Speeding Up the Reed-Solomon Encoder and Decoder

EDIC Doctoral Project
Ghid Maatouk
Supervisor: Prof. Amin Shokrollahi
ALGO, I&C, EPFL

I. Introduction

Reed-Solomon (RS) codes are used in many applications in communications and storage, with fast, optimized hardware implementations. In [1], the author proposes a method to speed up by a factor of roughly p the encoding and decoding of RS codes over finite fields where there exist pth roots of unity, at the small hardware cost of one or several additional Discrete Fourier Transform (DFT) units. In this project, we consider the problem of achieving a comparable speedup for RS codes over finite fields where there exist no roots of unity. We start by reminding the reader of RS codes and their encoding and decoding algorithms, then move on to describe the (generalized) RS code proposed in [1], along with its encoding and decoding algorithms. We then describe the various approaches we have taken to perform a similar parallelization to that proposed in [1], but in fields where there exist no roots of unity.

We briefly remind the reader of the definition of RS codes over a finite field $\mathbb{F}_q$ and outline their encoding and decoding algorithms (without proof of their correctness). This will allow us, in the next section, to show how the main ideas of [1] come into play so as to provide a factor p speedup for (most stages of) these algorithms. From now on, we will consider only finite fields $\mathbb{F}_q = \mathbb{F}_{2^l}$ of characteristic 2, and throughout the report we let α denote a primitive element of $\mathbb{F}_q$.

RS codes of length n and dimension k can be essentially viewed in two equivalent ways: as *evaluation codes* where a codeword corresponds to a polynomial of degree less than k and its coordinates to the evaluation of this polynomial at a set of evaluation points; or as a *cyclic code* with a generator polynomial whose roots are a set of $n-k$ consecutive powers of some primitive element α of $\mathbb{F}_q$. We give here the cyclic code formal definition, as it naturally gives rise to a systematic encoding algorithm.

Definition 1: The cyclic code over F_q with generator $g(x) = \prod_{i=0}^{n-k-1}(x - \alpha^i)$ is an $[n, k, n-k+1]_q$-code known as a Reed-Solomon code.

a) Encoding: Systematic encoding of Reed-Solomon codes works as follows: let $G(x) \in \mathbb{F}_q[x]_{<k}$ be the message to transmit and let $h(x)$ be the remainder of the division of $x^{n-k}G(x)$ by the generator polynomial $g(x)$. Then the corresponding codeword consists of the coefficients of the polynomial $x^{n-k}G(x) + h(x)$.

b) Decoding: Suppose that errors occur at t positions of the codeword, with $t \leq \frac{n-k}{2}$, so that the word $(y_0, \ldots, y_{n-1})$ is received at the decoder, corresponding to the polynomial $y(x)$. Decoding works as follows:

1) Compute *syndromes* $s_0, \ldots, s_{n-k-1}$ corresponding to the evaluation of y at the roots of the generator polynomial, i.e., $s_i := y(\alpha^i)$.
2) Use the Berlekamp-Massey algorithm to compute the shortest linear recurrence for the syndrome sequence, which corresponds to the coefficients of (the reversal of) the *error locator polynomial* $u(x)$, of degree t (the Berlekamp-Massey algorithm also outputs another auxiliary polynomial used for the computation of the error values).
3) Find the indices j such that $u(\alpha^{-j}) = 0$ using a *Chien search* (see below). These will correspond to the error positions. For each position j in error, compute the value ϵ_j of the error. It is given by a simple polynomial evaluation using the output of the Berlekamp-Massey algorithm.

c) Hardware implementation of the Chien Search: In step 3 of the decoding, the task is to find the roots of a polynomial $u(x)$ among a set of possible roots. The way this is implemented in hardware is through a very simple circuit that evaluates $u(x)$ at all elements in the set. What makes the circuit simple is the fact that all those elements can be expressed as consecutive powers of α^{-1}. Figure 1 shows how the consecutive evaluations can be implemented by multiplying the polynomial coefficients by hardwired elements. At step i, the circuit outputs $u(\alpha^{-i})$.

II. Speeding up RS Encoding and Decoding on Some Fields of Characteristic 2

In [1], the author proposes a method to speed up the RS encoder, the syndrome calculation and the Chien search (stages 1 and 3 of decoding) by a factor of p, provided the field $\mathbb{F}_q$ contains pth roots of unity. Actually, the resulting code

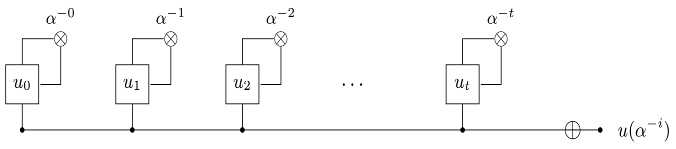

Fig. 1. Traditional Chien search

is a systematic generalized RS code. In what follows, we will present the main ideas of [1] for the case $p = 3$. These ideas can be easily extended to apply for any other value of p when pth roots of unity exist in $\mathbb{F}_q$. From now on, we will be only considering values of q such that 3 divides $q-1$, and we will let ρ denote a 3rd root of unity in $\mathbb{F}_q$.

A. Speeding up the Chien search

We explain in detail the modifications to the Chien search proposed in [1]. This will illustrate the fundamental ideas behind the speedup of this and other stages of RS encoding and decoding. We will then briefly show how these ideas extend to the encoding of a generalized RS code (to be defined) and to the syndrome calculation for this code.

The algorithm proposed in [1] takes as input a polynomial $u(x) = \sum_{i=0}^{t} u_i x^i$ and computes its evaluation at a set of candidate roots $\{\rho^{i \bmod 3}\alpha^{\lfloor i/3 \rfloor}\}_{i=1,\ldots,n}$ (why this is the set of candidate roots will become clear in the context of the generalized RS code designed in [1]). It computes, at each step, three values in one shot. More precisely, at step i, it will output the evaluation of $u(x)$ on the set $\{\alpha^{-i}\rho^{-j}\}_{j=1,2,3}$. Note that for every y in this set, $u(y) = b_i(y)$, where $b_i(x) := u(x) \bmod (x^3 - (\alpha^{-i})^3)$. But it is easy to see that

$$b_i(x) = a_0^{(i)} + a_1^{(i)}(\alpha^i x) + a_2^{(i)}(\alpha^i x)^2,$$

where (dropping the superscripts for simplicity),

$$\begin{aligned} a_0 &:= \sum_{\substack{0 \le j \le t \\ j \equiv 0 \bmod 3}} u_j \alpha^{-ij} \\ a_1 &:= \sum_{\substack{0 \le j \le t \\ j \equiv 1 \bmod 3}} u_j \alpha^{-ij} \\ a_2 &:= \sum_{\substack{0 \le j \le t \\ j \equiv 2 \bmod 3}} u_j \alpha^{-ij}. \end{aligned} \tag{1}$$

Given this expression for $b_i(x)$, the evaluations of $b_i(x)$ on the three points $\{\alpha^{-i}\rho^{-j}\}_{j=1,2,3}$ can then be computed as

$$\begin{pmatrix} b_i(\alpha^{-i}) \\ b_i(\alpha^{-i}\rho^{-1}) \\ b_i(\alpha^{-i}\rho^{-2}) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & \rho & \rho^2 \\ 1 & \rho^2 & \rho \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix}. \tag{2}$$

The algorithm for the modified Chien search thus works as follows:

On input $u(x)$, for $i = 0, \ldots, n/3 - 1$,

1) Calculate the coefficients a_0, a_1, a_2 of $b_i(x)$ as given by (1)
2) Calculate the evaluations of $b_i(x)$ on the points $\{\alpha^{-i}\rho^{-j}\}_{j=1,2,3}$ as given by (2).

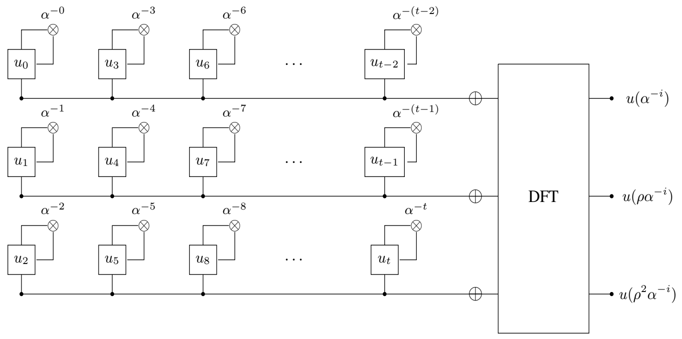

Fig. 2. Fast Chien search

In hardware, the computation in (1) can be carried similarly to that of the consecutive evaluations in the traditional Chien search. Note that the total number of gates required for the computation of the three coefficients a_0, a_1, a_2 is exactly the same as the number of gates required in the traditional case. As for the values in (2), they can be computed with a DFT circuit. Figure 2 shows the hardware implementation of the algorithm.

B. Encoding

We start by describing the generalized RS code designed in [1]. The code has length $n = 3m < q$ and dimension $k = 3t \le n$. In what follows, let ℓ denote $m - t = \frac{n-k}{3}$.

Definition 2: [1] Let

$$\begin{aligned} g_0 &:= \prod_{\substack{0 \le i < n-k \\ i \equiv 0 \bmod 3}} (x - \alpha^i) \\ g_1 &:= \prod_{\substack{0 \le i < n-k \\ i \equiv 1 \bmod 3}} (x - \alpha^i) \\ g_2 &:= \prod_{\substack{0 \le i < n-k \\ i \equiv 2 \bmod 3}} (x - \alpha^i). \end{aligned}$$

Then the code $C(n, k; \alpha, \rho)$ is defined as

$$\begin{aligned} C(n, k; \alpha, \rho) := \{\, (H_0, H_1, H_2) \mid H_i \in \mathbb{F}_q[x]_{<m}, \\ \{H_i\} \text{ satisfy (3) } \}, \end{aligned}$$

with

$$\begin{aligned} H_0 + H_1 + H_2 &\equiv 0 \bmod g_0 \\ H_0 + \rho H_1 + \rho^2 H_2 &\equiv 0 \bmod g_1 \\ H_0 + \rho^2 H_1 + \rho H_2 &\equiv 0 \bmod g_2. \end{aligned} \tag{3}$$

As the following theorem states, the code thus defined is a generalized RS code.

Theorem 1: [1] Let $\alpha_i := \rho^{i \pmod 3}\alpha^{\lfloor i/3 \rfloor}$ and $\Delta_i := \frac{1}{\prod_{j \ne i}(\alpha_i - \alpha_j)}$. Then C is an $[n, k, n-k+1]_q$-code and is given by

$$C = \{\, (\Delta_1 f(\alpha_1), \ldots, \Delta_n f(\alpha_n)) \mid f \in \mathbb{F}_q[x]_{<k} \}.$$

□

Encoding then works as follows: given a message

$$(G_0, G_1, G_2),\ G_i(x) \in \mathbb{F}_q[x]_{<t},$$

the corresponding codeword will be of the form

$$(x^\ell G_0 + h_0, x^\ell G_1 + h_1, x^\ell G_2 + h_2),$$

with $h_i(x) \in \mathbb{F}_q[x]_{<\ell}$. As the codeword satisfies (3), we have that

$$\begin{aligned} h_0 + h_1 + h_2 &= x^\ell (G_0 + G_1 + G_2) \bmod g_0 \\ h_0 + \rho h_1 + \rho^2 h_2 &= x^\ell (G_0 + \rho G_1 + \rho^2 G_2) \bmod g_1 \\ h_0 + \rho^2 h_1 + \rho h_2 &= x^\ell (G_0 + \rho^2 G_1 + \rho G_2) \bmod g_2, \end{aligned}$$

and thus the following algorithm will produce h_0, h_1, h_2:

On input $G_0, G_1, G_2 \in \mathbb{F}_q[x]_{<t}$,

1) Compute

$$\begin{aligned} f_0 &:= x^\ell (G_0 + G_1 + G_2) \bmod g_0 \\ f_1 &:= x^\ell (G_0 + \rho G_1 + \rho^2 G_2) \bmod g_1 \\ f_2 &:= x^\ell (G_0 + \rho^2 G_1 + \rho G_2) \bmod g_2. \end{aligned}$$

2) Compute

$$\begin{aligned} h_0 &:= f_0 + f_1 + f_2 \\ h_1 &:= f_0 + \rho^2 f_1 + \rho f_2 \\ h_2 &:= f_0 + \rho f_1 + \rho^2 f_2. \end{aligned}$$

As in the case of the Chien search, the hardware complexity of the circuit implementing this algorithm is essentially the same as that for traditional RS encoding with the same parameters n and k, but it performs three computations in parallel, thus gaining roughly a time factor of 3, at the cost of an additional DFT unit and inverse DFT unit.

C. Decoding

Recall that RS decoding works in three steps: syndrome computation, finding the error locator polynomial using the Berlekamp-Massey algorithm, and finding the error positions corresponding to the roots of the error locator using a Chien search. For the generalized RS code of [1], the syndrome computation is also sped up almost by a factor of 3 at the cost of $n/3$ extra DFT units. Basically, given a received codeword $(y_0, \ldots, y_{n-1})$, instead of evaluating $y(x)$ at the points α^i, the syndrome computation circuit evaluates three (shorter) polynomials in parallel at the points α^{3i}, α^{3i+1} and α^{3i+2}, respectively.

The syndromes are fed to the Berlekamp-Massey algorithm, which runs as is, without any speedup. It outputs a polynomial $u(x)$. Then the fast Chien search of Section II-A is performed to find the roots of $u(x)$ among the set $\{\alpha_i = \rho^{i \pmod 3} \alpha^{\lfloor i/3 \rfloor}\}$ of evaluation points of the code (as given by Theorem 1). This immediately gives the error positions, as it is easy to see from the analysis in Section II-A and the definition of the code that a root $\alpha^{-i}\rho^{-j}$ of $u(x)$ corresponds to an error at position $3i + j$.

III. Speeding up RS Encoding and Decoding on All Fields of Characteristic 2

We saw that [1] proposes a scheme to speed up the encoding and some decoding stages of (generalized) RS codes by a factor of p, if pth roots of unity exist in the field $\mathbb{F}_q$. But what if a practical application calls for computations on a field $\mathbb{F}_q$ where $q - 1 = 2^\ell - 1$ is a Mersenne prime (recall that a Mersenne prime is a prime number of the form $2^\ell - 1$ where ℓ is prime)? In this case, as the multiplicative order of a field element must divide the order $q - 1$ of the multiplicative group $\mathbb{F}_q^*$, all nontrivial field elements will have order $q - 1$, and there will be no pth roots of unity for $p < q - 1$.

In what follows, we would like to use the intuitions of [1] to design RS codes on general fields $\mathbb{F}_q$ while achieving a similar speedup for encoding and decoding. We will focus for now on the Chien search as this will illustrate the main ideas at play. This will later need to be extended into the design of a code such that the encoding and syndrome computations can be sped up by the same factor.

In the fast Chien search of [1], the polynomial $u(x)$ was evaluated at each step at a multiplicative coset of $\{1, \rho^{-1}, \rho^{-2}\}$. At each step i, what was actually evaluated at these points was the remainder $b_i(x)$ of the division of $u(x)$ by the polynomial vanishing on the set $\{\alpha^{-i}\rho^{-j}\}_{j=1,2,3}$. The clever choice of the evaluation sets $\{\alpha^{-i}\rho^{-j}\}_{j=1,2,3}$ allowed

- the coefficients of $b_i(x)$ to be obtained from the coefficients of $b_{i-1}(x)$ with a very simple circuit using only hardwired multiplications;
- the three evaluations to be obtained in "one shot" from the coefficients of $b_i(x)$ using a DFT module.

We would like to find another way to group the elements of $\mathbb{F}_q$ in sets, so that the evaluations on one set can be easily related to the evaluations on the previous set. A first approach is by considering additive cosets of a subspace of $\mathbb{F}_q$ when $\mathbb{F}_q$ is viewed as a vector space.

A. The subspace approach

We identify $\mathbb{F}_q = \mathbb{F}_{2^l}$ to the vector space $\mathbb{F}_2^l$. Let V be a subspace of $\mathbb{F}_q$. The idea is to evaluate $u(x)$ on sets of the form $V + a = \{v + a | v \in V\}$. Recall that in [1], what was evaluated on a set of points was $u(x)$ modded by the polynomial vanishing at these points. In the previous case, this polynomial was of the form $(x^3 - (\alpha^{-i})^3)$, which gave a very simple expression for the coefficients of the remainder polynomial $b_i(x)$, and which made it possible to relate these coefficients for successive values of i. In our case, the polynomial vanishing on the subspace (the *subspace polynomial*) is not as simple, but it has an interesting property: it is linearized, which gives us a simple expression for the polynomials vanishing at other evaluation sets (the *coset polynomials*) in terms of this polynomial. As an added benefit, the fact that the subspace polynomial is linearized makes it sparse, which gives us the hope of it resulting in a simple hardware implementation even if the dimension of the subspace (and the degree of the polynomial) grows. We now precise these notions.

Definition 3: Let V be a subspace of $\mathbb{F}_q$. The subspace polynomial $L_V(x)$ of V is the polynomial over $\mathbb{F}_q$ that vanishes on all elements of V, i.e.,

$$L_V(x) = \prod_{v \in V} (x - v). \tag{4}$$

The following is an important property of the subspace polynomial.

Fact 1: Let V be a subspace of $\mathbb{F}_q$ of dimension d. Then $L_V(x)$ is a linearized polynomial, i.e., its only nonzero coefficients are those of the monomials whose degree is a power of 2. It can be written as

$$L_V(x) = x^{2^d} + \mu_{d-1} x^{2^{d-1}} + \cdots + \mu_1 x + \mu_0.$$

Proof: Consider the $d+1$ linear forms over V $\{x^{2^i}\}_{i=0,\dots,d}$. As the dual space of V is of dimension d, there must be a linear dependency among these $d+1$ linear forms. So there exist $c_0, \dots, c_d$ such that $\sum c_i x^{2^i} = 0$ for all v in V. Therefore $L_V(x) = \prod_{v \in V}(x - v)$ divides $\sum c_i x^{2^i}$. As these two polynomials are of the same degree, they must be scalar multiples of each other so that there exist $\mu_0 = c_0 c_d^{-1}, \dots, \mu_{d-1} = c_{d-1} c_d^{-1}$ such that $L_V(x) = x^{2^d} + \mu_{d-1} x^{2^{d-1}} + \cdots + \mu_1 x + \mu_0$. ∎

We can relate the subspace polynomial to the coset polynomials as follows.

Fact 2: Let $V + a$ be a coset of V and define the coset polynomial $L_{V+a}(x)$ as the polynomial that vanishes on every element of $V + a$. Then

$$L_{V+a}(x) = L_V(x) + L_V(a).$$

Proof: $L_{V+a}(x) = L_V(x + a) = L_V(x) + L_V(a)$, by linearity of $L_V(x)$. ∎

We considered the simplest case possible, namely, we let $V = \{0, v\}$ be a subspace of dimension 1. In this case, the evaluation cosets are of the form $\{a, v + a\}$ and the coset polynomials of the form $L_{V+a}(x) = x^2 + vx + c_a$, where c_a is a constant depending on the coset. However, our attempts to derive either closed-form expressions for the coefficients of the remainders of $u(x)$ divided by the cosets polynomials, or simple relations between the coefficients of the remainder polynomial from one step to the next, were all infructuous. The intuition behind this is that the relation between the successive evaluation points, as well as between the successive coset polynomials, is additive, which makes it hard to relate values from one step to the next. In our next approach, we try to address this issue.

TABLE I
GOOD ELEMENTS FOR SOME FIELDS $\mathbb{F}_q$ WITH $q-1$ MERSENNE PRIME

$\mathbb{F}_q$	minimal polynomial of ω	$1+\omega$	$\lvert \log_\omega(1+\omega) - q/2 \rvert$
$\mathbb{F}_{2^5}$	$x^5 + x^3 + 1$	ω^{14}	2
$\mathbb{F}_{2^7}$	$x^7+x^6+x^5+x^4+x^2+x+1$	ω^{73}	9
$\mathbb{F}_{2^{13}}$	$x^{13}+x^{10}+x^9+x^8+x^6+x^5+x^4+x^3+1$	ω^{4067}	29
$\mathbb{F}_{2^{17}}$	$x^{17}+x^{16}+x^{13}+x^{10}+x^7+x^4+x^2+x+1$	ω^{65576}	40
$\mathbb{F}_{2^{19}}$	$x^{19}+x^{18}+x^{15}+x^{11}+x^{10}+x^9+x^8+x^6+x^4+x^3+1$	ω^{262133}	11

B. The primitive element approach

We would like to be able to define our evaluation sets in such a way as to have a multiplicative relation, both between the successive evaluation points, and between the successive coset polynomials.

As the only case in which the algorithm of [1] cannot be applied is for those fields $\mathbb{F}_q$ where $q-1$ is a Mersenne prime, we restrict our attention to such fields. Then any nontrivial field element ω is a primitive element of the field. For one such ω, define the evaluation cosets as follows: for $i \geq 1$, let $C_i = \{\omega^i, \omega^{i-1}(1+\omega)\}$. Thus $C_1 = \{\omega, 1+\omega\}$ and $C_{i+1} = \{\omega \cdot \gamma \,|\, \gamma \in C_i\}$. Of course, the problem with such evaluation cosets is that they might overlap. For them not to overlap too much, ω must be such that $1+\omega = w^j$ for j "close enough" to $\frac{q}{2}$. As every nontrivial field element is a generator, there is hope that there exists a "good" element, i.e., one for which $1+\omega = w^j$ for $|j - \frac{q}{2}| < c$ for some small constant c.

We have not yet been able to formally prove the existence of such an ω. However, even though such a result would be of theoretical interest, the fields $\mathbb{F}_q$ with $q-1$ Mersenne prime used in practice are very few. For our purposes, it is enough to simply assign an element ω for every such field $\mathbb{F}_q$. Table I gives one of the best elements ω in a few fields (there are often several best elements). Given a fixed ω, we know that we will have covered the whole field after $q/2 + 2|\log_\omega(1+\omega) - q/2|$ evaluation steps (producing two evaluations at each step), and there will be $2|\log_\omega(1+\omega) - q/2|$ repeated evaluations that we must make sure to discard.

Given the evaluation coset $C_i = \{\omega^i, \omega^{i-1}(1+\omega)\}$, the corresponding coset polynomial is $x^2 + \omega^{i-1}x + (\omega^2)^{i-1}(\omega + \omega^2)$. In this form, it is not very useful. Rather, we will consider its *reverse polynomial* $p_i(x) = 1 + \omega^{i-1}x + (\omega^2)^{i-1}(\omega + \omega^2)x^2$. Then we have that

$$p_{i+1}(x) = p_i(\omega x),$$

with $p_1(x) = 1 + x + (\omega + \omega^2)x^2$.

Let $b_i(x) = u(x) \bmod p_i(x) = a_1^{(i)} x + a_0^{(i)}$. $b_i(x)$ is the polynomial that we are interested in evaluating at the points of C_i. First note that $a_0^{(i)}$ and $a_1^{(i)}$ are linear in the coefficients of $u(x)$, since

$$\begin{aligned} u(x) \bmod p_i(x) &= \left(\sum_i u_i x^i\right) \bmod p_i(x) \\ &= \sum_i u_i (x^i \bmod p_i(x)) \\ &= \sum_i u_i (c_i x + d_i), \end{aligned} \tag{5}$$

for scalars $\{c_i\}$ and $\{d_i\}$ independent of the $\{u_i\}$. Then it is easy to see with a simple change of variables that if

$$b_i(x) = u(x) \bmod p_i(x) = \left(\sum_i u_i c_i\right) x + \left(\sum_i u_i d_i\right),$$

then

$$\begin{aligned} b_{i+1}(x) = u(x) \bmod p_i(\omega x) = \left(\sum_i u_i c_i \omega^{-1}\right) \omega x \\ + \left(\sum_i u_i d_i \omega^{-1}\right). \end{aligned} \tag{6}$$

Hence the coefficients $a_0^{(i+1)}$ and $a_1^{(i+1)}$ of $b_{i+1}(x)$ can be related to those of $b_i(x)$ in a very similar fashion as that in which the corresponding coefficients were related in (1), in the sped-up Chien search of [1].

To see why it is alright to consider the reverse polynomial $p_i(x)$ for each coset, consider the coset $C_i = \{\omega^i, \omega^{i-1}(1+\omega)\} = \{\omega^i, \omega^{j-1+i}\}$, where j is such that $1+\omega = \omega^j$. Then the reverse polynomial of C_i vanishes at $\{\omega^{-i}, \omega^{-j+1-i}\} = \{\omega^{q-1-i}, \omega^{q-2-j-i}\} = C_{q-2-j-i}$, so that it is nothing but the coset polynomial of another coset. Thus a simple relabeling of the evaluations resolves the issue.

This analysis is merely the beginning of the design process that should result in a full-fledged sped-up RS code over fields $\mathbb{F}_q$ with $q-1$ a Mersenne prime. In the conclusion section below, we point out what remains to be done to achieve this goal.

IV. Conclusion - Future Work

We have presented the generalized RS code designed in [1] with its encoding and decoding algorithms and have discussed the main intuitions behind the speedup factor they achieve. We have then tried to extrapolate the same kind of intuitions to the case where the RS code is over a finite field containing no roots of unity, considering for now only the Chien search problem. Our first, vector space based approach was not successful as the evaluation cosets were additive and made it difficult to relate the modded polynomial at each computation step to the one at the previous step, and its evaluations to the evaluations at the previous step. Our second approach, based on a "good" primitive element, seems more promising. Indeed, in that case, the evaluation cosets are multiplicative as are the ones in [1], and the coefficients of the modded polynomials at each step are related to those in the previous step in a similar fashion as in [1].

But up to this point we have only skimmed the surface of the problem. What still needs to be done is, first, to design the actual circuit for the Chien search using the evaluation cosets $\{\omega^i, \omega^{i-1}(1+\omega)\}$. Then, if we are indeed able to achieve a speedup factor of 2 for the Chien search, we should next use the same ideas to design an actual code where the encoding and syndrome calculation stages are also sped up by a factor of 2.

Another important question is whether we can generalize this approach to achieve other speedup factors using larger evaluation sets. Ideally, our design should scale to achieve any speedup factor that is a power of 2.

A question that would be interesting, but not essential, to answer, is the existence of "good" primitive elements over any field $\mathbb{F}_q$ where $q-1$ is a Mersenne prime.

Finally, it is yet to be determined whether the Berlekamp-Massey algorithm can be sped up as well, whether by a factor of p on fields where pth roots of unity exist, or by whatever speedup factor we can achieve on other fields for the encoding and other decoding stages.

References

[1] A. Shokrollahi, "RS-Codes with Faster Encoding and Decoding Algorithms", March 2008, unpublished.