Md. Taj, P Srinivas, S. Nagaraju / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 3, Issue 4, Jul-Aug 2013, pp.1607-1612

Design and Implementation of FPGA System to Reduce Reed-Solomon Errors

Md. Taj[1], P Srinivas[2], S. Nagaraju[3]

[1]Dept.of ECE Nimra College of Engineering & Technology, Jupudi, Vijayawada-India.
[2]Assistant Professor Dept.of ECE Nimra College of Engineering & Technology, Vijayawada, India.
[3]Assistant Professor Dept.of ECE VLITS Vadlamdui Guntur, India.

ABSTRACT

The data reliability has become an important issue in most communication and storage systems for high speed operation and mass data process. Various error correction code are provided for improving data reliability. A Reed-Solomon code is quite suitable for burst errors, but in case of random errors, it has some difficulty. For MLC NAND flash memories, Bose- Chaudhuri-Hocquenghem (BCH) codes are frequently used. BCH codes provide flexible code length and variable range of error correcting capability. However, NAND flash memory systems process with the large size of data such as a page or a block unit. Hence, BCH codes may not be appropriate for a NAND flash controller.

We propose product Reed- Solomon (RS) code for non-volatile NAND flash memory systems. Reed-Solomon codes are the most diversely used in data storage systems, but powerful for burst errors only. In order to correct multiple random errors and burst errors, another efficient decoding algorithm is required. The product code composing of column-wise Reed-Solomon codes and row-wise Reed-Solomon codes may allow decoding multiple errors beyond their error correction capability. The proposed code consists of two shortened Reed-Solomon codes and a conventional Reed- Solomon code. We implement the proposed coding scheme on a FPGA-based simulator with using an FPGA device. The proposed code can correct 16 symbol errors.

KEY WORDS: Bose-Chaudhuri-Hocquenghem (BCH), Reed-Solomon codes, Berlekamp-Massey Algorithm, flipped, Euclid's Algorithm, mass data process, decoding, FPGA-based simulator, NAND flash controller, and Altera.

I. INTRODUCTION

The data reliability has become an important issue in most communication and storage systems for high speed operation and mass data process. Various error correction code are provided for improving data reliability. A Reed-Solomon code is quite suitable for burst errors, but in case of random errors, it has some difficulty. For MLC NAND flash memories, Bose-Chaudhuri-Hocquenghem (BCH) codes are frequently used. BCH codes provide flexible code length and variable range of error correcting capability. However, NAND flash memory systems process with the large size of data such as a page or a block unit. Hence, BCH codes may not be appropriate for a NAND flash controller.

The Reed- Solomon (RS) code for non-volatile NAND flash memory systems. Reed-Solomon codes are the most diversely used in data storage systems, but powerful for burst errors only. In order to correct multiple random errors and burst errors, another efficient decoding algorithm is required. The product code composing of column-wise Reed-Solomon codes and row-wise Reed-Solomon codes may allow to decoding multiple errors beyond their error correction capability. The proposed code consists of two shortened Reed-Solomon codes and a conventional Reed- Solomon code. We implement the proposed coding scheme on a FPGA-based simulator with using an FPGA device. The proposed code can correct 16 symbol errors.

Fig.1 Concept of forward error correction

II. Reed Solomon Encoder

The encoder is the easy bit. Since the code is systematic, the whole of the block can be read into the encoder, and then output the other side without alteration. Once the kith data symbol has been read in, the parity symbol calculation is finished, and the parity symbols can be output to give the full n symbols. Gross simplification coming up. The idea of the parity words is to create a long polynomial (n coefficients long it contains the message and the parity) which can be divided exactly by the RS generator polynomial. That way, at the decoder the received message block can be divided by the RS generator polynomial. If the remainder of the division is zero, then no errors are detected.

If there is a remainder, then there are errors. Dividing a polynomial by another is not conceptually easy, but if you follow the maths in some of the references its not too hard to understand. The encoder acts to divide the polynomial represented by the k message symbols d(x) by the RS generator polynomial g(x). This generator polynomial is not the same as the Galois Field generator polynomial, but is derived from it

$$x(n-k).d(x)/g(x) = q(x) + r(x)/g(x) \quad (1)$$

The term x(n-k) is a constant power of x, which is simply a shift upwards n-k places of all the

polynomial coefficients in d(x). It happens as part of the shifting process in the architecture below. The remainder after the division r(x) becomes the parity. By concatenating the parity symbols on to the end of the k message symbols, an n coefficient polynomial is created which is exactly divisible by g(x).

The encoder is a 2t tap shift register, where each register is m bits wide. The multiplier coefficients g0 to g (2t-1) are coefficients of the RS generator polynomial. The coefficients are fixed, which can be used to simplify the multipliers if required. The only hard bit is working out the coefficients, and for hardware implementations the values can often be hard coded.

At the beginning of a block all the registers are set to zero. From then on, at each clock cycle the symbol in each register is added to the product of the feedback symbol and the fixed coefficient for that tap, and passed on to the next register. The symbol in the last register becomes the feedback value on the next cycle. When all n input symbols have been read in, the parity symbols are sitting in the register, and it just remains to shift them out one by one.

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

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

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

$$\mathbf{g}(X) = g_0 + g_1 X + g_2 X^2 + \ldots + g_{2t-1} X^{2t-1} + X^{2t} \quad (3)$$

The degree of the generator polynomial is equal to the number of parity symbols. R-S codes are a subset of the Bose, Chaudhuri, and Hocquenghem (BCH) codes; hence, it should be no surprise that this relationship between the degree of the generator polynomial and the number of parity symbols holds, just as for BCH codes.

Since the generator polynomial is of degree 2*t*, there must be precisely 2*t* successive powers of α that are roots of the polynomial. We designate the roots of **g**(X) as $\alpha, \alpha^2, \ldots, \alpha^{2t}$. It is not necessary to start with the root α; starting with any power of α is possible. Consider as an example the (7, 3) double-symbol-error correcting R-S code. We describe the generator polynomial in terms of its 2t = n - k = 4 roots, as follows

$$g(X) = (X - \alpha)(X - \alpha^2)(X - \alpha^3)(X - \alpha^4)$$
$$= (X^2 - (\alpha + \alpha^2)X + \alpha^3)(X^2 - (\alpha^3 + \alpha^4)X + \alpha^7)$$
$$= (X^2 - \alpha^4 X + \alpha^3)(X^2 - \alpha^6 X + \alpha^0)$$
$$= X^4 - (\alpha^4 + \alpha^6)X^3 + (\alpha^3 + \alpha^{10} + \alpha^0)X^2 - (\alpha^4 + \alpha^9)X + \alpha^3$$
$$= X^4 - \alpha^3 X^3 + \alpha^0 X^2 - \alpha^1 X + \alpha^3$$

Following the low order to high order format, and changing negative signs to positive, since in the binary field +1 = –1, **g**(*X*) can be expressed as follows:

$$\mathbf{g}(X) = \alpha^3 + \alpha^1 X + \alpha^0 X^2 + \alpha 3 X^3 + X^4 \quad (4)$$

III. Encoding in Systematic Form

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

$$X^{n-k} m(X) = q(X) g(X) + p(X) \quad (5)$$

where q(*X*) and p(*X*) are quotient and remainder polynomials, respectively. As in the binary case, the remainder is the parity. Equation (23) can also be expressed as follows

$$p(X) = X^{n-k} m(X) \text{ modulo } g(X) \quad (6)$$

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

$$U(X) = p(X) + X^{n-k} m(X) \quad (7)$$

We demonstrate the steps implied by Equations (5) and (6) by encoding the following three-symbol message:

$$\underbrace{010}_{\alpha^1} \quad \underbrace{110}_{\alpha^3} \quad \underbrace{111}_{\alpha^5}$$

with the (7, 3) R-S code whose generator polynomial is given in Equation (5). We first multiply (upshift) the message polynomial $\alpha^1 + \alpha^3 X + \alpha^5 X^2$ by X n - k = X^4, yielding $\alpha^1 X^4 + \alpha^3 X^5 + \alpha^5 X^6$. We next divide this upshifted message polynomial by the generator polynomial in Equation (5), $\alpha^3 + \alpha^1 X + \alpha^0 X^2 + \alpha^3 X^3 + X^4$. Polynomial division with nonbinary coefficients is more tedious than its binary counterpart, because the required operations of addition (subtraction) and multiplication (division). It is left as an exercise for the reader to verify that this polynomial division results in the following remainder (parity) polynomial.

$$p(X) = \alpha^0 + \alpha^2 X + \alpha^4 X^2 + \alpha^6 X^3 \quad (8)$$

Then, from Equation (7), the codeword polynomial can be written as follows

$$U(X) = \alpha^0 + \alpha^2 X + \alpha^4 X^2 + \alpha^6 X^3 + \alpha^1 X^4 + \alpha^3 X^5 + \alpha^5 X^6 \quad (9)$$

Systematic Encoding with an (n - k)–Stage Shift Register Using circuitry to encode a three-symbol sequence in systematic form with the (7, 3) R-S code described by g(X) in Equation (4) requires the implementation of a linear feedback shift register (LFSR) circuit, as shown in Figure 6. It can easily be verified that the multiplier terms in Figure 6, taken from left to right, correspond to the coefficients of the polynomial in Equation (4) (low order to high order). This encoding process is the nonbinary equivalent of cyclic encoding. Here,the (7, 3) R-S nonzero codewords are made up of 2m - 1 = 7 symbols, and each symbol is made up of m = 3 bits.

IV. Reed Solomon Decoder

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

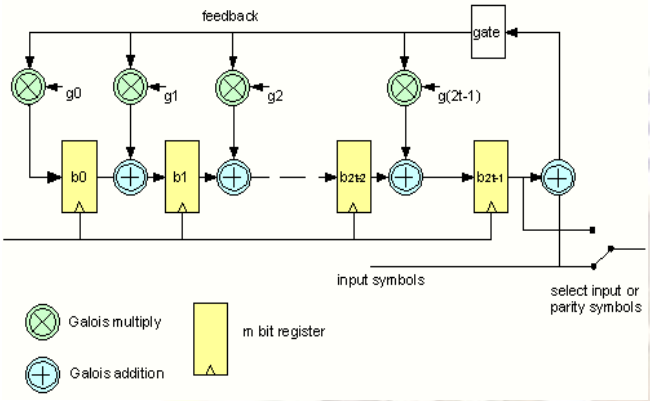

Fig2. Encoder with LFSR

$e(X) = \sum e_n X^n$, (10) Where n = 0, 1, 2, 3, 4, 5, 6

For this example, let the double-symbol error be such that

$e(X) = 0 + 0X + 0X^2 + \alpha^2X^3 + \alpha^5X^4 + 0X^5 + 0X^6$

$= (0\ 0\ 0)+(0\ 0\ 0)X+(0\ 0\ 0)X^2+(0\ 0\ 1)X^3+(1\ 1\ 1)X^4+(0\ 0\ 0)X^5+(0\ 0\ 0)X^6$ (11)

In other words, one parity symbol has been corrupted with a 1-bit error (seen as α^2), and one data symbol has been corrupted with a 3-bit error (seen as α^5). The received corrupted-codeword polynomial, r(X), is then represented by the sum of the transmitted-codeword polynomial and the error-pattern polynomial as follows

$r(X) = U(X) + e(X)$ (12)

Following Equation (12), we add U(X) from Equation (11) to e(X) and r(X), as follows

$r(X) = (100) + (001)X + (011)X^2 + (100)X^3 + (101)X^4 + (110)X^5 + (111)X^6$

$\alpha^0\alpha^2X\alpha^4X^2\alpha^0X^3\alpha^6X^4\alpha^3X^5\alpha^5X^6$ (13)

In this example, there are four unknowns—two error locations and two error values. Notice an important difference between the non binary decoding of r(X) that we are faced with in Equation (13) and binary decoding; in binary decoding, the decoder only needs to find the error locations. Knowledge that there is an error at a particular location dictates that the bit must be "flipped" from 1 to 0 or vice versa. But here, the nonbinary symbols require that we not only learn the error locations, but also determine the correct symbol values at those locations. Since there are four unknowns in this example, four equations are required for their solution.

Decoding is a far harder task than encoding. Typically about ten times more resources (be it logic, memory or processor cycles) are required to decode and correct the corrupted data The decode operation takes several stages. There are plenty of sources available for software implementations of the various algorithms required. Hardware implementations (FPGA or ASIC) are a little harder to come by, especially those with parameterised specifications. Texas Instruments give their software decoder away for free, whilst by comparison FPGA manufacturers such as Xilinx or Altera can charge many thousands of dollars for their hardware implementation.

V. Berlekamp-Massey

The second step is to find the error polynomial lambda. This requires solving 2t simultaneous equations, one for each syndrome. The 2t syndromes form a simultaneous equation with t unknowns. The unknowns are the locations of the errors. In general there are many possible solutions to the set of equations, but we assume that the one with the least number of errors is the correct one.

This assumption is the reason that more than t errors can actually cause the decoder to corrupt the received signal further (if allowed to). If more than t errors occur, then there will exist a possible solution to the equations with less than t errors. Unfortunately this solution is unlikely to correct the right symbols. The process of solving the simultaneous equations is usually split into two stages. First, an error location polynomial is found.

This Polynomial has roots which give the error locations. Then the roots of the error polynomial are found. There are several methods of finding the error polynomial lambda, the two most popular are Euclid's Algorithm (easier to implement) and the Berlekamp-Massey Algorithm (more efficient use of hardware resources). The algorithm iteratively solves the error locator polynomial by solving one equation after another and updating the error locator polynomial. If it turns out that it cannot solve the equation at some step, then it computes the error and weights it, increases the size of the error polynomial, and does another iteration.

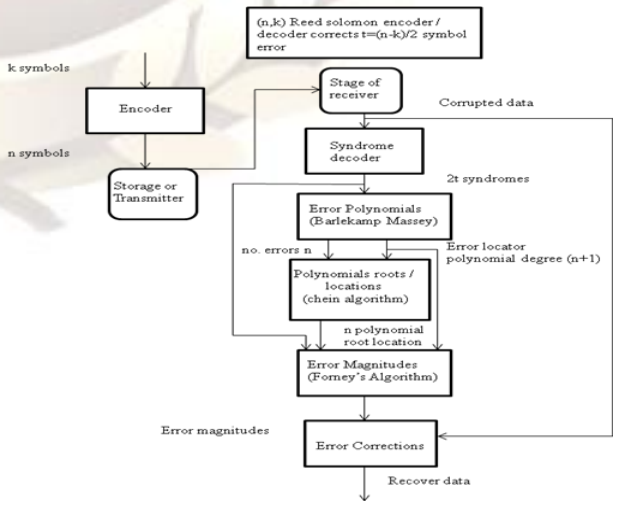

Fig3.Flow chart for Reed Solomon Decoder

A maximum of 2t iterations are required. For n symbol errors, the algorithm gives a polynomial with n coefficients. At this point the decoder fails if there are more than t errors, and no corrections can be made. Doing so might actually introduce more errors than there were originally

Suppose there are v errors in the codeword at location X j1 , X j2 , ... , X j. Then, the error polynomial e(X) shown in Equations (10) and (11) can be written as follows

$$e(X) = e_{j1}X^{j1} + e_{j2}X^{j2} + \ldots + e_{jv}X^{jv} \quad (14)$$

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

$$S_1 = r(\alpha) = e_{j1}\beta_1 + e_{j2}\beta_2 + \ldots + e_{jv}\beta_v$$
$$S_2 = r(\alpha^2) = e_{j1}\beta_1^{\,2} + e_{j2}\beta_2^{\,2} + \ldots + e_{jv}\beta_v^{\,2}$$
$$S_{2t} = r(\alpha^{2t}) = e_{j1}\beta_1^{\,2t} + e_{j2}\beta_2^{\,2t} + \ldots + e_{jv}\beta_v^{\,2t} \quad (15)$$

There are 2t unknowns (t error values and t locations), and 2t simultaneous equations. However, these 2t simultaneous equations cannot be solved in the usual way because they are nonlinear (as some of the unknowns have exponents). Next, it is necessary to learn the location of the error or errors. An error-locator polynomial, σ(X), can be defined as follows

$$\sigma(X) = (1 + \beta_1 X) (1 + \beta_2 X) \ldots (1 + \beta_v X)$$
$$= 1 + \sigma_1 X + \sigma_2 X^2 + \ldots + \sigma_v X^v \quad (16)$$

The roots of σ(X) are 1/β1, 1/β2, … ,1/βv. The reciprocal of the roots of σ(X) are the error-location numbers of the error pattern e(X). Then, using autoregressive modeling techniques, we form a matrix from the syndromes, where the first t syndromes are used to predict the next syndrome. That is,

$$\begin{bmatrix} S_1 & S_2 & S_3 & \ldots & S_{t-1} & S_t \\ S_2 & S_3 & S_1 & \ldots & S_t & S_{t+1} \\ & & & & \cdot & \\ & & & & \cdot & \\ S_{t-1} & S_t & S_{t+1} & \ldots & S_{2t-3} & S_{2t-2} \\ S_t & S_{t+1} & S_{t+2} & \ldots & S_{2t-2} & S_{2t-1} \end{bmatrix} \begin{bmatrix} \sigma_t \\ \sigma_{t-1} \\ \cdot \\ \cdot \\ \sigma_2 \\ \sigma_1 \end{bmatrix} = \begin{bmatrix} -S_{t+1} \\ -S_{t+2} \\ \cdot \\ \cdot \\ -S_{2t-1} \\ -S_{2t} \end{bmatrix} \quad (17)$$

We apply the autoregressive model of Equation (17) by using the largest dimensioned matrix that has a nonzero determinant. For the (7, 3) double-symbol error correcting R-S code, the matrix size is 2 × 2, and the model is written as follows

$$\begin{bmatrix} S_1 & S_2 \\ S_2 & S_3 \end{bmatrix} \begin{bmatrix} \sigma_2 \\ \sigma_1 \end{bmatrix} = \begin{bmatrix} S_3 \\ S_4 \end{bmatrix}$$
$$\begin{bmatrix} \alpha^3 & \alpha^5 \\ \alpha^5 & \alpha^6 \end{bmatrix} \begin{bmatrix} \sigma_2 \\ \sigma_1 \end{bmatrix} = \begin{bmatrix} \alpha^6 \\ 0 \end{bmatrix} \quad (18)$$

To solve for the coefficients σ1 and σ2 and of the error-locator polynomial, σ(*X*), we first take the inverse of the matrix in Equation (18). The inverse of a matrix [*A*] is found as follows

$$Inv\,[A] = \frac{Cofactor[A]}{\det\,[A]}$$

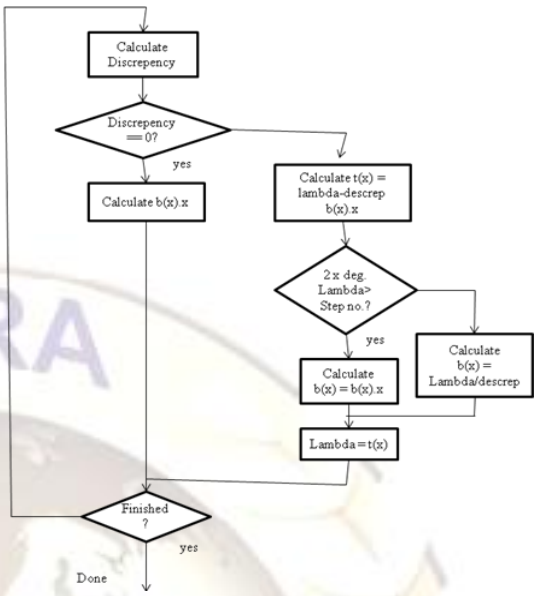

Fig4. Berlekamp-Massey Algorithm

Therefore

$$Det \begin{bmatrix} \alpha^3 & \alpha^5 \\ \alpha^5 & \alpha^6 \end{bmatrix} = \alpha^3\alpha^6 - \alpha^5\alpha^5 = \alpha^9 + \alpha^{10} = \alpha^2 + \alpha^3 = \alpha^5$$

Cofactor $\begin{bmatrix} \alpha^3 & \alpha^5 \\ \alpha^5 & \alpha^6 \end{bmatrix} = \begin{bmatrix} \alpha^6 & \alpha^5 \\ \alpha^5 & \alpha^3 \end{bmatrix}$ (19)

$$Inv \begin{bmatrix} \alpha^3 & \alpha^5 \\ \alpha^5 & \alpha^6 \end{bmatrix} = \frac{\begin{bmatrix} \alpha^6 & \alpha^5 \\ \alpha^5 & \alpha^3 \end{bmatrix}}{\alpha^5} = \alpha^{-5} \begin{bmatrix} \alpha^6 & \alpha^5 \\ \alpha^5 & \alpha^3 \end{bmatrix}$$
$$\alpha^2 \begin{bmatrix} \alpha^6 & \alpha^5 \\ \alpha^5 & \alpha^3 \end{bmatrix} = \begin{bmatrix} \alpha^8 & \alpha^7 \\ \alpha^7 & \alpha^5 \end{bmatrix} = \begin{bmatrix} \alpha^1 & \alpha^0 \\ \alpha^0 & \alpha^5 \end{bmatrix} \quad (20)$$

VI. Design Summery Report

Number of errors: 0
Number of warnings: 257

1.	Number of Slice Registers	4,737 out of 28,800 16%
2.	Number of Slice LUTs	6,302 out of 28,800 21%
3.	Number used as logic	6,261 out of 28,800 21%
4.	Number of route-thrus	53 out of 57,600 1%

Table 5.2 Logic Utilization

1.	Number of occupied Slices	2,502 out of 7,200 34%
2.	Number with an unused Flip Flop	2,949 out of 7,686 38%
3.	Number with an unused LUT	1,384 out of 7,686 18%
4.	Number of fully used LUT-FF pairs	3,353 out of 7,686 43%

Table5.3 Logic Distribution

VII. 6. RESULTS

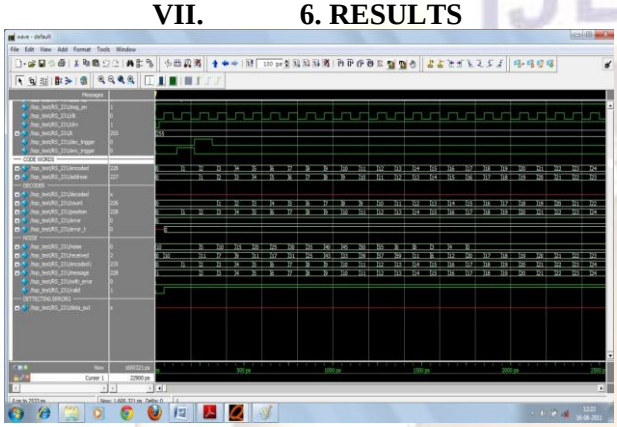

Fig5. Generation Of code words

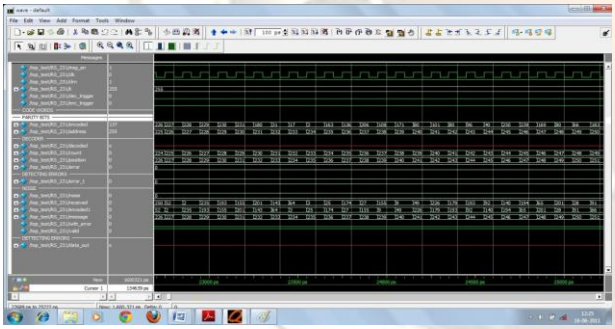

Fig6. Generation Of Parity Bits

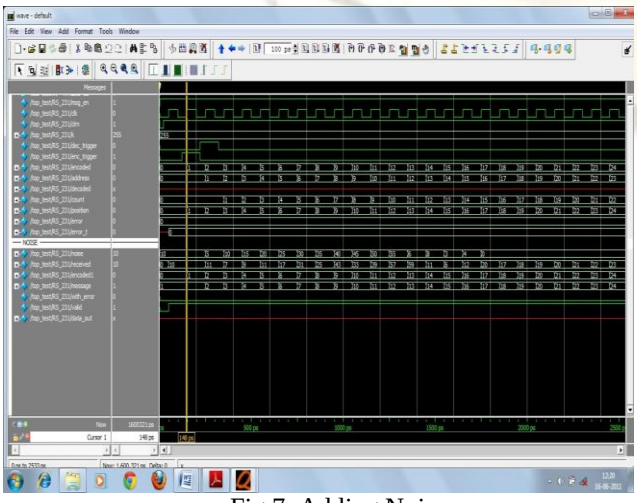

Fig 7. Adding Noise

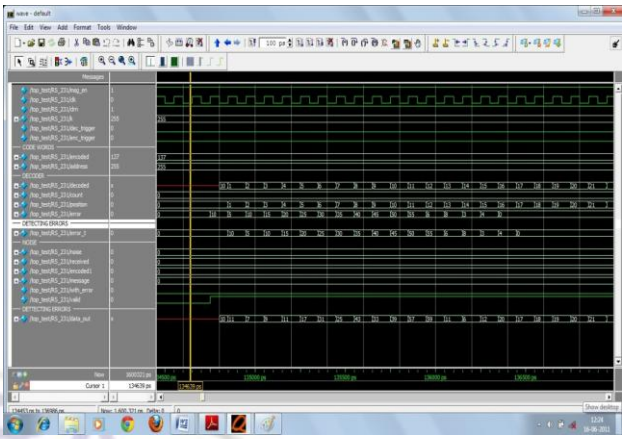

Fig8. Detection of Error Values

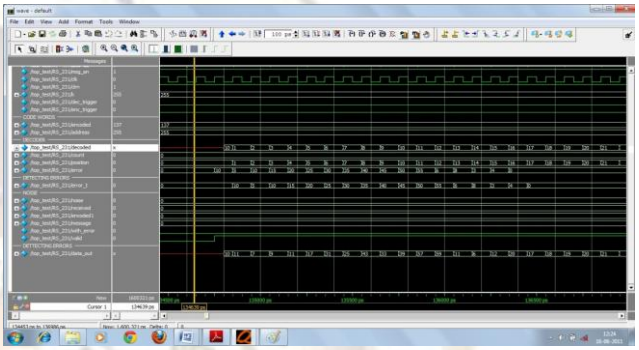

Fig8.Decoded Symbols

VIII. Conclusion

This paper proposes a product Reed-Solomon code for multiple random errors and burst errors. The proposed code takes two dimensional array data consisted of two shortened Reed-Solomon codes in a column-wise and a conventional Reed-Solomon code in a row-wise. The proposed code becomes powerful against multiple random errors and burst errors. The proposed code can corrects 16 symbol errors. The code has the coding gain of 1.8 dB and the bandwidth of 1.07 Gbps when operated at 290 MHz with the power consumption of 26.4 mW.
Future Scope of the project is to provide effective code for detecting 20 error symbols in 255 code words. Code can implement with low area and low power

References

[1] Polynomial Codes Over Certain Finite Fields by Reed, I. S. and Solomon, G. SIAM Journal of Applied Math.
[2] Information Theory and Reliable Communication by Gallager, R. G.
[3] Digital Communications: Fundamentals and Applications, Second Edition by Sklar, B.
[4] The Application of Error Control to Communications by Berlekamp, E. R., Peile, R. E., and Pope, S. P.
[5] Forward Error Correction Coding for Fading Compensation in Mobile Satellite Channels by Hagenauer, J., and Lutz, E.

[6] Theory and Practice of Error Control Codes by Blahut, R. E.
[7] Reed-Solomon Codes and Their Applications by Wicker, S. B. and Bhargava, V. K.
[8] ASIC Implementation of Reed-Solomon Error Correction Circuits for Low Area Overhead on Memory System by S. P. Kang, C. G. Kim, S. W. Rhee, and Y. Jee.
[9] X., Yang, "Industrial Data Communication and Control Networks", Beijing: TUP, 2003.6[9] B. Zeidman, "Designing with FPGAs & CPLDs", CMP Books, 2002
[10] C. E. Cummings, "Simulation and Synthesis Techniques for Asynchronous FIFO Design with Asynchronous Pointer Comparisons", SNUG San Jose 2002