International Journal of Embedded and Real-Time Communication Systems
Volume 12 • Issue 1 • January-March 2021

Design and Implementation of RS(450, 406) Decoder:

Forward Error Correction by Reed Solomon Decoding

Akhilesh Yadav, National Institute of Technology, Kurukshetra, India
https://orcid.org/0000-0001-9135-7328
Poonam Jindal, National Institute of Technology, Kurukshetra, India
Devaraju Basappa, NXP Semiconductors, India

ABSTRACT

Nowadays, in the field of data transmission between receiver and transmitter, the Reed Solomon code is used very frequently. FEC codes have two foremost and influential operations: (1) calculating parity symbols at the encoder side and (2) transmitting message symbols with parity symbols and decoding the received codeword at the second side by using the decoding algorithms. Gigabit automotive ethernet is used in the automotive car to provide better bandwidth for every kind of applications to connect functional components of the vehicles. This error correction technique is used in the gigabit automotive ethernet to reduce the channel noise during data transmission. RS (450, 406) is a powerful error correction techniques used in automotive ethernet. This paper focused only on the analysis of Reed Solomon decoding. Reed Solomon decoding is more efficient decoding techniques for correcting both burst and random errors. The critical steps of the Reed Solomon decoding are to solve the error evaluator and error calculator polynomial, which is also known as KES solver.

KEYWORDS

Cadence ncSim, Cadence Simvision, Chien-Search, Error Corrector, Forney Algorithm, Forward Error Correction, Galois Field (GF), Gigabit Ethernet, Inversionless Berlekamp Massey (iBM), Reed Solomon

1. INTRODUCTION

With the development of advanced vehicle technology, electronic systems are increasing in vehicle to refine their interpretation and new features. Considering the features of a car, its electronics system is divided into many functional elements, and every element has self-dependent control. Various complex controls and sensors are used in cars to maximize their efficiency and power. To conserve vehicles in normal operation, components in different domain or same domain need to communicate properly to each other. Therefore, to complete this communication inside vehicles different vehicle networks technology has been developed, like as Flex Ray, MOST (Media Oriented System Transport),

DOI: 10.4018/IJERTCS.20210101.oa2

LIN (Local Interconnect Network), LVDS (Low-Voltage Differential Signaling) and Controller Area Network (CAN) etc. These networks are developed for specific applications or domains. Initially CAN BUS is used in vehicle network but due to some limitations (like restriction on cable length, bit rate and module synchronization etc.), Automotive Ethernet (AE) is replacing CAN network technology. Automotive Ethernet (AE) is used for providing connection in between electronic systems.

AE is designed to meet bandwidth requirements, synchronization requirements, latency requirements and network management requirements. It has wide range of applications including: Diagnostics, Infotainment, Advance Driver Assistance Systems (ADAS) and in vehicle connectivity. In Ethernet data is transferred in the form of packets between nodes, it provides bidirectional communication. AE is a wired hierarchical homogeneous network. Gigabit or 1000BASE-T1 Ethernet is a next generation Automotive Ethernet, can serve as a backbone of the Autonomous car. The Automotive Ethernet (Sana Ullah et al., 2013, pp. 1-12) is used in cars to connect the different electronic systems for providing better and fast communication between them. AE is the physical network used to connect different electronic components used in the vehicles by a wired network. It provides better bandwidth, latency and management requirements.

The Physical Coding Sublayer (PCS) service interface allows the 1000BASE-T1 PCS to transfer information to and from a PCS client. In PCS transmission code is used for improving the transmission characteristic of any type of information to be transferred. In PCS Forward Error Correction (FEC) technique is used for error detection and correction. FEC is a powerful transmission code, it correct limited number of error without the need of retransmission. Several Error Correction Codes (ECC) available for FEC are:

1. Block codes
2. Convolutional codes
3. Hamming Codes
4. Binary Convolution code
5. Low – Density Parity check code
6. Cyclic Code
7. BCH (Bose Chaudhari and Hocquenghen) code
8. Reed Solomon Code

Except Reed Solomon (RS) Code, other error correction codes are not used in Physical layer (or PCS) due to their limitations such as: less error correction capability, less data rate and poor bandwidth. Moreover, these all EC codes are independent of Galois Field (GF) and primitive polynomial except RS code. They can correct error up to several bits. Hamming codes can detect two-bit error, or they can fix only one-bit error without detection of uncorrected errors. The most significant difference between BCH code and Reed Solomon are:

- BCH codes correct bits, while Reed Solomon code corrects symbols.
- BCH codes correct t bit error errors, while RS code corrects t symbols.

BCH codes can correct only random error, while RS code can correct both random and burst error during data transmission. Hence, due to the error correction capability RS codes are preferred over other BCH codes.

In physical layer of AE, RS encoder and decoder are used. They work simultaneously to provide full duplex communication. RS code is a powerful FEC code. RS encoding and decoding is used in the Gigabit Ethernet for better bandwidth and for reducing channel noise during the data transmission. RS code (E. R. Berlekamp, 1984; R. E. Blahut, 1983) is one of the most popular FEC (M. Kaur & V. Sharma, 2010) code. It adds the parity symbol in the message symbol and makes the receiver

Table 1. Different version of Reed Solomon Codes

Reed Solomon Codes	Primitive Polynomial	Galois Field (GF)	Size of Single Message	Error Correction Capability (Symbol)
RS(7, 3)	$1+x^1+x^3$	GF$\left(2^3\right)$	$4-$ bit symbol	2
RS(32, 28)	$1+x^2+x^5$	GF$\left(2^5\right)$	$5-$ bit symbol	2
RS(64, 56)	$1+x^1+x^6$	GF$\left(2^6\right)$	$6-$ bit symbol	4
RS(128, 112)	$1+x^3+x^7$	GF$\left(2^7\right)$	$7-$ bit symbol	8
RS(255, 243)	$1+x^2+x^3+x^4+x^8$	GF$\left(2^8\right)$	$8-$ bit symbol	6
RS(255, 239)	$1+x^2+x^3+x^4+x^8$	GF$\left(2^8\right)$	$8-$ bit symbol	8
RS(255, 251)	$1+x^2+x^3+x^4+x^8$	GF$\left(2^8\right)$	$8-$ bit symbol	2
RS(450, 406)	$1+x^4+x^9$	GF$\left(2^9\right)$	$9-$ bit symbol	22

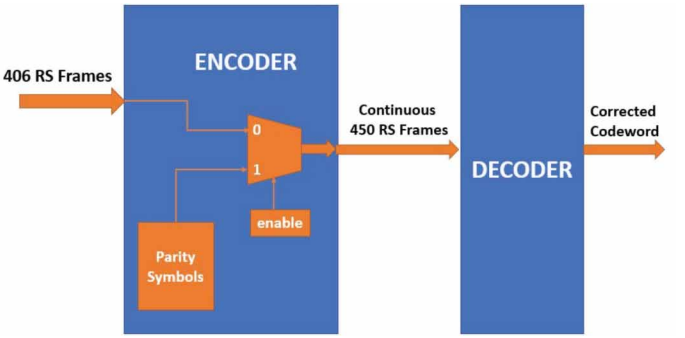
capable to identify the error and to correct the limited number of errors present in a received codeword. RS code is the important subclass of the BCH code. This RS encoding and decoding is required at PCS of Gigabit Ethernet. Every RS code is based on finite field also known as Galois Field (GF). A GF field is conventionally defined as finite field GF $\left(q\right)$ (D. Gorenstein & N. Zierler, 1961; Jimmy K. Omura & James L. Massey, 1986) where $q-1=n$ is the maximum length $\left(n=k+2t\right)$ of the codeword. And GF addition, subtraction, multiplication, divisions are correctly defined, where k is the length of message symbol, and $2t$ is the length of parity symbols, It is able to correct the errors up to $t=\dfrac{\left(n-k\right)}{2}$.

The several version of Reed Solomon Codes are: RS (7, 3) (Z. Sana & R. Gupta, 2019), RS(32, 28) (C. Peng et al., 2015), RS(64, 56) (J. Sha et al., 2009), RS(128, 112), RS(255, 243) (P. Dayal & R. Kumar, 2013), RS(255, 239) (Y. Lin et al., 2014; F. Garcia-Herrero et al., 2011), RS(255, 251) (A. S. Das et al., 2013) etc., And comparison has been done in terms of primitive polynomial, Galois Field, size of single message and number of error correction capability, as given in Table 1. Some literature review of RS code and its decoding algorithm is given in related work section.

In this paper, we proposed RS (450, 406) decoder based on Finite Field of GF $\left(2^9\right)$. This RS code are used in 1000BASE-T1 AE because it compact channel noise during data transmission. RS(450, 406) code has more error correction capability as compare to other RS codes, it can correct up to 22 error symbols.

We have designed RS(450, 406) decoder for 1000BASE-T1 PHY, which is one of the Gigabit Ethernet family with full duplex network capable to operate at 1000Mb/s. In this RS code, length of message symbol $\left(k\right)$ is equal to 406, and each message is of $9-$ bit symbol and length of the codeword

Figure 1. The architecture of RS Encoder and Decoder



$\left(n\right)$ is 450. RS (450, 406) is also known as shortened Reed Solomon code, and it can correct error up to $\left(n-k\,/\,2=22\right)$. For RS (450, 406) primitive code polynomial is x^9+x^4+1. The received codeword at the receiver side is also referred as RS frame. One of the essential elements of both RS encoding and decoding is Galois Field (GF) multiplication and division. RS decoder first checks the RS frame is valid or not then it will check the error. If there is error, then it will correct the codeword. If the codeword has error more than the correcting capability, then RS decoding will almost fail every time. At RS encoder (A. Yadav et al., 2019; Mustafa et al. 2013) side, the sender sends a continuous 406 RS frame to the encoder and decoder circuit. After 406 clock cycles, it starts sending calculated 44 parity symbols to the end of message symbols, so the input to the RS decoder circuit is continuous RS frame of size 450 and output of the decoder will be the corrected codeword as shown in Figure 1. Initially enable signal will be low for 406 RS frame, and after that, it becomes high for sending the calculated parity symbols, Multiplexer will continuously transmit 450 RS frames to the decoder. There are several steps to solve the corrected codeword, for every step, there are different types of decoding algorithms used to adjust the codeword. If the codeword is not adequately corrected, then low latency is achieved, and hence it will fail to decode the received codeword.

2. RELATED WORK

This chapter represents some background overview of Reed Solomon encoder and decoder. It provides brief idea of different version of RS encoder and decoder and also about different techniques used for calculating the Key Equation Solver.

2.1. Diplaxmi Chaudhari et al.

Authors Diplaxmi et al. (2019) introduced the FPGA implementation and VHDL design of RS(7, 3) encoder and decoder. In this paper author implemented the RS(7, 3) encoder and decoder in Verilog and also implemented hardware in Actel ProASIC3 FPGA kit. They used Berlekamp Massey algorithm for key equation solver that reduced the hardware complexity as the other algorithm like Euclidean algorithm. But RS(7, 3) can correct error up to 2 error symbols, So it has less error correction capability.

2.2. Rajeev Kumar and Priyanka Dayal

Authors Dayal & Rajeev (2013) implemented Reed Solomon encoder and decoder on the FPGA for the wireless network. They improved performance of the RS(255, 239) for IEEE 802.16 standard and also compared between RS(255, 239) and RS(255, 243) in the terms of the LUT's used. But RS(255, 239) and RS(255,243) can correct error up to 8 and 6 error symbol respectively. So both RS(255, 239) and RS(255, 243) has less error correction capability in comparison to our RS(450, 406) code.

2.3. Yi-Min Lin, et al.

Authors Y. Lin et al. (2014) introduced a 2.56 Gb/s Soft RS(255, 239) decoder chip to increase the error correction capability performance with area-efficient architecture. They proposed decision-confined algorithm to enhance decoding efficiency. Instead of generating number of possible codeword and calculating correct codeword, they produces only single codeword by confining degree of error location polynomial, so complexity of hardware reduced by removing decision making unit. But RS(255, 239) has less error correction capability in comparison to RS(450, 406). RS(255, 239) can corrects error up to 8 error symbol.

2.4. Anindya Sundar Das et al.

The Authors A. S. Das et al. (2013) designed FPGA implementation for RS(255, 251) encoding and decoding. They discussed all the step of encoder and decoder, and shown the synthesis results. They implemented in Verilog language and simulation is done in ModelSim and synthesized design by Xilinx ISE 7.1 I tool. Authors used Berlekamp Massey (BM) algorithm for key equation solver (KES) polynomial. RS(255, 251) also has less error correction capability in comparison to our proposed design. It can corrects error up to 2 error symbol.

2.5. Chia – Chun Peng et al.

Authors introduced IP (Intellectual Property) generator of Reed Solomon code. In this paper authors show new and different concept of RS codes IP generator to produce ten different kinds of RS codec including RS (208, 192), RS(72, 64), RS(255, 239), RS(255, 223), RS(207, 187), RS(204, 188), RS(28, 24), RS(36, 22), RS(182, 172) and RS(72, 64) for targeting different communication standard. These RS code IP generator perform implementation and hardware design. They also discussed the fixed architecture of Galois Field multiplier and RS encoder & decoder.

2.6. Dilip V. Sarwate and N. R. Shabhag

The authors D. V. Sarwate and N. R. Shabhag (2001) introduced a Reed Solomon decoder with High Speed architecture, in this paper authors focused on only the different algorithm of the Berlekamp Massey Algorithm (BMA) for calculating the key equation solver (KES), which is most important, also hardest part of Reed Solomon Decoder and its architecture design and complexity. Authors implemented the reformulation of the Inversion-less Berlekamp Massey algorithm (iBM) and shown the RiBM synthesized architecture.

2.7. Chan-ho Yoon

The Author introduced Forward Chien Search (FCS) algorithm for Reed Solomon decoder circuit. He used the Chien Search algorithm for calculating the error location and also discussed about the Forney algorithm which is used for calculating the error values of the Reed Solomon decoding. Author Shown the hardware design of the Chien Search and Forney algorithm. He also discussed about the Seed generator, used for polynomial multiplication.

3. REED SOLOMON DECODING

Reed Solomon Decoding is popular FEC decoding techniques used in communication system and satellite communication during data transmission. Input to the Reed Solomon (RS) decoder (D. V. Sarwate & N. R. Shanbhag, 2001; Z. Wang & J. Ma, 2006; T. Zhang & K. K. Parthi, 2002) is the received codeword, which needs to be decoded. The decoder first checks that RS received codeword is valid or not. If it is not a valid codeword, that means there are more or fewer errors in the received codeword during the data transmission. This part of the decoder circuit is called as error detection. If there are errors in the codeword, then the decoder circuit tries to correct all the inaccuracies by

using correction part, Figure 2 shows the flow diagram of RS decoding. As given in Figure 2, there are five steps to calculate the corrected codeword:

1. Syndrome calculation
2. KES (Key Equation Solver)
3. Calculation of error position
4. Calculation of error values
5. Error corrector

We implemented the Reed Solomon Decoder in Verilog by using design architecture of every step of the decoder. The Architecture of every step is shown in Figures. Every step of the decoder has a different algorithm, and by using these algorithms, we implemented our design. The RS (450, 406) decoder is being implemented for 1000BASE-T1 AE or Gigabit Automotive Ethernet technology. For every step, a different algorithm is used to solve the received codeword, Delay registers (FIFO registers) are used to store RS frame sent by the sender. This decoding is also known as syndrome dependent decoding. Let the transmitted codeword be $C\left(x\right)$. $R\left(x\right)$ received RS codeword polynomial, and error polynomial is $E\left(x\right)$. So the received codeword is represented as:

$$R\left(x\right) = C\left(x\right) + E\left(x\right) \tag{1}$$

where $C\left(x\right)$ can also be represented as:

$$C\left(x\right) = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + \ldots + c_{n-1} x^{n-1} \tag{2}$$

where, if $e > 0$ errors have generated in the codeword during data transmission. Then the error polynomial can be represented by:

$$E\left(x\right) = Y_1 x^{i_1} + Y_2 x^{i_2} + Y_3 x^{i_3} + \ldots + Y_e x^{i_e} \tag{3}$$

where Y_1, Y_2,..., Y_e are the error values occurred at the locations $X_1 = \alpha^{i_1}, X_2 = \alpha^{i_2}, \ldots\ldots, X_e = \alpha^{i_e}$. Unknown parameter for the decoder is the error polynomial $E\left(x\right)$, which needs to be calculated by using steps of RS decoding. After calculating the value of $E\left(x\right)$ just XOR with the received codeword polynomial $C\left(x\right)$. So the decoder tries to calculate the error polynomial from the input polynomial $R\left(x\right)$. Figure 2 shows the Architecture of the Reed Solomon Decoding. Design of RS decoder is shown in Figure 2, input to design is continuous RS frames. More than one memory is used to store the continuous RS frames and, memory selector is used to select individual memory one at a time for calculating corrected codeword. We used three memory for storing continuous RS frames to avoid data overlapping. We used buffer circuit for storing input data coming from output of the previous module. The time period, number of clock cycles and frequency of operation of each step or module is given in the Table 2.

Figure 2. Flow and Architecture of Reed Solomon Decoding

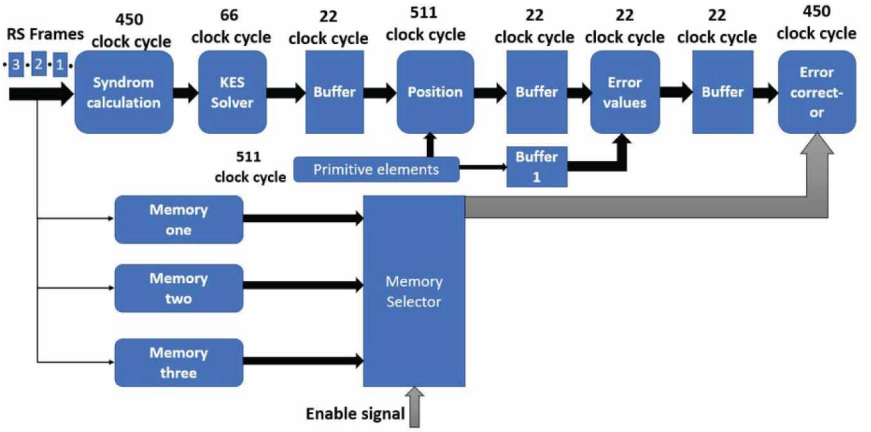


Table 2. Clock cycle, frequency and the time period of each module

Module	Clock Cycle	Frequency (MHz)	Time Period (ns)
Syndrome	450	125	3600
KES solver	66	125	528
Position	511	250	2044
Error Values	22	125	176
Buffer	66	125	528
Buffer1	511	250	2044
Primitive elements	511	250	2044
Error corrector	450	125	3600

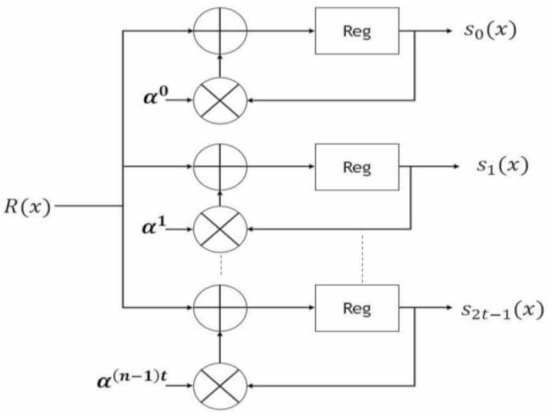
4. SYNDROME CALCULATION

The first step of RS decoding is to calculate the syndrome polynomial by using the received codeword, and Syndrome values also decide that, there is an error or not. If all the syndrome values are zero, it means there is no error. The number syndrome values are equal to $2t = n - k$ and the syndrome polynomial is given in equation (4):

$$S\left(x\right) = s_0 + s_1 x + s_2 x^2 + \ldots + s_{(2t-1)} x^{(2t-1)} \tag{4}$$

$$S\left(x\right) = \sum_{i=0}^{(2t-1)} s_i x^i \tag{5}$$

Figure 3. The architecture of Syndrome calculation



The syndrome values are the value of received codeword polynomial calculated at the different primitive elements α^i where $i = 0,1,\ldots,2t-1$. Basically we substitute all 44 primitive elements start from α^0 to α^{43} or α^1 to α^{44} into the received codeword. The Architecture of syndrome calculation is given in Figure 3.

Let $R(x) = C(x) + E(x)$. So:

$$s_i = R(\alpha^i) = c(\alpha^i) + E(\alpha^i) = E(\alpha^i), where\ 0 \le i \le 2t-1$$

If all $2t$ syndrome values are zero, it means the received codeword $R(x)$ will be equal to transmitted codeword $C(x)$ and it also indicate that there is no error in received codeword. Otherwise, decoder calculates the error polynomial $E(x)$. We analyzed our design by taking sample data (input message symbol (k)) of length 406, this sample data will act as input to syndrome calculator module.

Let the:

$$frame = \{9'h0a1, 9'h051, 9'h1a9, 9'h114, 9'h0aa, 9'h165, 9'h0ba, 9'h161, 9'h092\}$$

So sample data of length 406 given is:

$$\text{sample data} = \{45\ times\ of\ frame, 9'h000\}$$

Every data symbol is $9-bit$ Hexadecimal number.

So, total 44 $(n-k=44)$ parity symbols calculated through Reed Solomon encoder[8] circuit by using the sample data is:

$$\text{parity symbols} = \{9'h188, 9'h0cd, 9'h016, \ldots 9'h1d8, 9'h020, 9'h1a1\}$$

Now the codeword after appending the parity symbols to the end of sample data is:

Codeword $C\left(x\right) = \{\text{sample data, parity symbols}\}$

Let us assume that 10 number of errors are introduced during the transmission of the codeword from Physical Coding Sublayer (PCS) transmitter side to Physical Coding Sublayer (PCS) receiver side. The decoder then needs to correct all the errors introduced, So the decoder starts decoding step by step. The output of the first step or module is given in Table 3. The modular schematic of syndrome calculation is shown in Figure 4, where syndrome_out are the output of Syndrome module (or syndrome values) and samp_data (sample data), rst_n (reset signal), clk (clock signal) are input to the Syndrome module, and nk and ff are internal parameters.

5. KEY EQUATION SOLVER (KES)

After computing the syndrome polynomial coefficient, it needs to calculate the error position and respective error values. KES (Key Equation Solver) is used to calculate the error evaluator and error locator polynomial. By using these polynomials, we can calculate error position and respective error values and it is the hardest part of RS decoder. The syndrome polynomial is used to calculate error evaluator and locator polynomial. For this, all the syndrome values act as an input to the fundamental equation solver module. And, several algorithms are available to calculate error locator and evaluator polynomial. Let $\Lambda\left(x\right)$ is the locator polynomial of degree e and $\Omega\left(x\right)$ is the evaluator polynomial of at most degree $e-1$ as given below:

$$\Lambda\left(x\right) = \prod_{j=1}^{e}\left(1 - X_j x\right) = 1 + \lambda_1 x + \lambda_2 x^2 + \ldots + \lambda_e x^e \tag{6}$$

$$\Omega\left(x\right) = \sum_{i=1}^{e} Y_i X_i^{m_0} \prod_{j=1, j\neq i}^{e}\left(1 - X_j x\right) = \omega_0 + \omega_1 x + \omega_1 x^2 + \ldots + \omega_{e-1} x^{e-1} \tag{7}$$

The $\Lambda\left(x\right)$ and $\Omega\left(x\right)$ equations are correlated with each other through an equation, known as Key equation as shown in equation (8):

$$\Lambda\left(x\right) S\left(x\right) = \Omega\left(x\right) mod x^{2t} \tag{8}$$

Figure 4. Modular Schematic of Syndrome calculator

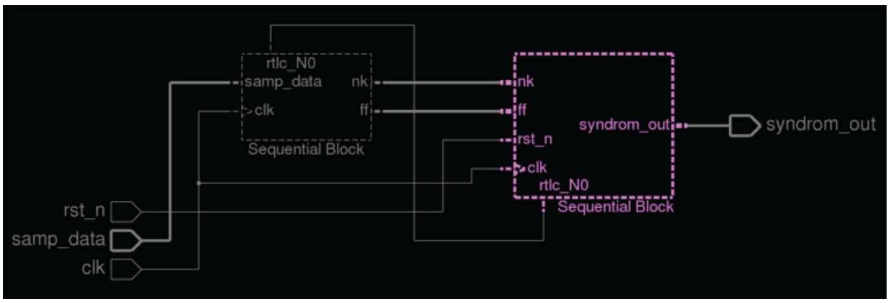


Table 3. Coefficient of Syndrome polynomial

Syndrome Coefficient	Values (9-bit number)	Syndrome Coefficient	Values (9-bit number)
S_0	9'b000011100	S_{22}	9'b110010011
S_1	9'b111001100	S_{23}	9'b111101011
S_2	9'b000000001	S_{24}	9'b111010101
S_3	9'b010110110	S_{25}	9'b010110010
S_4	9'b011010010	S_{26}	9'b101010011
S_5	9'b101001010	S_{27}	9'b101010001
S_6	9'b010000100	S_{28}	9'b110111110
S_7	9'b000010001	S_{29}	9'b011000101
S_8	9'b011111110	S_{30}	9'b000010100
S_9	9'b010001001	S_{31}	9'b110010011
S_{10}	9'b011110000	S_{32}	9'b111111010
S_{11}	9'b111010101	S_{33}	9'b001100001
S_{12}	9'b000100010	S_{34}	9'b100101100
S_{13}	9'b001110110	S_{35}	9'b111111110
S_{14}	9'b001111110	S_{36}	9'b100011110
S_{15}	9'b000100111	S_{37}	9'b110011000
S_{16}	9'b100000100	S_{38}	9'b111111100
S_{17}	9'b000110011	S_{39}	9'b000111001
S_{18}	9'b101110000	S_{40}	9'b101010111

continued on following page

Table 3. Continued

Syndrome Coefficient	Values (9-bit number)	Syndrome Coefficient	Values (9-bit number)
S_{19}	9'b001100000	S_{41}	9'b100100000
S_{20}	9'b000011101	S_{42}	9'b001000101
S_{21}	9'b011000000	S_{43}	9'b011111110

By using the equation (8) it is required to calculate error locator $\Lambda(x)$ and evaluator $\Omega(x)$ polynomial, which is most important and the hardest part of the Reed Solomon decoding of any order. The three algorithms that are generally used to calculate the locator $\Lambda(x)$ and evaluator $\Omega(x)$ polynomial.

- PGZ (Peterson Gorenstein Zieter)
- Berlekamp Massey (BM) decoding Algorithm
- Sugiyama's Euclidean (SE) and Extended Euclidean (EE)

Berlekamp Massey (BM) algorithm (J. L. Massey, 1969; E. R. Berlekamp, 1984; E. R. Berlekamp et al., 1994) is a famous algorithm to solve the Key equation. There are several version of Berlekamp Massey algorithm as:

- Inversionless BM (iBM)
- Simplified iBM
- Reformulated RiBM

If $e \le t$, it is easy to calculate $\Lambda(x)$ and $\Omega(x)$ but if $e > t$ then the algorithms almost always fail to calculate error locator $\Lambda(x)$ and error evaluator $\Omega(x)$ polynomial. Once the $\Lambda(x)$ and $\Omega(x)$ are computed, then the decoder can easily find the error location and its respective error values. Syndrome calculator and key Equation Solver steps are correlated to Key equation as given in equation (8), because Key equation depends upon the syndrome polynomial. After calculating the Key equation, error locator and error evaluators are much straight forward to compute error position and error values. The iBM algorithm is an iterative process to solve the key equation. Basically iBM (J. H. Hung et al., 2016) algorithm calculate the scalar multiples of $\beta \cdot \Lambda(x)$ and $\beta \cdot \Omega(x)$ in the place of $\Lambda(x)$ and $\Omega(x)$. It is clear that Chien – search and Forney algorithm finds the correct error location and error values respectively. But in iBM algorithm, it stores $\lambda_0 = \beta$.

Input to the iBM algorithm is syndrome values, $s_i \; for \; i = 0,1,2 \ldots 2t-1$. The pseudo code of the iBM algorithm is given below.

5.1. The iBM Algorithm

Initialization

$\lambda_0(0) = b_0(0) = 1, \lambda_i = b_i = 0 \; for \; i = 1,2,3,\ldots t \quad and \; k(0) = 0, \gamma(0) = 1$

Input: $s_i \; for \; i = 0,1,2 \ldots 2t-1$

$for \; r = 0 \; \textbf{step} 1 \; \textbf{until} \; 2t-1 \; \textbf{do}$

begin

step iBM.1 $\delta(r) = s_r.\lambda_0(r) + s_{r-1}.\lambda_1(r) + \ldots + s_{r-t}.\lambda_t(r)$

step iBM.2 $\lambda_i(r+1) = \gamma(r).\lambda_i(r) - \delta(r) b_{i-1}(r), (i = 0,1,2,\ldots\ldots,t)$

step iBM.3 ***if*** $\delta(r) \neq 0$ ***and*** $k(r) \geq 0$

then

begin

$b_i(r+1) = \lambda_i(r), \quad (i = 0,1,2,\ldots\ldots,t)$

$\gamma(r+1) = \delta(r)$

$k(r+1) = -k(r) - 1$

end

else

begin

$b_i(r+1) = b_{i-1}(r), \quad (i = 0,1,2,\ldots\ldots,t)$

$\gamma(r+1) = \gamma(r)$

$k(r+1) = k(r) + 1$

end

end

for $i = 0$ ***step*** 1 ***until*** $t-1$ ***do .***

$$\omega_i(2t) = s_i.\lambda_0(2t) + s_{i-1}.\lambda_1(2t) + \ldots + s_0.\lambda_i(2t)$$

So the **output** $\lambda_{i(2t)}, i = 0,1,2,\ldots.t, \omega_i(2t), i = 0,1,2,\ldots t-1$.

For $r < t$, step **iBM.1** contains the terms $s_{-1}.\lambda_{r+1}(r), s_{-2}.\lambda_{r+2}(r), \ldots\ldots, s_{r-t}.$ $\lambda_t(r)$ having unknown values $s_{-1}, s_{-2}, \ldots\ldots, s_{r-t}$. Fortunately for the degree of $\Lambda(r,x) \leq r$, it is known that $\lambda_{r+1}(r) = \lambda_{r+2}(r) = \ldots = \lambda_t(r) = 0$ so unknown value s_i does not affect the value of $\delta(r)$. There is a similarity between the steps **iBM.1** and **iBM.4,** so it can easily simplify DC and ELU architecture. RS(450, 406) decoding can correct errors up to $t(22)$. Basically, t is the number of errors that can be corrected. The data in Table 2 can be for any value of t from 1 to 22.

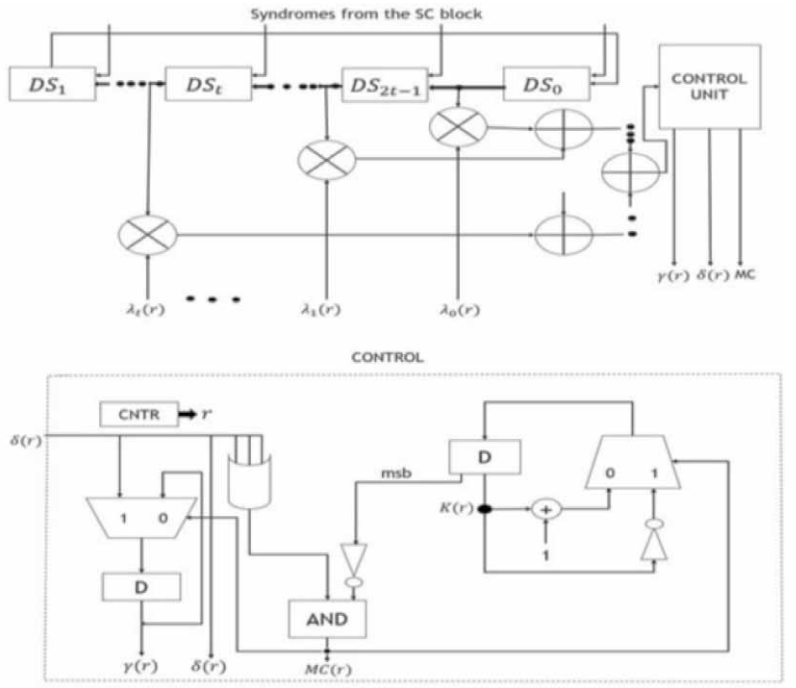
5.2. The Architecture of iBM Algorithm

Due to the similarity in between **step1** and **step2**, iBM architecture is divided into the two ciphering structures, one is Discrepancy Computation architecture and other one is ELU architecture, as given Figure 5 and Figure 6 respectively, These Architecture are as:

- DC(Discrepancy computation) for calculating step **iBM.1**
- ELU (error locator update) block for computing step **iBM.2** and step **iBM.3**

Inside the DC block latches are used to store all the 44 syndrome value s_i where i is from o to 43 arithmetic unit for arithmetic operation based on GF(2^m) where m is equal to 9 and also

Figure 5. (a) Discrepancy Block Architecture (b) Control Unit



the control unit for the entire architecture. The important signal of control unit are $\delta(r)$, $MC(r)$ and $\gamma(r)$.

It is directly connected to the error locator block which has latches for containing $\Lambda(r,x)$ and $B(r,x)$ and also the arithmetic unit of Galois field. In every first $2t$ clock cycles discrepancy block calculates the $\delta(r)$ and handovers this $\delta(r)$ to the ELU block along with $\gamma(r)$ and $MC(r)$ control signal that modifies coefficient of polynomial in the same clock cycle.

5.3. Discrepancy Computation Block Architecture

The discrepancy block architecture is shown in Figure 5, in which shift registers $DS_1, DS_2, \ldots, DS_{2t-1}, DS_0$ are initialized with the syndrome values $s_1, s_2, \ldots, s_{2t-1}, s_0$ respectively. In every first $2t$ clock cycles, $t+1$ multiplier circuits compute product in **iBM step1,** and calculated value act as input to **iBM step2** . A control unit for the combined architecture is shown in Figure 5, Outputs of the control unit are $\delta(r)$, $MC(r)$ and $\gamma(r)$. The control has two counters for variable r and $k(r)$ and storage element for storing $\gamma(r)$ value. The counter units compute OR operation of the discrepancy $\delta(r)$ to check if $\delta(r)$ is zero or non-zero as given in **iBM step3**. The counter can be implemented by using Ring counter or by 2's complement. If the counter for variable $k(r)$ is implemented in 2's complement, then the condition $k(r) \geq 0$ becomes true if the MSB bit of the $k(r)$ counter is zero, as given in **iBM step3**. When the $MC(r)$ signal is generated, the counter for variable $k(r)$ will change. $2t$

number of clock cycles are required to calculate the error locator polynomial $\Lambda\left(x\right)$ and next t clock cycles are required for calculating error evaluator polynomial $\Omega\left(x\right)$.

5.4. ELU Architecture

In ELU block, coefficients of the polynomial in **iBM step1** and **step2** updates as the $MC\left(r\right)$ signal becomes available. ELU block calculates the next value of the variable by using the previous values of the coefficients of the polynomial, as shown in Figure 6. The processor elements $\left(PE0_{i}\right)$ are used to update the coefficients $\lambda\left(r\right)$ and $B\left(r\right)$. The coefficients of $\delta\left(r\right), B\left(r\right), and\ MC\left(r\right)$ are sent to the processor which updates old value by a new value, processor $PE0$ is initialized with zero. The calculated values of the coefficients of error locator and evaluator polynomial are given in the Table 4.

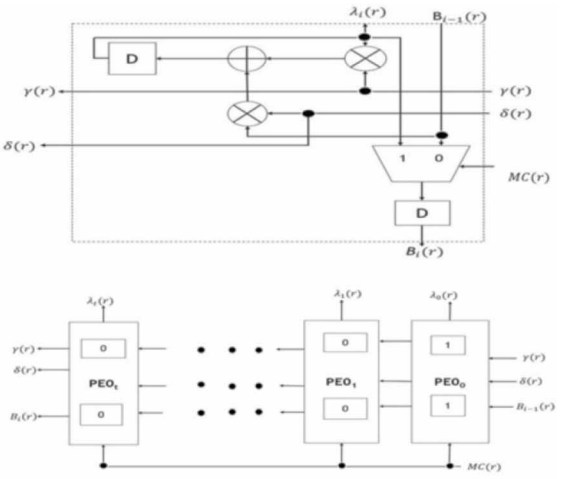
6. ERROR POSITION CALCULATION

The next step after calculating error locator and evaluator polynomial is to calculate the position of errors in received codeword, the logic behind position calculation is to substitute primitive elements

Table 4. Coefficient of Error Locator and Evaluator polynomial

S. No.	Coefficient of $\lambda\left(r\right)$	Values (9-bit Number)	Coefficient of $\Omega\left(r\right)$	Values (9-bit Number)
1.	λ_{0}	$9'b001000000$	ω_{0}	$9'b100110011$
2.	λ_{1}	$9'b100101101$	ω_{1}	$9'b101001011$
3.	λ_{2}	$9'b011010101$	ω_{2}	$9'b011011101$
4.	λ_{3}	$9'b100100100$	ω_{3}	$9'b111111101$
5.	λ_{4}	$9'b011101010$	ω_{4}	$9'b000010011$
6.	λ_{5}	$9'b000100110$	ω_{5}	$9'b111110000$
7.	λ_{6}	$9'b111010010$	ω_{6}	$9'b001100110$
8.	λ_{7}	$9'b011110010$	ω_{7}	$9'b110011011$
9	λ_{8}	$9'b001100110$	ω_{8}	$9'b000010100$
10.	λ_{9}	$9'b001000000$	ω_{9}	$9'b100010101$
11.	λ_{10}	$9'b111010010$	ω_{10}	$9'b000000000$

Figure 6. (a) the architecture of ELU (b) Processor Elements ELU

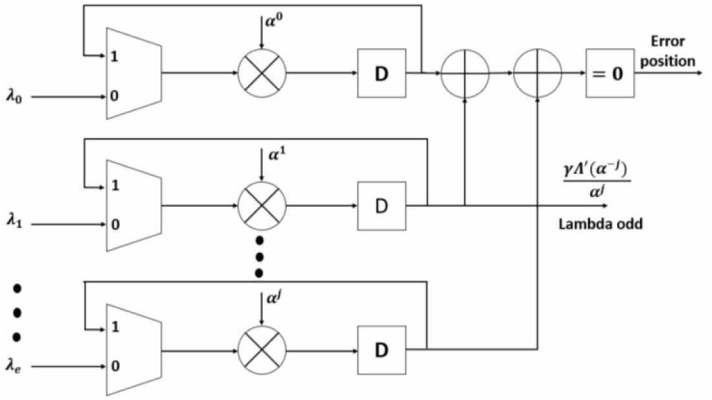


from α^0 to α^{510} in error locator polynomial $\Lambda(x)$. Chien – search (H. C. Chang et al., 2011; C. H. Yoon, 2013) algorithm is used to calculate the error position. It is the most popular method of calculating roots of the error locator polynomial. Chien – search algorithm uses the error locator polynomial for calculating error position. In this algorithm, it calculates the roots of the error locator polynomial (ELP) $\Lambda(x)$ by putting the value of primitive elements of Galois Field (GF), and the inverse of the roots will be the error position. It is not easy to calculate the roots of the ELP. The Architecture of the Chien – Search algorithm is shown in Figure 7.

Inputs to the Chien – Search Algorithm is the coefficients of the error locator polynomial, So this algorithm calculates the $\Lambda(\alpha^i)$ at every value of $i = 1,2,3 \ldots n$. After substituting the elements of the finite field to the error locator polynomial, it checks the condition:

$$\Lambda(\alpha^i) = 0 \tag{9}$$

Figure 7. The architecture of Chien – Search Algorithm



If this condition holds, then an error is to be in the position of $\left(n-i\right)$. And if it's not, then there is no error. In Figure 7, odd values of ELP $\left(\lambda_1,\lambda_3,\lambda_5\ldots\ldots\right)$ are calculated in one side and even values on the other side $\left(\lambda_2,\lambda_4,\lambda_6,\ldots\right)$. If in any clock cycle $\left(<n\right)$, the summation of all these values is zero, then the error position will be equal to the position of the clock. An important element of Chien – Search algorithm are Galois Field multiplication, division, primitive element and primitive polynomial. For example, if the ELP is:

$$\Lambda\left(x\right)=1+\lambda_1 x+\lambda_2 x^2+\lambda_3 x^3$$

To evaluate $\Lambda\left(x\right)$ at every non-zero elements in finite field of GF $\left(2^m\right)$ in sequence:

$$x=\alpha^1,x=\alpha^2,x=\alpha^3,x=\alpha^4,\ldots,x=\alpha^{2^m-1}$$

So, after putting these values:

$$\Lambda\left(\alpha^1\right)=1+\lambda_1(\alpha^1)+\lambda_2\left(\alpha^1\right)^2+\lambda_3\left(\alpha^1\right)^3$$

$$\Lambda\left(\alpha^2\right)=1+\lambda_1(\alpha^2)+\lambda_2\left(\alpha^2\right)^2+\lambda_3\left(\alpha^2\right)^3$$

…

$$\Lambda\left(\alpha^{2^m-1}\right)=1+\lambda_1(\alpha^{2^m-1})+\lambda_2\left(\alpha^{2^m-1}\right)^2+\lambda_3\left(\alpha^{2^m-1}\right)^3$$

For any value of α^{2^m-1} if $\Lambda\left(x\right)$ value is zero then α^{2^m-1} will be the root of $\Lambda\left(x\right)$, and inverse of root will be error position.

The modular Schematic of the error position algorithm is shown in Figure 8, Position_out indicates the output of the position module. The calculated results of the Error Position Calculation module is given in the Table 5.

Figure 8. Modular Schematic of Position Calculation

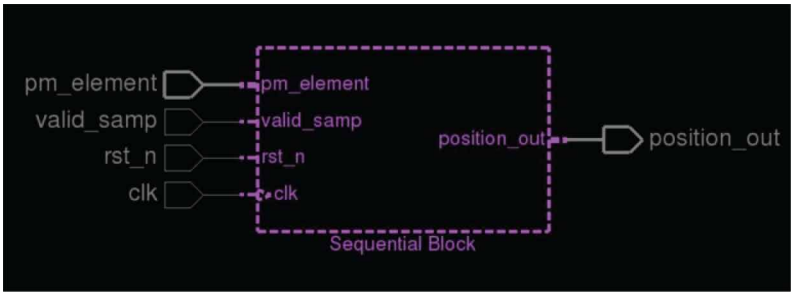


Table 5. Calculated Root value, error position and error values

S. No.	Root Values	Position Values	Coefficient of the Error Polynomial	The Error Values (9-bit Number)
1.	α^{62}	$\alpha^{449} = X^{449}$	Y_1	$9'b000000001$
2.	α^{63}	$\alpha^{448} = X^{448}$	Y_2	$9'b000000001$
3.	α^{64}	$\alpha^{447} = X^{447}$	Y_3	$9'b110100000$
4.	α^{65}	$\alpha^{446} = X^{446}$	Y_4	$9'b000000100$
5.	α^{66}	$\alpha^{445} = X^{445}$	Y_5	$9'b100000000$
6.	α^{83}	$\alpha^{428} = X^{428}$	Y_6	$9'b000010100$
7.	α^{84}	$\alpha^{427} = X^{427}$	Y_7	$9'b010111010$
8.	α^{85}	$\alpha^{426} = X^{426}$	Y_8	$9'b001100000$
9.	α^{86}	$\alpha^{425} = X^{425}$	Y_9	$9'b101110001$
10.	α^{87}	$\alpha^{424} = X^{424}$	Y_{10}	$9'b100000111$

7. ERROR VALUES CALCULATION

After computing the Syndrome value, Key Equation Solver and error position, Now the next step of Reed Solomon decoding algorithm is to calculate the error values with respect to the every error position value. The two popular methods of calculating the error values are:

- Transform Decoding Algorithm
- Forney Algorithm

In the frequency domain, the Transform decoding algorithm is used, and in the time domain Forney algorithm (G. D. Forney, 1965; Yingquan Wu & Yu Kou, 2009) is used. Transform decoding process does not use FFI or Chien – search algorithms and also have large complex circuit than Chien – search, so it occupies a large area. Therefore, Forney algorithm is generally used due to its lesser circuit complexity, more accuracy and more efficiency.

7.1. Forney Algorithm

The root value and the coefficients of $\Lambda\left(x\right)$ and $\Omega\left(x\right)$ are the inputs to the Forney algorithm and it is related to Chien - Search algorithm. In the Forney algorithm, only odd coefficients of error locator polynomial $\Omega\left(x\right)$ are used. It also uses the finite field multiplier and division just like Chien – Search algorithm. The formula to calculate the error values $\left(Y_i\right)$ is:

$$Y_i = \frac{-x^{m0}\Omega(x)}{x\Lambda'(x)} \tag{10}$$

where $x = \alpha^{-j}$ is point to root as calculated by the Chien – Search algorithm. The architecture of this algorithm is shown in the Figure 9. The equation (10) gives valid and correct results if there are errors in the codeword. The $\Lambda'(x)$ in equation (10) is the first derivative of the error polynomial $\Lambda(x)$ so the $\Lambda'(x)$:

$$\Lambda'(x) = \lambda_1 + 2\lambda_2 x + 3\lambda_3 x^2 + \ldots \tag{11}$$

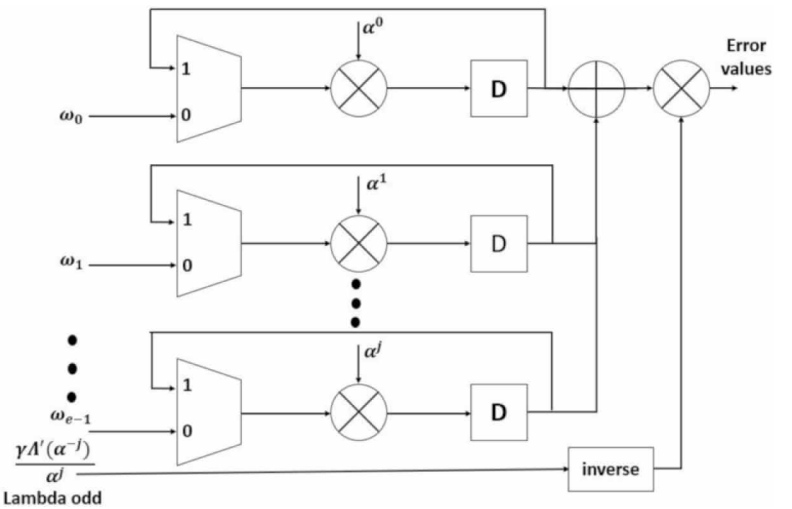
The $\Lambda'(x)$ is the derivative of $1 + \lambda_1 x + \lambda_2 x^2 + \ldots + \lambda_e x^e$. After multiplying by x to the equation (11):

$$x\Lambda'(x) = \lambda_1 x + \lambda_3 x^3 + \ldots \tag{12}$$

The equation (12) is just like the odd terms of the locator polynomial $\Lambda(x)$ and these odd terms can be calculated during the evaluation of the error position, so no need to calculate separately. The equation (10) can be easily modified by taking $m_0 = 0$. The inverse of the **Lambda odd** is calculated by the Brute-force search algorithm as shown in Figure 9, steps of this algorithm is given below:

- Let $f(p)$ and $g(p)$ are the polynomials in $GF(p^m)$.
- Suppose $M(p)$ be the primitive polynomial or irreducible polynomial in $GF(p^m)$.
- So, the multiplicative inverse of $f(p)$ is given by $a(p)$.
- If $(f(p).a(p))(mod\, m(p)) = 1$, then $a(p)$ is the inverse of the $f(p)$.

Figure 9. The architecture of Forney Algorithm



In Forney algorithm, we use both error locator $\Lambda(x)$ and error evaluator polynomial $\Omega(x)$. But only odd term of error locator polynomial $\Lambda(x)$ is used. The calculated error values by using the Forney algorithm is given in Table 5. So error polynomial $E(x)$ for ten number of error can be represented in terms of polynomial as given in equation (13):

$$E(x) = Y_1 x^{449} + Y_2 x^{448} + Y_3 x^{447} + \ldots + Y_9 x^{425} + Y_{10} x^{424} \tag{13}$$

8. ERROR CORRECTOR

The last step of RS decoding algorithm, after getting the error location and respective error values is error corrector. As now we know the error polynomial $E(x)$ is given in equation (13). So the corrected codeword can be calculated by just XORing the received codeword by the error polynomial $E(x)$:

$$C(x) = R(x) + E(x) \tag{14}$$

Also the modular Schematic of error corrector module is shown in Figure 10. The crr_data is the output after XORing with $E(x)$ and $R(x)$.

9. ANALYSIS OF SIMULATION RESULTS

In this paper Reed Solomon (RS) decoding is implemented in Verilog. Analysis of simulation results has been done in the Cadence SimVision and Verilog coding of RS decoding is also done in the Cadence tool NCsim. The static verification has been done in Cadence lint tool HAL and functional verification in Synopsys tool Spyglass. HAL is a super linting tool and it generates Schematic tracer for better analysis of design. The synthesis has been done in Vivado 2017.4, The Synthesis is the process of transforming an RTL design into a gate level representation, Verilog is more famous for synthesis designs because it is less tedious than traditional VHDL.

The RTL schematic of syndrome calculating is shown in Figure 11. As key enable is high syndrome modules starts sending the syndrome values to the key equation solver module for calculating the locator and evaluator polynomial, mod1_out is the syndrome values as shown in Figure12.

Figure 10. Modular Schematic of error corrector module

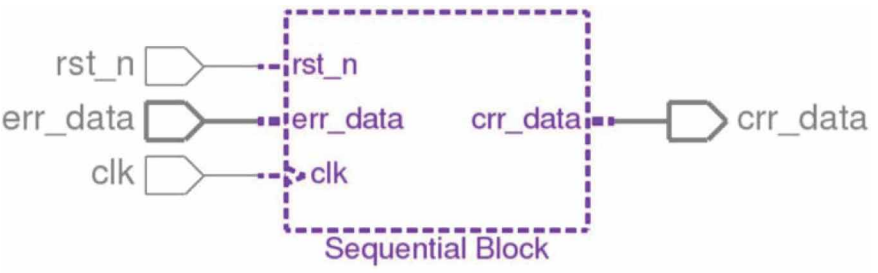


Figure 11. RTL Schematic of Syndrome Calculator

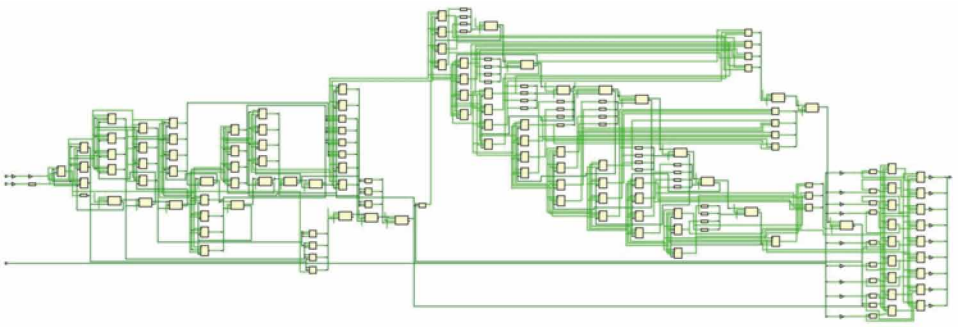
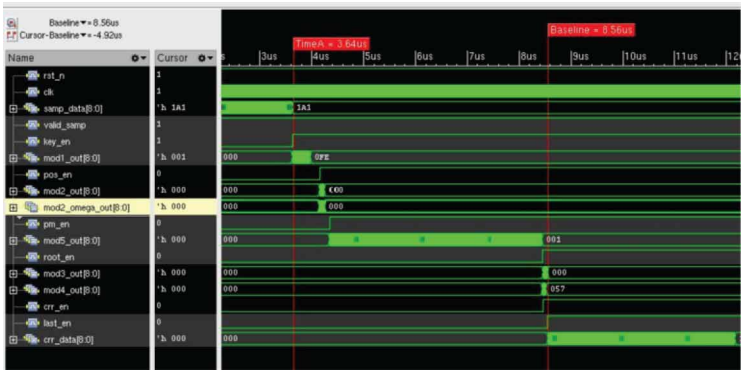


Figure 12. Simulation Results of Syndrome calculator and latency for 3rd case



Simulation results of the key equation solver block are shown in Figure 13. The lemda_values and omega_values showing the coefficients of error locator and error evaluator polynomial respectively, these are calculated by the iBM algorithm. The simulation results of the error position calculation steps are shown in the Figure 14. The variable root_s and position are showing the root value and position value of the error position calculation steps, respectively.

The simulation results of the error values calculation module are shown in Figure 15. The variables err_values are showing the error values at the respective error position. Simulation results of the last steps of the Reed Solomon decoding is shown in Figure 16. The variables err_data and crr_data are indicating the error codeword and corrected codeword, respectively.

Figure 13. Simulation results of key equation solver

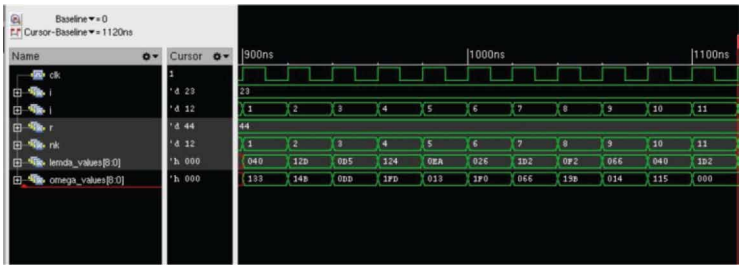


Figure 14. Simulation Results of the position calculator

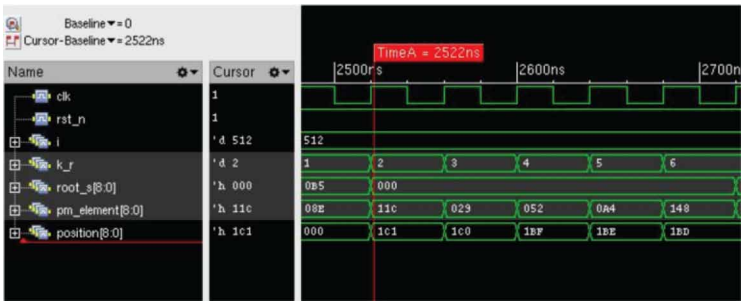


Figure 15. Simulation Results of Error value calculator

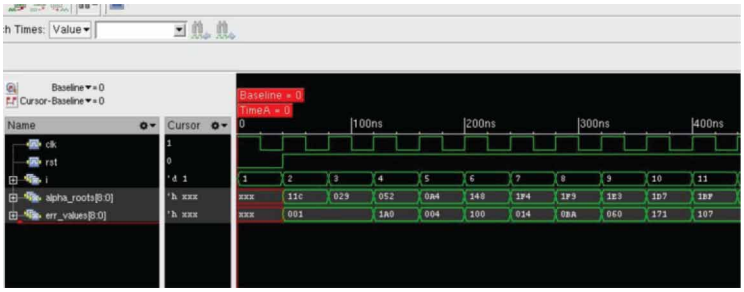
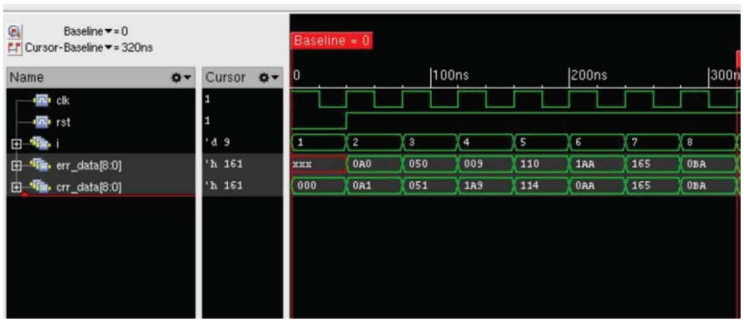


Figure 16. Simulation results of error corrector



Operating frequency of design depends upon the hardware requirements, circuit complexity and more important thing is the latency. We have used two clock signal in our model. Some modules of the model are operating on 1st clock and some other on 2nd clock. We considered three cases as given below:

1. All the design modules are operating at 125 MHz frequency.
2. Some modules are operating at 125 MHz and some at 250 MHz frequency.
3. Some modules are operating at 125 MHz and some at 750 MHz frequency.

Analysis of latency is vital in the Reed Solomon decoding. Latency also depends upon the clock frequency in each module of the decoder and the circuit complexity. Best latency of our

design is obtained at the 3rd case, and it is most suitable for our Gigabit Automotive Ethernet design, it means all design module is working correctly and efficiently. The simulation results for the 3rd case is shown in Figure 17. Time period for all three cases is given in Table 6. If the 125 MHz and 250 MHZ (2nd case) frequency is used in the decoder module then the simulation results of latency is shown in the Figure 12.

The latency is the time required to move from one point to other point within computer system and it is generally measured in nanoseconds, So the latency is:

$$\text{Latency } = 8.56\mu s - 3.64\mu s = 4.92\mu s$$

If latency is calculated according to the Table 2, where some modules are operating in 125 MHz & 250 MHz then latency obtained is:

$$\text{Latency } = 3.026\mu s \text{ approx.}$$

Figure 17. Simulation result of latency for 3rd case

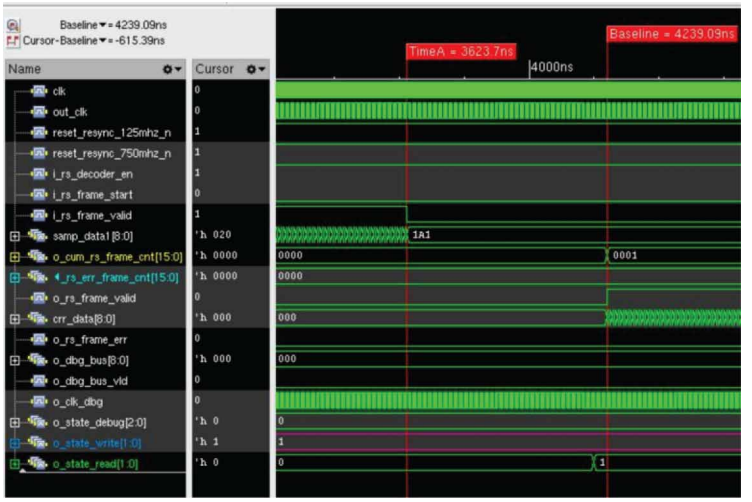


Table 6. Time period of design for all three Cases

Module Name	Time Period (1st Case) (ns)	Time Period (2nd Case) (ns)	Time Period (3rd Case) (ns)
Syndrome	8	8	8
KES Solver	8	8	1.332
Position	8	4	1.332
Error Values	8	8	1.332
Buffer	8	8	1.332
Buffer1	8	4	1.332
Primitive Elements	8	4	1.332
Error Corrector	8	8	8

If the first and last module is operating on the 125 MHz & other modules are operating on 750 MHz frequency then simulation results of the latency is shown in Figure 17.

So:

$$\text{Latency } = 4239.09ns - 3623.7ns = 615.39ns$$

10. CONCLUSION AND CHALLENGES

RS encoding and decoding are powerful error correction techniques, that's why they are used in the Gigabit Ethernet. This decoding technique provides better bandwidth in comparison to the other decoding techniques. The RS(450, 406) decoding is also known as the shortened RS code. We have shown that the latency depends on the frequency of each module. And we implemented this decoding techniques in Verilog. In every step of the decoding algorithm technique, different algorithms are used to decrease the complexity and power consumption. The RS (450, 406) codes are used in 1000BASE-T1, to minimize channel noise. This RS code architecture can be pipelined for gaining high speed. The hardest part of the decoding is to solve key equation, this done by using iBM algorithm. $2t$ clock cycles are required to calculate the error locator polynomial and t clock cycles for calculating the error evaluator polynomial. iBM algorithm is a power-efficient algorithm. The decoder can check whether the number of roots are equal to the degree of locator polynomial or not. If not, then the received codewords cannot be corrected. Latency depends upon the frequency of operation of each module. If input RS frame is more than one RS frame, it needs to use more memory to store the continuous frames. Every Reed Solomon encoding and decoding algorithm is based on the finite field which is also known as Galois field, For RS (450, 406) decoding GF$\left(2^m\right)$ is used.

In the automotive industry, more advanced electronics is being used in vehicles. It is a challenging task to use Gigabit Automotive Ethernet for connecting different functional components in the car. RS encoding and decoding are powerful error correction techniques used in 1000BASE-T1 to reduce channel noise during data transmission. The most Challenging part of Reed Solomon decoding is the calculation of Key Equation Solver (KES) efficiently because solving key equation is the crucial and hardest part of the RS decoding. Achieving low latency is also a challenge in RS encoding and decoding.

REFERENCES

Berlekamp, E. R. (1984). Algebraic Coding Theory. New York: McGraw-Hill.

Berlekamp, E. R. (1984). Algebraic Coding Theory. New York: McGraw-Hill.

Blahut, R. E. (1983). *Theory and Practice of Error-Control Codes*. Addison-Wesley.

Chang, H. C., Wu, J. Y., & Liao, Y. C. (2011). *Method and apparatus for decoding shortened BCH code for Reed Solomon code.* US Patent, Number 7,941,734 B2.

Das, A. S., Das, S., & Bhaumik, J. (2013). Design of RS (255, 251) Encoder and Decoder in FPGA. *International Journal of Soft Computing and Engineering*, *2*(6).

Dayal, P., & Patial, R. K. (2013). FPGA Implementation of Reed-Solomon Encoder and decoder for Wireless Network 802.1. *International Journal of Computers and Applications*, *68*(16).

Forney, G. (1965). On decoding BCH codes. *IEEE Transactions on Information Theory*, *11*(4), 549–557. doi:10.1109/TIT.1965.1053825

Garcia-Herrero, F., Valls, J., & Mehe, P. K. (2011). High-Speed RS (255, 239) Decoder Based on LCC Decoding. *Circuits, Systems, and Signal Processing*, *30*(6), 1643–1669. doi:10.1007/s00034-011-9327-4

Gorenstein, D., & Zierler, N. (1961). A Class of Error-Correcting Codes in P^m . *Journal of the Society for Industrial and Applied Mathematics*, *9*(2), 207–214. doi:10.1137/0109020

Hung, J. H., & Yen, C. N. (2016). *Simplified inversionless Berlekamp Massey algorithm for binary search BCH code and circuit implementing therefor.* US Patent, Number 9,459,836 B2, 4.

Kaur, M., & Sharma, V. (2010). Study of Forward Error Correction using Reed-Solomon Code. *International Journal of Electronics Engineering*, *2*, 331–333.

Lin, Y. M., Hsu, C. H., Chang, H. C., & Lee, C. Y. (2014). A 2.56 Gb/s Soft RS (255, 239) Decoder Chip for Optical Communication Systems. *IEEE Transactions on Circuits and Systems. I, Regular Papers*, *61*(7), 2110–2118. doi:10.1109/TCSI.2014.2298282

Massey, J. L. (1969). Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, *IT-15*(1), 122–127. doi:10.1109/TIT.1969.1054260

Mustapha, E. H., & Belkasmi, M. (2012). VHDL Design and FPGA Implementation of Fully Architecture of Iterative Decoder of Majority Logic codes for High Data Rate Applications. *Journal of Wireless Networking and Communication*, *2*(4), 35–42. doi:10.5923/j.jwnc.20120204.02

Omura, J. K., & Massey, J. L. (1986). *Computational method and apparatus for finite field arithmetic.* United States Patent 458762.

Peng, C. C., Liao, C. H., & Chen, R. J. (2015). IP generator of Reed Solomon codecs. *2015 IEEE International Conference on Consumer Electronics*, 392-393. doi:10.1109/ICCE-TW.2015.7216961

Sana, Z., & Gupta, R. (2019). A Comparison of RS (7, 3) and RS (15, 9), Employing Reed-Solomon Encoder and Decoder. *International Journal of Electrical, Electronics and Data Communication*, *7*(12).

Sarwate, D. V., & Shanbhag, N. R. (2001). High-speed architectures for Reed-Solomon decoders. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *9*(5), 641–655. doi:10.1109/92.953498

Sha, J., Lin, J., Wang, Z., Li, L., & Gao, M. (2009). Decoder Design for RS-Based LDPC Codes. *IEEE Transactions on Circuits and Wystems. II, Express Briefs*, *56*(9), 724–728. doi:10.1109/TCSII.2009.2027945

Ullah, S., Mohaisen, M., & Alnuem, M. A. (2013). A Review of IEEE 802.15.6 MAC, PHY, and Security Specifications. *International Journal of Distributed Sensor Networks*, 1-12.

Wang, Z., & Ma, J. (2006). High-Speed Interpolation Architecture for Soft-Decision Decoding of Reed–Solomon Codes. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *14*(9), 937–950. doi:10.1109/TVLSI.2006.884046

Wicker, S. B., & Bhargava, V. K. (1994). *Reed-Solomon Codes and Their Applications. IEEE*. doi:10.1109/9780470546345.index

Wu, Y., & Kou, Y. (2009). *Degree limited polynomial in Reed Solomon decoding,* US Patent, Number 7,613,988, B1, 3.

Yadav, A., Jindal, P., Basappa, D., & Prakashaiah, M. (2019). Forward Error Correction for Gigabit Automotive Ethernet using RS (450, 406) Encoder. *International Journal of Innovative Technology and Exploring Engineering*, *9*.

Yoon, C. H. (2008). *Forward Chien search type Reed Solomon decoder circuit.* US Patent, Number 7,406,651 B2, 29.

Zhang, T., & Parhi, K. K. (2002). On the high-speed VLSI implementation of errors-and-erasures correcting Reed-Solomon Decoders. *GLSVLSI '02: Proceedings of the 12th ACM Great Lakes symposium on VLSI*, 89–93. doi:10.1145/505306.505326

Akhilesh Yadav received the Bachelor of technology engineering in electronics engineering department from the Dr. A.P.J Abdul Kalam technical University Lucknow in 2018, Currently pursuing Master of technology in electronics and communication engineering department from the NIT Kurukshetra, and working as Student Intern in NXP Semiconductor Bangalore India. He has published 2 paper in International Journals. His research interest include Image processing, Computer Vision, Error correcting codes and Digital VLSI Design and RTL Design.

Poonam Jindal working since 2008 with ECE Department in National Institute of Technology Kurukshetra. She received her Ph.D. in Electronics and Communication Engineering from NIT Kurukshetra in 2016. She acquired degrees of M.Tech and B.Tech in 2005 and 2003 respectively. She has published 56 papers in various International Journals, Conferences and Book Chapters. Her research interests include wireless network security, wireless communication, physical layer security, Internet of Things, Security optimization in wireless networks. She is a reviewer of various reputed International Journals and conferences. She has guided 19 M.Tech dissertations and 30 B.Tech projects in the area of wireless networks.

Devaraju Basappa having an experience of 16 years in ASIC front end design and implementation and is currently working with NXP Semiconductor Bangalore as a Design Engineer. He received Bachelor of Engineering in Electronics and Communication.