Implementation of RS Encoder and RS Decoder using UHD Architecture

Naresh.B
M.Tech Student, ECE
Vardhaman College of Engineering,
Hyderabad, INDIA

S.Srinivas
Assistant Professor (Sr.), ECE
Vardhaman College of Engineering,
Hyderabad, INDIA

ABSTRACT
Reed Solomon (RS) codes are a sort of non-binary cyclic codes. This code is widely used in wireless and mobile communication units. RS encoder along with RS decoder using UHD architecture is designed in this paper. In this brief, a novel low complexity reformulated inverse-free burst-error correction algorithm is developed. Then based on the Proposed RiBC algorithm, a Unified VLSI architecture is designed. It will be shown that, it can achieve high-speed, throughput and improved error correcting capability than Hard Decision Decoding (HDD) design with less area. A design of (7, 3) Reed Solomon encoder and Decoder are implemented using Verilog hardware description language (HDL) code, simulated and synthesized by XILINX ISE simulator.

General Terms
RiBC algorithm,RS codes,UHD.

Keywords
Burst error correction, Hard decision decoding, unified VLSI Architecture

1. INTRODUCTION
Reed-Solomon (RS) codes are concerned with the detection and correction of errors in symbols. RS codes are widely used for correcting the errors in storage and communication systems [1]. During transmission error may happen for a number of reasons e.g. (scratches on CD, radio frequency interference with mobile phone reception, noise etc.) At the receiving side, the decoder detects and corrects a limited predetermined number of errors occurred during transmission.

When we are dealing with the RS codes as forward error correction, the errors are generated in transmission procedure are divided into burst errors, random errors and erasures[2]. In this brief, a low-complexity high speed RS encoding and decoding architecture will improve the overall system performance significantly. In this a low complexity reformulated inverse free burst-error correcting (RiBC) algorithm is developed for practical applications. Then, based on the proposed reformulated inverse free burst error correcting algorithm, a unified VLSI architecture that is capable of correcting random errors, as well as burst errors and erasures, is firstly presented for multi-mode decoding requirements. It will be shown that, being the first RS Encoder and then decoder owning enhanced burst-error correcting capability, it can achieve significantly better burst error correcting capability than hard-decision decoding (HDD) design.

In this paper, area efficient RS encoder along with RS decoder using UHD architecture are implemented.Section2 discusses the Reed Solomon encoding procedure.Section3 introduces the proposed RiBC algorithm and Section4 covers the proposed unified hybrid decoding Architecture. Experimental results are showing the simulation results of proposed design.

2. REED SOLOMON ENCODING

2.1 Reed- Solomon codes
Reed-Solomon code is a block code and can be specified as RS (n, k) as shown in Fig1.The variable n is the size of the code word with the unit of symbols, k is the number of data symbols and 2t is the number of parity symbols. Each symbol consists of m number of bits. Reed Solomon codes works on Galois field. The Reed Solomon code allows correct up to t number of symbol errors.Where t is given by $t = (n-k)/2$.

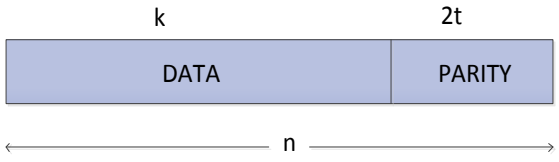

Fig 1 : The structure of RS Codeword

2.2 Reed-Solomon Encoder
Reed-Solomon codes operate on the information by dividing the message stream into blocks of data, adding redundancy per block, dependent only on the current inputs. The symbols in Reed-Solomon coding are elements of a Galois Field (finite field). Encoding procedure is done by dividing the message polynomial by generator polynomial, then it gives the Galois field remainder. The Galois field remainder is appended to the message[1]. This division is done by a Linear Feedback Shift Register (LFSR) implementation.Reed-Solomon Encoder using LFSR are shown in the Fig 2. The Linear Feedback Shift Register is the main computational element of the Reed-Solomon Encoder. The RS encoding procedure is based on Finite Field operations. The generating polynomial for an RS code takes the form

$$g(z) = g^0 + g^1 z + g^2 z^2 + ... + g^{2t-1} z^{2t-1} + g^{2t} z^{2t}$$

$$g(z) = 1 + z + z^3$$

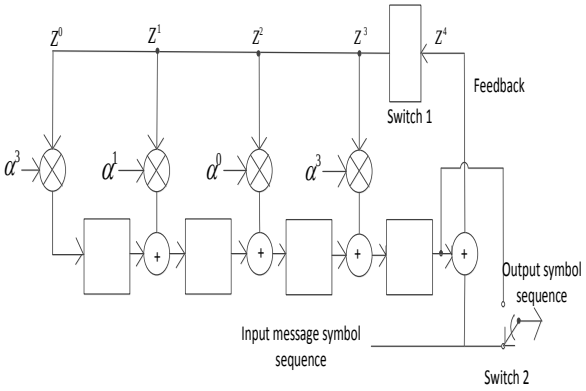

Fig2: Reed-Solomon Encoder Block Diagram (LFSR)

3. PROPOSED RIBC ALGORITHM

RS codes are very effective for correcting long burst errors. However, previous decoding algorithms in [5] and [6] are infeasible for hardware implementation due to their high computation complexity, long data path.

3.1 Proposed Reformulated Inverse-free Burst-Error Correcting (RiBC) Algorithm

In [12] the original burst error correcting algorithm consists of inversion operation and some computational steps contains long data dependency and long data path. To resolve these problems we reformulate BC algorithm to the proposed reformulated inverse free burst error correction algorithm.

The Reformulated inverse free burst error correction algorithm is a kind of list decoding algorithm [4]. In this proposed RiBC algorithm eight polynomials are updated simultaneously in each iteration. After every 2β inner iterations, $\overset{\sim(2\beta)}{\Lambda}(z)$, 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 $\overset{\sim(2\beta)}{\Lambda}(z)$, that is identical for longest consecutive l, and record the last element l^* of the consecutive l's. Then the corresponding $\Lambda^{(2\beta)}(z)$ and $\overset{\sim(2\beta)}{\Lambda}(z)$, at the l^*-th loop are marked as overall error locator polynomial $\Lambda^*(z)$ and error evaluator polynomial $\Omega^*(z)$ respectively

Algorithm for Reformulated Inverse - free Burst-Error Correction (RiBC):

f -length ($f<(2t-2\beta)$) burst of errors plus maximum β random errors):

Input: Syndromes $S_0, S_1, S_2, \ldots\ldots S_{2t-1}$;

A1: Compute

$$\Xi(z) = (1-\alpha^{1-(2t-2\beta)}z)(1-\alpha^{2-(2t-2\beta)}z).....(1-\alpha^{-1}z)(1-z)$$

$$= 1+\xi_1 z+\xi_2 z^2+....+\xi_{2t-2\beta}z^{2t-2\beta}$$

A2: For l=0 step 1 until n-1 do

A2.1: Compute

$$\Phi(z) = \Xi(\alpha^l z) = 1+\Phi_1 z+\Phi_2 z^2+...+\Phi_{2t-2\beta}z^{2t-2\beta};$$

A2.2: Compute

$$\Psi(z) = \Psi_0+\Psi_1 z+.....+\Psi_{2t-1}z^{2t-1},$$

$$where \Psi_i = \sum_{j=0}^{2t-2\beta}\phi_j S_{i-2t-2\beta-j};$$

A2.3: Initialize

$$\Lambda^{(0)}(z) = \lambda_0^{(0)}+\lambda_1^{(0)}z+....+\lambda_{2t-2}^{(0)}z^{2t-2} = \Phi(z);$$

$$B^{(0)}(z) = b_0^{(0)}+b_1^{(0)}z+....+b_{2t-2}^{(0)}z^{2t-2} = \Phi(z);$$

$$\overset{\sim(0)}{\Lambda}(z) = \overset{\sim(0)}{\lambda_0}+\overset{\sim(0)}{\lambda_1}z+.....+\overset{\sim(0)}{\lambda_{2t-2}}z^{2t-2} = 1;$$

$$\overset{\sim(0)}{B}(z) = \overset{\sim(0)}{b_0}+\overset{\sim(0)}{b_1}z+.....+\overset{\sim(0)}{b_{2t-2}}z^{2t-2} = 1;$$

$$\overset{\sim(0)}{\Delta}(z) = \overset{\sim(0)}{\delta_0}+\overset{\sim(0)}{\delta_1}z+.....+\overset{\sim(0)}{\delta_{2t-1}}z^{2t-1} = \psi(z);$$

$$\overset{\sim(0)}{\Theta}(z) = \overset{\sim(0)}{\theta_0}+\overset{\sim(0)}{\theta_1}z+.....+\overset{\sim(0)}{\theta_{2t-1}}z^{2t-1} = \psi(z);$$

$$\overset{\sim*(0)}{\Delta}(z) = \overset{\sim*(0)}{\delta_0}+\overset{\sim*(0)}{\delta_1}z+.....+\overset{\sim*(0)}{\delta_{2t-1}}z^{2t-1} = S(z);$$

$$\overset{\sim*(0)}{\Theta}(z) = \overset{\sim*(0)}{\theta_0}+\overset{\sim*(0)}{\theta_1}z+.....+\overset{\sim*(0)}{\theta_{2t-1}}z^{2t-1} = S(z);$$

$$\gamma^{(0)} = 1, k^{(0)} = 0;$$

A2.4: For $r = 0$ Step1 Until $2\beta-1$ do

A2.4.1: Compute

$$\overset{\sim(r+1)}{\delta_i} = \delta^r \overset{\sim(r)}{\delta_{i+1}} - \overset{\sim(r)}{\delta_0}\overset{\sim(r)}{\theta_i};$$

$$\overset{\sim*(r+1)}{\delta_i} = \delta^r \overset{\sim*(r)}{\delta_{i+1}} - \overset{\sim(r)}{\delta_0}\overset{\sim*(r)}{\theta_i};$$

$$\lambda_i^{r+1} = \gamma^{(r)}\lambda_i^{(r)} - \overset{\sim(r)}{\delta_0} b_{i-1}^{(r)};$$

$$\overset{\sim(r+1)}{\lambda_i} = \gamma^{(r)}\overset{\sim(r)}{\lambda_i} - \overset{\sim(r)}{\delta_0}\overset{\sim(r)}{bi-1};$$

A2.4.2: If $\overset{\sim(r)}{\delta_0} \neq 0$ and $k^{(r)} \geq 0$ then $a = 1$; else $a = 0$;

A2.4.3:

$$\begin{pmatrix} b_i^{(r+1)} \\ \overset{\sim(r+1)}{b_i} \\ \overset{\sim(r+1)}{\theta_i} \\ \overset{\sim*(r+1)}{\theta_i} \\ \gamma^{(r+1)} \\ k^{(r+1)} \end{pmatrix} = \begin{pmatrix} \lambda_i^{(r)} & b_{i-1}^{(r)} \\ \overset{\sim(r)}{\lambda_i} & \overset{\sim(r)}{b_{i-1}} \\ \overset{\sim(r)}{\delta_{i+1}} & \overset{\sim(r)}{\theta_i} \\ \overset{\sim*(r)}{\delta_{i+1}} & \overset{\sim*(r)}{\theta_i} \\ \overset{\sim(r)}{\delta_0} & \gamma^{(r)} \\ -k^{(r)}-1 & k^{(r)}+1 \end{pmatrix} \begin{pmatrix} a \\ \bar{a} \end{pmatrix}$$

A3: Track the longest consecutive $\tilde{\Lambda}^{(2\beta)}(z)$ that are identical, recorded the last element l^* of the consecutive $l's$ then the overall error locator polynomial $\Lambda^*(z) = \Lambda^{(2\beta)}(z)$ at the l^*-th outer loop.

The overall evaluator polynomial $\Omega^*(z)$ is corresponding $\tilde{\Delta}^{(2\beta)}(z)$ at the l^*-th outer loop.

Output $\Lambda^*(z), \Omega^*(z)$

The proposed Reformulated Inverse-free burst error correction algorithm is targeted for correcting burst (local) errors plus some random (global) errors. If the channel condition guarantees that only single long burst of errors occurred. Wu [7] presented a low complexity single long burst of errors correcting (sLBC) for that case. The single long burst of errors correcting algorithm is a special version of Reformulated inverse-free burst error correction algorithm.

3.2 Algorithm for single long Burst of Errors Correction (sLBC)

Input: Syndromes $S_{0,}S_1, S_2, \ldots\ldots S_{2t-1}$;

B1: Compute

$$\Xi(z) = (1-\alpha^{1-(2t-1)}\, z)(1-\alpha^{2-(2t-1)}\, z)\ldots.(1-z)$$
$$= 1+\xi_1 z+\xi_2 z^2+\ldots.+\xi_{2t-1}z^{2t-1}$$

B2: Compute

$$\Gamma(z) = s_{2t-1}+s_{2t-2}\xi_1 z+s_{2t-3}\xi_2 z^2\ldots.+s_0\xi_{2t-1}z^{2t-1}$$

B3: B3.1: Find roots of $\Gamma(z)$;

B3.2: Track the longest consecutive root sequence

$\alpha^{s+f-1}, \alpha^{s+f}, \ldots.. \alpha^{s+2t-2}$;

B3.3: Use t to calculate the value of s, f,

Then it can claim that a burst of errors starts at position α^{-s} with length f;

B4: Compute new error locator polynomial

$$\Lambda^*(z) = \Xi(\alpha^{s+2t-2}\, z)$$

B5: Compute new error value polynomial

$$\Omega^*(z) = \omega_0+\omega_1 z+\ldots..+\omega_{2t-1}z^{2t-1}$$

Where $\omega_i = \Lambda_0^* s_i+\Lambda_1^* s_{i-1}+\ldots..+\Lambda_i^* s_0$

for $i = 0,1,2\ldots\ldots.2t-1$;

Output $\Lambda^*(z), \Omega^*(z)$

3.3 Algorithm for Random Error and Erasure Correction (rEEC)

Input: Syndromes $S_{0,}S_1, S_2, \ldots\ldots S_{2t-1}$;

Erasure locations: $z_0, z_{1,}\ldots.., z_{\rho-1}; \rho \le 2t-2$

C1: Compute

$$\Xi(z) = (1-z_0 z)(1-z_1\, z)\ldots.(1-z_{\rho-1}\, z)$$
$$= 1+\xi_1 z+\xi_2 z^2+\ldots..+\xi_\rho z^\rho$$

C2: Compute

$$\Psi(z) = \Psi_0+\Psi_1 z+\ldots.+\Psi_{2t-1}z^{2t-1}$$

where $\Psi_i = \sum_{j=0}^{\rho}\xi_j s_{i+\rho-j}$;

C3: Initialization:

$$\Lambda^{(0)}(z) = B^{(0)}(z) = \Xi(z)$$
$$\Delta^{(0)}(z) = \delta_0^{(0)}+\delta_1^{(0)}z+\ldots.+\delta_{2t-1}^0 z^{2t-1} = \Psi(z)$$
$$\Theta^{(0)}(z) = \theta_0^{(0)}+\theta_1^0+\ldots.+\theta_{2t-1}^0 z^{2t-1} = \Psi(z)$$
$$k^{(0)} = 0, \lambda^{(0)} = 1$$

C4: For r=0 step 1 until $2t-\rho-1$ do

C4.1: Compute

$$\Delta^{(r+1)}(z) = x^{-1}\gamma^{(r)}\Delta^{(r)}(z)-\delta_0^{(r)}\Theta^{(r)}(z);$$
$$\Lambda^{(r+1)}(z) = \gamma^{(r)}\Lambda^{(r)}(z)-x\,\delta_0^{(r)}(z)-x\,\delta_0^{(r)}B^r(z)$$

C4.2: If $\delta_0^{(r)} \neq 0$ and $k^{(r)} \ge 0$ then a=1; else a=0;

$$\begin{bmatrix} B^{(r+1)}(z) \\ \Theta^{(r+1)}(z) \\ \gamma^{(r+1)} \\ k^{(r+1)} \end{bmatrix} = \begin{pmatrix} \Lambda^{(r)} & xB^{(r)}(z) \\ x^{-1}\Delta^{(r)}(z) & \Theta^{(r)}(z) \\ \delta_0^{(r)} & \gamma^{(r)} \\ -k^{(r)}-1 & k^{(r)}+1 \end{pmatrix} \begin{bmatrix} a \\ \bar{a} \end{bmatrix}$$

Output:, $\Omega^*(z) = \Delta^{(2t-\rho)}(z)$

4. UHD ARCHITECTURE

The above three algorithms are share many similar computation steps. Based on this interesting similarity, a UHD architecture is designed. Figure 3 shows the overall architecture of Unified Hybrid Decoding decoder. Three types of lines illustrate data flows for different work modes: solid line for mode-1 for burst combined with random error correction RiBC algorithm, dashed line for mode-2 for only burst error correction and big dashed line for mode-3 for only random error correction. Different blocks are used to process different steps in algorithm.

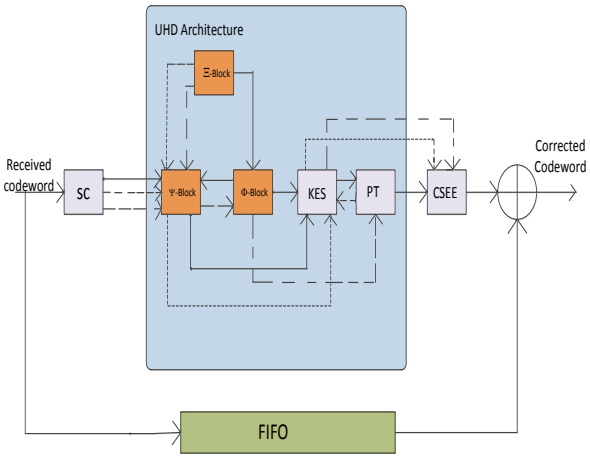

Fig 3:Overall architecture of UHD decoder

A Reed–Solomon decoder consists of mainly four blocks:

- The syndrome computation (**SC**) block.
- The key Equation solver (**KES**) block.
- Position tracking (**PT**) block.
- The Chien search and error evaluator (**CSEE**) block

4.1 Syndrome Calculation

The first step in decoding process is the received symbol is to determine the data syndrome.The syndrome calculation block is used to check whether the received polynomial contains errors or not. The syndrome generation block evaluates the received codeword polynomial. If the received codeword consists an error then the syndrome polynomial is non zero.If received codeword contains an error, it indicates that the received syndrome polynomial is zero,and the data is passed through the decoder without any error correction.The syndromes can then be calculated by substituting the 2t roots of the generator polynomial G(z) into R(z). The syndrome polynomial is generally represented as, Where α is the primitive element.

$$s(z) = s_0 + s_1 z + \cdots \ldots\ldots\ldots\ldots\ldots\ldots + s_{2t-1} z^{2t-1}$$

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

4.1.1 *ΞBlock :*

Ξ- block is used to carryout steps A1, B1, or C1 in different work modes. No matter which work mode is selected, the computation of Ξ(z) is always carried out as follows:

$$\Xi(z) = \Pi_{i=1}^{\rho}(1 - A_i z) = 1 + \xi_1(z) + \xi_2 z^2 + \ldots\ldots\ldots\ldots + \xi_\rho z^\rho$$

Where A_i denotes $\alpha^{i-(2t-2\beta)}, \alpha^{i-(2t-1)}$ orZ_{i-1}, and ρ denotes 2t-2 β, 2t-1.

4.1.2 *Φ Block:*

Φ block used to process Steps A2.1 and B3.1. For these steps, the common operation is multiply accumulate for each coefficient of the polynomial. Only a slight difference exists in step B3.1 it is a Chien Search-like step, hence an extra adder tree is required to verify the validity of current received symbol. For each l in step A2.1 since

$$\Phi[z] = \Xi(\alpha^l z) = 1 + \phi_1 z + \ldots\ldots + \phi_{2t-\beta} z^{2t-2\beta}$$

should be maintained within 2t+3 cycles

4.1.3 *ΨBlock:*

Ψ-block is used to process steps A2.2, B2 and C2. Actually, the inherent nature of steps A2.2, B2 and C2 is the multiplication of two polynomials.

4.2 Key Equation Solver Block Architecture

This is the heart of the Reed-Solomon Decoder. This KES block generates the Error Locator polynomial $\Lambda(z)$. After the Error Locator polynomial has been found, it is used to determine the Error Evaluator polynomial $\Omega(z)$.

In this step with the help of **S**(*z*), KES block will calculate error locator polynomial **Λ**(**z**) and error evaluator polynomial $\Omega(z)$ by solving key equation: **Λ**(*z*)**S**(*z*)= $\Omega(z)$ mod z^{2t}.

Key equation solver block determines the overall system performance significantly.The overall decoding process is done by using this key equation solver only.

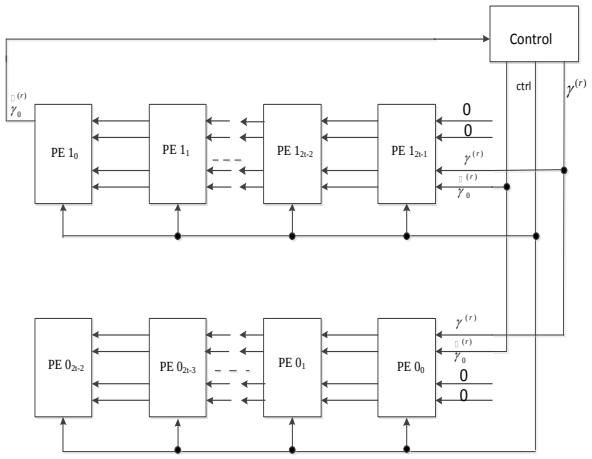

Fig 4:Overall architecture of Key Equation Solver block

In UHD decoder KES block is carry out steps A2.4, B4, B5 and C4. Fig 4 represents the overall architecture of KES block and internal structures of its two types of processing elements (PE): PE0 and PE1 are shown in the Fig 5 and Fig 6. As shown in fig 4 the KES block consist of 2t-1PE0's and 2t PE1's in the rth iteration. Each register in PE0$_i$ / PE1$_i$ stores the corresponding coefficients of different polynomials(Fig5, Fig6).

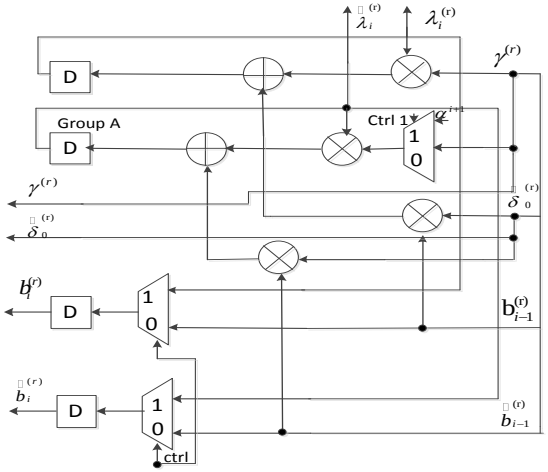

Fig 5: Block diagram of $PE0_i$

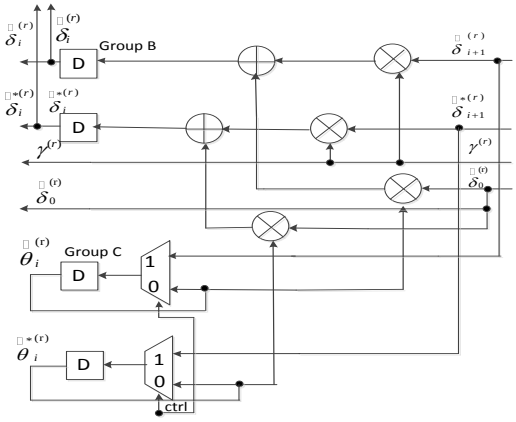

Fig 6: Block diagram of $PE1_i$

4.3 Position Track (PT) Block

PT block is used to track the longest consecutive polynomials that are identical or positions of roots that is steps B3.2 and B3.3.

The inputted $\lambda_i^{\sim(2\beta)}, \lambda_i^{(2\beta)}, \delta_i^{\sim(2\beta)}$ from KES block at the l^{th} outer iteration are denoted as $\tilde{\lambda}_i(l)$, $\lambda_i(l)$, and $\tilde{\delta}_i(l)$. In addition, $\lambda_i(l)$ (*temp*) represents $\tilde{\lambda}_i(l-1)$, while $\tilde{\lambda}_i(l)$ (*store*) are the coefficients of current continuously identical $\Lambda^{\sim(2\beta)}(z)$. Moreover, $\lambda_i^{\sim}$ (longest) stores the coefficients of current longest continuously identical $\Lambda^{\sim(2\beta)}(z)$. Control signals shift and equalare generated from signal generation Schedule. After *l* reaches n, λ_i (longest) and $\tilde{\delta}_i$ (longest) are outputted as the coefficients of overall error locator polynomial $\Lambda^*(z)$ and overall error evaluator polynomial $\Omega^*(z)$.

4.4 Chien Search and Error Correction

In this approach error locator polynomial and error value polynomial values are passed to the Chien search and Forney algorithm blocks. To find the error location, Chien search is the very good efficient method. Chien search method is used for finding error positions. Fig7 shows the basic Chien search block. Error correction block is shown in the Fig8.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 error positions, the following Forney algorithm is applied to determine the error value.

$$\Upsilon_i = \frac{\Omega(X_i^{-1})}{\Lambda'(X_i^{-1})}$$

where Y_i indicates error magnitude for the i^{th} erroneous symbol.

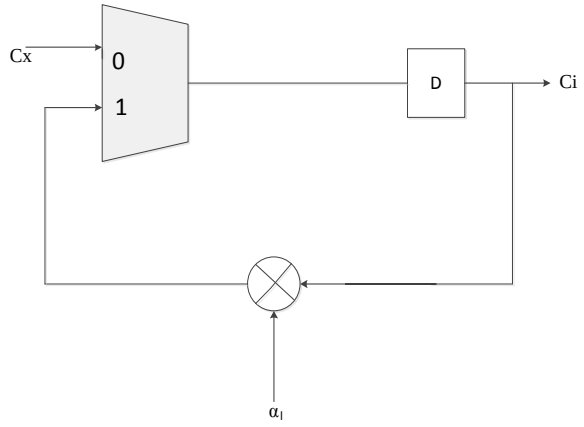

Fig 7: Basic Chien search cell

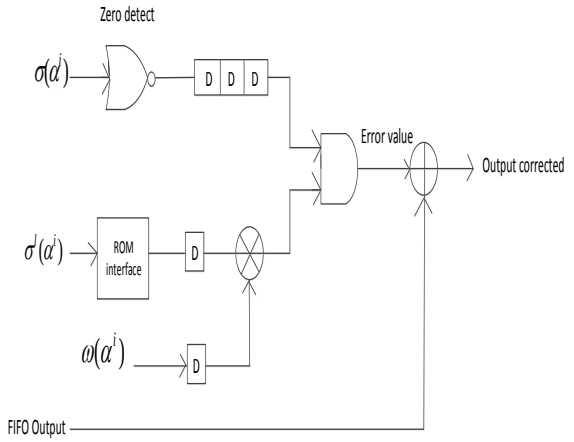

Fig 8: Error Correction block

The block diagram of proposed architecture is shown in the Fig9. The encoded output being given to the decoder along with introduction of noise. The decoder is fed with all the information need for decoding after a certain latency it will check whether all the code words are valid code words, if so it will output or else it will correct them, if the number of corrupted code words are less than the maximum error correcting capability of the RS Decoder.

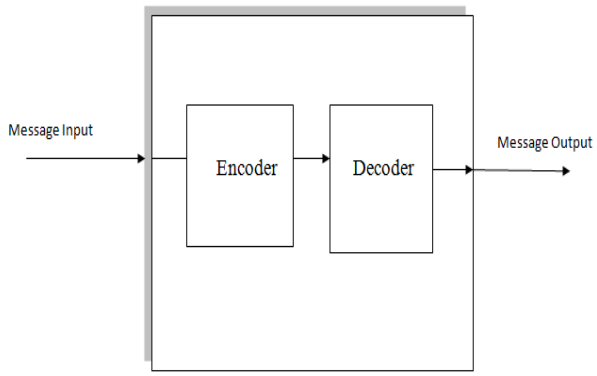

Fig 9: Block Diagram of Proposed Architecture

5. EXPERIMENTAL RESULTS

The Simulation results of proposed architecture are carried out using Xilinx ISE simulator.

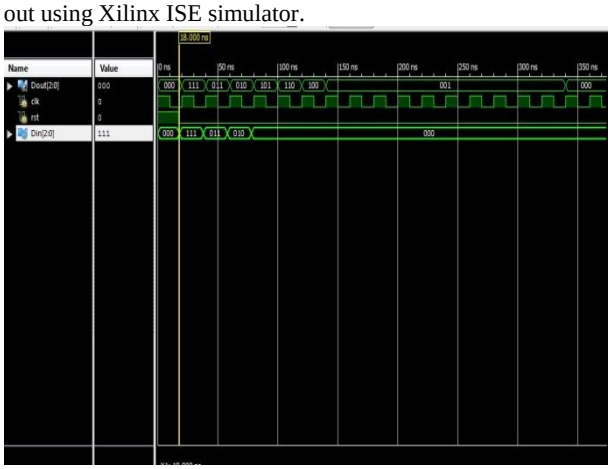

Fig10: Simulation result of encoder

Fig11: Simulation result for Syndrome

Fig12: Simulation result for decoder

Fig13: Simulation result for Encoder _Decoder

Fig14: Simulation result for Encoder_ Decoder

Fig 15: Final output for proposed Architecture

Message will be passing from encoder to decoder.The decoder checks the resulted data and gives the error free data. Fig15 shows the output of proposed architecture.

6. CONCLUSION

In this paper, a Unified Hybrid Architecture of the area-efficient Reed Solomon encoder along with Reed Solomon decoder using UHD architecture is developed. The architecture that can support three decoding modes is presented for the first time. The proposed architecture is used to capable of correct burst (local) errors and random (global) errors at a time. It will provide applications like Bar code scanning, high speed optical communications, and Data storage devices. Compare to traditional RS decoder the proposed architecture can achieve efficient burst error correcting capability.

7. REFERENCES

[1] B.Sklar, " Digital Communication, Fundamental and Application" 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] Amina, P. Chio, I.A. Sahagun and D. J. Sabido IX "VLSI Implementation of A (255,223) Reed- Solomon Error- Correction Codec," Roc. Of Second National ECE Conference.

[4] D.V. Sarwate and N.Rshanbhag, "High-speed Architectures for Reed-Solomon Decoder," IEEE transaction on VLSI system, OCT, 2001.

[5] E. Dawson and A. Khodkar, "Burst error-correcting algorithm for Reed-Solomon codes," Electron. Lott, vol. 31, pp. 848–849, 1995.

[6] L.Yin, J.Lu, K.B.Letaief and Y.Wu Burst-error-correcting algorithm for Reed-Solomon codes. Electronics Letters.,vol. 37, no. 11, pp.695-697, may 2001.

[7] Y. Wu, "Novel burst error correcting algorithms for Reed-Solomon codes," in Proc. IEEE Allerton Conf. Commun., Control, Comput.2009, pp. 1047–1052.

[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] K. Bhargava, Eds. Piscataway, NJ: IEEE Press, 1994. Algorithms and architectures for a VLSI Reed– Solomon codec.

[11] T. Zhang and K. K. Parhi, "On the high-speed VLSI implementation of errors-and-erasures correcting Reed-Solomon decoders," in *Proc. ACMGreat Lake SympVLSI (GLVLSI)*, 2002.

[12] Li Li, Bo Yuan "Unified Architecture for Reed-Solomon decoder combined with burst error correction", IEEE transaction on VLSI systems,JULY 2012.