrée par un exemple. Considérons le décodage par Chabanne du code Jensen tel qu'il est exposé dans l'article [9]. C'est un code binaire à deux variables dont les mots font 45 bits de long. Ces mots sont constitués de 17 bits consacrés à l'information de l'utilisateur ainsi que de 28 bits redondants (voir le document [19]). Les mots codés sont des chaînes binaires qui représentent le coefficient des polynômes dans un I idéal de $A = \mathrm{GF}(2)[X,Y]/(X^3+1, Y^{15}+1)$. Ici, I est généré par

$$\begin{aligned} g \;=\;& X^2.Y^{12} + X^2.Y^5 + X^2.Y^3 + X^2 + X.Y^{10} + X.Y^9 + X.Y^6 + X + Y^{14} + Y^{13} + \\ & Y^{11} + Y^8 + Y^7 + Y^4 + Y^2 + Y + 1 \end{aligned}$$

La base de Gröbner de I (calculée par MuPAD) se compose des 4 polynômes suivants :

$$\begin{aligned} p_1 &= X^3 + 1 \\ p_2 &= X^2.Y^4 + X^2.Y + X^2 + X.Y^7 + X.Y^6 + X.Y^4 + X.Y^2 + Y^7 + Y^6 + Y^2 + Y + 1 \\ p_3 &= X.Y^{10} + X.Y^8 + X.Y^5 + X.Y^4 + X.Y^2 + X.Y + X + Y^{10} + Y^8 + Y^5 + Y^4 + \\ & \quad Y^2 + Y + 1 \\ p_4 &= Y^{14} + Y^{13} + Y^{12} + Y^{11} + Y^{10} + Y^9 + Y^8 + Y^7 + Y^6 + Y^5 + Y^4 + Y^3 + Y^2 + \\ & \quad Y + 1 \end{aligned}$$

Le principe de l'algorithme de décodage de Chabanne est le suivant (voir la preuve d'exactitude dans le document [9]). Si le message m est reçu, on le convertit en un polynôme à deux variables M sur $\{X, Y\}$. On a alors deux cas :

- Si M appartient à l'idéal I, M est alors un mot codé du code de Jensen, et il n'y a pas eu d'erreur de transmission.
- Si M n'appartient pas à I idéal, des erreurs ont eu lieu. Dans ce cas nous essayons de trouver le polynôme de I le plus proche de M. Pour cela, on calcule le reste dans I de M et des autres polynômes déduits de M par les automorphismes de I.

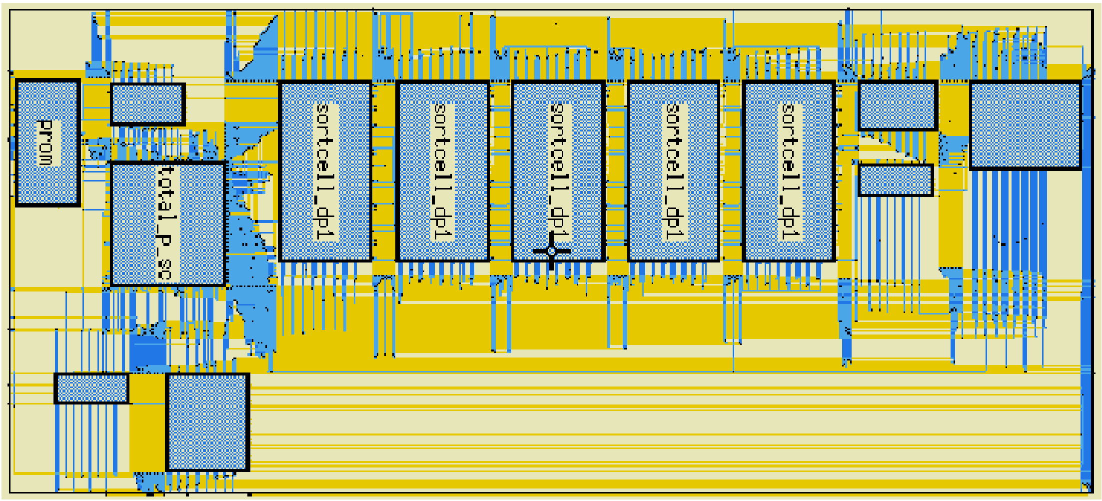

FIG. 2 – *Conception finale, synthèse structurée à 5 cellules, placement automatique des blocs (surface : 1,6 mm2).*

Dans notre exemple, l'ensemble G des automorphismes comprend 320 éléments (obtenus par associations de 3 automorphismes de base). En général, on utilise seulement un sous-ensemble d'automorphismes. Donc, si une erreur survient sur M, pour $\pi \in G$ connu, on calcule le reste R de $\pi(M)$ à partir de I idéal par les réductions de Gröbner. Si la distance de Hamming de R est inférieure à une valeur constante (dite capacité de correction du code), il est possible de corriger M par $\pi^{-1}(\pi(M) + R)$. Si la distance de Hamming est trop importante, la procédure continue par itérations avec l'automorphisme suivant.

Supposons, par example, que le mot émis est $E = g \times (X^2 + X.Y + 1)$ et que le mot reçu est $M = E + X^2.Y^3 + X.Y^{12} + 1$ (erreur de distance de Hamming de 3). Pour le train de bits de M, cela correspond à trois erreurs binaires dans le mot (si on imagine les coefficients dans un tableau, une erreur à la ligne 2, colonne 3, la seconde à la ligne 1, colonne 12 et la dernière à la ligne 0, colonne 0).

Par réduction de Gröbner, le reste de M sur l'idéal du code est :

$$X^2.Y^3 + X.Y^8 + X.Y^7 + X.Y^6 + X.Y^5 + X.Y^3 + X.Y + X + Y^{12} + Y^8 + Y^7 + Y^6 + Y^5 + Y^3 + Y$$

dont la distance de Hamming est de 15. Etant donné que la capacité du code est seulement de 5, la correction n'est pas possible.

Mais, par exemple, le reste de $h_2(M)$ ($h_2(M) = X^2.M$ est un des automorphismes de I) est $X^2 + Y^{12} + X.Y^3$ et on peut corriger le mot reçu par $h_2^{-1}(h_2(M) + X^2 + Y^{12} + X.Y^3) = M + h_2^{-1}(X^2 + Y^{12} + X.Y^3) = M + X^3 + X.Y^{12} + X^2.Y^3 = M + 1 + X.Y^{12} + X^2.Y^3$ et nous retrouvons E.

Comme précisé dans l'article [9], cette méthode nous permet en général de supprimer la plupart des erreurs jusqu'à la limite de capacité de correction du code binaire. Les erreurs irrécupérables dépendent de l'ordre de terme utilisé. Par ailleurs, cette procédure de correction ne rajoute jamais d'erreur.

6.2 Performances du réseau

Afin d'estimer les performances que nous pouvons attendre, nous devons évaluer le nombre de réductions qui seront nécessaires pour calculer un reste. Les bases de Gröbner sont triangulaires (chaque polynôme de la base étant plus grand que le précédent) et nous réduisons donc successivement par rapport à p_1, p_2, p_3 et enfin p_4. Notre polynôme d'entrée aura pour terme principal maximum $X^4.Y^{28}$ (les polynômes de A peuvent avoir une puissance maximale de X égale à 2 et de Y égale à 14 mais les automorphismes peuvent doubler les degrés).

Avec p_1, un terme du polynôme risque d'être effacé. Ce terme doit être sous la forme $X^i.Y^j$ avec $i \geq 3$ et $j \geq 0$ (sinon la réduction n'est pas possible). Les réductions ne génèrent pas de termes avec des degrés supérieurs, le processus est donc incrémental. Le nombre maximum de réductions nécessaires pour arriver à ce que le polynôme soit inférieur à p_1 est de 58 (de X^4Y^{28} à X^3Y^0).

De même, nous pouvons avoir jusqu'à 25 réductions avec p_2 (termes décroissants de $X^2.Y^{28}$ à $X.Y^{31}$). En effet, alors que le terme principal de p_2 est avec Y^4, il y a un terme avec Y^7 et la réduction d'un terme peut introduire le terme $X.Y^{31}$. Il nous faudra également 22 réductions avec p_3 (termes décroissants de $X.Y^{31}$ à $X.Y^9$), et enfin 15 réductions avec p_4 (termes de forme $X^0.Y^i$).

Le processus global de réduction nécessite donc $58 + 25 + 22 + 15 = 120$ réductions. Dans le pire des cas, nous aurons donc à effectuer cette opération pour toutes les images du polynôme, opération multipliée par les 320 automorphismes. Nous devrons donc éxécuter $320 \times 120 = 38400$ réductions.

Supposons que notre réseau possède une horloge cadencée à 40 Mhz (comme proposé dans les tests de la section 5.2). Les exposants maximum sont: 4 pour X et 31 pour Y. Bien que les termes ne devraient jamais être tous présents simultanément, nous pouvons prendre $5 * 32 = 160$ comme nombre de cellules, pour avoir une limite supérieure. Si nous avons suffisamment de cellules (donc plus de 160), alors une nouvelle réduction peut être lancée tous les 320 cycles.

Nous pouvons concevoir le processus maître de sorte que le résultat d'une réduction n'est pas nécessaire avant une nouvelle utilisation du système. De fait, nous pouvons diviser notre ensemble d'automorphismes en deux parts égales et effectuer les réductions alternativement pour chaque sous-ensemble. Nous obtenons donc une étape de pipeline complète.

Il convient maintenant de négliger le temps nécessaire pour vider le réseau après la dernière réduction (ce temps sera en fait utilisé pour la première réduction de la correction suivante). La durée totale d'une réduction est de $320 * 38400/(40 * 10^6 Hz) \simeq 0.32s$. Si un mot codé a une longueur de 45 bits avec une capacité de 17 bits non redondants, le taux de données est de 157 bits/s pour les données brutes et de 55 bits/s pour des données réelles.

Ces performances sont relativement décevantes, mais il faut noter que, dans la pratique, les polynômes n'atteignent pas une longueur de 160 monômes et que le nombre d'automorphismes à traiter jusqu'à ce que le décodage soit possible n'est pas de 320 mais seulement de 20 ou 30. De plus, concernant le cycle d'horloge, il est à noter que les technologies BiCMOS ou AsGa permettraient d'acheminer un volume de données plus élevé.

Pourtant, même avec un taux de rendement de mille, on obtient approximativement 7 Ko par seconde (c'est à dire moins que la plupart des modems) pour un coût élevé. Ceci est compensé par le fait que la procédure de Chabanne est le seul algorithme pratique pour le code Jensen (l'algorithme "naïf" nécessitant une base de données de $2^{45} * 45$ bits). On peut en conclure que : l'implémentation du matériel nécessaire à cette procédure est indispensable en raison de sa complexité, et que les améliorations algorithmiques restent nécessaires pour obtenir des systèmes fiables de décodage du matériel pour ce code.

7 Conclusion

Nous avons décrit dans cet article un réseau systolique dédié aux calculs des bases de Gröbner pour des polynômes à deux variables avec un coefficient en corps de Galois. Les résultats obtenus montrent que l'intégration dans une puce de 16 mm^2 de polynômes d'au moins 50 variables est effectivement possible. Dans une limite de 100 mm^2 (taille très courante pour des puces fabriquées en petites séries), il est possible de traiter des polynômes de 300 coefficients. Les évaluations de temps révèlent une performance d'environ 40 Mbits/s. Ceci n'est qu'une valeur indicative : ce taux de données est une estimation donnée par le compilateur de circuit Compass. Cependant, ces résultats sont concluants, c'est à dire que les dessins de masque générés fonctionneraient à une telle fréquence.

Nous avons également montré qu'un matériel spécialisé était indispensable à l'implémentation du décodage du code Jensen. Une telle implémentation pourra certainement être élaborée efficacement dans quelques années. On peut également noter qu'au cours de ce travail, le circuit a presque entièrement été obtenu automatiquement à partir de la spécification Alpha (seules les optimisations de placement des blocs à l'aide de Compass ont été effectuées manuellement). Cette méthode de synthèse haut niveau permet de définir une architecture à partir d'une spécification haut niveau, et aide à obtenir une conception valable.

Bibliographie

1. B. Buchberger. – *An algorithm for finding a basis for the residue class ring of a*

zero-dimensional polynomial ideal. – PhD thesis, Math. Inst. University of Innsbruck, Austria, 1965.

2. M. Sweedler. – Using Gröbner bases to determine the algebraic and transcendental nature of field extensions: return of the killer tag variables. – In *Proc. 10th International Symposium, AAECC-10*, pages 66–75, 1993.
3. B. Buchberger. – A survey on the method of Gröbner bases for solving problems in connection with systems of multivariate polynomials. – In *Proc. 2nd Int. Symp. on Symbolic and Algebraic Computation by Computers*, pages 7–1–7–15, 1985.
4. Tanguy Risset and Yannick Saouter. – A linear systolic array for the computation of Gröbner basis. – Technical Report Internal Publication 1069, IRISA, Rennes, France, 1996.
5. S. Lösch. – Parallel redundant manipulator based on open and closed normal Assur chains. – In J-P. Merlet and B. Ravani, editors, *Computational Kinematics*, pages 251–260. Kluwer, 1995.
6. D. Lazard. – Stewart platforms and Gröbner basis. – In *Proc. of Advance in Robot Kinematics*, 1992.
7. S.M. Jennings. – A Gröbner basis view of the Welch-Berlekamp algorithm for Reed-Solomon codes. – *IEE Proceedings Communications*, 142(6):349–351, December 1995.
8. X. Chen, I.S. Reed, T. Helleseth, and T.K. Truong. – Use of Gröbner bases to decode binary cyclic codes up to the true minimum distance. – *IEEE Transactions on Information Theory*, 40(5):1654–1661, September 1994.
9. H. Chabanne. – Permutation decoding of abelian codes. – *IEEE Transactions on Information Theory*, 38(6):1826–1829, November 1992.
10. L. Robbiano. – *The Curves Seminar at Queens's - Volume V.* – Queen's University, 1988.
11. A. Poli and L. Huguet. – *Error correcting codes: Theory and applications.* – Prentice-Hall, 1992.
12. R.J. Mac Eliece. – *Finite Fields for Computer Scientists and Engineers.* – Kluwer, 1987.
13. C.S. Yeh, I.S. Reed, and T.K. Truong. – Systolic multipliers for finite fields GF(2^m). – *IEEE Trans. Comp.*, C-33(4):357–360, 1984.
14. H. Brunner, A. Curiger, and M. Hofstetter. – On computing multiplicative inverses in GF(2^m). – *IEEE Trans. Comp.*, 48(8):1010–1015, 1993.
15. H.T. Kung. – Why systolic architectures? – *Computer*, 15(1):37–46, 1982.
16. D.I. Moldovan and J.A.B. Fortes. – Partitioning and mapping algorithms into fixed-size systolic arrays. – *IEEE Transactions on Computers*, 35(1):1–12, jan 1986.
17. D. K. Wilde. – The Alpha language. – Research Report 999, IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France, Jan 1994.
18. P. Le Moenner, L. Perraudeau, S. Rajopadhye, T. Risset, and P. Quinton. – Generating regular arithmetic circuits with AlpHard. – In *Massively Parallel Computing Systems (MPCS'96)*, May 1996.
19. J.M. Jensen. – The concatenated structure of cyclic and abelian codes. – *IEEE Tr. Inform. Theory*, IT-31:788–793, 1985.