IMPLEMENT REED SOLOMON ENCODER/ DECODER USING SPARTAN FPGA

Manju Mangtani

*Assistant Professor, Department of Electronics and Communication,
Poornima Institute of Engineering and Technology, Jaipur, Rajasthan, India
{E-mail : manju.mangtani@poornima.org}*

Abstract

In this paper, (15, 11) reed Solomon codes have been designed and implement using Spartan field programmable gate array device. The design is carried out by writing VHDL code. The waveforms are tested using the package ISIM simulator and synthesis report and programming file are obtained using the Spartan 6. Simulation waveforms show that (15, 11) reed Solomon decoder could correct up to 2 error in given polynomial to the encoder.

Key Words : FPGA, Galois Field, RS Code, BCH Code.

1. Introduction

In the wireless communication, a wide problem is being faced out i.e. bit error (Digital Communication). During the receiving operation some amount of error is being incorporated with the received data stream. But at the receiver during decoding process the information is not correct due to the bit errors. To overcome these errors some coding techniques is being used like Linear block coding technique, hamming code technique, Reed Solomon coding technique .these techniques are used to detect the error and correct the same bit error to provide the correct information or data stream. We have found after reviewing REED-SOLOMON Codes are the best forward error correcting code. These codes are currently used in wide variety of application starts from satellite communication to data storage system.

The error correction system used on CD and DVD is based on REED SOLOMON codes. REED SOLOMON code, in which redundant information is added to data so that it can be used to recover errors in transmission or storage and retrieval. The error correction system used on CD and DVD is based on the REED SOLOMON code. A REED SOLOMON Code is a linear code (adding two code words produce another codeword) and it is a cyclic code (cyclically shifting the symbols of codeword produce another codeword). It belongs to the family of BCH codes, but it is distinguish by having multi bit symbols. This makes the code particularly good at dealing with burst off error because, although a symbol may have all its bits in error, this count as only one symbol error in terms of correction capacity of the code.[5]

Fig.1 : Bit in Reed-Solomon Codes

2. System Models

The system model of REED SOLOMON Code is given below as in Fig.2

Fig. 2 : System model of Reed-Solomon Code[2]

It consists of three major blocks which are

1. Reed-Solomon Encoder
2. Communication Channel
3. Reed-Solomon Decoder

The Reed-Solomon encoder takes a block of digital data and adds extra “redundant” bits. Errors occur during the transmission of bits or storage of information by noise or

interference, scratches on a CD. A Reed-Solomon code is specified as RS (n, k) with s-bit symbols. This means that the encoder takes k data symbols of s bits each and adds parity symbols to make an n symbol codeword. There are n-k parity symbols of s bits each.

The Reed-Solomon decoder tries to correct errors for each codeword. Based upon the syndromes the decoder is able to determine the number of errors in the received block. If there are errors present, the decoder tries to find the locations of the errors using the berlekamp-massey algorithm by creating an error locator polynomial. A Reed-Solomon decoder can correct up to t symbols that contain errors in a codeword, where $2t = n - k$.

3. System Description

3.1 FPGA Implementation of Reed-Solomon Encoder

(15, 11) reed Solomon codes has been implemented using SPARTAN FPGA. This code could correct up to 2 errors. The word length in each symbol is 4 bits and codes are based on Galois field 16 elements respectively. The code generator polynomial for the code is:

$$G(x) = x^4 + 15x^3 + 3x^2 + x + 12$$

The structure of implemented (15, 11) Reed-Solomon encoder is shown in fig (3). All data paths in fig.3 provide for 4 bit values. The 4 storage elements are 4 bit registers, labeled a0 through to a3. These registers are called as parity registers. The circuit performs polynomial division of the message polynomial, m(x) by the field generator polynomial, G(x). The remainder of the division, b(x) is stored in the a₀ to a₃ parity registers. The code word c(x) is the concatenation of message polynomial m(x) followed by remainder polynomial b(x).

Operation:

Initially the registers are in 0 states. Without lack of generality, we assume that then message is divided equally into m-bits words, where each word is now considered a Galois field element, where m is the degree of the field generator polynomial. Each m-bit word is then associated with an increasing power of x, starting with x^{k-1} and ending with $x^0 = 1$, thus forming a polynomial, m(x) over GF (2m). The message polynomial is of degree k-1, that is, there are k coefficients. Each coefficient enters the circuit one coefficient every clock cycle, with the most significant coefficient entering first. For the first k clock cycles, corresponding to when the message is entering and the remainder are being calculated, switches 1 and 2 are in position B. the encoded data thus corresponds with the message polynomial for the first k clock cycles. During these first k clock cycles, the remainder is being calculated in the registers. When the message has finished entering into the encoder, switches 1 and 2 are set to position A. Since the output of switch 1 is 0, the resulting

multiplications are 0, and the resulting additions, and consequently the inputs to the registers are just the value of the preceding register. In the case of the first register, its input is 0. Since it sees the output of the switch, the output of switch 2 is now the output of the last register. Thus the entire remainder polynomial, b(x) is shifted out one element at a time. Once all the register contents have been shifted out, the codeword generation is complete.[7]

Fig. 3 : (15, 11) Reed-Solomon Encoder [8]

3.2 FPGA Implementation of Reed Solomon Decoder

Fig.4 shows the block diagram of the implemented reed-solomon decoder. In this fig., the first process is to calculate the syndrome value from the incoming codeword polynomial. For a t- error correcting reed Solomon code, there are 2t syndromes that must be calculated. Another crucial step in the decoding process is to find the location of the errors in the received codeword. these codeword is then used to evaluate the coefficients of the error locator Polynomial $\Lambda_1 \dots \dots \Lambda_v$ using berlikamp algorithm. An iterative process, called the chien search is the most efficient means of doing this. The error locations are identified by the chien search and the error values are calculated using forney's method. As these calculations involve all the symbols of the received codeword, it is necessary to store the message until the results of the calculations are available. Then, to correct the errors, each error values are added (modulo-2) to the appropriate location in the received codeword.

Fig. 4 : Double error correcting RS decoder block diagram[11]

3.2.1 Syndrome Calculation

The syndrome calculation is the first step in decoding process. For a t -error correcting reed Solomon code, there are $2t$ syndromes that must be calculated. They are calculated as follows:

$s(j) = r(\alpha^{m_0+j})$ where $j = \{0, \dots, 2t-1\}$ in the equation above, $r(x)$ is the received codeword polynomial and m_0 is the log of the initial root of the code generator polynomial. Since the received codeword polynomial is the sum of the message polynomial and

error polynomial. We can reduce this to $s(j) = e(\alpha^{m_0+j}) + e(\alpha^{m_0+j}) = \sum_{t=0}^t Y_t * X_t^{1+m_0}$, where $j = \{0, \dots, 2t-1\}$

Where the Y_t are the error values at the error locations X_t .

Original Massey- Berlekamp Algorithm

Consider Reed-Solomon code capable of correcting up to ' t ' errors and without loss of generality, let $m_0 = 0$. The original Massey-Berlekamp algorithm is defined as follows.[7] first initialization are made, let :

$$\Lambda(x)^{(0)} = 1, B(x)^{(0)} = 1, \Gamma(x)^{(0)} = 0, A(x)^{(0)} = x^{-t}, L^{(0)} = 0$$

Where

$\Lambda(x)$ = The Error Locators Polynomial,
 $B(x)$ = The Error- Locator Support Polynomial,
 $\Gamma(x)$ = error- evaluator polynomial,
 $A(x)$ = error-evaluator support polynomial.
 L = integer variable.

The algorithm proceeds iteratively, and the superscripts define the iteration level.

Let the syndromes be represented by the syndrome polynomial:

$$S(x) = \sum_{j=0}^{2t-1} S_j \cdot x^j$$

The algorithm iterates for $2t$ steps. At the $(k+1)^{st}$ step, calculate the following term:

$$\Delta^{(k+1)} = \sum_{j=0}^{L^{(k)}} \Lambda_j^{(k)} S_{k-j}$$

Let

$$\Lambda(x)^{(k+1)} = \Lambda(x)^{(k)} - \Delta^{(k+1)} \cdot B(x) \cdot x$$

$$\Gamma(x)^{(k+1)} = \Gamma(x)^{(k)} - \Delta^{(k+1)} \cdot A(x) \cdot x$$

If $\Delta^{(k+1)} = 0$ or $2L^{(k)} > k$ then,

$$B(x)^{(k+1)} = x \cdot B(x)^{(k)}$$

$$A(x)^{(k+1)} = x \cdot A(x)^{(k)}$$

$$L^{(k+1)} = L^{(k)}$$

Otherwise

$$B(x)^{(k+1)} = \frac{\Lambda(x)^{(k)}}{\Delta^{(k+1)}} \quad (2)$$

$$A(x)^{(k+1)} = \frac{\Gamma(x)^{(k)}}{\Delta^{(k+1)}} \quad (3)$$

$$L^{(k+1)} = k+1-L^{(k)}$$

Galois field inversion is required in step (2) and (3), however it is the inversion of the same value, $\Delta^{(k+1)}$. Thus only one inverter is needed.

3.2.2 Chien Search for Error Positions

Another crucial step in the decoding process is to find the location of the errors in the received codeword. An iterative process, called the chien search is the most efficient means of doing this. Consider the error locator polynomial, $\Lambda(x)$, which is a t degree polynomial whose roots are the errors locations, X_t .

$$\Lambda(x) = \prod_{t=1}^t (x - X_t) = x^t + \Lambda_{t-1}x^{t-1} + \dots + \Lambda_1x + \Lambda_0$$

Now, it can be shown, that the coefficient of the error-locator polynomial is related to the syndrome values, $S(j)$, through a set of relations called Newton's identities:

$$S(t+j) + \Lambda_1 S(t+j-1) + \dots + \Lambda_t S(j) = 0 \quad \forall j$$

The circuit shown in fig 5 will implement such a search. One at a time (per clock cycle). [6] Initially, the values of the registers are loaded with the coefficients of the error locator polynomial. A sum of the registers is formed, resulting in the chien sum, and if it is 0, then an error has been located.

Since the received message polynomial, $r(x)$ is received one symbol at a time, starting with the symbol for the first position, a pipelined approach to the decoder is possible.

4. Top Level Design

The performance of the encoder described above was verified with a VHDL test bench. The test bench has the structure shown in fig 6. The clock and stimulus generator provides the 4 input signals to the encoder. Fig 7 shows a typical encoder and decoder simulation, showing inputs, outputs, identical control signals.

Fig. 5 : Top level design Files of Reed-Solomon Encoder

Fig. 6 : Simulation waveforms for (15, 11) Reed-Solomon Code Encoder

(a)

(b)

Fig. 7 : Top level design File of Reed-Solomon Decoder

Fig. 8 : Simulation waveforms for (15, 11) Reed-Solomon Code Decoder

5. Synthesis Reports

There is a large number of synthesis reports (hardware and software reports) obtained from synthesis operation. They describe all what concern the implementation process like storage resources required, I/O resources required, computation resources required, time delay at different points inside the chip ...etc. [6]. Table (1) shows the summary of hardware synthesis reports for (15, 11) Implemented Reed-Solomon encoder/decoder.

Table 1 : The Summary Of Hardware Synthesis Reports for (15, 11) Implemented Reed-Solomon Encoder

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slice Registers	33	54576	0%
Number of Slice LUTs	42	27268	0%
Number of fully used LUT-FF pairs	33	42	78%
Number of bonded IOBs	16	296	5%
Number of BUFG/BUFGCTRLs	1	16	6%

Table 2 : The Summary of Hardware Synthesis Reports for (15, 11) Implemented Reed-Solomon Encoder

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slice Registers	211	10224	1%
Number of Slice LUTs	206	9112	3%
Number of fully used LUT-FF pairs	140	349	42%
Number of bonded IOBs	15	232	6%
Number of BUFG/BUFGCTRLs	1	16	6%

6. Conclusions

Reed-Solomon codes are block-based error correcting codes with a wide range of applications in digital communications and storage. The usage of FPGA Technology to implement these codes provides many advantages like efficient reconfigurability and universal chip implementation. The design procedure using FPGA Technology is done by writing a hardware description programs (using language like VHDL) to each element in the system and a main program to control the influence of signals in the system. A schematic top-level design is then required to specify pins needed for real hardware interfacing. After correct compilation, obtaining timing analysis, synthesis reports and simulation, a programming file is finally downloaded to the FPGA board.

Reference

- [1] Joaquin Garcia, Rene Cumplido Department of Computer Science , “On the design of an FPGA-Based OFDM modulator for IEEE 802.16-2004” INAOE, Puebla, Mexico 2005 International Conference on Reconfigurable Computing and FPGAs (ReConFig 2005).
- [2] K. Harikrishna, T. Rama Rao, and Vladimir A. Labay, “FPGA Implementation of FFT Algorithm for OFDM Based IEEE 802.16d (Fixed WiMAX) Communications” Journal Of Electronic Science And Technology, Vol. 8, No. 3, September 2010 .
- [3] Miljko Bobrek, Kenyon H. Clark, Austin P. Albright “FPGA Implementation of Reed-Solomon Decoder for IEEE 802.16 WiMAX Systems using Simulink-Sysgen Design Environment” Oak Ridge National Laboratory Cognitive Radio Program 1 Bethel Valley Rd Oak Ridge ,TN 37831.
- [4] M.A. Mohamed, A.S. Samarah, M.I. Fath Allah , “Implementation of the OFDM Physical Layer Using FPGA” IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 2, No 2, March 2012.
- [5] Yuval Cassuto, Jehoshua Bruck, Robert J. McEliece “On the Average Complexity of Reed–Solomon List Decoders” IEEE Vol. 59, No. 4, April 2013.
- [6] Li Chen et.al. “Progressive Algebraic Soft-Decision Decoding of Reed-Solomon Codes” IEEE Vol. 61, No. 2, February 2013.
- [7] Yeow Meng Chee et.al. “Estimates on the Size of Symbol Weight Codes” IEEE Vol. 59, No. 1, January 2013.
- [8] Sian-Jheng Lin, Wei-Ho Chung “An Efficient (n, k) Information Dispersal Algorithm for High Code Rate System over Fermat Fields” IEEE Vol. 16, No. 12, December 2012.
- [9] F. Abdelkefi, J. Ayadi “Reed-Solomon code-based sparse channel estimation for OFDM systems” IEEE Vol. 48 27th September 2012.
- [10] Li Li et. al. “Unified Architecture for Reed-Solomon Decoder Combined With Burst-Error Correction” IEEE Vol. 20, No. 7, July 2012.
- [11] Mario Huemer et.al. “On the Exploitation of the Redundant Energy in UW-OFDM: LMMSE versus Sphere Detection” IEEE Vol. 19, No. 6, June 2012.
- [12] Xinmiao Zhang, Yu Zheng “Systematically Re-encoded Algebraic Soft-Decision Reed–Solomon Decoder” IEEE Vol. 59, No. 6, June 2012.
- [13] J.-I. Park, H. Lee “Area-efficient truncated Berlekamp-Massey architecture for Reed-Solomon decoder” 17th February 2011.
- [14] Mostafa El-Kham et.al. “Iterative Algebraic Soft-Decision List Decoding of Reed–Solomon Codes” IEEE Vol. 24, No. 3, March 2006
- [15] Kenny Chung Chung Wai, Dr. Shanchieh Jay Yang” Field Programmable Gate Array Implementation of Reed-Solomon Code, RS(255,239)” In proceeding of 9th Annual Military and Aerospace programmable logic Device International Conference, 2 September 2011.
- [16] Joaquin Garcia, Rene Cumplido Department of Computer Science, “On the design of an FPGA-Based OFDM modulator for IEEE 802.16-2004” INAOE, Puebla, Mexico 2005.