Susrutha Babu Sukhavasi, Suparshya Babu Sukhavasi, Satyanarayana P, S R Sastry K, Krishna Karthik T, Arafath Ali / International Journal of Engineering Research and Applications (IJERA)
ISSN: 2248-9622 www.ijera.com
Vol. 2, Issue 3, May-Jun 2012, pp.3009-3016

Design Of High-Throughput Interpolator Structural Design For Low-Complexity Pursue Decoding Of RS Codes

Satyanarayana P*, Susrutha Babu Sukhavasi*, Suparshya Babu Sukhavasi*
S R Sastry K, Krishna Karthik T**, Arafath Ali ***.**
*Faculty, Department of ECE, K L University, Guntur, AP, India.
**M.Tech -VLSI Student, Department of ECE, K L University, Guntur, AP, India.
***M.Tech –VLSI Student, Department of ECE, AEC, Nalgonda, AP, India.

ABSTRACT

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.

Keywords* - *RScodes, Guruswami-Sudan algorithm, Registers, Multiplexers ,D-flip flop.

I. INTRODUCTION

The algebraic soft-decoding (ASD) of Reed–Solomon (RS) Codes provides significant coding gain over hard-decision Decoding with polynomial complexity. The decoder in this case has three main functions:

1. Multiplicity assignment
2. Interpolation
3. Factorization of bi-variate 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.

EXISTING SYSTEM

RS (n, k) code over GF (2^q), with n=2^q-1 , and systematic encoding of the message. In such a case, 2t redundant symbols are added to the original k-symbol message to obtain the codeword c(x), where 2t=n-k. After the message is transmitted by a noisy channel, the RS decoder receives r(x) =c(x) + e(x), where e(x) is the polynomial representation of the noise. The first step of the ASD algorithms is the computation of the multiplicity matrix. There are two levels of reliabilities in the LCC decoder: m=0 for the unreliable symbols and m=1 for the most reliable symbols. To reduce the complexity of the decoder, re-encoding and coordinate transformation can be applied, which result is k- zeros on the reliable locations (corresponding to m=1), and 2t non-zero symbols in less reliable positions (corresponding to m=0). The k zeros are directly interpolated during re-encoding, making a univariate polynomial. The remaining 2t symbols require a bivariate interpolation.

Existing system disadvantages

Efficient re-encoder architecture has been presented. To match with the latency amounting to 528 clock cycles, a backward interpolator was introduced with 525 clock cycles of latency.

PROPOSED SYSTEM

Proposed modified nielson's algorithm for Low-latency interpolation Here, we present the modified

Nielson's interpolator algorithm to reduce the total latency. Since the LCC algorithm works with maximum multiplicity of m=1, and maximum y-degree of l=1, we include Nielson's interpolation algorithm for these parameters in Algorithm 1, where deg0 anddeg1 are the degrees of the bivariate polynomials to interpolate g0(x, y) and g1(x, y) respectively. Algorithm 1 has two stages: stage-1 (A1) performs the polynomial evaluation (PE) or discrepancy computation. Stage-2 (A2 and A3) performs the polynomial update (PU). There are two ways to update a polynomial: if the polynomial has minimum order and discrepancy is different from zero, then the x-degree of the polynomial is increased by one, while the y-degree remains unchanged, (step A3); otherwise, a linear combination of the two polynomials and their respective discrepancies, delta0 and delta1, are computed (step A2). This last updating mode doesn't modify the degree of the polynomial. When re-encoding is applied and the maximum multiplicity is taken to be one, the number of iterations needed in the interpolation algorithm is 2t. Considering the worst-case situation, in which one of the two polynomials is updated during all iterations by applying step A3, the highest possible order of a polynomial will be 2t x-degree. In each iteration, the number of coefficients of the minimum x-degree polynomial is increased only by one, until it reaches a maximum of 2t+1coefficients in the last iteration. So the required number of cycles changes from 2t+1* 2t, if the 2t coefficients are covered in each iteration, to (2t+1)*(2t+2)/2, if the growth of the order of polynomial is considered.

We have determined experimentally that the cost of hard-decision decoding with multiplicity rn at each point is an upper bound to the cost of soft-decision decoding with a maximum multiplicity of m (for high SNR, soft-decision decoding reduces to hard decision decoding). The cost is C I (n / 2) (m) (m+ l), which has a squared dependence on m and a linear dependence on n. This "multiplicity explosion", combined with a large value of n (e.g. n = 255) results in very high costs. For example, for a (255, k) Reed-Solomon code and m = 4, the cost is C = 2550 iterations. Each received symbol imposes 10 linear constraints on the interpolation problem. The second problem is the large memory requirement. Algorithm 1 needs to store b bivariate polynomials. Since a homogeneous linear system must have more unknowns than equations, the length (number of terms) of the polynomials must be at least C. The third problem is the implementation of the Hasse derivative which is a computationally demanding operation.

ADVANTAGES OF PROPOSED SYSTEM

We have formulated a 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.

II. 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)

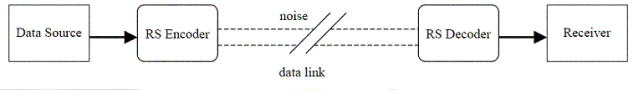

Fig 1: Applications of RS code

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).

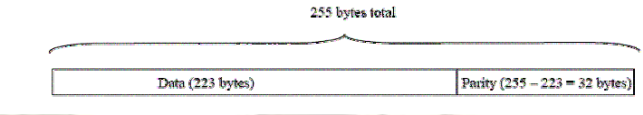

Fig 2: Modified REED SOLMON code

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 = $bd_{min}/2c$ 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.

III. 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.

A. 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.

B. 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.

C. 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.

D. A PRIMITIVE 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.

E. 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

MODULE DESCRIPTIONS

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.

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.

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.

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.

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.

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):

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

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).

IV. SIMULATION RESULTS

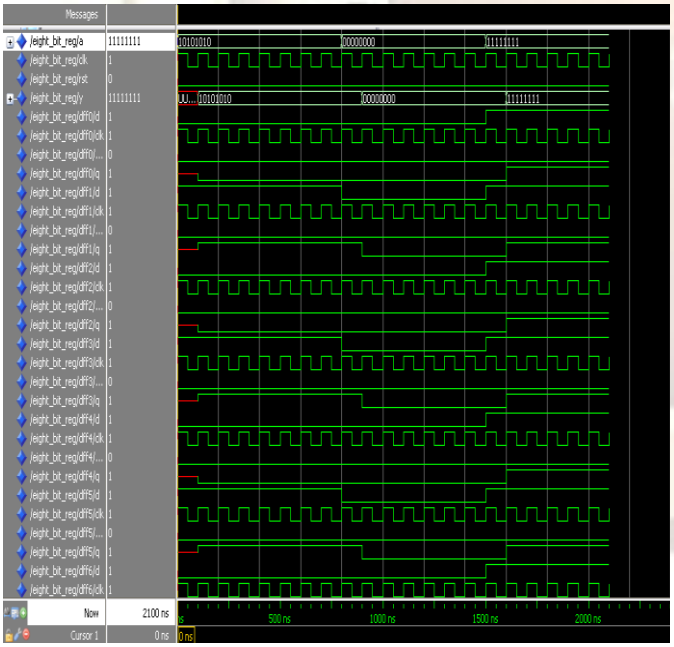

Fig 3: 8 - BIT REGISTER

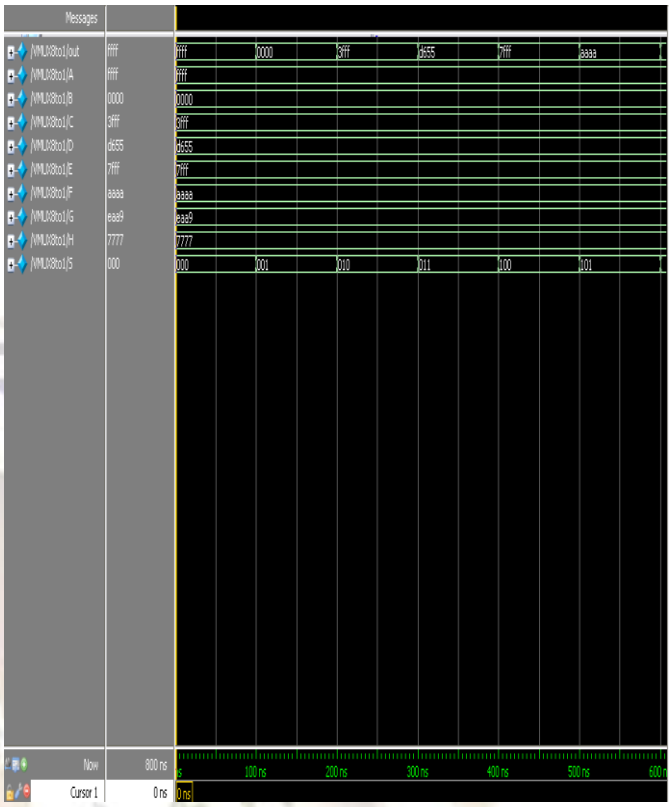

Fig 4: MULTIPLEXER

Fig 5: GF(2^8) ADDITION

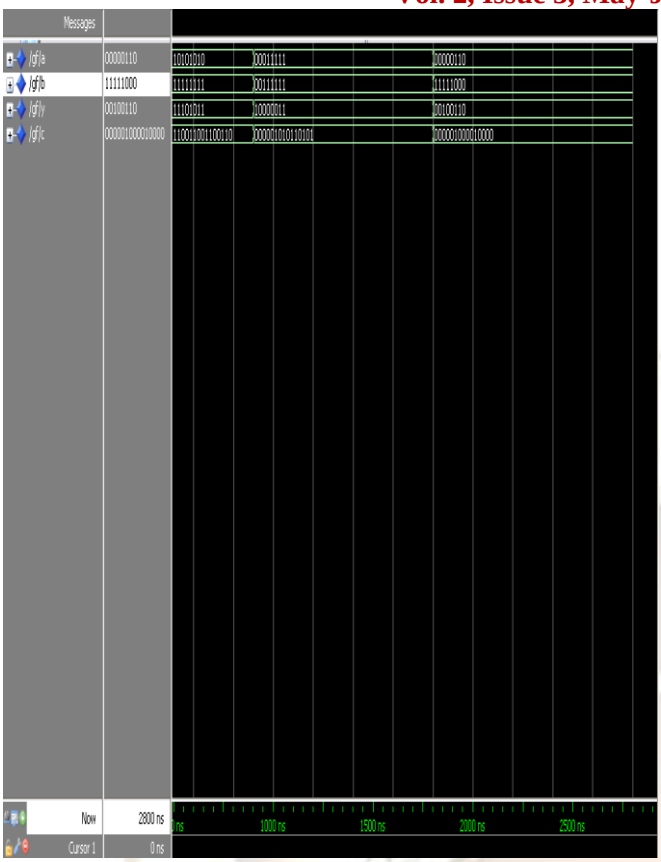

Fig 6: GF(2^8)) MULTIPLIER

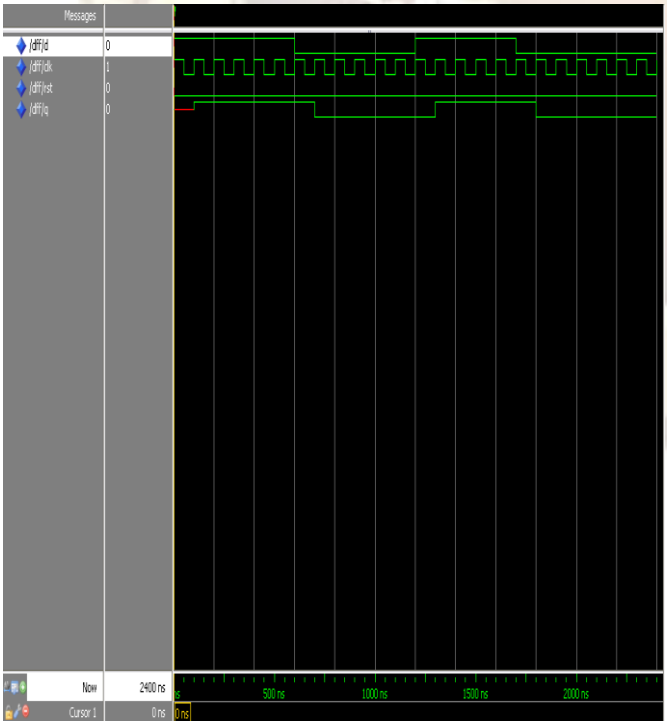

Fig 7: D flip-flop

V. 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.

VI. 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.*

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.

Mr. P.Satyanarayana was born on 20th May, 1977 in A.P, India. He received the B.Tech degree from Nagarjuna University in 2000, A.P and M.Tech degree from JNTU Kakinada in 2004. He is working as Associate Professor in ECE dept at K L University. His interested fields include wireless communication, VLSI, DSP and Embedded Systems. His research area is Wireless Communication.

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.

T. Krishna Karthik: was born in Gudivada, Krishna(dist),AP, India. He received B.Tech. in Electronics & Communication Engineering from GEC,AP. India Presently he is pursuing M.Tech VLSI Design in KL University, Vijayawada, AP, India. He has undergone 2 international conferences and 2 publishment in IEEE.

Arafath Ali 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 Anurag Engineering College. His research interests include Low Power VLSI Design.