International Journal of Modern Engineering Research (IJMER)
www.ijmer.com Vol.2, Issue.5, Sep-Oct. 2012 pp-3769-3772 ISSN: 2249-6645

Iterative Multivariate Interpolation for Low Complexity Reed-Solomon Codes

N. Sireesha[1], V. Prasanth[2]
[1] PG Scholar, ECE, Pragati Engineering College, JNTUK, AP, INDIA
[2] Associate Professor, ECE, Pragati Engineering College, JNTUK, AP,INDIA

ABSTRACT : *The algebraic soft-decoding (ASD) of Reed–Solomon (RS) codes provides significant coding gain over hard-decision decoding with polynomial complexity. In order to reduce the complexity 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. 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. We have implemented the proposed interpolator in CYCLONE III FPGA which provides 1.3 Gb/s throughput.*

Keywords: *Algebraic soft-decision decoding, interpolation, low-complexity chase (LCC), low latency, Nielson's algorithm, Reed–Solomon (R-S) codes.*

I. INTRODUCTION

The Reed-Solomon codes (RS codes) are non binary cyclic codes with code symbols from a Galois field .we want to transmit a message f. The bits of the message can be grouped in to 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. (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.

c = (f(x0), f(x1), . . . , f(xn−1)), xi □X. (2)

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} (3)

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 [5] 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 [14]with coefficients chosen from a finite field:

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

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[3] 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 (e6= 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].

II. INTERPOLATION

The GS algorithm is an interpolation-based list decoder with two main steps [11].

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 codewords 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.

Primitive polynomials are of interest here because they are used to define the Galois field. A popular choice for a primitive polynomial is:

$$p(x) = x^8 + x^7 + x^2 + x^1 + 1 \quad (4)$$

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.

A. FINITE FIELDS

In order to understand the encoding and decoding principles of nonbinary 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 elements by progressively multiplying the last entry by α.

$$F = \{0, 1\alpha, \alpha 2, \ldots, \alpha j, \ldots\} = \{0, \alpha 0, \alpha 1, \alpha 2, \ldots, \alpha j, \ldots\} \quad (5)$$

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:

$$\alpha(2m-1) + 1 = 0 \quad (6)$$

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:

$$\alpha(2m+ n) = \alpha(2m-1)\, \alpha n+1 = \alpha n+1 \quad (7)$$

B. REVIEW OF LCC DECODING OF RS CODES

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.

III. SYNDROME COMPUTATION

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

$$0 < k < n < 2m + 2 \quad (8)$$

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,

$$(n, k) = (2m - 1, 2m - 1 - 2t) \quad (8a)$$

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. In 2*t* syndromes are calculated by evaluating received polynomial for powers of q.

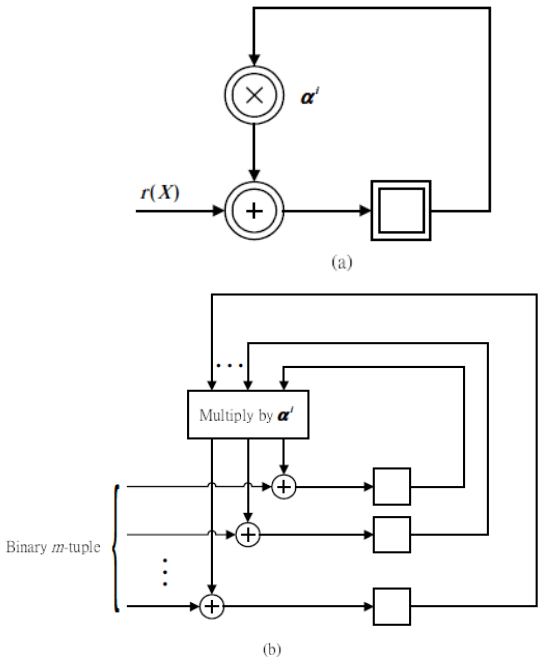

Figure 1: Syndrome computation circuits for Reed-Solomon codes: (a) over GF(2*m*); (b) in binary form.

Assume the transmitted code vector is

$\mathbf{t}(X) = t_0 + t_1X + t_2X^2 + \ldots + t_{n-1}X^{n-1}$,

and the received vector is

$\mathbf{r}(X) = r_0 + r_1X + r_2X^2 + \ldots + r_{n-1}X^{n-1}$.

The first step in decoding a Reed-Solomon code is to calculate the 2t syndrome components as:

$S_0 = \mathbf{r}(a^0) = r_0 + r_1 + r_2 + \ldots + r_{n-1}$

$S_1 = \mathbf{r}(a^1) = r_0 + r_1(a) + r_2(a)^2 + \ldots + r_{n-1}(a)^{n-1}$

$S_2 = \mathbf{r}(a^2) = r_0 + r_1(a^2) + r_2(a^2)^2 + \ldots + r_{n-1}(a^2)^{n-1}$

$S_{2t-1} = \mathbf{r}(a^{2t-1}) = r_0 + r_1(a^{2t-1}) + r_2(a^{2t-1})^2 + \ldots\ldots + r_{n-1}(a^{2t-1})^{n-1}$.

The syndrome polynomial is

$\mathbf{S}(X) = S_0 + S_1X + S_2X^2 + \ldots + S_{2t-1}X^{2t-1}$.

The second step in decoding a Reed-Solomon code is to find the error location polynomial $\mathbf{L}(X)$ and the error evaluation polynomial $\mathbf{W}(X)$.

The error location polynomial is

$\mathbf{L}(X)=1+L_1X+L_2X^2+\ldots+L_eX^e$,

and the error evaluation polynomial is

$\mathbf{W}(X)=W_0+W_1+W_2X^2+\ldots+W_{e-1}X^{e-1}$,

where e is the number of errors. The error location polynomial and the error evaluation polynomial are related to the syndrome polynomial through the key equation is

$\mathbf{L}(X)\mathbf{S}(X)=\mathbf{W}(X) \bmod X^{2t}$.

The popular iterative Berlekamp - Macey algorithmis

used to solve for L(X) and M(X).The last step in decoding a Reed-Solomon code is to find the error location and the error value. The error location is obtained using Chan's searching algorithm. Basically X is substituted with a^n in **L**(X) for all possible n in a code to find the root of **L**(X). The inverse of the root of the error location polynomial is the error position. After an error location is found, the error value is calculated via Forney's error evaluation algorithm. Once the error value is found, it is added to the corrupted symbol to correct the error.

IV. 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.

V. PROPOSED ARCHITECTURE

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

A. REGISTERS

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.

B. 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.

C. 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.

D. 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.

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: b7b6b5b4b3b2b1b0. The 8-bit elements of the field are regarded as polynomials with coefficients in the field Z2: b7x7 + b6x6 + b5x5 + b4x4 + b3x3 + b2x2 + b1x1 + b0. The field elements will be denoted by their sequence of bits, using two hex digits.

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) = x8 + x4 + x3 + x + 1 = 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 x8 + x4 + x3 + x + 1 just write the exponents of each non-zero term. (Remember that terms are either zero or have a 1 as coefficient.)

E. GF (2^8) ADDER

To add two field elements, just add the corresponding polynomial coefficients using addition in Z2. 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).

SIMULATION RESULTS

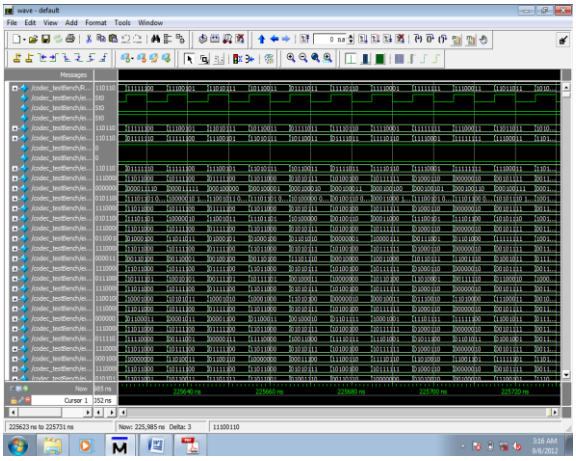

Figure 2: Simulated Output.

AREA UTILIZATION REPORT

Flow Summary	
Flow Status	Successful - Thu Sep 06 03:20:19 2012
Quartus II Version	11.0 Build 208 07/03/2011 SP 1 SJ Web Edition
Revision Name	test
Top-level Entity Name	RS8FreqDecode
Family	Cyclone III
Total logic elements	7,012 / 15,408 (46 %)
Total combinational functions	7,012 / 15,408 (46 %)
Dedicated logic registers	2,479 / 15,408 (16 %)
Total registers	2479
Dedicated logic registers	21 / 169 (12 %)
Total virtual pins	0
Total memory bits	2,048 / 516,096 (< 1 %)
Embedded Multiplier 9-bit elements	0 / 112 (0 %)
Total PLLs	0 / 4 (0 %)
Device	EP3C16F256C6
Timing Models	Final

Figure 3: Flow Summary Report

V. CONCLUSIONS

The proposed one is a hardware implementation of modified Nielson's algorithm, which works with a different scheduling, leads 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 multivariate interpolator architecture. This will reduce the number of iterations required for LCC decoder. Using our low-latency interpolator is found to be at least 45% more efficient in terms of area-delay product over the best of previous works. This architecture has been implemented in a CYCLONE-III FPGA device which provides a throughput of 1.3 Gb/s .

REFERENCES

[1] F. García-Herrero, M. J. Canet, J. Valls, and P. K. Meher "High-Throughput Interpolator Architecture for Low-Complexity Chase Decoding of RS Codes" *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 3, pp. 568–573, Mar. 2011

[2] 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.

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

[11] T. K. Moon, *Error Correction Coding: Mathematical Methods and Algorithms*. Hoboken, NJ: Wiley, 2004.

[12] 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.

[13] F. Parvaresh and A. Vardy, "Multiplicity assignments for algebraic soft-decoding of Reed–Solomon codes," in *Proc. ISIT*, 2003, pp. 205–205.

[14] 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.

[15] 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.