International Journal of Modern Engineering Research (IJMER)
www.ijmer.com Vol.2, Issue.4, July-Aug 2012 pp-2938-2946 ISSN: 2249-6645

A Primitive Polynomial to Define the Finite Field Modules for High-Throughput Interpolator to Encode and Decode Of Non Binary Cyclic Codes

Susrutha Babu Sukhavasi[1], Suparshya Babu Sukhavasi[1]
S. R. Sastry K.[2], M. Aravind Kumar[3], P. Bosebabu[4],
G. Roopa Krishna Chandra[5]

[1]Faculty, Department of ECE, K L University, Guntur, AP, India.
[2]M.Tech -VLSI Student, Department of ECE, K L University, Guntur, AP, India.
[3]M.Tech -VLSI Student, Padmasri Dr.B.V.Raju Institute of Technology, Medak, A.P, India.
[4] Faculty, Department of ECE, Andhra Loyola College Of Eng &Tech ,Vijayawada, AP, India.
[5]M.Tech - Student, Department of ECE, Lakireddy Bali Reddy College of engineering LBR, Guntur, AP, India

Abstract *: This paper represents The algebraic soft-decoding (ASD) of Reed–Solomon (RS) codes provides significant coding gain over hard-decision decoding with polynomial complexity. The low-complexity chase (LCC) algorithm is proposed for reducing the complexity of interpolation, which interpolates over 2^n test vectors, being attractive for VLSI implementation. The interpolation is simplified in LCC decoding by restricting the multiplicity to m=1 and replacing the factorization step with Chien's search and Forney's algorithm. In this paper, high-throughput interpolator architecture for soft-decision decoding of Reed–Solomon (RS) codes based on low-complexity chase (LCC) decoding is presented. We have formulated a modified form of the Nielson's interpolation algorithm, using some typical features of LCC decoding. The proposed algorithm works with a different scheduling, takes care of the limited growth of the polynomials, and shares the common interpolation points, for reducing the latency of interpolation. Based on the proposed modified Nielson's algorithm we have derived low-latency architecture to reduce the overall latency of the whole LCC decoder. An efficiency is low, in terms of area-delay product, has been achieved by an LCC decoder, by using the proposed interpolator architecture, over the best of the previously reported architectures for an RS(255,239) code with eight test vectors.*

Keyword: *RScodes,Guruswami-Sudan algorithm, Registers,Multiplexers,D-flipflop.*

I. Introduction

2. Interpolation
3. Factorization of bivariate polynomials

It's being the interpolation and the most computation-intensive one. Several architectures based on Nielson's algorithm and Lee–O'Sullivan algorithm are found in the literature for the VLSI implementation of interpolation stage. However, their hardware complexity is still high. An interpolation architecture for LCC with m=3, called backward interpolation, is proposed in [9], which could be considered as the best of the current approaches. Backward interpolator shares the computation of common points of the test vectors. These points are ordered in such way that a pair of adjacent vectors differ only at one point. Due to this feature, the backward interpolation architecture involves less area and provides higher speed than its prior ones.

II. Reed Solomon Software Performance

The following data details performance numbers for a number of specific RS (n, k) implementations for two general purpose processing architectures and one digital signal processor. Numbers are provided for both decode in the presence of no error, as well as decode in the presence of maximum channel error. Note that correcting errors requires more processing power than simply validating blocks, and that the required processing power increases linearly with the error rate. Typical applications tend to keep the error rate low such that active correction is not required. The two digit hexadecimal number in each column specifies the GF (255) primitive polynomial used to generate the underlying Galois field.In 1960, Irving Reed and Gus Solomon published a paper in the Journal of the Society for Industrial and Applied Mathematics. This paper described a new class of error-correcting codes that are now called Reed-Solomon (R-S) codes. These codes have great power and utility, and are today found in many applications from compact disc players to deep-space applications. This article is an attempt to describe the paramount features of R-S codes and the fundamentals of how they work. Reed-Solomon codes are non-binary cyclic codes with symbols made up of m-bit Sequences, where m is any positive integer having a value greater than 2. R-S (n, k) codes on m-bit symbols exist for all n and k for which

$0 < k < n < 2m + 2$

Where k is the number of data symbols being encoded, and n is the total number of code symbols in the encoded block. For the most conventional R-S (n, k) code,

$(n, k) = (2m - 1, 2m - 1 - 2t)$

Where t is the symbol-error correcting capability of the code, and n - k = 2t is the number of parity symbols. An extended R-S code can be made up with n = 2m or n = 2m + 1, but not any further.Reed-Solomon codes achieve the largest possible code minimum distance for any linear code with the same encoder input and output block lengths. For non-binary codes, the distance between two code words is defined (analogous to Hamming distance) as the number

of symbols in which the sequences differ. For Reed-Solomon codes, the code minimum distance
$d_{min} = n - k + 1$

The code is capable of correcting any combination of t or fewer errors, where t can be expressed as

$T= [(d_{min}-1) / 2] = [(n-k) / 2]$

where [x] means the largest integer not to exceed x. Equation illustrates that for the case of R-S codes, correcting t symbol errors requires no more than 2t parity symbols. Equation lends itself to the following intuitive reasoning. One can say that the decoder has n - k redundant symbols to "spend," which is twice the amount of correctable errors. For each error, one redundant symbol is used to locate the error, and another redundant symbol is used to find its correct value. The erasure-correcting capability, ρ, of the code is
$\rho = d_{min} - 1 = n - k$ (5)
Simultaneous error-correction and erasure-correction capability can be expressed as follows:
$2\alpha + \gamma < d_{min} < n - k$ (6)

Where α is the number of symbol-error patterns that can be corrected and γ is the number of symbol erasure patterns that can be corrected. An advantage of non-binary codes such as a Reed-Solomon code can be seen by the following comparison. Consider a binary (n, k) = (7, 3) code. The entire n-tuple space contains 2n = 27 = 128 n-tuples, of which 2k = 23 = 8 (or 1/16 of the n-tuples) are code words. Next, consider a non-binary (n, k) = (7, 3) code where each symbol is composed of m = 3 bits. The n-tuple space amounts to 2nm = 221 = 2,097,152 n-tuples, of which 2km = 29 = 512 (or 1/4096 of the n-tuples) are code words. When dealing with non-binary symbols, each made up of m bits, only a small fraction (i.e., 2km of the large number 2nm) of possible n-tuples are code words. This fraction decreases with increasing values of m. The important point here is that when a small fraction of the n-tuple space is used for code words, a large dmin can be created. Any linear code is capable of correcting n - k symbol erasure patterns if the n – k erased symbols all happen to lie on the parity symbols. However, R-S codes have the remarkable property that they are able to correct any set of n - k symbol erasures within the block. R-S codes can be designed to have any redundancy. However, the complexity of a high-speed implementation increases with Reed-Solomon Codes redundancy. Thus, the most attractive R-S codes have high code rates (low redundancy).

2.1 REED-SOLOMOERROR PROBABILITY

The Reed-Solomon (R-S) codes are particularly useful for burst-error correction; that is, they are effective for channels that have memory. Also, they can be used efficiently on channels where the set of input symbols is large. An interesting feature of the R-S code is that as many as two information symbols can be added to an R-S code of length n without reducing its minimum distance.

For R-S codes, error probability is an exponentially decreasing function of block length, n, and decoding complexity is proportional to a small power of the block length. The R-S codes are sometimes used in a concatenated arrangement. In such a System, an inner convolution decoder first provides some error control by operating on soft-decision demodulator outputs; the convolutional decoder then presents hard-decision data to the outer Reed-Solomon decoder, which further reduces the probability of error.

2.2 FINITE FIELDS

In order to understand the encoding and decoding principles of non binary codes, such as Reed-Solomon (R-S) codes, it is necessary to venture into the area of finite fields known as Galois Fields (GF). For any prime number, p, there exists a finite field denoted GF(p) that contains p elements. It is possible to extend GF(p) to a field of pm elements, called an extension field of GF(p), and denoted by GF(pm), where m is a nonzero positive integer. Note that GF(pm) contains as a subset the elements of GF(p). Symbols from the extension field GF(2m) are used in the construction of Reed-Solomon (R-S) codes. The binary field GF(2) is a subfield of the extension field GF(2m), in much the same way as the real number field is a subfield of the complex number field. Besides the numbers 0 and 1, there are additional unique elements in the extension field that will be represented with a new symbol α. Each nonzero element in GF(2m) can be represented by a power of α. An infinite set of elements, F, is formed by starting with the elements {0, 1, α}, and generating additional elementsby progressively multiplying the last entry by α, which yields the following:
F = {0, 1, α, α2, …, α j, …} = {0, α0, α1, α2, …, α j, …}

To obtain the finite set of elements of GF(2m) from F, a condition must be imposed on F so that it may contain only 2m elements and is closed under multiplication. The condition that closes the set of field elements under multiplication is characterized by the irreducible polynomial shown below:

α(2m−1) + 1 = 0

or equivalently Using this polynomial constraint, any field element that has a power equal to or greater than 2m - 1 can be reduced to an element with a power less than 2m - 1, as follows:

α(2m+ n) = α(2m−1) αn+1 = αn+1

2.3 ADDITION IN THE EXTENSION FIELD GF(2M)

Each of the 2m elements of the finite field, GF(2m), can be represented as a distinct polynomial of degree m - 1 or less. The degree of a polynomial is the value of its highest-order exponent. We denote each of the nonzero elements of GF(2m) as a polynomial, ai (X), where at least one of the m coefficients of ai (X) is nonzero. For i = 0,1,2,…,2m - 2,

αi = ai (X) = ai, 0 + ai, 1 X + ai, 2 X 2 + … + ai, m - 1 X m - 1

Consider the case of m = 3, where the finite field is denoted GF(23). the mapping (developed later) of the seven elements {αi} and the zero element, in terms of the basis elements {X 0, X 1, X 2} described. Since Equation (10) indicates that α0 = α7, there are seven nonzero elements or a total of eight elements in this field. Each row in the mapping comprises a sequence of binary values representing the coefficients ai, 0, ai, 1, and ai, 2 in Equation (14). One of the benefits of using extension field elements {αi} in place of binary elements is the compact notation that facilitates the mathematical representation of nonbinary encoding and decoding processes. Addition of two elements of the finite field is then defined as the modulo-2 sum of each of the polynomial coefficients of like powers,

αi + αj = (ai, 0 + aj, 0) + (ai, 1 + aj, 1) X + … + (ai, m - 1 + aj, m - 1) X m - 1

2.4 APRIMITIVE POLYNOMIAL IS USED TO DEFINE THE FINITE FIELD

A class of polynomials called primitive polynomials is of interest because such functions define the finite fields GF(2m) that in turn are needed to define R-S codes. The following condition is necessary and sufficient to guarantee that a polynomial is primitive. An irreducible polynomial f(X) of degree m is said to be primitive if the smallest positive integer n for which f(X) divides X n + 1 is n = 2m - 1. Note that the statement A divides B means that A divided into B yields a nonzero quotient and a zero remainder. Polynomials will usually be shown low order to high order. Sometimes, it is convenient to follow the reverse format.

2.4.1 ENCODER

The Reed-Solomon encoder reads in k data symbols, computes the n - k parity symbols, and appends the parity symbols to the k data symbols for a total of n symbols. The encoder is essentially a 2t tap shift register where each register is m bits wide. The multiplier coefficients are the coefficients of the RS generator polynomial. The general idea is the construction of a polynomial; the coefficients produced will be symbols such that the generator polynomial will exactly divide the data/parity polynomial.

2.4.2 DECODER

The Reed-Solomon decoder tries to correct errors and/or erasures by calculating the syndromes for each codeword. Based upon the syndromes the decoder is able to determine the number of errors in the received block. If there are errors present, the decoder tries to find the locations of the errors using the Berlekamp-Massey algorithm by creating an error locator polynomial. The roots of this polynomial are found using the Chien search algorithm. Using Forney's algorithm, the symbol error values are found and corrected. For an RS (n, k) code where n - k = 2T, the decoder can correct up to T symbol errors in the code word. Given that errors may only be corrected in units of single symbols (typically 8 data bits), Reed-Solomon coders work best for correcting burst errors.

2.5 REED-SOLOMON ENCODING

The most conventional form of Reed-Solomon (R-S) codes in terms of the parameters n, k, t, and any positive integer m > 2.

(n, k) = (2m - 1, 2m - 1 - 2t) (20)

where n - k = 2t is the number of parity symbols, and t is the symbol-error correcting capability of the code. The generating polynomial for an R-S code takes the following form:

g(X) = g0 + g1 X + g2 X 2 + … + g2t - 1 X 2t - 1 + X 2t (21)

The degree of the generator polynomial is equal to the number of parity symbols. R-S codes are a subset of the Bose, Chaudhuri, and Hocquenghem (BCH) codes; hence, it should be no surprise that this relationship between the degree of the generator polynomial and the number of parity symbols holds, just as for BCH codes. Since the generator polynomial is of degree 2t, there must be precisely 2t successive powers of α that are roots of the polynomial. We designate the roots of g(X) as α, α2, …, α2t. It is not necessary to start with the root α; starting with any power of α is possible.

2.6 ENCODING IN SYSTEMATIC FORM

Since R-S codes are cyclic codes, encoding in systematic form is analogous to the binary encoding procedure. We can think of shifting a message polynomial, m(X), into the rightmost k stages of a codeword register and then appending a parity polynomial, p(X), by placing it in the leftmost n - k stages. Therefore we multiply m(X) by X n - k, thereby manipulating the message polynomial algebraically so that it is right-shifted n - k positions. Next, we divide X n - k m(X) by the generator polynomial g(X), which is written in the following form:

X n - k m(X) = q(X) g(X) + p(X)

where q(X) and p(X) are quotient and remainder polynomials, respectively. As in the binary case, the remainder is the parity. Equation can also be expressed as follows:

p(X) = X n - k m(X) modulo g(X)

The resulting codeword polynomial, U(X) can be written as

U(X) = p(X) + X n - k m(X)

2.6.1 SYSTEMATIC ENCODING WITH AN (N-K) – STAGE SHIFT REGISTER

Using circuitry to encode a three-symbol sequence in systematic form with the (7, 3) R-S code described by g(X) in the implementation of a linear feedback shift register (LFSR) circuit. It can easily be verified that the multiplier terms taken from left to right, correspond to the coefficients of the polynomial in low order to high order.
This encoding process is the non-binary equivalent of cyclic encoding. Here, corresponding to the (7, 3) R-S nonzero code words are made up of 2m - 1 = 7 symbols, and each symbol is made up of m = 3 bits.

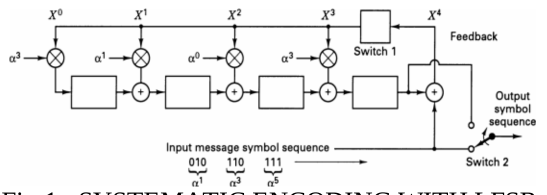

X^0
X^1
X^2
X^3
X^4
Feedback
Switch 1
α^3
α^1
α^0
α^3
Output symbol sequence
Input message symbol sequence
$\underbrace{010}_{\alpha^1}$ $\underbrace{110}_{\alpha^3}$ $\underbrace{111}_{\alpha^5}$
Switch 2

Fig 1: SYSTEMATIC ENCODING WITH LFSR

Here the example is non-binary, so that each stage in the shift register of Figure holds a 3-bit symbol. In the case of binary codes, the coefficients labeled g1, g2, and so on are binary. Therefore, they take on values of 1 or 0, simply dictating the presence or absence of a connection in the LFSR. However in Figure 9, since each coefficient is specified by 3-bits, it can take on one of eight values. The non-binary operation implemented by the encoder of Figure, forming code words in a systematic format, proceeds in the same way as the binary one. The steps can be described as follows:

1. Switch 1 is closed during the first k clock cycles to allow shifting the message symbols into the (n - k)–stage shift register.
2. Switch 2 is in the down position during the first k clock cycles in order to allow simultaneous transfer of the message symbols directly to an output register.
3. After transfer of the kth message symbol to the output register, switch 1 is opened and switch is moved to the up position.
4. The remaining (n - k) clock cycles clear the parity symbols contained in the shift register by moving them to the output register.
5. The total number of clock cycles is equal to n, and the contents of the output register is the codeword polynomial p(X) + X n - k m(X), where p(X) represents the parity symbols and m(X) the message symbols in polynomial form.

2.7 REED-SOLOMON DECODING

Earlier, a test message encoded in systematic form using a (7, 3) R-S code resulted in a codeword polynomial described. Now, assume that during transmission this codeword becomes corrupted so that two symbols are received in error. This number of errors corresponds to the maximum error-correcting capability of the code. For this seven-symbol codeword example, the error pattern, **e**(X), can be described in polynomial form as follows:

$$e(x) \;\; = \sum_{n=0}^{6} e_n X^n$$

For this example, let the double-symbol error be such that one parity symbol has been corrupted with a 1-bit error (seen as α2), and one data symbol has been corrupted with a 3-bit error (seen as α5). The received corrupted-codeword polynomial, **r**(X), is then represented by the sum of the transmitted-codeword polynomial and the error-pattern polynomial as follows:

r(X) = U(X) + e(X)

Following Equation, we add U(X)

$R(x) =(100)+(001)x+(011)x^2+(100)x^{3+}(101)x^4+ (110)x^5+(111)x^6$

In this example, there are four unknowns—two error locations and two error values. Notice an important difference between the nonbinary decoding of **r**(X) that we are faced with in Equation and binary decoding; in binary decoding, the decoder only needs to find the error locations. Knowledge that there is an error at a particular location dictates that the bit must be "flipped" from 1 to 0 or vice versa. But here, the non-binary symbols require that we not only learn the error locations, but also determine the correct symbol values at those locations. Since there are four unknowns in this example, four equations are required for their solution.

2.7.1 ERROR LOCATION

Suppose there are ν errors in the codeword at location X j1 , X j2 , ... , X jν . Then, the error polynomial **e**(X) shown

$E(x) = e_{j1}x^{j1}+ e_{j2}x^{j2}+\ldots.+ e_{j\nu}x^{j\nu}$

The indices 1, 2, … ν refer to the first, second, …, νth errors, and the index j refers to the error location. To correct the corrupted codeword, each error value e jl and its location X jl , where l = 1, 2, ..., ν, must be determined. We define an error locator number as jl l β = α . Next, we obtain the n - k = 2t syndrome symbols by substituting αi into the received polynomial for i = 1, 2, … 2t

There are 2t unknowns (t error values and t locations), and 2t simultaneous equations. However, these 2t simultaneous equations cannot be solved in the usual way because they are nonlinear (as some of the unknowns have exponents). Any technique that solves this system of equations is known as a Reed-Solomon decoding algorithm.

III. LOW-COMPLEXITY CHASE DECODING OF RS CODES

Reed Solomon codes are error-correcting codes that have found wide-ranging applications throughout the fields of digital communication and storage. Some of which include:

- Storage Devices (hard disks, compact disks, DVD, barcodes, etc.)
- Wireless Communication (mobile phones, microwave links, etc.)
- Digital Television
- Broadband Modems (ADSL, xDSL, etc.)
- Deep Space and Satellite Communications Networks (CCSDS)

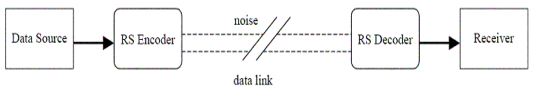


Fig 2: Applications of RS code

RS codes are systematic linear block codes, residing in a subset of the BCH codes called non-binary BCH. It is block because the original message is split into fixed length blocks and each block is split into m bit symbols; linear because each m bit symbol is a valid symbol; and systematic because the transmitted information contains the original data with extra CRC or 'parity' bits appended. These codes are specified as RS (n, k), with m bit symbols. This means that the encoder takes k data symbols of m bits each, appends n - k parity symbols, and produces a code word of n symbols (each of m bits).

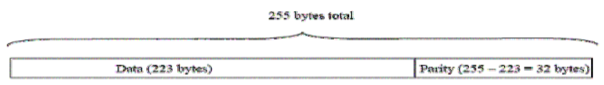


Fig 3: Modified REED SOLMON code

Reed Solomon codes are based on a specialized area of mathematics known as Galois fields (a.k.a. finite fields). These fields are of the form GF (p^m), where p is prime. RS makes use of Galois fields of the form GF (2^m), where elements of the field can be represented by m binary bits. Hence, RS codes of the form RS (2^8) lend themselves well to digital communication.Reed-Solomon codes are powerful error-correcting codes that can be found in a wide variety ofdigital communications systems, from digital media to wireless communications and deep-spaceprobes. The ubiquitous nature of these codes continues to fuel research into decoding algorithmssome forty years after their introduction. Reed-Solomon codes have been employed in a wide spectrum of digital communications systems because they provide powerful error correction with only a small number of overhead symbols. Reed-Solomon codewords consist of non-binary symbols and therefore the correction of a single symbol could result in the correction of more than one of the constituent bits. For this reason, Reed-Solomon codes are well suited to the correction of burst errors.Classical decoders for Reed-Solomon codes of length n and dimension k can correct up to t = bdmin/2c errors where dmin = (n−k+1) is the minimum distance of the code. Recently, a new class of list decoding algorithms has been introduced that can sometimes correct an even greater number of errors. The list decoding problem is to find the set of codewords at a Hamming distance of t0 from the received word. If t0 > dmin/2 there might not be a unique codeword so the decoder returns a list of candidate codewords. The Guruswami-Sudan (GS) list decoding algorithm has t0 as large as n – p nk errors. To improve the error-correction capability of a decoder even further, the decoder should take advantage of the soft reliability information available from the channel. Soft-decision decoders can provide an asymptotic gain of 2-3 dB on Gaussian channels and 10 dB or more on Rayleigh fading channels. Traditional hard-decision Reed-Solomon decoding algorithms are efficient because they are algebraic; that is, they exploit the underlying algebraic structure of the code to generate a system of equations that is solved using finite field arithmetic. However, an algebraic decoder based on finite field arithmetic does not appear to be compatible with the real-valued, soft information available from the channel and therefore it has been a research challenge to develop an algebraic soft-decision Reed-Solomon decoder. Koetter and Vardy have recently proposed an algebraic soft decision decoding algorithm by extending the list decoder of Guruswami and Sudan to include a method for converting soft information into algebraic conditions. The Koetter-Vardy (KV) algorithm can achieve up to about 4 dB of coding gain at a frame-error-rate (FER) of 10−3 on a Gaussian noise channel (with a practical range of 1–1.5 dB) and gains of 2–7 dB on a Rayleigh fading channel.The Koetter-Vardy soft-decision decoding procedure shows a lot of promise from the point of view of error correcting performance. At a first glance, the algorithm seems to be quite computationally complex and not straightforward to implement in VLSI. This paper aims to introduce techniques that reduce the complexity of interpolation-based decoders to the point where an efficient VLSI implementation is possible. A review of the GS and KV list-decoding algorithms. The techniques for significantly reducing the complexity and memory requirements of interpolation-based decoders. A VLSI architecture is then developed that reduces the complexity of evaluating the Hasse derivative, one of the main tasks in interpolation.

IV. INTERPOLATION-BASED LIST DECODING ALGORITHM

We want to transmit a message f. The bits of the message can be grouped into log2(q)- bit symbols chosen from the finite field with q elements, GF(q). An (n, k) Reed-Solomon code over GF(q) represents the k-symbol message, f = (f0, f1, f2, . . . , fk−1) by an n-symbol codeword, c = (c0, c1, c2, . . . , ck−1, . . . , cn−1), where n > k and usually n = q − 1. The k symbols of the message f can be considered to be the coefficients of the up to degree (k − 1) univariate message polynomial:

f(x) = f0 + f1x + f2x2 + . . . + fk−1xk−1.

We use the classical view of Reed-Solomon codes taken from the original definition, with this evaluation map encoding method, a codeword is formed by evaluating the message polynomial f(x) at n elements of GF(q). If the set of evaluation elements is X = {x0, x1, . . . , xn−1}, the codeword c is:

c = (f(x0), f(x1), . . . , f(xn−1)), xi **E** X.

We will always assume that n = q −1 and the set of evaluation elements X is the set of nonzero elements of GF(q):

X4={x0, x1, x2, . . . , xn−1}4={1, _, _2, . . . , _n−1}

where Xn is a primitive n’th root of unity. The evaluation map encoding method is useful because, it provides insight leading to interpolation-based decoding algorithms.

Guruswami-Sudan algorithm

An interpolation-based decoder takes the point of view that a codeword is a message polynomial evaluated at points in a finite field and uses polynomial interpolation to try to reconstruct that polynomial. The Guruswami-Sudan (GS) algorithm is an interpolation-based list decoder for Reed- Solomon codes. To describe the algorithm, we will first need to review some notation and facts about bivariate polynomials, which are the basic data structures in the algorithm. Consider the bivariate polynomial with coefficients chosen from a finite field:

$$P(x,y) = \sum_{a=0}^{\infty}\sum_{b=0}^{\infty} p_{a,b} x^a y^b \in \mathrm{GF}(q)[x,y] \; .$$

Consider the received word y = c + e, where e is an error vector with components drawn from GF(q). Since each component of c was generated by evaluating f(x) at a unique value of x 2 X, a unique xi can be associated with each received yi 2 GF(q) to form the list of points,

L = {(x0, y0), (x1, y1), . . . , (xn−1, yn−1)}.

If there is no noise (e = 0), then yi = f(xi), 0 _ i < n, and a bivariate polynomial, P(x, y) = y−f(x), passes through all the points in L with a multiplicity of one. This suggests that an interpolation-based approach can be used to decode Reed-Solomon codes. In the presence of noise (e 6= 0), the interpolation polynomial will pass through some points that are not part of the codeword. The GS algorithm ensures that under certain conditions, the codeword polynomial “lives inside” the interpolation polynomial [2, 3]. The GS algorithm is an interpolation-based list decoder with two main steps:

1. Interpolation Step: Given the set of points L and a positive integer m, compute P(x, y) of GF(q)[x, y]\{0} of minimal (1, k −1)-weighted degree that passes through all the points in L with multiplicity at least m.

2. Factorization Step: Given the interpolation polynomial P(x, y), identify all the factors of P(x, y) of the form y − f(x) with deg f(x) < k. The output of the algorithm is a list of the code words that correspond to these factors.

A complete factorization of P(x, y) is not necessary since we are just looking for linear y-roots of degree < k. An appropriate root-finding algorithm is given. The multiplicity, m, functions as a user-selectable complexity parameter. The error-correcting ability of the GS algorithm increases as the value of m increases. Unfortunately, so does the decoding complexity. Primitive polynomials are of interest here because they are used to define the Galois field.

A popular choice for a primitive polynomial is:

$$p(x) = x^8 + x^7 + x^2 + x^1 + 1$$

This is also known as the 0x87 polynomial, corresponding to the binary representation of the polynomial's coefficients excluding the MSB (i.e. 10000111). This specific polynomial is used in the CCSDS specification for a RS (255, 223). In GF (2^8) there are 16 possible primitive polynomials.

The VOCAL implementation has the ability to perform all combinations of RS (n, k) [n = 255, and 0 < k < n], for any of the 16 possible Galois fields, including the 0x87 field used by CCSDS. Additionally, the VOCAL RS modules can use any arbitrary generator polynomial for the calculation of the parity symbols.

V. REED SOLOMON IMPLEMENTATIONS

The implementations below can be customized to work with other RS (n, k) codes to yield similar results in performance. Optimized Software Implementation: The pure software implementation is dominated computationally by multiplication over a finite field (Galois Field multiplication). The encoder requires 71,181 cycles per codeword on a MIPS32 processor and the decoder requires 66,045 cycles. Scalar GF Multiply Support: This is the simplest form of VOCAL's hardware acceleration. The Scalar GF Multiply Support extends the capabilities of the MIPS32 processor by taking advantage of MIPS Technologies CorExtend capability to decrease the number of cycles to 23,305 cycles to encode and 9,174 cycles per codeword to decode on the MIPS32 processor.SIMD GF Multiply Support: The SIMD GF Multiply Support requires 128 bytes of local ROM Memory, but increases the performance to 3,918 cycles per megabit to encode and 3,078 cycles per codeword to decode. RS Encode Kernel. The RS Encode Kernel uses 1024 bytes of local ROM memory to encode. The number of cycles to process a codeword on a MIPS32 CPU falls to 2,702 cycles for encoding and decoding only consumes 828 cycles with this implementation.

5.1 METHODOLOGIES

Methodologies are the principles and explanations of High-Throughput Interpolator Architecture for Low-Complexity Chase Decoding of RS Codes. And here we have Five types of modules are used.

MODULES

1. Registers
2. Multiplexers
3. D-flipflop
4. Gf(2^8) multiplier
5. Gf(2^8) adder
6. Polynomial Evaluation
7. Polynomial update

MODULE DESCRIPTIONS

5.1.1 REGISTERS

Actual definition of Register is "a combinational of flip-flops". Flip-flops are used as data storage elements. Such data storage can be used for storage of computer science, and such a circuit is described as sequential logic. **Shift Register** is another type of sequential logic circuit that is used for the storage or transfer of data in the form of binary numbers and then "shifts" the data out once every clock cycle, hence the name "shift register". It basically consists of several single bits "D-Type Data Latches", one for each bit (0 or 1) connected together in a serial or daisy-chain arrangement so that the output from one data latch becomes the input of the next latch and so on. The data bits may be fed in or out of the register serially, i.e. one after the other from either the left or the right direction, or in parallel, i.e. all together. The number of individual data latches required to make up a single **Shift Register** is determined by the number of bits to be stored with the most common being 8-bits wide.

The Shift Register is used for data storage or data movement and are used in calculators or computers to store data such as two binary numbers before they are added together, or to convert the data from either a serial to parallel or parallel to serial format. The individual data latches that make up a single shift register are all driven by a common clock signal making them synchronous devices.

Generally, shift registers operate in one of four different modes with the basic movement of data through a shift register being:

- **Serial-in to Parallel-out -** In this serial-in to parallel-out, the register is loaded with serial data, one bit at a time, with the stored data being available in parallel form.
- **Serial-in to Serial-out -** In this serial-in to serial-out, the register is loaded with the serial data is shifted serially "IN" and "OUT" of the register, one bit at a time in either a left or right direction under clock control.
- **Parallel-in to Serial-out -** In this parallel-in to serial-out, the register is loaded with the parallel data is loaded into the register simultaneously and is shifted out of the register serially one bit at a time under clock control.
- **Parallel-in to Parallel-out -** In this parallel –in to parallel-out, the register is loaded with the parallel data is loaded simultaneously into the register, and transferred together to their respective outputs by the same clock pulse.
- **Universal shift registers-** Today, high speed bi-directional "universal" type Shift Registers are available as a 4-bit multi-function devices that can be used in either serial-to-serial, left shifting, right shifting, serial-to-parallel, parallel-to-serial, and as a parallel-to-parallel multifunction data register, hence the name "Universal". These devices can perform any combination of parallel and serial input to output

operations but require additional inputs to specify desired function and to pre-load and reset the device.

5.1.2 MULTIPLEXERS

A 2n-to-1 multiplexer sends one of 2n input lines to a single output line. A multiplexer has two sets of inputs: 2n data input lines, n select lines, to pick one of the 2n data inputs. The mux output is a single bit, which is one of the 2n data inputs. A 2n-to-1 multiplexer routes one of 2n input lines to a single output line. Just like decoders, muxes are common enough to be supplied as stand-alone devices for use in modular designs. Muxes can implement arbitrary functions. Smaller muxes can be combined to produce larger ones. It can add active-low or active-high enable inputs. As always, we use truth tables and Boolean algebra to analyze things. Tune in tomorrow as we start to discuss how to build circuits to do arithmetic.

5.1.3 D-FLIP-FLOP

There are some circuits that are not quite as straight forward as the gate circuits. However, we still need to learn about circuits that can store and remember information. They're the kind of circuits that are used in computers to store program information - RAM memory.
The combination of two flip-flops constitutes a D-type flip-flop. That's D because the output of the flip-flop is delayed by the time of one clock pulse. Set a value for the data and pulse the clock ON and OFF. We'll find a copy of the data appearing at the output on the trailing edge of the clock pulse. Now, if we consider the combination of two flip-flops as a unit, we have a D flip-flop. It's called a D flip-flop because it delays the signal. The signal appears at the output of the circuit delayed by the time of one clock pulse.

5.1.4 GF (2^8) MULPTIPLIER

Galois Field Theory (GFT) deals with numbers that are binary in nature, have the properties of a mathematical "field," and are finite in scope. Although some Galois computations don't exist in ordinary mathematics, many Galois operations match those of regular math. Addition (Ex-Or) and multiplication are common Galois operations, and logarithms, particularly, are handy for checking multiplication results. For over 40 years, Galois Field multipliers have been used both for coding theory and for cryptography. Both areas are complex, with similar needs, and both deal with fixed symbolic alphabets that neatly fit the extended Galois Field model.

This application note will focus primarily on cryptographic applications of GFT, and will present some practical design solutions that have been synthesized and simulated for ready use. While the basic multiplier structure used by the solutions clearly has its roots in the designs of Berlekamp and Massey from the 1960s, the specific structure used here comes from a more recent paper by Johann Großschädl at IAIK (Graz University of Technology, Austria). This application note does not delve deeply into GFT, although its appendices point out some enlightening tutorial material for interested readers. Its goal instead is to deliver a series of multiplier solutions and verify their correctness and usabilty. The specific results and tools presented will then be applicable to other multiplier versions of varying lengths. To this end, we first present a 4-bit multiplier and its verification. We then expand it into an 8-bit multiplier, doing the same, and finally into a 163-bit multiplier. The larger multiplier can eventually be used as part of a solution for Elliptic Curve Cryptography using one of the NIST-recommended curves and the NIST chosen irreducible polynomial. A complete verification of this larger multiplication is an ordeal, but a few examples will be presented to assure readers of its validity.

5.1.5 The Finite Field GF(2^8).

The case in which n is greater than one is much more difficult to describe. In cryptography, one almost always takes p to be 2 in this case. This section just treats the special case of p = 2 and n = 8, that is. GF(2^8), because this is the field used by the new U.S. Advanced Encryption Standard (AES). The AES works primarily with bytes (8 bits), represented from the right as:
$b_7b_6b_5b_4b_3b_2b_1b_0$.
The 8-bit elements of the field are regarded as polynomials with coefficients in the field Z_2:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0.$$

The field elements will be denoted by their sequence of bits, using two hex digits.

5.1.6 Multiplication in GF(2^8)

Multiplication is this field is much more difficult and harder to understand, but it can be implemented very efficiently in hardware and software. The first step in multiplying two field elements is to multiply their corresponding polynomials just as in beginning algebra (except that the coefficients are only 0 or 1, and 1 + 1 = 0 makes the calculation easier, since many terms just drop out). The result would be up to a degree 14 polynomial -- too big to fit into one byte. A finite field now makes use of a fixed degree eight irreducible polynomial (a polynomial that cannot be factored into the product of two simpler polynomials). For the AES the polynomial used is the following (other polynomials could have been used):

$$m(x) = x^8 + x^4 + x^3 + x + 1 = \text{0x11b (hex)}.$$

The intermediate product of the two polynomials must be divided by m(x). The remainder from this division is the desired product. This sounds hard, but is easier to do by hand than it might seem (though error-prone). To make it easier to write the polynomials down, adopt the convension that instead of $x^8 + x^4 + x^3 + x + 1$ just write the exponents of each non-zero term. (Remember that terms are either zero or have a 1 as coefficient.)

5. GF (2^8) ADDER

To add two field elements, just add the corresponding polynomial coefficients using addition in Z_2. Here addition is modulo 2, so that 1 + 1 = 0, and addition, subtraction and exclusive-or are all the same. The identity element is just zero: 00000000 (in bits) or 0x00 (hex).

VI. SIMULATION RESULTS

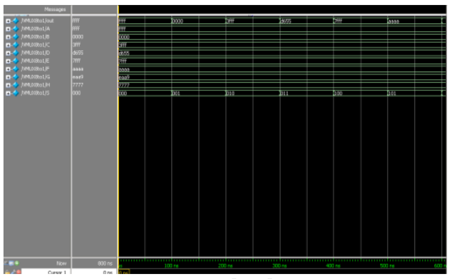


Fig 4: Multiplexer

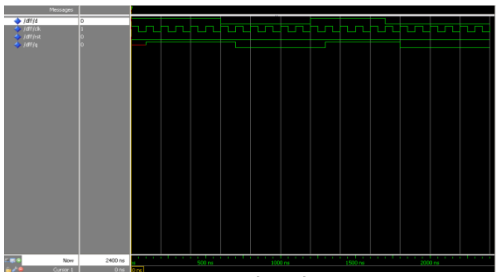


Fig 5: D flip-flop

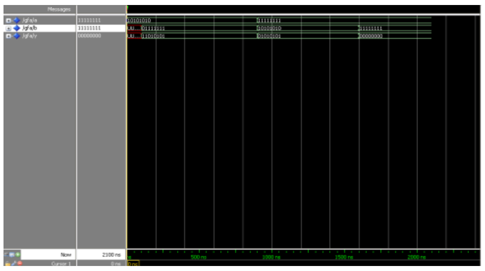


Fig 6: GF(2^8) ADDITION

Fig 7: GF(2^8)) MULTIPLIER

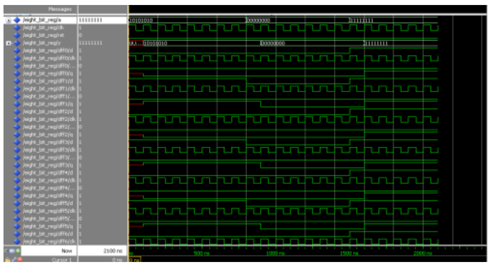


Fig 8: 8 - BIT REGISTER

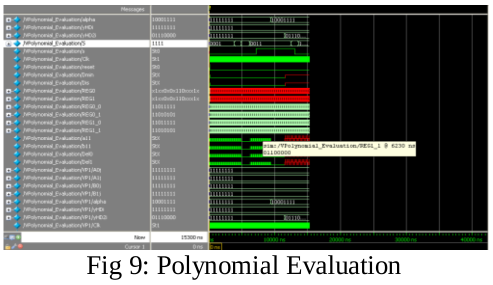


Fig 9: Polynomial Evaluation

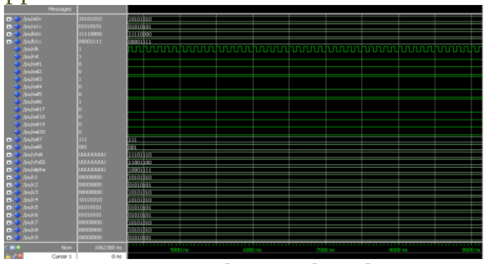
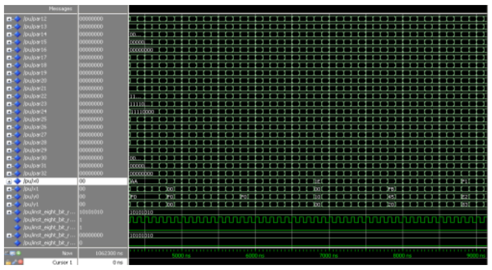


Fig 10,11: Polynomial Update



VII. CONCLUSION

The modified Nielson's algorithm, which works with a different scheduling, takes care of the limited growth of the polynomials and shares the common interpolation points, for reducing the latency of interpolation. Based on the proposed modified Nielson's algorithm, we have derived a low-latency interpolator architecture. An LCC decoder using our low-latency interpolator is found to be at least 39% more efficient in terms of area-delay product over the best of previous works.

REFERENCES

[1] R. Koetter and A. Vardy, "Algebraic soft-decision decoding of Reed–Solomon codes," IEEE Trans. Inf. Theory, vol. 49, no. 11, pp. 2809–2825, Nov. 2003.

[2] A. Ahmed, N. R. Shanbhag, and R. Koetter, "An architectural comparision of Reed–Solomon soft-decoding algorithm," Signals, Syst. Comput., pp. 912–916, 2006.

[3] W. J. Gross, F. R. Kschischang, R. Koetter, and P. G. Gulak, "Architecture and implementation of an interpolation processor for soft-decision Reed–Solomon decoding," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 15, no. 3, pp. 309–318, Mar. 2007.

[4] X. Zhang, "Reduced complexity interpolation architecture for soft-decision Reed–Solomon decoding," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 14, no. 10, pp. 1156–1161, Oct. 2006.

[5] Z. Wang and J. Ma, "High-speed interpolation architecture for softdecision decoding of Reed–Solomon codes," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 14, no. 9, pp. 937–950, Sep. 2006.

[6] J. Zhu and X. Zhang, "Efficient VLSI architecture for soft-decision decoding of Reed–Solomon codes," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 55, no. 10, pp. 3050–3062, Nov. 2008.

[7] J. Bellorado and A. Kavcic, "A low-complexity method for Chase-type decoding of Reed–Solomon codes," Proc. ISIT, pp. 2037–2041, Jul. 2006.

[8] X. Zhang and J. Zhu, "High-throughput interpolation architecture for algebraic soft-decision Reed–Solomon decoding," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 57, no. 3, pp. 581–591, Mar. 2010.
[9] J. Zhu, X. Zhang, and Z. Wang, "Backward interpolation architecture for algebraic soft-decision Reed–Solomon decoding," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 17, no. 11, pp. 1602–1615, 2009.
[10] T. K. Moon, Error Correction Coding: Mathematical Methods and Algorithms. Hoboken, NJ: Wiley, 2004.
[11] W. J. Gross, F. R. Kschischang, R. Koetter, and P. G. Gulak, "A VLSI architecture for interpolation-based in soft-decision list decoding of Reed–Solomon decoders," J. VLSI Signal Process., vol. 39, no. 1–2, pp. 93–111, 2005.
[12] F.Parvaresh and A. Vardy, "Multiplicity assignments for algebraic soft-decoding of Reed–Solomon codes," in Proc. ISIT, 2003, pp. 205–205.
[13] X. Zhang, "High-speed VLSI architecture for low-complexity Chase soft-decision Reed–Solomon decoding," in Proc. Inf. Theory Applic. Workshop, San Diego, CA, Feb. 2009.
[14] J. Ma, A.Vardy, and Z.Wang, "Reencoder design for soft-decision decoding of an (255,239) Reed–Solomon code," in Proc. IEEE Int. Symp. Circuits Syst., Island of Kos, Greece, May 2006, pp. 3550–3553.

Authors

S.Susrutha Babu was born in India, A.P. He received the B.Tech degree from JNTU, A.P, and M.Tech degree from SRM University, Chennai, Tamil Nadu, India in 2008 and 2010 respectively. He worked as **Assistant Professor** in Electronics Engineering in Bapatla Engineering College for academic year 2010-2011 and from 2011 to till date working in **K L University**. He is a member of Indian Society for Technical Education and International Association of Engineers. His research interests include antennas, FPGA Implementation, Low Power Design and wireless communications and Digital VLSI. He has published articles in various international journals and Conference in IEEE.

S.Suparshya Babu was born in India, A.P. He received the B.Tech degree from JNTU, A.P, and M.Tech degree from SRM University, Chennai, Tamil Nadu, India in 2008 and 2010 respectively. He worked as **Assistant Professor** in Electronics Engineering in Bapatla Engineering College for academic year 2010-2011 and from 2011 to till date working in **K L University**. He is a member of Indian Society for Technical Education and International Association of Engineers. His research interests include antennas, FPGA Implementation, Low Power Design and wireless communications and Robotics. He has published articles in various international journals and Conference in IEEE

S R Sastry Kalavakolanu was born in A.P,India. He received the B.Tech degree in Electronics & communications Engineering from Jawaharlal Nehru Technological University in 2010. Presently he is pursuing M.Tech VLSI Design in KL University. His research interests include Low Power VLSI Design. He has undergone 3 International Journals and 1 publishment in IEEE.

M.Aravind Kumar was born in A.P,India He received his B.Tech degree in Electronics and Communication Engineering from S.V.H college of Engineering and Technology in the year 2005. He is presently pursuing masters in VLSI system design (4th semester) at Padmasri Dr.B.V.Raju Institute of Technology, Medak(dst), A.P.India.

P.Bose Babu was born in A.P,India, He Completed M.Tech in VLSI system Design from MVGR COLLEGE OF ENGG.&TECH, Vizianagaram in 2011 and B.Tech from QIS College of engineering, in 2009 in Electronics &Communication engineering. Presently he is Working as Asst.Professor in ANDHRA LOYOLA college of Engg.& Technology, Vijayawada,A.P,India.

G.Roopa Krishna Chandra was born in A.P,India He received his B.Tech degree in Electronics and Communication Engineering from Sri Sarathi Institute of Engineering and Technology year 2009. He has completed M.Tech in Lakireddy Bali Reddy College of engineering in 2012. His Research areas are Adaptive Signal Processing, Embedded Systems.