Modified Welch Berlekamp Algorithm to Decode Reed Solomon Codes

Ali Mahmudi*, *Sentot* Achmadi, and *Michael*

Departement of Software Engineering, National Institute of Technology (ITN Malang).
Jalan Sigura-gura 2, Malang 65145, Indonesia.

Abstract. In this paper, the Reed Solomon Code is decoded using the Welch-Berlekamp Algorithm. The RS Decoder is implemented using Hardware Description Language VHDL (VHSIC hardware Description Language) and simulated on Modelsim software. Some modifications have been carried out on the Welch Berlekamp algorithm in such a way that it is easier to implement. A pilot design double error correction RS(63, 59) decoder has been written in VHDL and simulated. The XILINX FPGA layout RS(63, 59) is then obtained.

Key words: FPGA, hard decision decoding, Reed Solomon Code, Welch Berlekamp algorithm

1 Introduction

The introduction of error control codes in data transmission and data communication is due to the addition of unwanted noise over the noisy channel. These error control codes are used to minimize the error. These codes are capable of detecting and correcting errors to achieve minimum error level.

Decoding Reed Solomon codes can be classified into two categories: soft decision decoding and hard decision decoding. The soft decision decoding is quite complex and very hardware intensive to implement. Generalized Minimum Distance Decoding is one example of the soft decision decoding. The hard decision decoder is very popular because it is very simple to implement [1, 2]. The Berlekamp-Massey Algorithm and the Euclidean Algorithm are the two well-known algorithms to solve the key equation in hard decision decoding [3–5]. However, the use of the Welch Berlekamp Algorithm to solve the key equation is not much presented [1, 6].

RS code is a cyclic linear block code that has been used in many modern applications. Advanced television [7], DVD applications [8], data hiding [9, 10] and data transmission with nano satellite [11] are some few applications in our modern life.

This paper presents the hard decision decoding for RS codes RS(63, 59), especially on the VHDL implementation of the Welch Berlekamp algorithm to solve the key equation for RS(63, 59) decoder.

This paper is primarily concerned with the VHDL implementation of the Welch

* Corresponding author: amahmudi@hotmail.com

Berlekamp algorithm to decode Reed Solomon Codes over GF(2^6), especially RS(63, 59). Some modifications have been carried out on the algorithm so that it is easier to implement. This paper is organized as follows. In section 2, a brief explanation of RS codes is given. Section 3 presents the RS encoder and section 4 deals with the RS decoder. Section 5 and section 6 deal with the implementation and results, respectively. Finally, section 7 is the conclusions.

2 RS code

The *RS* code is normally specified as *RS*(*n*, *k*) where $n = 2^m - 1$ and $k = n - 2t$ are the number of symbols in every codeword and the number of information symbols in every codeword [1, 2, 4], respectively. It works over $GF(2^m)$ with *m* bits per symbol.

RS(*n*, *k*) is able to correct up to *t* symbols. Then,

$d = 2t + 1$ is the designed distance

n-k is the number of redundancy or parity check symbols. The relation between *t* and *n-k* can be written as

$2t = n\text{-}k$

The RS code can be shown in Figure 1 below.

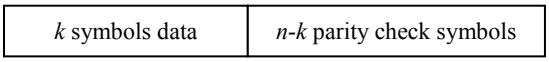

Fig. 1. The systematic *RS* code.

In the case of RS(63, 59), this code works over $GF(2^6)$ with 6 bits per symbol. It has 63 symbols in every codeword, where it has 59 information symbols and the rest is four parity check symbols. This code can correct up to two symbols.

3 RS encoder

The *RS*(*n*, *k*) code, is easily explained by using its special polynomial, called *generator polynomial*, *g*(*x*). The general form of the *RS*(*n, k*) generator polynomial is shown in Equation 1, Equation 2, and Equation 3

$$g(x) = \prod_{i=0}^{n-k-1} (x - \alpha^{b+i}) \tag{1}$$

where *b* is a non-negative integer selected by the designer. In this paper, it is assumed that $b = 0$. In the case of *RS*(*63*, *59*), the generator polynomial *g*(*x*) is

$$g(x) = (x - \alpha^0)(x - \alpha^1)(x - \alpha^2)(x - \alpha^3) \tag{2}$$
$$g(x) = x^4 + \alpha^{18}.x^3 + \alpha^{39}.x^2 + \alpha^{21}.x + \alpha^6$$

All valid *RS*(*n, k*) code words can be regarded as being polynomials of degree (*n-1*) over $GF(2^m)$ which are divisible by its generator polynomial *g*(*x*).

$$I(x) = i_0 + i_1 x + \ldots + i_{k-1} x^{k-1} \tag{3}$$

Let be the information polynomial to be encoded. The systematic encoding works by multiplying the information polynomial *I*(*x*) by x^{n-k}. The result is then divided with *g*(*x*) to obtain the remainder polynomial of degree *n-k-1*. The encoded signal is then formed by subtracting *r*(*x*) from $I(x)x^{n-k}$. It can be written as Equation 4

$$c(x) = I(x).x^{n-k} + (I(x).x^{n-k} \bmod g(x)) \tag{4}$$

It should be noted that the remainder polynomial is, in fact, the parity check polynomial. This generalized encoding scheme is shown in Figure 2.

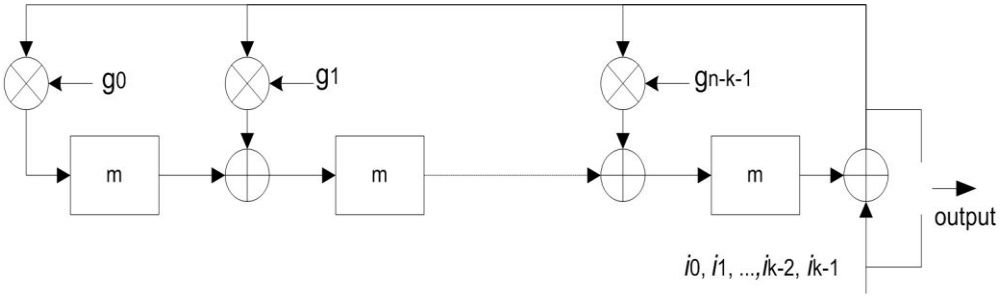

Fig. 2. The structure of *RS* encoder.

4 RS decoder

The RS decoder has three main stages: Re-Encoder, the Welch-Berlekamp algorithm and Chien Search and Error Evaluation

4.1 Re-Encoder

The Re-Encoding [2, 4, 12, 13] is similar to the encoding process. This Re-Encoding process is similar to the syndrome computation process. In the case of RS(63, 59) codes, this Re-Encoding process takes 59 received symbols. It internally generates four check symbols. The re-encoder output will be the mod two addition of four internally generated check symbols and four received check symbol. This is then called the 'parity check' symbol.

In the case of RS(63, 59), the parity check location is $\mathbf{H}= \{\alpha^0, \alpha^1, \alpha^2, \alpha^3\}$ and the parity check is $\mathbf{R} = \{R0, R1, R2, R3\}$.

4.2 Welch-Berlekamp algorithm

The check pair input to the Welch-Berlekamp algorithm will be (R_d, α_d): (R_0, α^0), (R_1, α^1), (R_2, α^2) and (R_3, α^3) consecutively. This algorithm is shown in Figure 3.

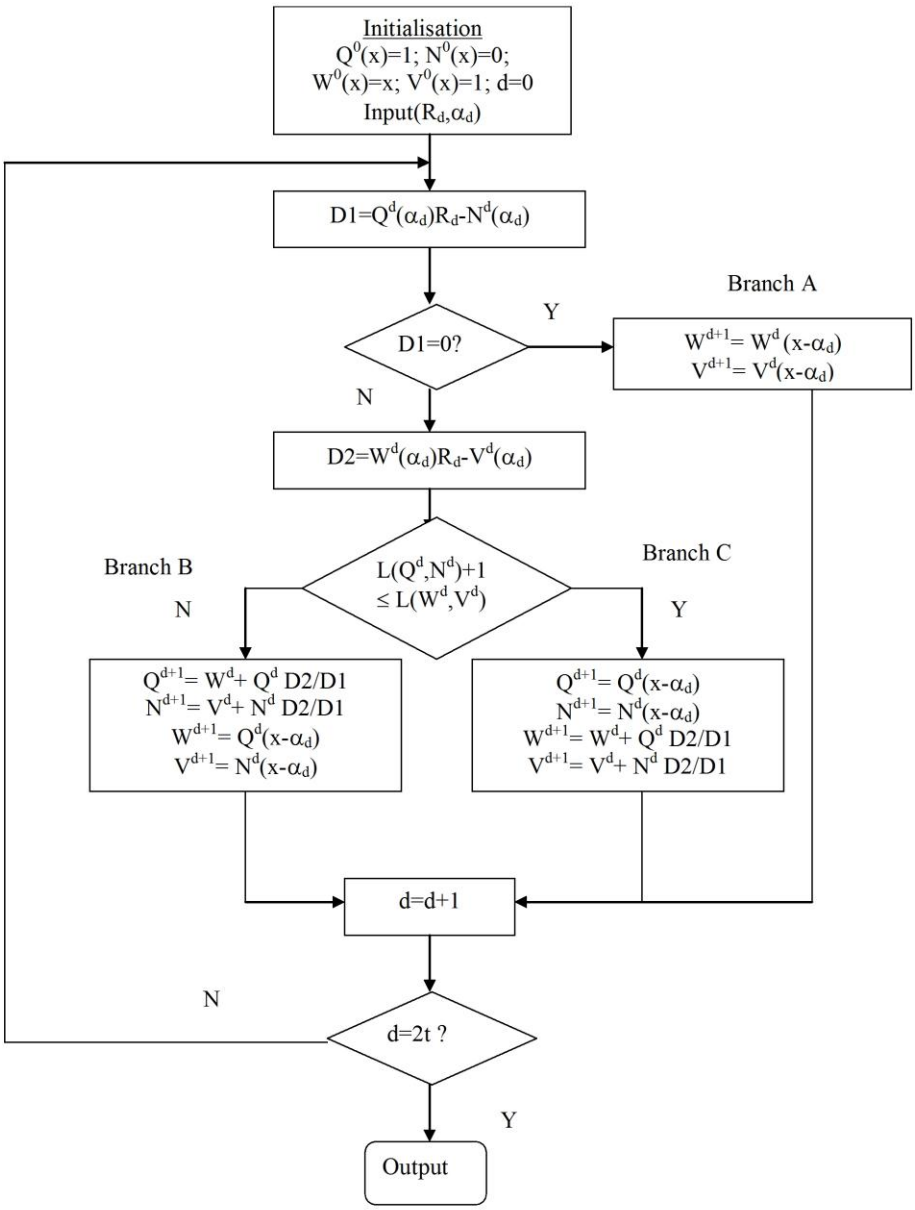

Fig. 3. The Welch Berlekamp algorithm.

The Figure 3 above can be written as shown on Listing 1.

```
1.      ---- Initialisation
2.      Q^0(x)=1; N^0(x)=0
3.      W^0(x)=x; V^0(x)=1
4.
5.      Input = (Rd, αd)
6.      for d=0 step 1 until less than 3
7.      begin loop
8.      D1= Q^d(αd)Rd- N^d(αd)

9.      if D1=0 then  % Branch A
10.     W^d+1=W^d(x-αd).        & V^d+1=V^d(x-αd)
11.     Q^d+1=Q^d.              & N^d+1=N^d
12.

13.     Else
14.     D2=[W^d(αd)Rd- V^d(αd)]

15.     If L(Q,N)+1 ≤ L(W,V) then  % Branch C
16.     W^d+1=W^d+Q^d. D2/D1  & V^d+1=V^d+N^d. D2 / D1
17.     Q^d+1=Q^d(x-αd)         & N^d+1=N^d(x-αd)
18.

19.     Else % Branch B
20.     W^d+1=Q^d(x-αd)         & V^d+1=N^d(x-αd)
21.     Q^d+1=W^d+Q^d .D2 / D1  & N^d+1=V^d+N^d. D2 / D1
22.

23.     End if
24.     End if
25.     End of loop
```

Fig. 4. Pseudocode Welch Berlekamp algorithm.

4.2 Modified Welch-Berlekamp algorithm

Then, if variable **J = L(W ^{d},V ^{d}) - L(Q ^{d},N ^{d})** is defined. The modification process of Figure 4 above Listing 1 above can be explained as follows:

Line 4

As the polynomial $Q^0(x)=1$ and $N^0(x)=0$ then $L(Q^0,N^0) = 0$. The polynomial $W^0(x) = x$ and $V^0(x) = 1$ then $L(W^0,V^0) = 1$. Hence

$L(W^0,V^0) - L(Q^0,N^0) = 1$

$J = 1$.

```
4.  J = 1
```

Line 12

As in line 10, it means that

$L(W^{d+1},V^{d+1}) = L(W^{d},V^{d}) + 1$

As in line 11, it means that

$L(Q^{d+1},N^{d+1}) = L(Q^{d},N^{d})$

Hence J = J + 1

9.	if D1=0 then % Branch A	
10.	$W^{d+1}=W^{d}(x-\alpha_d)$	& $V^{d+1}=V^{d}(x-\alpha_d)$
11.	$Q^{d+1}=Q^{d}$	& $N^{d+1}=N^{d}$
12.	**J = J+1**	

Line 18

As in line 16, it means

$L(W^{d+1},V^{d+1}) = L(W^{d},V^{d})$

As in line 17, it means

$L(Q^{d+1},N^{d+1}) = L(Q^{d},N^{d})+1$

Hence J = J - 1

15.	If $L(Q,N)+1 \leq L(W,V)$ then % Branch C	
16.	$W^{d+1}=W^{d}+Q^{d}.D2/D1$	& $V^{d+1}=V^{d}+N^{d}.D2/D1$
17.	$Q^{d+1}=Q^{d}(x-\alpha_d)$	& $N^{d+1}=N^{d}(x-\alpha_d)$
18.	**J = J – 1**	

Line 22

As seen on line 15, the condition on Branch C is $L(Q,N)+1 \leq L(W,V)$,

it can be shown that the condition on Branch B is $L(W,V) < L(Q,N)+1$ which means $L(W,V) = L(Q,N)$.

Then $L(Q^{d+1},N^{d+1}) = L(Q^{d},N^{d})$ because of line 21 and also $L(W^{d+1},V^{d+1}) = L(W^{d},V^{d}) +1$ because of line 20.

Hence J = 1.

19.	Else % Branch B	
20.	$W^{d+1}=Q^{d}(x-\alpha_d)$	& $V^{d+1}=N^{d}(x-\alpha_d)$
21.	$Q^{d+1}=W^{d}+Q^{d}.D2/D1$	& $N^{d+1}=V^{d}+N^{d}.D2/D1$
22.	**J = 1**	

Hence, the modified Welch Berlekamp algorithm can be shown as on Figure 5 and also on Figure 6.

The Welch Berlekamp algorithm output will be Q(x) and N(x) polynomials, the error locator polynomial and the error evaluation polynomial consecutively. These polynomials are able to correct up to t errors at the received signal. It will correct up to two errors, in the case of Q(x) and N(x) polynomials. Then, the Q(x) polynomial will be solved using a Chien-Search to find the root of Q(x).

1. ---- Initialisation
2. $Q^0(x)=1$; $N^0(x)=0$
3. $W^0(x)=x$; $V^0(x)=1$
4. J = 1
5. Input = (R_d, α_d)
6. for d=0 step 1 until less than 2t-1
7. begin loop
8. D1= $Q^d(\alpha_d)R_d$- $N^d(\alpha_d)$
9. if D1=0 then % Branch A
10. $W^{d+1}=W^d(x-\alpha_d)$ & $V^{d+1}=V^d(x-\alpha_d)$
11. $Q^{d+1}=Q^d$ & $N^{d+1}=N^d$
12. J = J + 1
13. Else
14. D2=[$W^d(\alpha_d)R_d$- $V^d(\alpha_d)$]
15. If L(Q,N)+1 ≤ L(W,V) then % Branch C
16. $W^{d+1}=W^d+Q^d$ .D2/D1 & $V^{d+1}=V^d+N^d$ .D2/D1
17. $Q^{d+1}=Q^d(x-\alpha_d)$ & $N^{d+1}=N^d(x-\alpha_d)$
18. J = J - 1
19. Else % Branch B
20. $W^{d+1}=Q^d(x-\alpha_d)$ & $V^{d+1}=N^d(x-\alpha_d)$
21. $Q^{d+1}=W^d+Q^d$ .D2/D1 & $N^{d+1}=V^d+N^d$ .D2/D1
22. J = 1
23. End if
24. End if
25. End of loop

Fig. 5. Pseudocode modified Welch Berlekamp algorithm

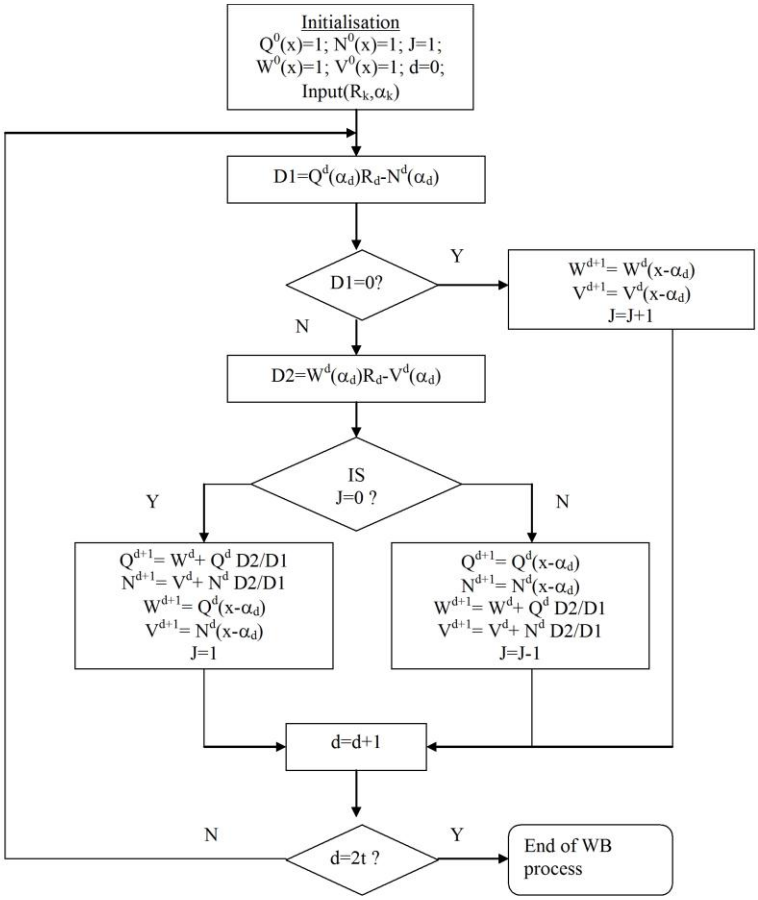

Fig.6. Modified Welch Berlekamp algorithm.

4.3 Chien search and error evaluation

Chien search [1, 3] is then used to find the root of $Q(x)$ polynomial. A Horner's rule is normally used to perform Chien Search (where $Q(x)$ is a polynomial of degree d, $d \le t$)

$$Q(x) = \sum_{i=0}^{d} q_i x^i = (\ldots(q_d x + q_{d-1})x + \ldots)x + q_0 \tag{5}$$

This process requires at most t multipliers and it takes at most tm clock cycles. In the case of RS(63, 59), this Chien search process requires two multipliers (since, $t = 2$) and it takes 12 clock cycles to complete the process. However, the Chien search can be modified as follows

$$Q(x) = q_d x^d + q_{d-1} x^{d-1} + \ldots + q_0 \tag{6}$$

Define $Q_l(x) = q_l x^l$ for $0 < l \le d$. Then

$$Q(x) = Q_d(x) + Q_{d-1}(x) + \ldots + q_0 \tag{7}$$

and it can be shown that

$$Q(\alpha^p) - Q_d(\alpha^{p+1})\alpha^{-d} + Q_{d-1}(\alpha^{p+1})\alpha^{1-d} + \ldots + q_0 \tag{8}$$

This modified Chien search requires *d* multipliers; however, whatever degree of the polynomial, it only takes *m* clock cycles. In the case of RS(63, 59), this modified Chien search needs two multipliers, but it only takes six clock cycles, whatever degree of the polynomial. Similar process is applied to the $N(x)$ polynomial.

If $Q(\alpha^g) = 0$ then α^g is the root of the error locator polynomial $Q(x)$. Then the error value E_g to the symbol s_g is given by [1, 3].

$$E_g = \begin{cases} \dfrac{N(\alpha^g)}{P(\alpha^g)Q'(\alpha^g)} : \text{if } \alpha^g \notin H_2 \\ n_g + \dfrac{N'(\alpha^g)}{P'(\alpha^g)Q'(\alpha^g)} : \text{if } \alpha^g \in H_2 \end{cases} \tag{9}$$

$$P(x) = \alpha^{10}x^4 + \alpha^{28}x^3 + \alpha^{49}x^2 + \alpha^{31}x + \alpha^{16}$$

Once the error value E_g is found using the Equation (9) at location α^g, the corrected symbol in that particular position can be found by

$$c_g = E_g + s_g \tag{10}$$

5 Implementation

A VHDL design of RS(63, 59) decoder has been designed and it is then synthesized onto a Xilinx XCV600 device.

How the RS decoder works is explained in the following steps.

Step 1: First of all, the received word is first re-encoded to obtain the remainder (parity check). This step needs *nm* clock cycles. It takes 63x6 clock cycles for RS(63, 59).

Step 2: The Welch-Berlekamp algorithm to solve the key equation. The output of the Welch-Berlekamp algorithm is $Q(x)$ and $N(x)$ polynomials, the error locator polynomial and the error evaluation polynomial consecutively.

Step 3: Find the roots of $Q(x)$ polynomial using Chien search is the next step. If $Q(x) = 0$ at a certain location, the $N(x)$ polynomial is then evaluated to find the error value. It is then added to the received symbol using modulo twoaddition to doing the correction.

6 Results

The resulting RS(63, 59) decoder circuit can do up to two errors corrections. The corresponding FPGA layout is shown below in Figure 5. This design requires 16 % (1 161 out of 6 912) of the total slices inside a Xilinx XCV600 package fg680 running at clock speed up to 27.343 MHz.

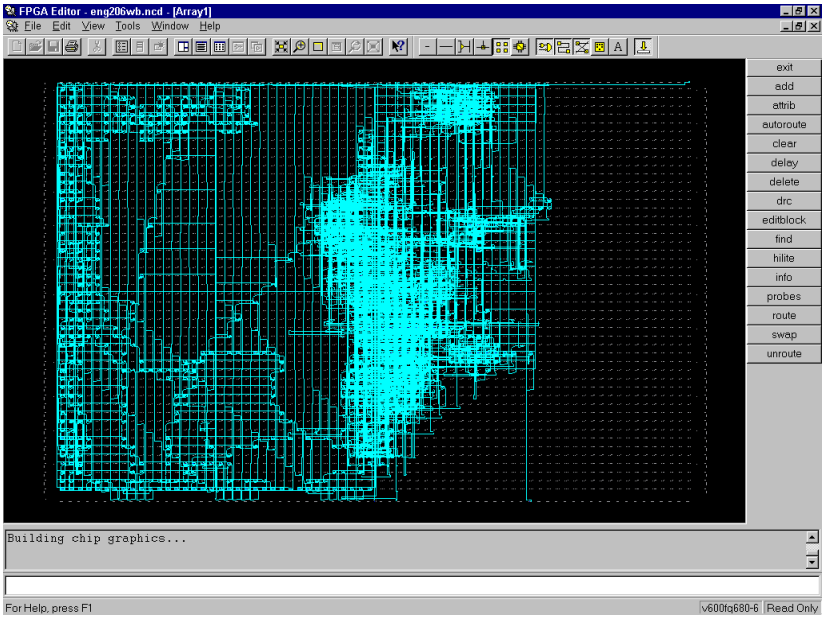

Fig. 5. FPGA layout RS(63, 59) Decoder.

7 Conclusions

The VHDL implementation of the Welch Berlekamp algorithm to decode RS code has been presented. A VHDL design RS(63, 59) has been successfully synthesized on a XILINX FPGA. The RS(63, 59) decoder has been simulated using Modelsim Altera software, and also tested for a number of error patterns. The simulation results show that the decoder works as an error only decoder and can correct up to two errors.

8 References

1. A. Mahmudi. *The investigation into generic VHDL implementation of generalised minimum distance decoding for Reed Solomon codes*. [Thesis]. University of Huddersfield, UK (2005). pp. 48–53. http://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.417302
2. A. Singh, M. Kaur. *International Journal of Innovative Research in Computer and Communication Engineering*, **1**,2:190–192 (2013). http://www.rroij.com/open-access/study-of-reed-solomon-encoder-.php?aid=42686
3. S. Czynszak. *Decoding algorithms of Reed Solomon code.* [Thesis]. Blekinge Institute of Technology, Sweden (2011). pp. 40-47. https://www.diva-portal.org/smash/get/diva2:833161/FULLTEXT01.pdf
4. P. Shrivastava, U.P. Singh. *International Journal of Advanced Research in Computer Science and Software Engineering*, **3**,8:965-969 (2013). https://pdfs.semanticscholar.org/7e94/64b704a9f4b59f9d7df9b437e1b8366b8912.pdf.
5. A.J. Han Vinck. *Coding concepts and Reed Solomon codes*. [Online] from www.martinvinck.com/page3/assets/bookHan.pdf (2013) [Accessed on 10 July 2017].
6. L.R. Welch, E.R. Berlekamp. *Error correction for algebraic block codes*. [Online] from https://www.google.com/patents/US4633470 (1986) [Accessed on 10 July 2017].
7. I.E. Richardson, *The H.264 Advanced Video Compression Standard*. 2nd Edition. UK :

Wiley Publication (2010). pp. 279. *https://books.google.co.id/books?id=k7nOAiIUo9IC&printsec=frontcover&dq=The+H.264+Advanced+Video+Compression+Standard&hl=en&sa=X&ved=0ahUKEwjix_nbz5jYAhWKYo8KHajEB54Q6AEIKDAA#v=onepage&q=The%20H.264%20Advanced%20Video%20Compression%20Standard&f=false*

8. J.P. Nguyen. *Applications of Reed Solomon codes on optical media storage.* [Thesis]. San Diego State University, California (2011). pp. 13–20. http://sdsu-dspace.calstate.edu/bitstream/handle/10211.10/1743/Nguyen_Johnny.pdf;sequence=1.
9. I. Diop, S.M. Farssi, O. Khouma, H.B. Diouf, K. Sylla. *International Journal of Distributed and Parallel Systems*, **3** (2012). https://pdfs.semanticscholar.org/b41c/9cbb3b3c13bee0d0f21c71a841271689f6da.pdf.
10. F.R. Ishengoma. *International Journal of Computer Applications*, **106**:28–31 (2014). https://arxiv.org/abs/1411.4790
11. A.N.U. Husain, Suwadi, G. Hendrantoro. *Jurnal Teknik POMITS*, **2**: A33–A38 (2013). http://ejurnal.its.ac.id/index.php/teknik/article/view/2319 [in Bahasa Indonesia]
12. J. Bhaumik, A.S. Das, J. Samanta. *International Journal of Soft Computing and Engineering*, **2**:395–399 (2013). http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=119E40402BD90D53B1B3EF68C680DC28?doi=10.1.1.301.6529&rep=rep1&type=pdf.
13. P. Sunitha, G.V. Ujwala. *International Research Journal of Engineering and Technology*, **2**:476–480 (2015). https://www.irjet.net/archives/V2/i6/IRJET-V2I676.pdf.