INSTITUT DE RECHERCHE EN INFORMATIQUE DE TOULOUSE

RAPPORT
INTERNE
N° 97-53-R

High level synthesis of a co-processor for Gröbner basis computations

Tanguy Risset and Yannick Saouter

CNRS (UMR 5505) - INP - UPS

INSTITUT DE RECHERCHE EN INFORMATIQUE DE TOULOUSE

RAPPORT
INTERNE
N° 97-53-R

High level synthesis of a co-processor for Gröbner basis computations

Tanguy Risset* and Yannick Saouter†

Rapport de recherche N° 97-53-R — Novembre 1997 — 18 pages

Key-words: Gröbner basis, Error correcting codes, Systolic architectures, High level synthesis.

Abstract: *Gröbner basis are a powerful tool with many applications in symbolic computation. In this article, we propose a linear systolic array which can be used as co-processor for efficient computation of Gröbner basis, especially dedicated for the domain of error-correcting codes. The design was made from a high level specification in the Alpha language, which provides many static analysis checks. A case study show how much time can be gain with such a co-processor.*

Résumé : *Les bases de Gröbner ont de nombreuses applications dans le domaine du calcul symbolique. Dans cet article, nous proposons un réseau systolique linéaire qui peut être utilisé comme co-processeur pour le calcul des bases de Gröbner, et tout particulièrement dans le domaine des codes correcteurs d'erreurs. Le circuit a été fait à partir d'une spécification dans le langage de haut niveau Alpha, qui permet de nombreuses vérifications statiques. Les performances du co-processeur sont évaluées autour d'un cas d'études.*

*IRISA, Campus de Beaulieu, F-35042, Rennes Cedex, France. `risset@irisa.fr`
†IRIT, 118 route de Narbonne, 31062 Toulouse Cedex, France. `saouter@irit.fr`

1 Introduction

Gröbner basis were introduced by Bruno Buchberger [2] in 1965 in the context of computations in residue classes of polynomial rings. This concept has many applications in symbolic computations [3, 4], but also in computational non-linear geometry, in automatic geometry proving and in modeling theory (see [5] for a survey of references). However those fields do not require real-time application and thus dedicated hardware is not necessary. Gröbner basis have also applications in robotics [6, 7], but the power of computation needed to fit real-time constraints is far away from what is possible in current technology. In recent years, several applications of Gröbner basis have appeared in the field of decoding error-correcting codes (Reed-Solomon codes [8], BCH and quadratic residue codes [9], abelian codes [10]). We present in this article a linear systolic array for implementation of Gröbner basis algorithms at real-time speed. In section 2, we introduce the context of Gröbner basis and the Buchberger's algorithm. In section 3, we precise the choices for the implementation. In section 4, we describe our linear systolic array. Section 5 is devoted to summarize our implementation results. In section 6, we tackle a practical decoding example and show a case study where hardware implementation is useful.

2 Definitions and theorems

In this section we briefly present the notions about Gröbner basis in rings of multivariate polynomials. We refer the reader to [11] for a full course on the subject. Let $K[X_1, X_2, ..., X_n]$ be the ring of polynomials with n indeterminates and coefficients over the field K.

Definition 2.1 *(terms - term-ordering) Let $T^n = \{X_1{}^{a_1} \dots X_n{}^{a_n} \mid a_i \in \mathbb{N}\}$. The elements of T^n are called terms. Let σ be a strict total ordering on the set of terms T^n. This order will be called a term-ordering if we have: (i) $1 <_\sigma t$ for all term $t \in T^n - \{1\}$ and (ii) if $t_1 <_\sigma t_2$, for t_1 and t_2 in T^n, then $t.t_1 <_\sigma t.t_2$, for all t in T^n.*

For instance, the lexicographical orders and the total degree orders are term-orderings.

Definition 2.2 *(leading term - leading coefficient) Let K be a field and σ a term ordering over $A = K[X_1, X_2, ..., X_n]$. Then every non zero polynomial $f = f(X_1, X_2, ..., X_n)$ can be written in a unique way as $f = \sum_{i=1}^{r} c_i T_i$, with $c_i \in K^*$ and $T_1 >_\sigma T_2 >_\sigma \dots >_\sigma T_r$. We denote $c(f, T_i) = c_i$, $Lt_\sigma(f) = T_1$ is called the leading term of f with respect to σ, and $Lc_\sigma(f) = c_1$ is called the leading coefficient of f with respect to σ.*

Definition 2.3 *(reduction) Let f and g be two polynomials of A and $F = \{f_1, ..., f_t\} \subset A - \{0\}$. We say that f reduces in one step as g along F if* *there exists* $f_i \in F$ *and* $T \in T^n$ *such that* $g = f - (c/Lc_\sigma(f_i)).T.f_i$, *where* $c = c(f, T.Lt_\sigma(f_i))$. *This notion extends immediately by transitive closure and we denote it* $\xrightarrow{F}$.

Such a relation $\xrightarrow{\mathrm{F}}$ is *confluent* if, whenever we have two possible reductions for x: $x \xrightarrow{\mathrm{F}} y_1$ and $x \xrightarrow{\mathrm{F}} y_2$, there exists z such that $y_1 \xrightarrow{\mathrm{F}} z$ and $y_2 \xrightarrow{\mathrm{F}} z$.

Definition 2.4 *(Gröbner basis) Let* $F = \{f_1, ..., f_t\} \subset A - \{0\}$, *and* (F) *the ideal generated by* F. *If the relation* $\xrightarrow{F}$ *is confluent then* F *is said to be a Gröbner basis of* (F).

The reader will referred again to [4, p B23] to have a proof of the fact that any ideal of multivariate polynomial has Gröbner basis. An important property is the following one:

Theorem 2.1 *Let* $F = \{f_1, ..., f_t\} \subset A - \{0\}$ *be a Gröbner basis of the ideal* (F). *Let* $f \in A$, *then we have* $f \in (F) \leftrightarrow f \xrightarrow{F} 0$.

Another basic operation between multivariate polynomials is the following one:

Definition 2.5 *(S-polynomial) Let* f_1 *and* f_2 *be two polynomials of* $A = K[X_1, X_2, ..., X_n]$, *and* σ *a term-ordering. Then the S-polynomial of* f_1 *and* f_2 *will be the polynomial:*

$$f = Lc_\sigma(f_2).u_1.f_1 - Lc_\sigma(f_1).u_2.f_2$$

where $u_1 = lcm(Lt_\sigma(f_1), Lt_\sigma(f_2))/Lt_\sigma(f_1)$ *and* $u_2 = lcm(Lt_\sigma(f_1), Lt_\sigma(f_2))/Lt_\sigma(f_2)$ *(lcm denotes the least common multiple of two polynomials).*

Gröbner basis, given a set of generators $F = \{f_1, ..., f_t\} \subset A - \{0\}$ of the ideal (F), are computed by the Buchberger's algorithm whose correctness proof is to be found in [2]:

Algorithm A
Input: *F, σ.*

- ***Initialization:*** *Set* $G = F$ *and* $B = \{\{f_1, f_2\} \mid f_1, f_2 \in F, f_1 \neq f_2\}$.
- ***Step 1:*** *If* $B = \emptyset$, *then* ***Stop***.
- ***Step 2:*** *Choose* $\{f_1, f_2\} \in B$ *such that* $lcm(Lt_\sigma(f_1), Lt_\sigma(f_2))$ *is minimal with respect to* σ. *Set* $B = B - \{f_1, f_2\}$.
- ***Step 3:*** *Compute* $h = S(f_1, f_2)$ *and compute* h' *such that* $h \xrightarrow{G} h'$.
- ***Step 4:*** *If* $h' \neq 0$, *then set* $B = B \cup \{\{g, h'\} \mid g \in G\}$ *and* $G = G \cup \{h'\}$. *Goto* ***Step 1***.

Output: *G.*

The final set G is a Gröbner basis for the ideal (F) with respect to the term-ordering σ. In this algorithm the iteration set G always grows. In fact, this set could be iteratively reduced by checking for any polynomial $f \in G$, whether it reduces into 0 along the set $G - \{f\}$. In this paper, we do not tackle the problem of reducing the number of polynomials generated by Buchberger's algorithm, we will show how to implement the basic operations of reduction and of computation of the S-polynomial in a linear systolic array.

3 Using Gröbner basis for error correcting codes

Gröbner basis are used in multi-variable codes (semi simple codes, abelian codes, see [12] for a survey). These codes manipulate elements of $K[X_1, \ldots, X_n]/I$ where K is a finite field (or Galois field, $GF(p^n)$ see [13]), and I is an ideal of $K[X_1, \ldots, X_n]$. The reduction operator and the computation of the S-polynomial are very often used when computing in $K[X_1, \ldots, X_n]/I$.

As many applications of Gröbner basis have $K = GF(2)$ (see section 6), we will limit our study to coefficients in $GF(2)$. This, of course, simplify the implementation of the operateurs of the Galois field: *Xor* for the addition and *And* for the multiplication. The version of this architecture extended to $K = GF(p^n)$ has been study in [5], and the principles are quite similar (the problem of deriving efficient operators for computations in Galois fields has already been studied and some regular [12, 14] and irregular [15] architecturesg have been proposed).

We will describe an architecture for $GF(2)[X, Y]$. The overall principles of the architecture can easily be extended to a greater number of indeterminates, the only things that will change are the operators related to the term ordering which, in this paper, is chosen to be the lexicographic ordering.

3.1 Implementation choices

The target architectural model will be a systolic linear unidirectional array. This model is synchronous, communications with the host are known statically [16]. The array is unidirectional (all communication channels go in the same direction), thus the algorithm can be partitioned and completed in several passes if the array is not big enough (LPGS partitioning [17]). Finally, this model can be implemented either in ASIC or FPGA technology.

We have choosen to represent a polynomial as a linked list of triplets (λ_i, x_i, y_i), where λ_i is boolean and x_i, y_i are integers. The triplet (λ_i, x_i, y_i) will code the monomial $\lambda_i X^{x_i} Y^{y_i}$. The first coefficient of the list is the one on which will be performed step 1 or step 1' of the algorithm (see section 3.2). Hence, in the case of the computation of a S-polynomial, this coefficient must be the leading term (the lexicographically greatest one on (x_i, y_i)), while in the case of a reduction, this coefficient must the term by which the reduction is performed.

We will suppose that this condition is respected, hence we will provide a result with terms ordered in lexicographical decreasing order on (x_i, y_i). For the implementation, we have chosen λ, x and y to be between 0 and 255. We also suppose that the coefficient λ_0 is non null.

3.2 Algorithms

In this section we rewrite the algorithms of reduction and of computation of the S-polynomials with the notation of linked lists and the previous conventions.

Algorithm B *(algorithms of the reduction and S-polynomial)*
Input: *two lists* $L_1(\lambda_i^1, x_i^1, y_i^1)$, *and* $L_2(\lambda_i^2, x_i^2, y_i^2)$ *of lexicographically sorted coefficients. The first coefficients (i = 0) of each list are the terms with which the operation is performed.*
Output: *a list* $L = (\lambda_i, x_i, y_i)$ *result of the operation. This list is lexicographically sorted on* (x_i, y_i).

- ***Step 1 (Reduction):*** $\Lambda_1 = \lambda_0^2$, $\Lambda_2 = \lambda_0^1$, $u_x^1 = 0$, $u_y^1 = 0$, $u_x^2 = x_{01} - x_{02}$ *and* $u_y^2 = y_{01} - y_{02}$. *If* $u_x^2 < 0$ *or* $u_y^2 < 0$ *then* ***Stop***.
- ***Step 1' (S-polynomial):*** $m_x = Max(x_0^1, x_0^2)$, $m_y = Max(y_0^1, y_0^2)$, $u_x^1 = m_x - x_0^1$, $u_x^2 = m_x - x_0^2$, $u_y^1 = m_y - y_0^1$, $u_y^2 = m_y - y_0^2$, $\Lambda_1 = \lambda_0^2$, $\Lambda_2 = \lambda_0^1$.
- ***Step*** *2* l_1*=Length(*L_1*),* l_2*=Length(*L_2*)*

  - *For* $i = 1$ *to* l_1
    $(\lambda_i^3, x_i^3, y_i^3) = (\lambda_i^1 \times \Lambda_1, x_i^1 + u_x^1, y_i^1 + u_y^1)$

  - *For* $i = 1$ *to* l_2
    $(\lambda_{i+l_1}^3, x_{i+l_1}^3, y_{i+l_1}^3) = (-\lambda_i^2 \times \Lambda_2, x_i^2 + u_x^2, y_i^2 + u_y^2)$
- ***Step*** *3:* $(\lambda, x, y) = LexSort(\lambda_i^3, x_i^3, y_i^3)$.

The **LexSort** procedure sorts the elements of L_3 lexicographically on (x_i, y_i). If two elements with same degree on x and y are encountered, the corresponding λ's are added. Algorithm B described above can compute the S-polynomial of P_1 and P_2 or the reduction of P_2 by P_1 using the first term of P_1. This particular reduction may not be possible, while the reduction using another term of P_1 is possible. This case can be detected in **Step 1**. The output signal *valid* indicates the host if the current reduction was not possible. With this mechanism, the host can stop the current computation and send again the two polynomials in the array, starting with the next coefficient of P_1.

In fact the algorithm of reduction described above is not the one of the definition 2.3: the reduced polynomial is multiplied by $Lc_\sigma(P_2)$. The Buchberger's algorithm still ends and leads to polynomials proportional with constants to the ones obtained by the original algorithm. From an architectural point of

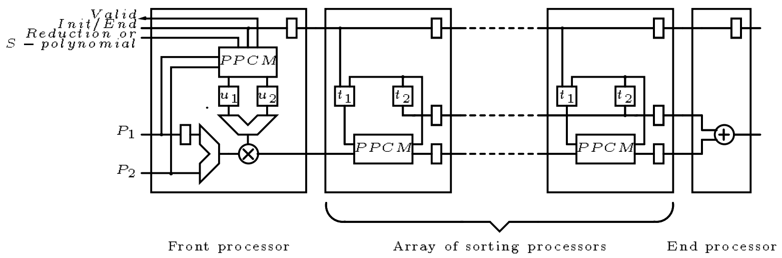

Figure 1: The systolic array: the first cell computes the monomials u_1 and u_2, then multiply P_1 and P_2 and interleave the resulting coefficients in one single stream. The other cells add the two results by first sorting the coefficients. The last cell adds coefficients of same degree.

view this new expression of the reduction avoid the operation of division and involves two multiplications which could be performed in a sequential fashion (hence sharing resource).

Finally, note that these algorithms might directly be implemented with FIFO files, but we choose a systolic design, since most of data exchanges with the host are known statically [16].

4 The systolic array

4.1 Architecture

The systolic array (see figure 1) is composed of three types of processors (or cells). The first and the last cell are special ones and will be dedicated to the steps **1**, **1'** and **2** of the algorithm B. All the other ones are identical, they compose an array which is almost a systolic sort array and will be execute the **LexSort** procedure in algorithm B.

The first cell determines the value of the monomials u_1 and u_2 to be multiplied by the monomials of P_1 and P_2, interleaves the coefficients of the two polynomials and executes the multiplications. Thus at the output of the first cell, we have only one stream of unsorted polynomial coefficients (a consequence of this merge is that the coefficients entering the first cell should arrive every two clock ticks).

The middle cells (or sorting cells) sort the coefficients in decreasing degree order. This is very close to a classical systolic sort: each cell stores the first coefficient it receives, after that, for each coefficient received, if it has a lower degree than the one stored, it is transmitted to the next cell, if the degree is higher, the internal coefficient is transmitted and the current coefficient is stored and if the degrees are equal, the current coefficient is stored in an additional register (this case can appear only once for each degree, when the two modified input polynomials have a coefficient of this degree).

The sorted coefficients of the polynomial are stored in the array. To empty the array, a control signal indicating the end of the computation is sent in the array, and the coefficients are output toward the end of the array. As there may be some degrees for which two coefficients have been stored, they are sent in parallel on two channels (which explain why there are two channels for the coefficients of one polynomial). These two coefficients are added in the last cell of the array which is connected to the host. The internal architecture of this array is precisely described in [5].

Note that one communication channel on figure 1 actually stands for three channels (for λ, x and y). Hence the multiplier of the first cell represent two integer adder (adding the powers for x and y) and one Galois multiplier which is implemented with an *And*. The comparator in the sorting cells is a lexicographic comparator on the degree (which are integral). Finally, the last cell simply add the two Galois coefficients with an *Xor*.

The array is initiated with the *Init* control signal which starts the computation of u_1 and u_2. If it is not possible to execute the reduction asked, a *Valid* signal indicates it to the host which can send the next input. The computation of u_1 and u_2 is the only part which is affected by the fact that the whole treatment represent a reduction or the computation of a S-polynomial. The coefficients u_1 and u_2 will be used during all other steps until the *End* signal is sent in the array (together with the last coefficients) which initiates the output of the coefficients towards the last cell. The first coefficient to be output is the coefficient stored in the first sorting cell, it is the coefficient of higher degree.

4.2 Complexity of the array

The number of cells of the array is not fixed. We suppose that we are given an array of with m cells and polynomials with n coefficients. If we suppose $m \geq 2n-2$, the time elapsed between the input of the first coefficient in the array and the output of the last coefficient at the end is: $4n+m$. However, a new computation can be started as soon as all the coefficients have been input, thus the delay between two successive computations is $2n$. The number of integral latches is $2m+8$, we need 2 integer adders (in the first cell), $2m$ integer comparators for lexicographic comparison.

4.3 Partitioning

If m is smaller than $2n$, the computation will be performed with partitioning.

Suppose that we are given two polynomials of n monomial values and if we have m cells with $m < 2n-2$, the host will receive as result $2n-2$ coefficients. The first $2n-2-m$ received coefficients will be unsorted and the last m ones will be all greater than the previous ones and sorted in decreasing order. In order to finish the computations the first $2n-2-m$ coefficients will have to be stored in the host and reintroduced in the sorting array (avoiding the first cell) while flushing out the first m results. This process is repeated until all coefficients are sorted. The process will then take $\lceil\frac{2n-2}{m}\rceil$ passes. The $k-th$ phase will last $2n-2-(k-1).m+1$ clock ticks for the sorting of m items and the last phase will last $((2n-2)\ mod\ m)+1$ clock ticks. Thus the latency, i.e. the time between two successive computations is, in terms of clock ticks:

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

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

5 Architectural synthesis

Once the theoretical complexity of the array is established, it is interesting to go further into the design process in order to obtain precise complexity report for a VLSI implementation. However, logic synthesis from high level description is a non trivial process. Obtaining a valid architectural description in VHDL is difficult because the language is not well suited for static checks. This section explains how we have used the Alpha tool as a help for the designer.

5.1 Methodology

The methodology we have used is the following one: first we described the array with the Alpha language [18, 19] which allow high level description, then we simulated it at the functional level. Then we automatically translated this specification into VHDL code which can be synthesized by commercial tools.

The VHDL specification was synthesized with the Compass software and we obtained reports on the size and clock cycle of the layout. The synthesis was done without manual editing (except for the placement procedure, see section 5.2). We asked for the best clock cycle without any resource constraints. The manual optimization of the layout would certainly improve the performances but it is out of the scope of the paper, we present these results because they give an idea of the real complexity and speed of the array.

The advantages of choosing Alpha for describing the architecture are the following ones:

- Alpha provides simultaneously a strong semantical analysis of the program (hence lots of statics checks) and a convenient way to express hierarchy and repetitive patterns (especially useful for systolic arrays). In table 1, it is shown that hierarchical design is necessary to obtain large layout.

- Alpha programs can be parameterized, this genericity is very important to re-use the design. For instance, the different arrays tested in section 5.2

Number of cell, design type	area report ($\lambda = 0.4\mu$)	surface (mm^2)	remarks
5 cells, flat, automatic placement	2042×3540	1.16	
5 cells, structured,automatic placement	3021×3459	1.67	
5 cells, structured, manual placement	2194×4638	1.6	
20 cells, flat, automatic placement,	3802×6591	4	routing failed
20 cells, structured, automatic placement,	6026×6856	6.61	
20 cells, structured, manual placement	11567×3180	5.88	
50 cells, flat, automatic placement,	5810×10251	9.5	routing failed
50 cells, structured, automatic placement,	8906×10024	14.3	routing failed
50 cells, structured, manual placement	18420×5401	15.9	

Table 1: Experimental results, the clock cycle is 26 ns. The tests have been done for arrays of 5, 20 and 50 cells, synthesis have been made for a flat (unstructured) VHDL description or with the structuring given in Alpha, and both automatically and with manual improvement of the placement of the blocks. When indicated, Compass was not able to route the wires.

where obtained by changing the value of the parameter in only one command of the design flow.

- Functional simulation and static analysis are both easier and more efficient than in VHDL code. For instance it was very easy in Alpha to simulate the interface to the host and to check that the exception protocol when reduction is not possible was correct.

5.2 Architectural synthesis results

The results of the synthesis are presented in table 1. The technology used is $\lambda = 0.4\mu$. The complete design process consists of the synthesis (generation of a net-list), the placement of the blocks and the routing of the wires. The last two phases can be done hierarchically or not (flat design, default in Compass). In both cases, we have tested a complete automatic design and a design where the placement procedure was performed by hand (which is very quick in practice). The clock cycle found (timing report) was 26 ns.

The area reports are realistic as they represent the material used. The timing reports are just evaluation (done by Compass): they do not change from one design to the other as this evaluation depends only on the net-list which is the same in each design. The clock cycle can be optimized by adding latches on the design of the systolic array. These timing optimizations are not tackled in this paper, just note that such optimizations can be easily performed with Alpha as this environment implement many automatic program transformations.

The tests show that, the best area performances are obtained with flat designs. But, as soon as there are more than 20 cells, Compass does not provide

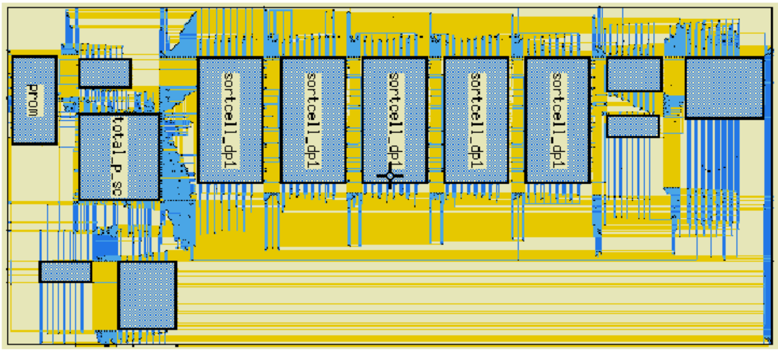

Figure 2: Final design, structured synthesis with five cells, automatic placement of the blocs (surface: 1.6 mm^2).

correct results. Actually, to synthesize quickly big descriptions, it is mandatory to have a hierarchical description with intelligent placement (neighboring cells are placed side by side).

With this clock cycle, and if $m = 2n = 50$, the complete computation is performed in 7.8 μs and 1000 computations can be done in 1.3 ms. The total array of such chip would be around 16 mm^2. If one wants more cells (for bigger polynomials), it is easy to spread cells on several chips as connections between successive cells are simple. The layout of an array of 5 cells is represented on figure 2.

6 A case study: Chabanne's decoding of Jensen code

In this section, we present the performances of Chabanne's decoding algorithm for two-dimensional codes [10].

6.1 Principles of the decoding

We illustrate here the use of the circuit by an example. We consider Chabanne's decoding for Jensen's code of article [10]. This code is a binary bivariate code whose words are 45 bits large. These words are made from 17 bits of user information together with 28 redundancy bits (see [20]). Codewords are binary chains representing coefficient of polynomials in an ideal I of $A = GF(2)[X,Y]/(X^3+1, Y^{15}+1)$. Here, I is generated by:

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

as may be computed by Reed's formula [12], when given the 28 zeroes of the code. The Gröbner base of the ideal I (computed by MuPAD [21]) is composed of the 4 following polynomials:

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

The principle of Chabanne's decoding algorithm is the following one (the exact proof of correctness of this decoding scheme is described in [10]). If the message m is received, one converts it into a bivariate polynomial M over $\{X, Y\}$. There is then two cases:

- If M belongs to the ideal I, then M is a codeword of the Jensen's code, no transmission errors have occurred.
- If M do not belongs to I, errors have occurred. In that case, we try to find the *closest to M* polynomial in I. This is done by successively computing, along I, the remainder of M and other polynomials obtained from M by taking the image by some automorphisms of I.

In our particular ideal I, the set G of automorphisms is composed of 320 elements (which can be obtained with the help of 3 basic automorphisms). In general, the set of automorphisms do not need to be the exact entire set of automorphisms.

Hence, when an error occurs on M, for $\pi \in G$ given, we compute the remainder R of $\pi(M)$ on the ideal I by Gröbner reductions. If the Hamming weight of R is less than constant value (called the power of correction of the code), we can correct M by $\pi^{-1}(\pi(M) + R)$. If the remainder has a Hamming weight too large, the procedure goes on iteratively with other permutations.

Example 1 *Let us suppose that we emitted the word* $E = g \times (X^2 + X.Y + 1)$ *and that an error, occurred such that the received word was* $M = E + X^2.Y^3 + X.Y^{12} + 1$ *(error of Hamming weight 3).*

For the bit-stream of M*, this corresponds to three binary errors in the word (one at line 2, column 3, the second one at line 1, column 12 and the last one at line 0, column 0 if one imagine the coefficients in a matrix).*

By Gröbner reductions, we find that the remainder of M *on the ideal of the code is:*

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

whose Hamming weight is 15. Since the code can only correct up to 5 errors, correction is not possible here.

But, for instance, the remainder of $h_2(M)$ *(*$h_2(M) = X^2.M$ *is one of the automorphisms of* I*) is* $X^2 + Y^{12} + X.Y^3$ *and thus we can correct the received word by* $h_2^{-1}(h_2(M) + X^2 + Y^{12} + X.Y^3) = M + h_2^{-1}(X^2 + Y^{12} + X.Y^3) = M + X^3 + X.Y^{12} + X^2.Y^3 = M + 1 + X.Y^{12} + X^2.Y^3$*. Thus we recover the error committed during transmission.*

As pointed out in article [10], this method enables to erase most of the errors up to the correction power of the binary code. However, depending on the ordering of terms used, some errors may resist. On the opposite point of view, we are ensured that this correction procedure never add errors.

6.2 Performances of the array

In order to estimate the performances we can achieve, we need to know how many reductions will be necessary to compute a remainder. The Gröbner bases are in triangular form (each polynomial of the basis is greater with respect to the order than the following one) and thus we reduce successively with respect to p_1, p_2, p_3 and at last p_4. Our input polynomial will have $X^4.Y^{28}$ as maximum leading term (polynomials of A can have power of X up to 2 and of Y up to 14 but the automorphisms may double the degrees).

With p_1, one term of the polynomial might be erased. This term needs to be of the form $X^i.Y^j$ with $i \geq 3$ and $j \geq 0$ (otherwise the reduction is not possible). Reductions do not generate terms with higher degrees hence, the process is incremental. The maximum number of reductions necessary to render the polynomial inferior to p_1 is 58 (from X^4Y^{28} to X^3Y^0)

Similarly, we may have up to 25 reductions with p_2 (terms decreasing from $X^2.Y^{28}$ to $X.Y^{31}$). Indeed, while the leading term is with Y^4, there is a term of p_2 with Y^7 and thus the reduction of a term can introduce the $X.Y^{31}$ term. We need also 22 reductions with p_3 (terms decreasing from $X.Y^{31}$ to $X.Y^9$), and 15 reductions with p_4 (terms of the form $X^0.Y^i$).

Thus the global reduction process takes 58+25+22+15 = 120 reductions. We have to perform that for the images of the polynomial by the 320 automorphisms of the group hence, we must execute $320 \times 120 = 38400$ reductions.

Suppose that our array has a 40 Mhz clock (as suggested by the tests of section 5.2). The maximum exponent for X is 4 and for Y is 31. Although all the terms should never be simultaneously present we can take $5 * 32 = 160$ as number of cells to have an upper limit. If we have enough cells (i.e. more than 160), then every 320 clock ticks a new reduction can be initiated.

We can design the managing code in such a way that the result of a reduction is not needed at the following use of the device. Indeed, we can split our set of automorphism in two equal parts and performed reductions alternately for each subset. Thus we have a complete pipeline task achievement.

Now we neglect the time to clear out the array after the last reduction (in fact, this time will be used for the first reduction of the next correction) and so the global time for reduction is $320 * 38400/(40 * 10^6 Hz) \simeq 0.32s$. Since a codeword is 45 bits long with a capacity of 17 irredondant bits, the data rate 157 bits/s for raw data and 55 bits/s for real data.

These performance may seem quite deceiving, however it has to be noted that this is a case at worst: in practical uses, the polynomials are not 160 monomials long and the number of automorphisms to consider until to be able to decode the word is not 320 but only around 20 or 30. As for the working frequency, the use of BiCMOS or AsGa technology would enable to address much higher data rate.

In spite of this and even if we gain a thousand ratio performances, it would lead to roughly 7 Kbytes per second (i.e. less than most modems) for a high cost. This remark is balanced by the fact that Chabanne's procedure is the only practical algorithm for Jensen's code (the rough algorithm would require a $2^{45} * 45$ bits large database and is thus unrealistic) and so we can conclude that, first, hardware implementation of this procedure is necessary because of its complexity, second, algorithmic improvements have still to be made to have an efficient hardware decoding devices for this code.

7 Conclusion

In this article, we have described a systolic array dedicated to computations of Gröbner basis for two variables polynomials with coefficient in Galois field. The results obtained show that integration is effectively possible for polynomials of at least 50 variables into a chip of $16mm^2$. If we assume a limit of $100mm^2$ (quite common for small series chips), it would then be possible to deal with 300 variables polynomials. The timing reports exhibit performance of approximatively 40 Mbits/s. This value is indicative: this data rate is an estimation given by the silicon compiler Compass which generally minimizes the potential of its own layout. However, these performances are effective (i.e. the derived layouts would work at such frequency).

We have also shown that specialized hardware is mandatory to implement the decoding of Jensen code, such implementation could certainly be be built efficiently in a few years. It has to be noted that this circuit has been obtained nearly entirely automatically (only optimizations of block placing with Compass have been manually made) from the Alpha specification. This high level synthesis process allow to specify the architecture from high level specification and help the designer to obtain a valid design.

References

[1] F.J. Mac Williams and N.J.A. Sloane. *The Theory of Error-Correcting Codes*. Amsterdam: North-Holland, 1977.

[2] 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.

[3] 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.

[4] 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.

[5] Yannick Saouter Tanguy Risset. A linear systolic array for the computation of gröbner basis. Technical Report Internal Publication 1069, IRISA, Rennes, France, 1996.

[6] 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.

[7] D. Lazard. Stewart platforms and Gröbner basis. In *Proc. of Advance in Robot Kinematics*, 1992.

[8] 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.

[9] 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.

[10] H. Chabanne. Permutation decoding of abelian codes. *IEEE Transactions on Information Theory*, 38(6):1826–1829, November 1992.

[11] L. Robbiano. *The Curves Seminar at Queens's - Volume V*. Queen's University, 1988.

[12] A. Poli and L. Huguet. *Error correcting codes: Theory and applications.* Prentice-Hall, 1992.

[13] R.J. Mac Eliece. *Finite Fields for Computer Scientists and Engineers.* Kluwer, 1987.

[14] 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.

[15] H. Brunner, A. Curiger, and M. Hofstetter. On computing multiplicative inverses in GF(2^m). *IEEE Trans. Comp.*, 48(8):1010–1015, 1993.

[16] H.T. Kung. Why systolic architectures? *Computer*, 15(1):37–46, 1982.

[17] 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.

[18] D. K. Wilde. The Alpha language. Research Report 999, IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France, Jan 1994.

[19] 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.

[20] J.M. Jensen. The concatenated structure of cyclic and abelian codes. *IEEE Tr. Inform. Theory*, IT-31:788–793, 1985.

[21] The MuPAD Group (Benno Fuchssteiner et al.). *MuPAD User's Manual.* John Wiley and sons, Chichester, New York, first edition, March 1996.

INSTITUT DE RECHERCHE EN INFORMATIQUE DE TOULOUSE

UPS, 118 route de Narbonne. 31062 Toulouse cedex 04
Tél. +33 (0) 561 556 765 / fax. +33 (0) 561 556 258

ENSEEIHT, 2 rue Camichel. 31071 Toulouse cedex
Tél. +33 (0) 561 627 862 / fax. +33 (0) 561 588 209