Analysis of a Decoding Approach for Goppa Codes

[1]LIRIDA ALVES DE BARROS NAVINER and [2]ZOUHAIR BELKOURA
[1] Département Communications et Electronique
Ecole Nationale Supérieure des Télécommunications
46, rue Barrault – 75637 – Paris CEDEX 13 - FRANCE
[2] Department of Electrical and Electronic Engineering
Imperial College of Science, Technology and Medicine
London SWT 2BT - ENGLAND
lirida.naviner@enst.fr http://www.enst.fr

Abstract: — Algebraic-geometry family of codes contains sequences with excellent asymptotic behaviour, but few work have been reported in the literature concerning their hardware implementation. In this paper, we investigate an algorithm for decoding AG codes under the hardware feasibility point of view. We modify the original strategy in order to obtain a new structure more suitable for hardware implementation.

Key Words: —Algebraic-geometry codes, linear codes, channel decoding, hardware implementation,VLSI.

1 Introduction

Algebraic-geometry (AG) codes are linear codes for error correction that are constructed by means of algebraic curves over finite fields. AG codes were developed by Goppa [1, 2]. At the time of publishing, it was not evident that these codes have better asymptotic properties than existing ones, because the necessary algebra had not been studied at the time. His publication led to the open questions being answered and subsequently the advantages of AG codes were shown. Further work in developping AG codes was done by Tsfasman, Vladuts, Zink, Justesen and others during the 1980s. Their results gave AG codes that exceeded a number of constraining bounds [3]. Garcia and Stichtenoth have simplified the construction and proofs of AG codes significantly with their papers in the 1990s [4, 5]. AG codes are asymptotically good, i.e. their rate $R>0$ and relative distance $\delta>0$. AG codes don't have a strict interdependence between alphabet size q and code length n. For some powerful codes, the alphabet has to grow with codeword length such that $q>n$. This is not the case with AG codes [6].

In despite to their good properties, few work have been reported in the literature concerning the hardware implementation of AG codes. In this work, we will investigate an algorithm for AG codes under the hardware feasibility point of view.

The paper is organized as follows. In Section 2 we present some basics about construction and decoding of codes based on algebraic geometry. In Section 3 we then describe the Berlekamp-Massey-Sakata (BMS) algorithm and hardware implementation issues of a such algorithm are investigated. Also, some modifications of the Sakata's strategy are proposed in order to obtain a structure more suitable for hardware implementation. Finally some conclusions and perspectives of this work are outlined in section 4.

2 Construction and Decoding of Algebraic-geometry Codes

Current research work on algebraic geometry codes centres around the design of AG codes and their decoding [7]. The recent work of Xing and Ling uses curves defined over an extension of $\mathbb{F}_q$ in order to construct good codes for small q [8]. This overcomes the problem of having only few rational points on curves for small q. In [9] [10], in addition to rational points, the authors make use of places of small degree on the curve. They also draw upon the idea of concatenated codes. Again, this targets the problem of finding good codes for small q. These and other works on AG codes and variants thereof allow to find many improvements to Brouwer's table. In fact, Brouwer publishes a table of the best linear codes available, many of which are AG codes [11]. Advances in code construction have been made by studying different types of curves, notably Hermitian curves, Klein curves and Elliptic curves [12].

Decoding algorithms are applied to perform error-correction to received codewords. For any error correction, error location and error evaluation needs to be performed. The Berlekamp-Massey (BM) algorithm was generalised by Sakata to be applied to AG codes [13]. An efficient algorithm for computing unknown syndromes using majority voting schemes, Hankel-block

matrices and Gaussian elimination was presented in [14]. O'Sullivan published a generalisation of the key equation to Hermitian codes, a subclass of AG codes [15]. Closer to the implementation side, Kotter presents a modified BM algorithm in [16]. A useful overview of other advances and algorithms can be found in [17].

3 Strategy Description of the Goppa Codes Decoder

This section deals with the strategy implementation proposed in [18, 19][1].

The structure supposes $m+1$ processors $P(l)$, where m is the dimension of the space $L(mP_\infty)$, given by the particular AG code C and the underlying curve χ. Each processor consists of ρ^2 basic cells $C_{i,j}$, where ρ is the first nongap in the nongap sequence of χ. Fig. 1 presents a simplified diagram for the arrangement of the processor and the cells. Control logic is not shown in the processor.

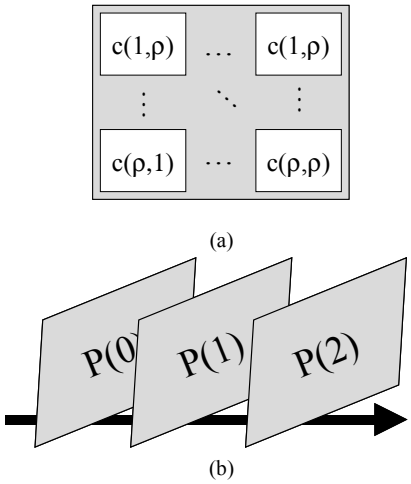

Fig 1. (a) A basic processor. (b) Processor arrangement.

The architecture is systolic in the sense that the processors only communicate via the basic cells. A cell $C_{i,j}$ in $P(l)$ communicates with cells $C_{i,j}$ and/or $C_{i+1,j}/C_{i-1,j}$ in the processors $P(l-1)$ and $P(l+1)$. The data communication is local and cells of the same processor do not pass data between each other.

From the diagram, it is easy to see that the proposed architecture is three-dimensional, an arrangement which is highly unusual if not unheard of for simple VLSI architectures. The reason to subdivide the structure in different processors is their common control signals and the calculation of those. These control signals are calculated as a function of l and i.

[1] The work on implementing Sakatas architecture was done with some help of Professor Shojiro Sakata (University of Electro-Communications, Tokyo, Japan) himself, who showed the kindness to give a few pieces of advice

The desired output of the BMS algorithm is a matrix of polynomials $\breve{f}_{i,j}$ which give error locating polynomials. Although these are multivariate polynomials, Sakata introduces a notation which allows to write each polynomial as a vector containing the coefficients $\in$ GF(8). This vector has size k for a polynomial of degree k and consequently the output produced is a three-dimensional arrangement of elements of GF(q) which can be indexed by $\breve{f}_{i,j,k}$ [20].

Other than the algorithm of [20] which treats complete polynomials in $m+1$ iterations, this version implements a pipeline structure. This means that values of index k are treated in $P(l)$, whilst $P(l-1)$ treats values of index $k-1$.

The structure takes as input a three-dimensional arrangement of values $\hat{v}_{i,j,k} \in$ GF(q) which are derived from the syndromes [19, 20]. For the calculations, two other auxiliary three-dimensional data structures are generated, $\breve{g}_{i,j,k} \in$ GF(q) and $\hat{w}_{i,j,k} \in$ GF(q). These are initially 0 and therefore cannot be considered as inputs, but do influence the calculations. Systolicity is ensured by a careful arrangement of cell positions $C_{i,j}$ in the different processors $P(l)$.

For different l, the cell $C_{i,j}$ finds itself in positions $c_{i',j}$ where $i' = \phi(l,i)$ for a certain function ϕ [19]. It should be noted that the indices i,j of $C_{i,j}$ correspond to the data indices and their position is only modified to ensure systolicity. Every cell has 10 inputs of which one is a control signal. The cell gives 4 outputs. The operations in a cell $C_{i,j}$ of processor $P(l)$ are defined as [20, 21]:

$$\breve{f}_{l+1,i,j,k} = \breve{f}_{l,i,j,k+\hat{\kappa}(l,i)-\breve{\kappa}(l+1,i)} - d_{l,i}\breve{g}_{l,\bar{j},j,k,\breve{\kappa}(l,i)-\breve{\kappa}(l+1,i)} \quad (1)$$

$$\breve{g}_{l+1,\bar{j},j,k} = \breve{g}_{l+1,\bar{j},j,k} \text{ or } d_{l,i}^{-1}\breve{f}_{l,i,j,k} \quad (2)$$

$$\hat{v}_{l+1,i,j,k} = \hat{v}_{l+1,i,j,k} - d_{l,i}\hat{w}_{l,\bar{j},j,k-\hat{\kappa}(l,i)} \quad (3)$$

$$\hat{w}_{l+1,\bar{j},j,k} = \hat{w}_{l+1,\bar{j},j,k} \text{ or } d_{l,i}^{-1}\hat{v}_{l,i,j,k+\hat{\kappa}(l,)} \quad (4)$$

Note that the index l represents the processor number. The coefficients $\hat{\kappa}(l,i)$, $\breve{\kappa}(l,i)$ and $\bar{j} = \eta(l,i)$ are predetermined integer values that are calculated from the curve characteristics (namely the set of pole orders [19]). By definition, $1 \le \bar{j} \le \rho$, which means that $\bar{j}$ merely defines a rule for dataflow between cells. The calculation of $\breve{g}_{l+1,\bar{j},j,k}$ and $\hat{w}_{l+1,\bar{j},j,k}$ is administered by a one bit control flag generated by the control logic. The left value is selected for flag=0. Finally, $d_{l,i}$ is a coefficient and equals $\hat{v}_{l,i,j,\hat{\kappa}(l,)}$.

The control logic computes two control signals $c_{l,i}$ and $s_{l,i}$. These are integer values. Again, these depend largely on l and i. Using these, the control logic

determines the values $c_{l+1,\bar{j}}$ and $s_{l+1,i}$ as well as the "flag" bit according to the following:

$$\text{flag} = (d_{l,i} \neq 0) \wedge (c_{l,\bar{j}} < \kappa(l,i) - s_{l,i}) \quad (5)$$

$$\text{if flag} = 1 \Rightarrow \begin{cases} s_{l+1,i} = \kappa(l,i) - c_{l,\bar{j}} \\ c_{l+1,\bar{j}} = \kappa(l,i) - s_{l,\bar{j}} \end{cases} \quad (6)$$

$$\text{if flag} = 0 \Rightarrow \begin{cases} s_{l+1,i} = s_{l,i} \\ c_{l+1,\bar{j}} = c_{l,\bar{j}} \end{cases} \quad (7)$$

Again, $\kappa(l,i)$ a value that is determined from the curve characteristics. From the re-definition of the cell equations given in [21], it can be seen that there are several modifications in the indices *k*. In fact, a cell $C_{i,j}$ in processor *P(l)* will have to take values with indices *k*, *k-1*, $k-\hat{\kappa}(l,i)$, $k+\hat{\kappa}(l,i)$ as input.

Assume a processor *P(l)* treats values with index *k* at a given time. Providing values with indices smaller thank *k* consists of adding delayers to the structure. However, to implement the input of a "future" value, indexed $k+\hat{\kappa}(l,i)$, the whole processing has to be delayed by $\hat{\kappa}(l,i)$ cycles, during which the values $\bar{f}_{l,i,j,k}$, $\bar{g}_{l,\bar{j},j,k}$, $\hat{v}_{l,i,j,k}$ and $\hat{w}_{l,\bar{j},j,k}$ have to be stored until values with indices $k+\hat{\kappa}(l,i)$ are available. Moreover, the value $d_{l,i} = \hat{v}_{l,i,\bar{j},\hat{\kappa}(l,i)}$ can only be calculated after $\hat{\kappa}(l,i)$ values have been treated. Since $d_{l,i}$ is needed for treatment of values with index smaller than $\hat{\kappa}(l,i)$ as well, all these have to be delayed until $d_{l,i}$ is known. This introduces more wasteful overhead.

In order to present a feasible example implementation of the proposed architecture, the three-dimensional structure in the original proposal would have to be mapped into two dimensions. Such a mapping would be easiest if the width of one of the three dimensions could be reduced to 1 gate. In the direction of the dataflow, there are always *m + 1* processors, therefore this does not provide a possibility for projection on 2-D.

By examining closely the interconnections between the different cells $C_{i,j}$, in two processors *P(l)* and *P(l+1)* however, it can be seen that there is no interaction between columns, i.e. between cells with different *j* index. A proposed mapping would thus consist of placing the columns above each other instead of next to each other (see Fig. 2). The only obvious disadvantage of this rearrangement is the fact that every row *i* actually shares common control signals. Whilst the transmission of those could easily be done if all the relevant cells are in one row, the penalties of routing could be significant for larger circuits. However, since this rearrangement seems the only straightforward way to a feasible implementation, these penalties remain subject to later examination. For the size of this implementation, the penalties due to the rearrangement are zero.

To properly design the circuit, a number of values and coefficients had to be computed from the underlying curve $\chi: X^3Y + Y^3Z = Z^3X$ for the code $C_\Omega(D, 13P_\infty)$ as for the example used by Kotter [16]. Therefore we have *m*+1 = 14 processors *P(l)* , $0 \leq l \leq 13$. The curve has genus *g* = 3 and has 24 rational points, one of which is P_∞ . The code has length *n* = 23, dimension *k* = 12 and minimum distance d_{min} = 9. The gaps of the curve are given {1, 2, 4}. From this, the set of pole orders $O = \{o_l \mid l \in \mathbb{Z}_0\} = \{0, 3\,(=\rho), 5, 6, 7, 8, \ldots\}$ are determined.

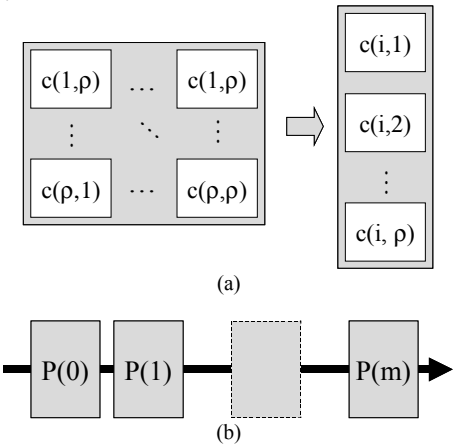

Fig. 2. (a) Mapping 2-D processor to 1-D processor. (b) Resulting 2-D structure.

A number of further values necessary to determine the interconnections between cells and other relevant details had to be computed from this. These are shown in the definitions below. As before, the index *l* corresponds to the processor and *i,j* to the cell or data indices.

$$o^{(i)} = \min\{o_l \in O \mid o_l \equiv i - 1 \bmod \rho\} \quad (1 \leq i \leq \rho) \quad (8)$$

$$\bar{o}^i(j,k) \in O = o^{(i)} + o^{(j)} + k\rho \quad (9)$$
$$if \quad \bar{o}^i(j,k) = o_l$$

$$\eta(l,i) \in \{1, \ldots, \rho\} = (o_l + 1 \bmod \rho) + 1 \quad (10)$$

$$\kappa(l,i) \in \mathbb{Z}_0 = (o_l - o^{(i)} - o^{\eta(l,i)})/\rho \quad (11)$$
$$else \quad \eta(l,i) = \kappa(l,i) = undefined$$

$$\kappa(i) = (o^{(i)} - i + 1)/\rho, \quad 1 \leq i \leq \rho \quad (12)$$

$$\bar{\kappa}(l,i) = (o_l - o^{(i)} - \eta(l,i) + 1)/\rho \quad (13)$$

$$\hat{\kappa}(i) = \bar{\kappa}(l,i) + \kappa(i) \quad (14)$$

The derived values from (8) to (14) have some relevant properties for the architecture. Firstly, $\eta(l,i)$ always takes a value between 1 and ρ, $1 \leq \eta(l,i) \leq \rho$. For $\kappa(l,i)$, $\bar{\kappa}(l,i)$ and $\hat{\kappa}(l,i)$ it is known that they can be any positive integer, $\kappa(l,i), \bar{\kappa}(l,i), \hat{\kappa}(l,i) \in \mathbb{Z}_0$. Finally, from the definition of $\bar{\kappa}(l,i)$ follows that the value of $\bar{\kappa}(l+1,i)$ is never exceeds the preceding value by more than 1, i.e. $\bar{\kappa}(l,i) \leq \bar{\kappa}(l+1,i) \leq \bar{\kappa}(l,i) + 1$.

4 Conclusions and Future Work

AG codes provide the tools to construct new classes of error-correcting codes with properties superior to the codes used today. Nevertheless, not many research can be found concerning their hardware implementation and it has been the motivation for this work.

We have investigated the hardware implementation issues of the BMS decoding strategy proposed by Sakata. We have modified the original strategy to obtain a more suitable hardware implementation by a 3-D to 2-D structure mapping. We have evaluated the impact of each equation under a hardware point of view. This is essential to obtain a efficient VLSI. Actual work includes a fine cost estimation in terms of logic gate count, and a simulated and validated for synthesis VHDL description of the decoder. Also we are considering a modified structure to take into account the recent algorithm modifications [21].

References

[1] V. D. Goppa, "A new class of linear error-correcting codes," *Problems of Information Theory 6*, pp. 207-212, 1970.

[2] V. D. Goppa, "Codes Associated with Divisors," *Problems of Information Transmission 13*, pp. 22-26, 1977.

[3] M. A. Tsfasman, S. G. Vladut, and T. Zink, "Modular curves, Shimura curves and Goppa codes, better than Varshamov-Gilbert bound," *Math. Nachr. 104*, pp. 13-28, 1982.

[4] A. Garcia and H. Stichtenoth, "A tower of Artin-Schreier extensions of function fields attaining the Drinfeld-Vladut bound," *Inventiones Mathematicae*, vol. 121, pp. 211-222, 1995.

[5] A. Garcia and H. Stichtenoth, "On the asymptotic behaviour of some towers of function fields over finite fields," *Journal of Number Theory*, vol. 61(2), pp. 248-273, 1996.

[6] L. H. C. Lee, *Error Control Block Codes*: Artech House Publishers, 2000.

[7] G. Lachaud, M. A. Tsfasman, J. Justesen, and V. K.-W. Wei, "Introduction to the Special lssue on Algebraic Geometry Codes," *IEEE Transactions on Information Theory*, vol. 41, pp. 1545, 1995.

[8] C. Xing and S. Ling, "A class of linear codes with good parameters from algebraic curves," *IEEE Transactions on Information Theory*, vol. 46, pp. 1527 - 1532, 2000.

[9] D. Cunsheng, H. Niederreiter, and X. Chaoping, "Some new codes from algebraic curves," *IEEE Transactions on Information Theory*, vol. 46, pp. 2638 - 2642, 2000.

[10] C. Xing, H. Niederreiter, and K. Y. Lam, "A Generalization of Algebraic-Geometry Codes," *IEEE Transactions of Information Theory*, vol. 45, pp. 2498-2501, 1999.

[11] A. E. Brouwer, "Bounds on Min Distance of Linear Codes, "http://www.in.tue.nl/~aeb/voorlincod.html, 2002.

[12] I. Blake, C. Hegaard, T. Hoholdt, and V. K. Wei, "Algebraic Geometry Codes," *IEEE Transactions of Information Theory*, vol. 44, pp. 2596-2618, 1998.

[13] S. Sakata, "Extension of the Berlekamp-Massey algorithm to N dimensions," *Informat. Comput.*, vol. 84, pp. 207-239, 1990.

[14] G. Feng, V. K. Wei, T. R. N. Rao, and K. K. Tzeng, "Simplified Understanding and Efficient Decoding of a Class of Algebraic-Geometric Codes," *IEEE Transactions of Information Theory*, vol. 40, 1994.

[15] M. E. O'Sullivan, "Decoding of Hermitian codes: the key equation and efficient error evaluation," *IEEE Transactions on Information Theory*, vol. 46, pp. 512 - 523, 2000.

[16] R. Kotter, "A fast parallel implementation of a Berlekamp-Massey algorithm for algebraic-geometric codes," *IEEE Transactions on Information Theory*, vol. 44, pp. 1353 - 1368, 1998.

[17] T. Hoholdt and R. Pellikaan, "On the Decoding of Algebraic-Geometric Codes," *IEEE Transactions of Information Theory*, vol. 41, 1995.

[18] S. Sakata and M. Kurihara, "A systolic array architecture for implementing a fast parallel decoding algorithm of one-point AG codes," presented at Proceedings of IEEE International Symposium on Information Theory, Ulm, Germany, 1997.

[19] S. Sakata and M. Kurihara, "A Systolic Array Architecture for Fast Decoding of One-Point AG Codes and Scheduling of Parallel Processing," *Proceedings of AAEC-14, Springer*, pp. 302-313, 1999.

[20] S. Sakata, "A Vector Version of the BMS Algorithm for Implementing Fast Erasure-and-Error Decoding of One-Point AG Codes," presented at AAEC, 1997.

[21] S. Sakata and H. Matsui, "BMS Algorithm." Manuscript unpublished, 2002.