Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 2, Issue. 4, April 2013, pg.116 – 125

RESEARCH ARTICLE

HDL IMPLEMENTATION OF ALGEBRAIC SOFT DECISION ALGORITHM FOR RS CODES

M. Revathy[1], R. Saravanan[2]
[1]Associate Professor, PSNA college of Engg.&Tech, Dindigul, India
[2]Professor, PSNA college of Engg.&Tech, Dindigul, India

[1] *revathim@psnacet.edu.in;* [2] *hodcse@psnacet.edu.in*

Abstract— Reed Solomon (RS) codes are widely used to detect and correct data errors in transmission and storage systems. Hence it is used in many digital communication and storage devices. In existing system Reformulated inversion less Burst error correcting (RiBC) algorithm is used. But it lacks in speed, throughput & area. To overcome this Algebraic Soft Decision (ASD) algorithm is proposed. This Proposed algorithm is based on Unified VLSI architecture for correcting burst errors as well as random errors. This new architecture is denoted as Unified Hybrid Decoding (UHD) architecture. It will be shown that, being the first RS decoder owning enhanced burst-error correcting capability, it can achieve high-speed, throughput and improved error correcting capability than traditional Hard Decision Decoding (HDD) design with less area.

Key Terms: - Algebraic Soft Decision; RiBC; Burst Error Correction; Hard decision Decoding

I. INTRODUCTION

The development of the information transmission technologies in the computer networks and in the telecommunication systems is inseparably connected with the problem of integrity and of ensuring high effectiveness of error detection and correction of errors which occur during data transmission.

The dynamic increase in the speed of information transmission in the buses of computer systems and the channels of computer networks brings about stringent requirements for the performance of the hardware implementation of the error detection and correction algorithm it must ensure the realization of the operations which are connected with the rate of error detection for data transmission.

A continuous increase in the speeds of data transmission in the telecommunications network systems is resulted in change of error character and effectiveness criterions of error detecting and correction. Today the main cause of data transmission errors is external impulse noise [1]. Data transmission speeds up the external impulse noise's influence on few adjacent channels signals. In consequence, the group of adjacent bits of transmitted data can be distorting. Such group of bits is named burst of errors. So, today bursts of bits distortions become dominating type of errors [1].

On the other hand, the dynamic increase of the data transmission speed has changed the importance such of tradition error control effectiveness criterions as number control bits and possibilities of detecting and correction errors in data rate transmission. It is clear that in modern condition of significant speeding up of data transmission, its increasing of the importance of computation complexity of detecting and correction errors procedure which impose the possibilities of real time error control. On the contrary, the modern tendency for increase of speed and values of data transmission decreases the importance of such tradition error control effectiveness criterions as number control bits.

Thus, the problem of detecting and correcting burst of errors in data rate transmission is very important and requires attention for the current and future technological developments in computer and telecommunications system. This specifies the urgency and the practical importance of the development of new methods and means to speed up the burst errors detection and correction procedures.

Reed-Solomon codes are a class of non-binary cyclic codes which have strong burst and erasure error correction capabilities. Their current uses include deep-space communications, compact disc recordings, cellular communications, and digital broadcast [2]. Their popularity stems from the existence of encoding and decoding algorithms that are simple to implement in hardware. In addition to their practical uses, these codes also have a nice theoretical structure for proving the feasibility of certain coding and storage schemes [3] [4].

Reed-Solomon code is a block code, means the transmitted message symbol is divided into separate blocks. RS codes are also known as linear codes in which a new code word produces by adding two code words. Generally RS codes are denoted as *(n, k)* code, where *k* is the data symbols and *n* is the total number of symbols in a codeword. The Reed Solomon codeword is constructed by adding the data symbols with self-generated parity symbols by the encoder, according to the message length and polynomial. The number of parity symbols added to the data symbols is *2t (n-k)*.

II. RiBC Algorithm

RS code is very effective in correcting long burst errors. However, previous RS burst decoding algorithms in [6] and [7] are infeasible for hardware implementation due to their high computation complexity. In [5], Wu proposed a new approach to track the position of burst of errors. By introducing a new polynomial that is a special linear function of syndromes, this approach can correct a long burst of errors with length up to $2t\text{-}1\text{-}2\beta$ plus a maximum of β random errors. Here β is a pre-chosen parameter that determines the specific error correcting capability. In this case, the miss correction probability is upper bounded by $(n\text{-}2f)(n\text{-}f)^{\beta}2^{m(\beta+f-2t)}$. Although the approach in [5] has reduced computation complexity, it still contains inversion operation and long data path, which impedes its efficient VLSI implementation, therefore, the algorithm in [5] was reformulated to the RiBC algorithm. The RiBC algorithm is a kind of list decoding algorithm. 8 polynomials are updated simultaneously in each iteration. After every 2β inner iterations, $\dot{\Lambda}^{(2\beta)}(x)$, as the candidate of the error locator polynomial of the random errors, is computed for current *l*-th outer iteration. When *l* reaches *n*, we track the $\dot{\Lambda}^{(2\beta)}(x)$, that is identical for longest consecutive *l*, and record the last element *l** of the consecutive *l*'s. Then the corresponding $\Lambda^{(2\beta)}(x)$ and $\dot{\Lambda}^{(2\beta)}(x)$, at the *l**-th loop are marked as overall error locator polynomial $\Lambda^*(x)$ and error evaluator polynomial $\Omega^*(x)$ respectively. Finally Forney algorithm is used to calculate the error value in each error position with the miscorrection probability up to $(n\text{-}2f)(n\text{-}f)^{\beta}2^{m(\beta+f-2t)}$. The RiBC algorithm is targeted for correcting burst error plus some random errors.

III. Proposed Algorithm

In many situations [12, 13] the decoder can be supplied with probabilistic reliability information concerning the received symbols. A decoding algorithm that utilises such information is generally referred to as soft-decoding algorithm. In this section, we present an overview of the RS soft – decoding algorithm. List decoding algorithms and the algebraic soft-decoding algorithm are based on operations on bi-variate polynomials. Basic definitions related to bi-variate polynomials and list-decoding can be found in [11]. We outline the three main steps involved in the algebraic soft decoding algorithm.

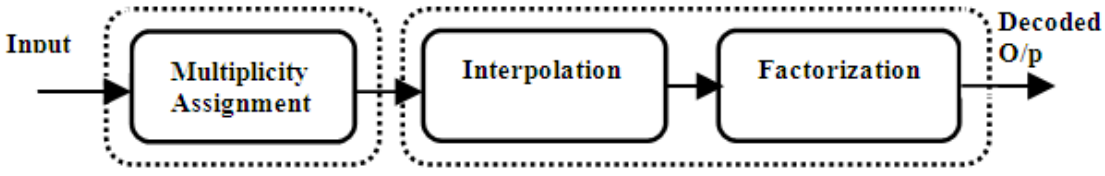

Fig 1. Block diagram of ASD Algorithm

The function of the multiplicity assignment step is to decide the multiplicity, $m_{i,j}$, of the interpolation point $(\alpha_j,\beta_{j,i})$ according to the reliability information from the channel. Different ASD algorithms have their distinct multiplicity assignment schemes. Nevertheless, they share the same interpolation and factorization steps.

A. *Multiplicity Assignment*

The error-correcting capability and complexity of ASD algorithms are mainly determined by the multiplicity assignment step. Various multiplicity assignment schemes have been developed [14,17]. Although the

multiplicity assignment schemes proposed in [15,16] lead to ASD algorithms with higher coding gain, these schemes involves very complex computations. For the purpose of practical implementation, ASD algorithms with simple multiplicity assignment schemes are highly desired. Moreover, the complexity of the interpolation and factorization steps grows fast with multiplicities. Hence multiplicity assignment schemes using small multiplicities are preferred.

For an [n, k] RS-code, given the data received from the channel, determine non-trivial interpolation points (xi,yi,j) and their corresponding multiplicities mi,j , where $0 \le i < n$, $0 \le j < si$, and si is the number of interpolation points associated with the ith code symbol position.

The KV decoding algorithm assigns the multiplicities based on symbol level reliability. As mentioned at the beginning of this chapter, for each code position j, the possible interpolation points include (αj,βj,i) for any βj,i. For practical applications, the simplified KV multiplicity assignment scheme is used. In this scheme, Pr(βj,i|rj), is computed for each βj,i. Then each mi,j can be calculated by

$$m_{i,j} = [\Pr(\beta_{j,i}|r_j)\lambda]$$

where λ is a pre-set real number and decides the maximum possible multiplicity, mmax. Since Pr(βj,i|rj)) is at most 1.0, mmax = [λ], and each non-trivial mi,j can range from 1 to mmax.

B. *Algorithm*

Input : reliability matrix II and a positive integer s, indicating the total number of interpolation points.

Output : Multiplicity matrix M.

Initialization step: Set II* := II and M := all-zero matrix.

Iteration step : Find the position (i,j) of the largest entry $\pi^*_{i,j}$ in II*, and set

$$\begin{aligned} \pi^*_{i,j} &:= \pi_{i,j} \\ m_{i,j} &:= m_{i,j} + 2 \\ m_{i,j} &:= m_{i,j} + 1 \\ s &:= s - 1 \end{aligned}$$

Control step : If s = 0, return M; otherwise go to the iteration step.

C. *Interpolation step*

Given the set of non-trivial interpolation points and their corresponding multiplicities, compute the non-trivial polynomial $Q_{min}(X, Y)$ of minimum $(1, k-1)$ -weighted degree that passes through each of the interpolation points $(x_i, y_{i,j})$ with corresponding multiplicity $m_{i,j}$. $Q_{min}(X, Y)$ is denoted the *minimal polynomial*.

The KV algorithm computes weights (or multiplicities) proportional to the soft information. These weights, stored in a multiplicity matrix, are used to control a biased interpolation in the GS algorithm. The (a,b)-weighted degree of a bivariate polynomial $P(x, y) \in GF(q)[x, y] \setminus \{0\}$ is $\deg^{(a,b)} P(x, y) = \max\{au + bv | p_{u,v} \neq 0\}$, where $a, b \in N$. If P(x, y) = 0, then $\deg^{(a,b)}(0) = -\infty$.

Suppose a nonnegative integer $m(\alpha)$ is assigned to each element $\alpha \in F$, and we are asked to construct a polynomial $f(x)$ of least degree that has a zero of multiplicity $m(\alpha)$, at $x = \alpha$, for all $\alpha \in F$. Clearly a minimum-degree solution to this one-dimensional interpolation problem is

$$f(x) = \prod_{\alpha \in F} (x - \alpha)^{m(\alpha)}$$

Given a required multiplicity m(α, β) for each (α, β) ∈ F2, construct a low-degree polynomial Q(x, y) that has zeros of the required multiplicity.

Let {m(α, β) : (α, β) ∈ F2} be a multiplicity function as above and let $\varphi 0 < \varphi 1 < \cdots$ be an arbitrary monomial order. Then there exists a nonzero polynomial Q(x, y) of the form

$$Q(x, y) = \sum_{i=0}^{C} a_i \, \phi_i(x, y)$$

where

$$C = \sum_{\alpha,\beta} \binom{m(\alpha, \beta) + 1}{2}$$

which has a zero of multiplicity m(α, β), at (x, y) = (α, β), for all (α, β) ∈ F2.

$Q(x, y)$ has a zero of multiplicity m at (α, β) if and only if $Q_{r,s}(\alpha, \beta) = 0$ for all (r, s) such that $0 \leq r + s < m(\alpha, \beta)$. There are $\binom{m(\alpha,\beta)+1}{2}$ choices for (r, s), each such choice imposes one homogeneous linear constraint on the coefficients a_i. In total there are C such linear constraints imposed on the $C+1$ coefficients $a_0, a_1, \cdots, a_C$. It follows that there must be at least one nonzero solution to this set of equations, which corresponds to a nonzero polynomial $Q(x, y)$ with the required multiplicities.

D. *Factorization step*

Determine all factors of the minimal polynomial of the form $Y - f(X)$ with $deg\ [f(X)] < k$. Each $f(X)$ corresponds to a valid data polynomial. The factorization problem can be solved by the iterative algorithm proposed by Roth and Ruckenstein [18]. Each iteration of this algorithm involves root computations over finite fields for a polynomial, whose degree can be as large as t. When the degree of the polynomial is higher than one, root computations over finite fields are traditionally carried out by exhaustive search. The prediction-based schemes in [19], [20] can be employed to circumvent the exhaustive search in most cases. If the prediction is correct, only one field multiplication and constant binary matrix multiplications are required to find the root. When the degree is at most four, the polynomial can be also converted to an affine format, in which the roots can be computed directly [21]. Particularly, when the degree of the polynomial is two, the conversion to the affine format and root computation are much less complicated.

IV. PROPOSED REED SOLOMON DECODER

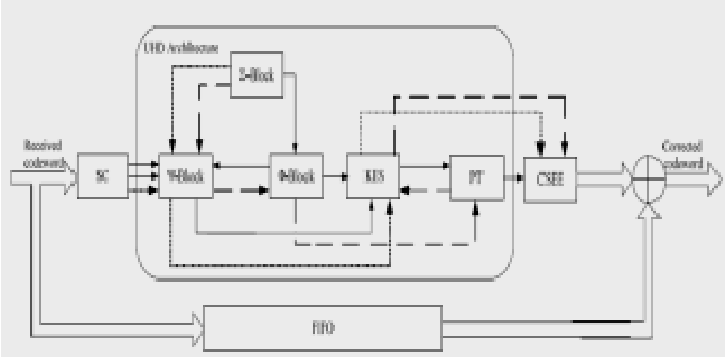

Fig 2. Overall UHD Architecture

The overall UHD architecture is shown in Fig 2. Here different blocks are used to process different steps in algorithm. Since excluding KES and PT blocks, other blocks are quite straight forward to be implemented; in this section we only introduce the architectures of KES and PT blocks.

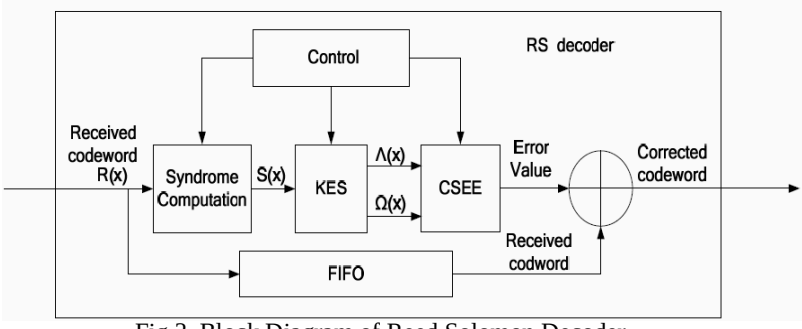

Fig 3. Block Diagram of Reed Solomon Decoder

A Reed–Solomon decoder consists of three blocks
- The syndrome computation (**SC**) block.
- The key-equation solver (**KES**) block.
- The Chien search and error evaluator (**CSEE**) block.

These blocks usually operate in pipelined mode in which the three blocks are separately and simultaneously working on three successive received words. The **SC** block computes the syndromes usually as the received word is entering the decoder. The syndromes are passed to the **KES** block to determine the error locator and error evaluator polynomials. These polynomials are then passed to the **CSEE** block, which calculates the error locations and error values and corrects the errors as the received word is being read out of the decoder. The throughput bottleneck in Reed–Solomon decoders is in the **KES** block in contrast, the **SC** and **CSEE** blocks are relatively straightforward to implement.

A. *Syndrome Calculation*

The very first step in the decoding process is to calculate the syndrome values which are used to check, whether the errors are occurred or not in the received polynomial. Due to the transmission channel, some symbols of the transmitted codeword can be corrupted and the decoder may receive the incorrect polynomial. So it's necessary to check the error occurrence. The Syndrome Transform (also called Syndrome Generation) block evaluates the received codeword of the generator polynomial. If the received data contains an error, the syndrome polynomial generated will be non-zero. If the received data has no error, the syndrome polynomial is zero, and the data is passed out of the decoder without any error correction. The syndrome calculation module receives codes from channel, denoted as *r(x)*, and calculates syndrome polynomial *S(x)*.

$$S(x) = \sum_{i=0}^{2t-1} S_i x^i$$

where *t* is the error correcting capability. Then, the coefficients of syndrome polynomial can be calculated by

$$S_i = r(\alpha^i) = \sum_{j=0}^{n-1} r_j \alpha^{ij}, \ i = 0, 1, \ldots, (2t-1)$$

where *n* is the code length and $\alpha^0, \alpha^1 \ldots \alpha^{2t-1}$ are the roots of the generation polynomial in encoding. The Eq.(2) can be represented in a recursive format as

$$S_i = r(\alpha^i) = (\ldots((r_{n-1}\alpha^i + r_{n-2})\alpha^i + r_{n-3})\ldots)\alpha^i$$

and each iteration only needs a finite-field multiplication and a finite-field addition. The processing unit of syndrome calculation module is shown below.

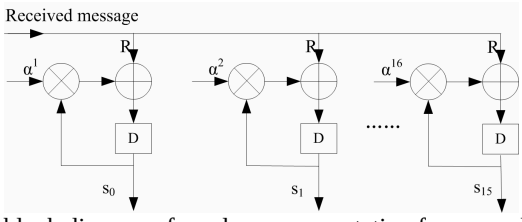

Fig 4. The block diagram of syndrome computation for example RS code

The syndrome is the result of a parity check performed on **r** to determine whether **r** is a valid member of the codeword set [3]. If in fact **r** is a member, the syndrome **S** has value **0**. Any nonzero value of **S** indicates the presence of errors. Similar to the binary case, the syndrome **S** is made up of *n* - *k* symbols, {*Si*} (i = 1, … , *n* - *k*). Thus, for this R-S code, there are four symbols in every syndrome vector; their values can be computed from the received polynomial, **r**(*X*). Note how the computation is facilitated by the structure of the code, and is rewritten below

From $\mathbf{U}(X) = \mathbf{m}(X)\,\mathbf{g}(X)$ this structure it can be seen that every valid codeword polynomial **U**(*X*) is a multiple of the generator polynomial **g**(*X*). Therefore, the roots of **g**(*X*) must also be the roots of **U**(*X*). Since **r**(*X*) = **U**(*X*) + **e**(*X*), then **r**(*X*) evaluated at each of the roots of **g**(*X*) should yield zero only when it is a valid codeword. Any errors will result in one or more of the computations yielding a nonzero result. The computation of a syndrome symbol can be described as follows:

$$S_i = \mathbf{r}(X)\big|_{X=\alpha^i} = \mathbf{r}(\alpha^i) \qquad i = 1, \cdots, n-k$$

Where, **r**(*X*) contains the postulated two-symbol errors. If **r**(*X*) were a valid codeword, it would cause each syndrome symbol *Si* to equal 0. For this example, the four syndrome symbols are found as follows

$$S_1 = \mathbf{r}(\alpha) = \alpha^0 + \alpha^3 + \alpha^6 + \alpha^3 + \alpha^{10} + \alpha^8 + \alpha^{11}$$
$$= \alpha^0 + \alpha^3 + \alpha^6 + \alpha^3 + \alpha^2 + \alpha^1 + \alpha^4$$
$$= \alpha^3$$
$$S_2 = \mathbf{r}(\alpha^2) = \alpha^0 + \alpha^4 + \alpha^8 + \alpha^6 + \alpha^{14} + \alpha^{13} + \alpha^{17}$$
$$= \alpha^0 + \alpha^4 + \alpha^1 + \alpha^6 + \alpha^0 + \alpha^6 + \alpha^3$$
$$= \alpha^5$$
$$S_3 = \mathbf{r}(\alpha^3) = \alpha^0 + \alpha^5 + \alpha^{10} + \alpha^9 + \alpha^{18} + \alpha^{18} + \alpha^{23}$$
$$= \alpha^0 + \alpha^5 + \alpha^3 + \alpha^2 + \alpha^4 + \alpha^4 + \alpha^2$$
$$= \alpha^6$$
$$S_4 = \mathbf{r}(\alpha^4) = \alpha^0 + \alpha^6 + \alpha^{12} + \alpha^{12} + \alpha^{22} + \alpha^{23} + \alpha^{29}$$
$$= \alpha^0 + \alpha^6 + \alpha^5 + \alpha^5 + \alpha^1 + \alpha^2 + \alpha^1$$
$$= 0$$

The results confirm that the received codeword contains an error (which we inserted), since $\mathbf{S} \neq \mathbf{0}$.

B. *KES*

This is the heart of the Reed-Solomon Decoder. This block generates the Error Locator polynomial $\Lambda(x)$ (also known as the "Key Equation" as it is the key to solve the decoding problem). After the Error Locator polynomial has been found, it is used to determine the Error Evaluator polynomial $\Omega(x)$.

With the help of inputted $\mathbf{S}(x)$, in this step, Key equation solver (KES) block will calculate error evaluator polynomial $\mathbf{\Omega}(x)$ and error locator polynomial $\mathbf{\Lambda}(x)$ by solving key equation: $\mathbf{\Lambda}(x)\mathbf{S}(x) \equiv \mathbf{\Omega}(x) \bmod x2t$. This part is the most important step in the whole RS decoding procedure, which usually dominates the performance of the overall decoder.

C. *ALGORITHM FOR KES BLOCK*

1. In the first ρ + 1 clock cycles, compute $\Phi(z)$ using the DC block only:

(a) Initialize all the DFF's in DC block with
$\mu_i^{(U)} = 0$ and $\mu_i^{(L)} = s_i$, for i = 0, 1,…, d-2, set
$\gamma^{(r)} = 1$and MC = 0.

(b) For $0 \le r \le \rho$, in each clock cycle, set $\delta^{(r)} = \Psi_r$

(c) After ρ + 1 clock cycles, the content of the upper DFF in each $PE1_i$ block, , is equal to Φ_i.

2. In the following d - ρ - 1 clock cycles, compute the $\Lambda(z)$ using ELU block and $\Omega^{\wedge}(z)$ using DC block.

(a) Re-initialize all the DFF's in DC and ELU blocks with $\mu_i^{(L)} = \mu_i^{(U)} = \Phi_i$ and $K_i^{(L)} = K_i^{(L)} = \Psi_i$

(b) For $0 \le r \le d - \rho - 1$, in each clock cycle, set $\delta^{(r)} = \delta^{\wedge}{}_0^{(r)}$ and generate $\gamma^{(r)}$ and $MC^{(r)}$

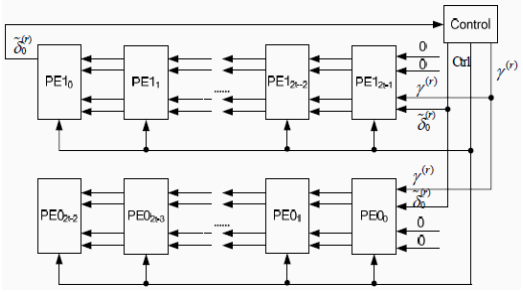

Fig 5. The overall architecture of KES block.

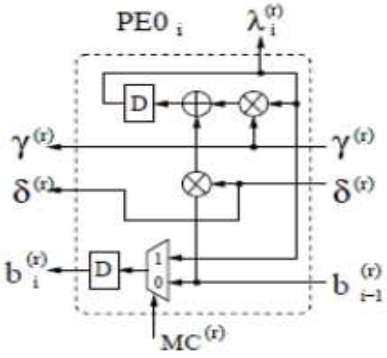

Fig 6. The block diagram of PE0*i*.

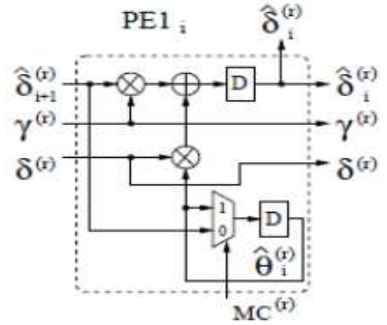

Fig 7. The block diagram of PE1*i*.

D. *Position Tracking Block*

PT block is used to track the longest consecutive polynomials that are identical. The input $\lambda_i^{\sim(2\beta)}$, $\lambda_i^{(2\beta)}$ and $\delta_i^{\sim(2\beta)}$ from KES block at the *l*-th outer iteration are denoted as $\lambda_i^{\sim}(l)$, $\lambda_i(l)$and $\delta_i^{\sim}(l)$. In addition, $\lambda_i^{\sim}$ (*temp*) represents $\lambda_i^{\sim}$ (*l* - 1) , while $\lambda_i^{\sim}$ (*store*) are the coefficients of current continuously identical $\Lambda^{\sim(2\beta)}(x)$. Moreover, $\lambda_i^{\sim}$ (*longest*) stores the coefficients of current *longest* continuously identical $\Lambda^{\sim(2\beta)}(x)$. Control signals *shift* and *equal* are generated from the signal generation schedule. After *l* reaches n, λ_i (*longest*) and $\delta_i^{\sim}$ (*longest*) are outputted as the coefficients of overall error locator polynomial $\Lambda^*(x)$ and overall error evaluator polynomial $\Omega^*(x)$.

E. *Chien Search and Error Correction*

After the error locator polynomial σi and error value polynomial ωi have been found, those values are passed to the Chien search and Forney algorithm blocks. To find the error location, Chien search is the best known method. Chien search is a widely employed approach to look for error position. Its basic idea is simple but efficient: If $\Lambda(\alpha^{-i})=0$ for current *i*, it indicates that the *i*-th symbol of the received codeword is wrong and needs to be corrected. After obtaining the position of error, the following Forney algorithm is applied to determine the error value

$$Y_i = -\frac{\Omega(x)}{x\Lambda'(x)}\bigg|_{x=\alpha^{-i}}$$

where *Yi* is the error magnitude for the *i*-th erroneous symbol.

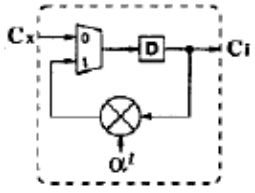

Fig 8. Basic Chien cell

The error locator polynomial is the polynomial, which lies in inverse position of the original error locations. The Chien search algorithm multiplies each coefficient of σi by primitive element α. Forney algorithm block works parallel with chien search block, to compute the error values. A Chien search cell is shown in Figure 8: and Chien search block is shown in Figure 9: Figure 10: represents the Forney algorithm block

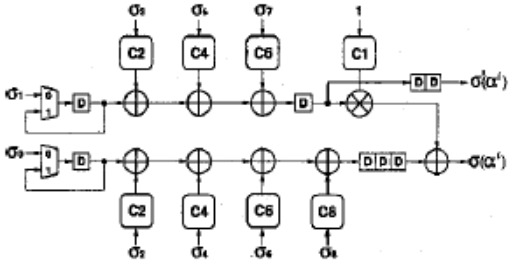

Fig 9. Chien Search Block

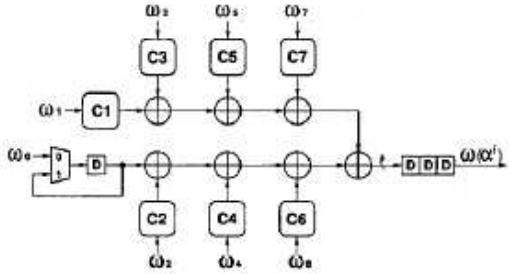

Fig 10. Forney Algorithm

This computed error values and error locations from Chien search and Forney blocks are given to error correction block which is shown in fig 11. The Zero detector circuit detects the value zero from Chien search block. If the value is zero, the decoder realize that error has occurred in this location and corrects that value, by adding the value corresponding value found by Forney block with the FIFO output. The FIFO memory is used for making desired delay to get corrected decoded sequence. In very rare case, if the number of error occurrence is exceeding the number of maximum error occurring probability, then decoder generates either the decoding fail signal or incorrect polynomial.

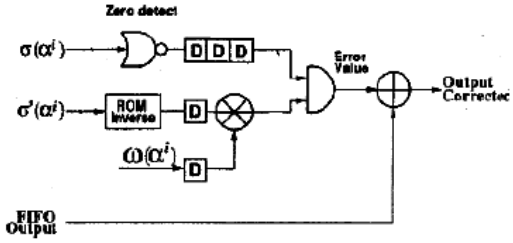

Fig 11. Error Correction block

V. SIMULATION RESULTS

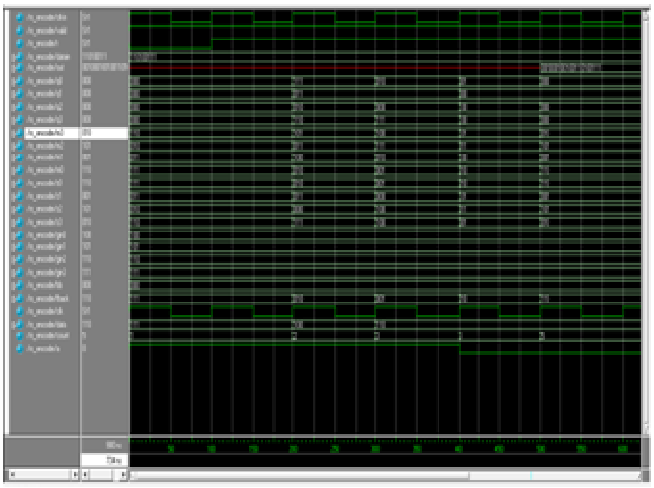

Fig 12. Simulation result of our proposed system

VI. CONCLUSION

Recent improvements in the decoding capabilities of Reed-Solomon codes using Algebraic Soft Decision Decoding makes it appealing to examine the decoding algorithms for Reed-Solomon product codes. In this paper, we have developed a Unified Hybrid Architecture of the area-efficient Reed Solomon decoder. It is found that the potential performance of ASD decoding of Reed Solomon codes is significantly better than the previous RiBC algorithm.

REFERENCES

[1] B. Sklar, "Digital Communication, Fundamental and Ap- plication," Prentice Hall, Upper Saddle River, 2001, p. 1104.
[2] S.B. Wicker and V. K. Bhargava, eds. "Reed-Solomon Codes and their Applications". New York: IEEE Press, 1994.
[3] E. Martinian, G.W. Wornell, and R. Zamir, Source Coding With Encoder Side Information,"IEEE Transactions on Information Theory, vol. submitted, 2004. ArXiV cs.IT/0412112.
[4] A. Dimakis, "Ubiquitous access to distributed data in large-scale sensor networks through decentralized erasure codes," in Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks, April 2005.
[5] Y. Wu. Novel burst error correcting algorithms for Reed-Solomon codes. proceeding of IEEE Allerton Conference on Communication, Control and Computing, Sep. 30 - Oct. 2 2009.
[6] E. Dawson and A. Khodkar. Burst error-correcting algorithm for Reed-Solomon codes. Electronics Letters 1995; 31(11) 848-849.
[7] L.Yin, J.Lu, K.B.Letaief and Y.Wu. Burst-error-correcting algorithm for Reed-Solomon codes. Electronics Letters 2001; 37(11) 695-697.
[8] R. E. Blahut, Theory and Practice of Error-Control Codes. Reading, MA: Addison-Wesley, 1983.
[9] H.-C. Chang and C. B. Shung, "New serial architectures for the Berlekamp–Massey algorithm," IEEE Trans. Commun., vol. 47, pp. 481–483, Apr. 1999.
[10] M. A. Hasan, V. K. Bhargava, and T. Le-Ngoc, Reed–Solomon Codes and Their Applications, S. B. Wicker and V. K. Bhargava, Eds. Piscataway, NJ: IEEE Press, 1994. Algorithms and architectures for a VLSI Reed–Solomon codec.
[11] R. Koetter and A. Vardy, "Algebraic soft-decision decoding of Reed-Solomon codes," Submitted to IEEE Trans. Inform. Theory, May 31, 2000.
[12] E.R.Berlekamp R.E. Peile and S.P.Pope, " The Application of Error control to Communications", IEEE commun. Mag., vol. 25, pp. 44-57, January 1987.
[13] A.Vardy and Y. Be'ery, "Bit-Level soft-decision decoding of reed-Solomon codes", IEEE trans. Commun., vol. 39, pp. 440-445 march 1991.
[14] R. Koetter and A. Vardy, Algebraic soft-decision decoding of Reed-Solomon codes," IEEE Transactions on

Information Theory, vol. 49,no. 11, pp. 2809{2825, Nov. 2003.
[15] F. Parvaresh and A. Vardy, Multiplicity assignments for algebraic softdecoding of Reed-Solomon codes," in Proc. of IEEE International Sym-posium on Information Theory, Yokohama, Japan, Jul. 2003, p. 205.
[16] N. Ratnakar and R. Koetter, Exponential error bounds for algebraic softdecision decoding of Reed-Solomon codes," IEEE Transactions on In-formation Theory, vol. 51, no. 11, pp. 3899{3917, Nov. 2005.
[17] S. Lee and B. Kumar, \Soft-decision decoding of Reed-Solomon codesusing successive error-and-erasure decoding," in Proc. of IEEE Global Telecommunications Conference, New Orleans, LA, Nov. 2008, pp. 1{5}.
[18] R. M. Roth and G. Ruckenstein, "Efficient decoding of Reed-Solomon codes beyond half the minimum distance,"IEEE Trans. on Info. Theory,vol. 46, no. 1, pp. 246-257, Jan. 2000.
[19] X. Zhang and K. K. Parhi, "Fast factorization in soft-decision Reed-Solomon decoding," IEEE Trans. on VLSI Systems, vol. 13, no. 4, pp. 413-426, Apr. 2005.
[20] X. Zhang, "Further exploring the strength of prediction in the factorization of soft-decision Reed-Solomon decoding," IEEE Trans. on VLSISystems, vol. 15, no. 7, pp. 811-820, Jul. 2007.
[21] J. Ma, A. Vardy and Z. Wang, "Low latency factorization architecture for algebraic soft-decision decoding of Reed-Solomon codes," IEEE Trans.on VLSI Systems, vol. 15, no. 11, pp. 1225-1238, Nov. 2007.
[22] Hanho Lee. "An Area-Efficient Euclidean Algorithm Block for Reed-Solomon Decoder," Proceedings of the IEEE Symposium on VLSI, 2003.
[23] Hanho Lee, Meng-Lin Yu, and Leilei Song, "VLSI Design of Reed-Solomon Decoder Architectures," IEEE International Symposium on Circuits and Systems, May 28-3 I , 2000, Geneva, Switzerland
[24] S. Lin and D. J. Costello, Error Control Coding (2nd edition). Upper Saddle River, NJ: Prentice-Hall, 2004.
[25] V. Guruswami and M. Sudan, \Improved Decoding of Reed-Solomon and Algebraic-Geometry Codes," IEEE Transactions on Information Theory, vol. 45, no. 6, pp. 1757{1767, 1999.
[26] H. Lee, "An Area-Efficient Euclidean Algorithm Block for Reed-Solomon Decoder", IEEE Annual Symposium on VLSI, 2003.
[27] H. Lee, M. Yu, and L. Song, "VLSI design of Reed–Solomon decoder architectures", IEEE International Symposium Circuits and System, Vol. 5, 2000.
[28] R. Roth and G. Ruckenstein, "Efficient Decoding of Reed–Solomon Codes beyond Half the Minimum Distance," IEEE Trans. Inform. Theory, vol. 46, no. 1, pp. 246–257, January 2000.
[29] R. Koetter, J. Ma, A. Vardy and A. Ahmed, "Efficient interpolation and factorization in algebraic soft decision decoding of Reed-Solomon codes," Proc. Of IEEE Symp. On Info. Theory, 2003.
[30] T. K. Troung, J. H. Jeng and I. S. Reed, "Fast algorithm for computing the roots of error locator polynomials upto degree 11 in Reed-Solomon decoders," IEEE Trans. on Comm., vol. 49, pp. 779 -783, May 2001.
[31] W. J. Gross, F. R. Kschsichang, R. Koetter and P. G. Gulak, "A VLSI architecture for interpolation in softdecision list decoding of Reed-Solomon codes," Proc. of IEEE Workshop on Signal Processing Systems, Oct. 2002.
[32] M. A. Hasan, and A. G. Wassal, VLSI Algorithms, Architectures, and Implementation of a Versatile GF(2m) Processor, IEEE Trans. on Computers, vol. 49, no. 10, Oct. 2000.
[33] Kavish Seth, Viswajith K N, S Srinivasan, and V Kamakoti, "Ultra Folded High-Speed Architectures for Reed-Solomon Decoders", in Proceedings of the 19th International Conference on VLSI Design (VLSID'06), 2006.
[34] S Lee, Hanho Lee, CS Choi, "Two parallel Reed-Solomon based FEC architecture for optical communications", in IEICE Electronics Express, vol.5, no.10, pp. 374-380, 2008.
[35] Hanho Lee, CS Choi, Jeong-In Park, Kihoon Lee, "High speed low complexity Reed Solomon decoder using pipelined Berlekamp Massey algorithm and its folded architecture", in Journal of semiconductor technology and science, vol.10, no.3, September,2010

