VISVESVARAYA TECHNOLOGICAL UNIVERSITY
Jnana Sangama, Belagavi, Karnataka-590018

Project Report on

Implementation of Reed Solomon Encoder

Submitted in Partial fulfilment of requirement for the award of degree of

BACHELOR OF ENGINEERING
in
ELECTRONICS & COMMUNICATION ENGINEERING

Submitted by

SUKRUTH S	**(1BI15EC164)**
VISHAL MARUTHI	**(1BI15EC185)**
SYED SAQLAIN AHMED	**(1BI15EC167)**

Carried Out at

U.R. RAO SATELLITE CENTRE
Department of Space, ISRO
Bangalore

Under the Guidance of

Dr. K V PRASAD	**Mr. SOMASUNDARAM S**
Professor and HOD	**Scientist-SF**
Department of ECE	**PDMG Division**
BIT, Bangalore	**Bangalore**

Department of Electronics & Communication Engineering
Bangalore Institute of Technology
K.R. Road, V.V. Puram, Bengaluru-560004
2018-2019

BANGALORE INSTITUTE OF TECHNOLOGY

K.R. Road, V.V Pura, Bengaluru -560004

Phone: 26613237/26615865, Fax:22426796

www.bit-bangalore.edu.in

Department of Electronics and Communication Engineering.

CERTIFICATE

Certified that the project work entitled **Implementation of Reed Solomon Encoder** carried out by **Mr. Sukruth S (1BI15EC164), Mr. Vishal Maruthi (1BI15EC185), Mr. Syed Saqlain Ahmed (1BI15EC167)** bonafide students of **Bangalore Institute of Technology** in partial fulfilment for the award of **Bachelor of Engineering** in **Electronics and Communication Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2018-2019. It is certified that all corrections / suggestions indicated for Internal Assessment have been incorporated in the Report deposited in the departmental library. The Project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said Degree.

Signature of Guide	Signature of HOD	Signature of Principal
-------------------------------	--------------------------------	-------------------------------
Dr. K.V. Prasad	**Dr. K.V. Prasad**	**Dr. Aswath. M. U**
Professor & HOD	Professor & HOD	Principal, BIT
Dept. of ECE, BIT	Dept. of ECE, BIT	

External Viva

Name of the examiners	Signature with date
1.	
2.	

ABSTRACT

Reed-Solomon error correction has several applications in satellite communication, in particular, forming part of the specification for the CCSDS digital communication standard.

Hardware implementation of coders and decoders for Reed-Solomon error correction are complicated and require some knowledge of the theory of the Galois fields on which they are based. This report describes the underlying mathematics and the algorithms used for coding and decoding, with particular emphasis on their realisation in logic circuits. Worked examples are provided to illustrate the processes involved.

The FPGA implementations of the respective Reed Solomon encoders and decoders were also done to understand the output and compare results of hardware and software. Despite the general nature of the topic taken this report is meant to give an approach to the hardware implementation of Reed Solomon encoder and verify the same using the developed c code.

Keywords: Digital television, error-correcting codes, CCSDS, hardware implementation,
Galois field arithmetic

ACKNOWLEDGEMENT

ISRO is one of the premier institutions of our country in the space technology sector. We are glad to have been part of this institution which enriched my technical knowledge in space technology.

We thank **PDMG, URSC** for giving us an opportunity to be a part of their team and to know about the various aspects of their division.

We express our sincere gratitude to Mr **Lalith Krishna**, Scientist-SG, Data processing and application Division, URSC and to our guide Mr **Somasundaram S**, Scientist-SF, Data processing and application Division, URSC for their support. In spite of their busy schedule and crucial work, they spent time to interact and share vital knowledge, both academic and life skills with the trainees.

We would like to express our sincere thanks to our internal guide **Dr K.V. PRASAD**, Professor and HOD, Dept. of ECE, BIT, Bangalore for providing constant support and guidance. We also thank **Dr C.R BYRAREDDY**, Professor, B.E Project Coordinator, Dept. of ECE, BIT, Bangalore for motivation and regular source of encouragement. We would like to express our gratitude to **Dr K.V. PRASAD**, HOD, Dept. of ECE, BIT, Bangalore for his valuable suggestions. We immensely thank **Dr ASWATH. M. U**, Principal, BIT, Bangalore for his encouragement.

It would be our privilege to express our heartfelt gratitude and respect to the Bangalore Institute of Technology, which has given us an opportunity to present this Project report.

SUKRUTH S	**(1BI15EC164)**
VISHAL MARUTHI	**(1BI15EC185)**
SYED SAQLAIN AHMED	**(1BI15EC167)**

DECLARATION

We hereby declare that the project report entitled “**Implementation of Reed Solomon encoder”** is a bonafide record of the project carried out at **Bangalore Institute of Technology** in partial fulfilment of the requirements for the award of degree **Bachelor of Engineering** in **Electronics and Communication Engineering of Visvesvaraya Technological University, Belagavi** during the academic year 2018-2019. We further declare that the project report is not submitted to any other universities in fulfilment of the requirements for the award of any degree.

SUKRUTH S	**(1BI15EC164)**
VISHAL MARUTHI	**(1BI15EC185)**
SYED SAQLAIN AHMED	**(1BI15EC167)**

Contents

LIST OF FIGURES AND TABLES

Chapter 1

INTRODUCTION

1.About Indian Space Research Organisation (ISRO)

India decided to go to space when Indian National Committee for Space Research (INCOSPAR) was set up by the Government of India in 1962. With the visionary Dr Vikram Sarabhai at its helm, INCOSPAR set up the Themba Equatorial Rocket Launching Station (TERLS) in Thiruvananthapuram for upper atmospheric research.

Indian Space Research Organisation, formed in 1969, superseded the erstwhile INCOSPAR. Vikram Sarabhai, having identified the role and importance of space technology in a Nation's development, provided ISRO the necessary direction to function as an agent of development. ISRO then embarked on its mission to provide the Nation space-based services and to develop the technologies to achieve the same independently.

Throughout the years, ISRO has upheld its mission of bringing space to the service of the common man, to the service of the Nation. In the process, it has become one of the six largest space agencies in the world. ISRO maintains one of the largest fleets of communication satellites (INSAT) and remote sensing (IRS) satellites, that cater to the ever-growing demand for fast and reliable communication and earth observation respectively. ISRO develops and delivers application specific satellite products and tools to the Nation: broadcasts, communications, weather forecasts, disaster management tools, Geographic Information Systems, cartography, navigation, telemedicine, dedicated distance education satellites being some of them.

To achieve complete self-reliance in terms of these applications, it was essential to develop cost efficient and reliable launch systems, which took shape in the form of the Polar Satellite Launch Vehicle (PSLV). The famed PSLV went on to become a favoured carrier for satellites of various countries due to its reliability and cost efficiency, promoting unprecedented international collaboration. The Geosynchronous Satellite Launch Vehicle (GSLV) was developed keeping in mind the heavier and more demanding Geosynchronous communication satellites.

Apart from technological capability, ISRO has also contributed to science and science education in the country. Various dedicated research centres and autonomous institutions for remote sensing, astronomy and astrophysics, atmospheric sciences and space sciences in general function under the aegis of Department of Space. ISRO's own Lunar and interplanetary missions along with other scientific projects encourage and promote science education, apart from providing valuable data to the scientific community which in turn enriches science.

Future readiness is the key to maintaining an edge in technology and ISRO endeavours to optimise and enhance its technologies as the needs and ambitions of the country evolve. Thus, ISRO is moving forward with the development of heavy lift launchers, human spaceflight projects, reusable launch vehicles, semi-cryogenic engines, single and two stage to orbit (SSTO and TSTO) vehicles, development and use of composite materials for space applications etc.

Vikram Sarabhai Space Centre (VSSC)

Vikram Sarabhai Space Centre (VSSC), Thiruvananthapuram, is the lead centre of ISRO responsible for the design and development of launch vehicle technology. The Centre pursues active research and development in the fields of aeronautics, avionics, materials, mechanisms, vehicle integration, chemicals, propulsion, space ordnance, structures, space physics and systems reliability. The Centre undertakes crucial responsibilities of design, manufacturing, analysis, development and testing related to the realisation of subsystems for the different missions. These are sustained by activities towards programme. Planning and evaluation, technology transfer, industry coordination, human resources development and safety. Planning, execution and maintenance of all civil works related to the Centre is also carried out. The Centre depends on administrative and auxiliary services for support.

U R Rao Satellite Centre (URSC)

U R Rao Satellite Centre (URSC), Bengaluru, formerly known as ISRO Satellite Centre (ISAC) is the lead centre for building satellites and developing associated satellite technologies. These spacecrafts are used for providing applications to various users in the area of Communication, Navigation, Meteorology, Remote Sensing, Space Science and interplanetary explorations. The Centre is also pursuing advanced technologies for future missions. URSC is housed with the state-of-the-art facilities for building satellites on end-to-end basis. ISRO Satellite Integration and Test Establishment (ISITE) is equipped with state-of-the-art clean room facilities for spacecraft integration and test facilities including a 6.5 Metre thermo vacuum chamber, 29 Ton vibration facility, Compact Antenna Test Facility and acoustic test facility under one roof. Assembly, Integration and Testing of all Communication and Navigation Spacecraft is carried out at ISITE. A dedicated facility for the product ionisation of standardised subsystems is established at ISITE.

URSC has a unit called Laboratory for Electro Optics System (LEOS), which is situated in Peenya, Bengaluru and is mainly responsible for research, development and product ionisation of Sensors for ISRO programmes.
Since inception, URSC has the distinction of building more than 100 satellites for various applications like scientific, communication, Navigation and remote sensing. Many private and public sector industries are also supporting ISAC in realising standard satellite hardware.

Payload Data Management Group (PDMG)

Payload Data Management Group (Data Handling Division and Data Storage Division) is responsible for delivering Data Handling and Storage Systems for all IRS missions, located in URSC. In addition to that, Group is also responsible for building test beds for testing various flight packages for different missions. Also, the group supports encoding systems for Satellite commanding across various ground stations. Group is also actively pursuing the development of high-end technologies required to meet future requirements.

As the number of Data handling and storage systems to be delivered for the upcoming IRS Missions increased significantly, to meet schedule requirements augmentation of manpower is necessary. In the present scenario, to carry out all the routine project activities without compromising on the development of advanced technologies required for future satellites is becoming increasingly difficult. Currently routine activities like testing, are being carried out with the help of 4 to 5 contract engineers. This support is no longer available. To meet the existing immediate project requirements additional manpower support is sought through service contract wherein the service provider will provide service by way of assistance in testing of electronics systems, technical documentation, design, simulations etc. on a need basis. Currently, Payload Data Management Group is working on more than 22 Projects (more than 550 flight cards). This number is expected to increase in the nearby future. Project work includes, test system builds up, harness fabrication, test & evaluation of on-board/ground Hardware & Software, Interface tests, technical documentation etc. The design work in Group includes FPGAs, ASICs, software, PCB design etc. for on-board use and ground use.

In order to meet the increased work load and the critical schedule requirements, skilled manpower support is required for the group, the details of which are explained further in this document.

1.1 Overview

Channel coding is a signal processing technique by which data can be sent from a source to a destination through a noisy channel so that distinct messages are easily distinguishable from one another. This allows reconstruction of the data with improved reliability. In spacecraft, the data source is usually digital, with the data represented as a string of ‘zeros’ and ‘ones. A channel encoder (or simply ‘encoder’) is then a device that takes this string of binary data and produces a modulating waveform as output. If the channel code is chosen correctly for the particular channel in question, then a properly designed decoder will be able to reconstruct the original binary data even if the waveforms have been corrupted by channel noise. If the characteristics of the channel are well understood, and an appropriate coding scheme is chosen, then channel coding provides higher overall data throughput at the same overall quality (bit error rate) as encoded transmission - but with less energy expended per information bit. Equivalently, channel coding allows a lower overall bit error rate than the encoded system using the same energy per information bit. There are other benefits that may be expected from coding.

First, the resulting ‘clean’ channel can benefit the transmission of compressed data. The purpose of data compression schemes is to map a large amount of data into a smaller number of bits. Adaptive compressors will continually send information to direct a ground decompressor how to treat the data that follows. An error in these bits could result in improper handling of subsequent data. Consequently, compressed data is generally far more sensitive to communication errors than uncompressed data. The combination of efficient low error rate channel coding and sophisticated adaptive data compression can result in significant improvement in overall performance.

Second, a low bit error rate is also required when adaptive (or self-identified) telemetry is used. Adaptive telemetry is much like adaptive data compression in that information on how various ground processors should treat the transmitted data is included as part of the data. An error in these instructions could cause improper handling of subsequent data and the possible loss of much information.

Third, low error probability telemetry may allow a certain amount of unattended mission operations. This is principally because the operations systems will know that any anomalies detected in the downlink data are extremely likely to be real and not caused by channel errors. Thus, operators may not be required to try to distinguish erroneous data from genuine spacecraft anomalies. In a typical space channel, the principal signal degradations are due to the loss of signal energy with distance, and to the thermal noise in the receiving system.

1.2 Problem Statement

The main challenge in any communication system is error free data link. To overcome the errors introduced during transmission one can either adopt retransmission or error detection and correction at the receiver. Retransmission, being a more power consuming alternative, is not preferred. Instead we opt for error detection and correction. This project deals with design and implementation of two of the popularly used codes namely, Reed-Solomon codes and convolutional codes.

1.3 Objective of the Project

To design the CCSDS recommended Reed Solomon encoder and Convolutional encoder and Viterbi decoder with the following features:

- Minimal hardware
- High frequency of operation
- Less latency

1.4 Outcome of the Project

Software and hardware implementation of the following encoders and decoder were completed:

- Generalized Reed Solomon encoder for different code rates.
- Software implementation of RS encoder for CCSDS standard.

1.5 Development Specification

• Language	:	VHDL
• Simulator	:	Actel Libero IDE 9.1
• Other Tools used	:	Microsoft Visual C++ 6.0
• Family of FPGA	:	Accelerator
• Die	:	AX2000
• Package	:	352 CQFP
• Speed	:	Standard (STD)

1.6 Why FPGA?

FPGAs are an uncommitted sea of logic gates that may be reconfigured according to the application requirements. FPGAs, due to their flexible and reconfigurable hardware architecture, allow large scale parallel processing and pipelining of dataflow. Latest FPGAs provide plenty of processing resources, huge on-chip RAM and support very high clock speeds. Efficient and effective pipelining in FPGAs is possible due to high I/O capability that allows FPGAs to access multiple RAM banks simultaneously.

FPGAs use multiple distributed memory banks for partitioning and pipelining of algorithms thus bringing about significant improvements in performance compared to the processor-based implementation. The use of multiple memory banks removes a significant performance bottleneck.

1.7 Application of ECC

- Compact disc players provide a growing application area for forward error control coding (FECC).
- In CD applications the powerful Reed-Solomon code is used since it works at a symbol level, rather than at a bit level, and is very effective against burst errors.
- The Reed-Solomon code is also used in computers for data storage and retrieval.
- Digital audio and video systems are also areas in which FEC is applied.
- Error control coding, generally, is applied widely in control and communications systems for aerospace applications, in mobile (GSM).
- Cellular telephony and for enhancing security in banking and barcode reader

Chapter 2

LITERATURE SURVEY

The history of error correcting coding (ECC) started with the introduction of the Hamming codes (Hamming 1974), at or about the same time as the seminal work of Shannon (1948). Shortly after, Golay codes were invented (Golay 1974). These two first classes of codes are optimal.

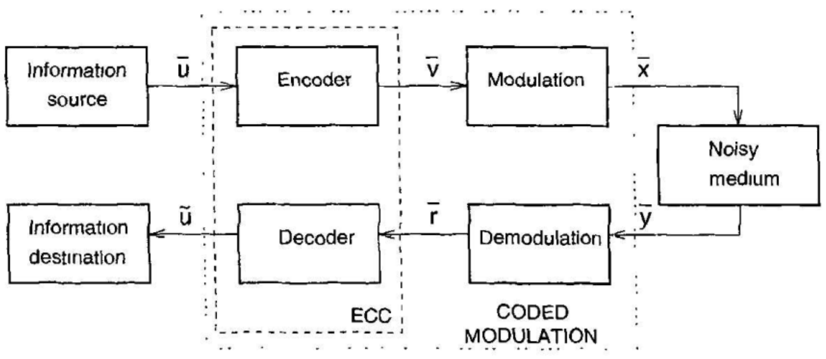


Figure 2-1: Concatenated Encoder and Decoder

Figure 2-1 shows the block diagram of a canonical digital communications/storage system. The information source and destination will include any source coding scheme matched to the nature of the information. The ECC encoder takes the input as information symbols from the source and adds redundant symbols to it, so that most of the errors- introduced in the process of modulating a signal, transmitting it over a noisy medium and demodulating it - can be corrected.

Usually, the channel is assumed to be such that samples of an additive noise process are added to the modulated symbols (in their equivalent complex base band representation). The noise samples are assumed to be independent from the source symbols. This model is relatively easy to track mathematically and includes additive white Gaussian noise (AWGN) channels, flat Rayleigh fading channels and binary symmetric channels (BSC). The case of frequency selective channels can also be included, as techniques such as spread spectrum and multicarrier modulation (MCM) effectively transform them into either AWGN channels or flat Rayleigh fading channels.

At the receiver end, the ECC decoder utilizes the redundant symbols and their relationship with the information symbols in order to correct channel errors.

In the case of error detection, the ECC decoder can be best thought of as a rs encoder of the received information, followed by a check that the redundant symbols generated are the same as those received.

In classical ECC theory, the combination of modulation, noisy medium and demodulation was modelled as a discrete memory less channel with input $\bar{v}$ and output$\bar{r}$. An example of this is binary transmission over an AWGN channel, which is modelled as a BSC. This is illustrated in fig 1.2. The BSC has a probability of channel error p- or transition probability- equal to the probability of an error for binary signalling over an AWGN channel,

$$p = Q\left(\sqrt{\frac{2E_b}{N_0}}\right) \qquad (2.1)$$

Where E_b/N_0 is the energy per bit to noise ratio also referred to as bit signal to noise ratio (SNR) or SNR per bit and

$$Q(x) = \frac{1}{\sqrt{2\pi}}\int_x^{\infty} e^{-z^2/2}\,dz \qquad (2.2)$$

Is the Gaussian Q function in terms of complementary error function, the Q function can be written as the

$$Q(x) = \frac{1}{2}\operatorname{erfc}\left(\frac{x}{\sqrt{2}}\right) \qquad (2.3)$$

The above equations are useful in analytical derivations and used in computation with C program or MATLAB scripts of performance and approximations.

Codes can be combined in several ways. An example of serial concatenation is the following. For years the most popular concatenated ECC scheme has been the combination of Reed –Solomon (RS code), through intermediate interleaving, and inner binary Convolutional codes. This scheme has been used in several applications ranging from space communication to digital broadcasting of high definition television. The basic idea is that the soft decision decoder of the Convolutional decoder produces bursts of errors that broken into smaller pieces by the de-interleaving process and handled effectively by the RS decoder .RS codes are non-binary codes that work with symbols composed of several bits and can deal with multiple bursts of errors. Serial concatenation has the advantage that it requires two separate decoders, one for inner code and one for outer code, instead of single but very complex decoder for the overall code.

2.1 Why channel coding?

Channel coding is a signal processing technique by which data can be sent from a source to a destination through a noisy channel so that distinct messages are easily distinguishable from one another. This allows reconstruction of the data with improved reliability.

$$\frac{C}{N_0} = 10\log\left(\frac{E_B}{N_0}\right) + 10\log\left(\frac{R}{B}\right) \qquad (2.4)$$

Where C is channel capacity, N_0 is noise power in watt,E_B is bit error rate, R is data rate and B is channel bandwidth in Hz.

For a given system, in order to obtain the fixed $\frac{C}{N_0}$ we must either have to increase $\frac{E_B}{N_0}$ or $\frac{R}{B}$ as noise in the channel reduces $\frac{C}{N_0}$ but in space application and mobile communication and both of these are fixed. The amount of $\frac{E_B}{N_0}$ that can be compensated by coding is called coding gain.

If the channel code is chosen correctly for the particular channel in question, then a properly designed decoder will be able to reconstruct the original binary data even if the waveforms have been corrupted by channel noise. If the characteristics of the channel are well understood, and an appropriate coding scheme is chosen, then channel coding provides higher overall data throughput at the same overall quality (bit error rate) as uncoded transmission - but with less energy expended per information bit. Equivalently, channel coding allows a lower overall bit error rate than the uncoded system using the same energy per information bit.

2.2 Error detection and error correction

When a message is transmitted or stored it is influenced by interference which can distort the message. Radio transmission can be influenced by noise, multipath propagation or by other transmitters. In different types of storage, apart from noise, there is also interference which is due to damage or contaminant sin the storage medium. There are several ways of reducing the interference. However, some interference is too expensive or impossible to remove. One way of doing so is to design the messages in such ways that the receiver can detect if an error has occurred or even possibly correct the error too. This can be achieved by Error – Correcting Coding.

In such coding the number of symbols in the source encoded message is increased in a controlled manner, which means that redundancy is introduced.

To make error correction possible the symbol errors must be detected. When an error has been detected, the correction can be obtained in the following ways: -

(1) Asking for a repeated transmission of the incorrect code word (Automatic repeat Request (ARQ)) from the receiver.

(2) Using the structure of the error correcting code to correct the error Forward Error Correction (FEC)).

It is easier to detect an error than it is to correct it. FEC therefore requires a higher number of check bits and a higher transmission rate, given that a certain amount of information has to be transmitted within a certain time and with a certain minimum error probability. The reverse is also true; if the channel offers a certain possible transmission rate, ARQ permits a higher information rate than FEC, especially if the channel has a low error rate. FEC however has the advantage of not requiring a reply channel. The choice in each particular case therefore depends on the properties of the system or on the application in which the error – correcting is to be introduced. In many applications, such as radio broadcasting or Compact Disc (CD), there is no reply channel. Another advantage of the FEC is that the transmission is never completely blocked even if the channel quality falls below such low levels that ARQ system would have completely asked for retransmission. In a system using FEC, the receiver has no Real-time contact with the transmitter and cannot verify if the data was received correctly. It must make a decision about the received data and do whatever it can to either fix it or declare an alarm. There are two main methods to introduce Error- Correcting Coding. In one of them the symbol stream is divided into block and coded. This consequently called Block Coding. In the other one a convolution operation is applied to the symbol stream. This is called Convolutional Coding.

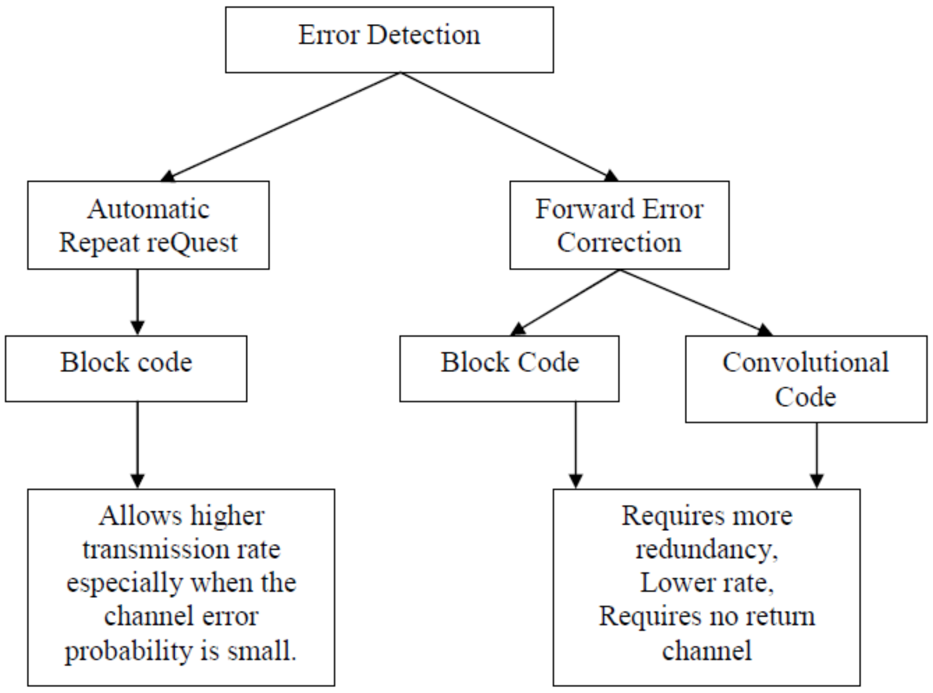

Error Detection
Automatic Repeat reQuest
Forward Error Correction
Block code
Block Code
Convolutional Code
Allows higher transmission rate especially when the channel error probability is small.
Requires more redundancy, Lower rate, Requires no return channel

Figure 2-2:Error Detection and Correction

FEC techniques repair the signal to enhance the quality and accuracy of the received information, improving system performance. Various techniques used for FEC are described in the following sections.

2.3 Error detection and correction codes

The telecom industry has used FEC codes for more than 20 years to transmit digital data through different transmission media. Claude Shannon first introduced techniques for FEC in 1948. These error-correcting codes compensated for noise and other physical elements to allow for full da-ta recovery.

For an efficient digital communication system early detection of errors is crucial in preserving the received data and preventing data corruption. This reliability issue can be addressed making use of the Error Detection and Correction (EDAC) schemes for concurrent error detection (CED).

The EDAC schemes employ an algorithm, which expresses the information message, such that any of the introduced error can easily be detected and corrected (within certain limitations), based on the redundancy introduced into it. Such a code is said to be e-error detecting, if it can detect any error affecting at most e-bits e.g. parity code, two-rail code, m-out-of-n, Berger ones etc.

Similarly, it is called e-error correcting, if it can correct e-bit errors e.g. Hamming codes, Single Error Correction Double Error Detection (SECDED) codes, Bose Choudhary – Hocquenqhem (BCH) Codes, Residue codes, Reed Solomon codes etc. As mentioned earlier both error detection and correction schemes require additional check bits for achieving CED. An implementation of this CED scheme is shown in figure below

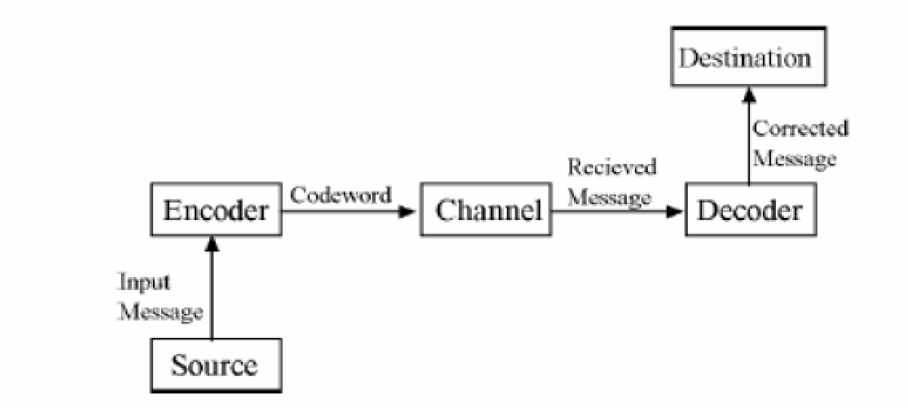


Figure 2-3:**Concurrent error detection implementations**

Here the message generated by the source is passed to the encoder which adds redundant check bits, and turns the message into a code word. This encoded message is then sent through the channel, where it may be subjected to noise and hence altered. When this message arrives at the decoder of the receiver, it gets decoded to the most likely message. If any error had occurred during its trans-mission, the error may either get detected and necessary action taken (Error Detection scheme) or the error gets corrected and the operations continue (Error Correction Scheme).

Detection of an error in an error detection scheme usually leads to the stalling of the operation in progress and results in a possible retry of some or all of the past computations. On the other hand the error correction scheme permits the process to continue uninterrupted, but usually requires higher amount of redundancy than the error detection. Each of these schemes has varied applications, de-pending on the reliability requirements of the system.

2.3.1 Terminologies and definitions used in EDAC codes

An EDAC Scheme is said to be

- **Unidirectional:** - When all components affected by a multiple error change their values in only one direction from, say, 0 to 1, or vice versa, but not both.

- **Asymmetric: -** If its detecting capabilities are restricted to a single error type (0 to 1 or 1 to 0). These codes are useful in applications which expect only single type of error during their operation.
- **Linear: -** If the sum of two code words (encoded data) is also a code word. Sum here means adding two binary block bits wise XOR.
- **Non-separable or Non-systematic**: - if the check-bits are embedded within the code word, and cannot be processed concurrently with the information, else is referred to as separable or systematic codes**.**
- **Block codes**: if the code word can be considered as a collection of binary blocks, all of the same length, say n. Such codes are characterized by the fact that the encoder accepts k information symbols from the information source and appends a set of r redundant symbols derived from the information symbols, in accordance with the code algorithm.
- **Convolutional codes:** Convolutional codes differ from block codes in that the encoder contains memory and the n encoder outputs at any time unit depend not only on the k inputs but also on m previous input blocks. An (n, k, m) convolutional code can be implemented with a k-input, n-output linear sequential circuit with input memory m.
- **Binary:** if the elements (or symbols) can assume either one of two possible states (0 and 1).
- **Cyclic:** if it is a parity check code with the additional property that every cyclic shift of the word is also a code word.
- **Forward Error Correcting**: - if enough extra parity bits are transmitted along with the original data, so as to enable the receiver to correct a predetermined maximum corrupt-ed data, without any further retransmissions.
- **Backward Error Correcting: -** if the redundancy is enough only to detect errors and re-transmission is required.

The Hamming Distance of two code words x, y $\in Fn$ denoted by d (x, y), is the minimum number of bits in which they differ.i.e., the number of 1s in a XOR b. A distance d code can detect (d-1) bit errors and correct [(d-1)/2] bit errors.

The Error Syndrome is a defined function which is used to exactly identify the error location by addition of the bad parity bit locations. It is the vector sum of the received parity digits and the parity check digits recomputed from the received information digits. A Code word is a

block of n symbols that carries the k information symbols and the r redundant symbols (n = k + r).

For a (n, k) block code, the rate of the code defined as the ratio of number of information bits to the length of code k/n.

2.3.2 Concurrent error detection schemes

Schemes for Concurrent Error Detection (CED) find wide range of applications, since only after the detection of error, can any preventive measure be initiated. The principle of error detecting scheme is very simple, an encoded code word needs to preserve some characteristic of that particular scheme, and a violation is an indication of the occurrence of an error. Some of the CED techniques are discussed below.

Parity Codes

These are the simplest form of error detecting codes, with a hamming distance of two (d=2), and a single check bit (irrespective of the size of input data). They are of two basic types: Odd and Even. For an even-parity code the check bit is defined so that the total number of 1s in the code word is always even; for an odd code, this total is odd. So, whenever a fault affects a single bit, the total count gets altered and hence the fault gets easily detected. A major drawback of these codes is that their multiple fault detection capabilities are very limited.

Checksum Codes

In these codes the summation of all the information bytes is appended to the information as b-bit checksum. Any error in the transmission will be indicated as a resulting error in the checksum. This leads to detection of the error. When b=1, these codes are reduced to parity check codes. The codes are systematic in nature and require simple hardware units.

m-out-of-n Codes

In this scheme the code word is of a standard weight m and standard-length n bits. Whenever an error occurs during transmission, the weight of the code word changes and the error gets detected. If the error is a 0 to 1 transition an increase in weight is detected, similarly 1 to 0

leads to a reduction in weight of the code, leading to easy detection of error. This scheme can be used for detection of unidirectional errors, which are the most common form of error in digital systems.

2.3.3 Concurrent Error Correction Schemes

Error-correcting codes (ECC) were first developed in the 1940s following a theorem of Claude Shannon that showed that almost error-free communication could be obtained over a noisy channel. The quality of the recovered signal will however depend on the error correcting capability of the codes.

Error correction coding requires lower rate codes than error detection, but is a basic necessity in safety critical systems, where it is absolutely critical to get it right first time itself. In these special circumstances, the additional bandwidth required for the redundant check-bits is an acceptable price.

Over the years, the correcting capability of the error correction schemes have gradually in-creased with constrained number of computation steps. Concurrently, the time and hardware cost to perform a given number of computational steps have also greatly decreased. These trends have led to greater application of these error-correcting techniques.

One application of ECC is to correct or detect errors in communication over channels where the errors appear in bursts, i.e. the errors tend to be grouped in such a way that several neighbouring symbols are incorrectly detected. Non – binary codes are used to correct such errors. Since the error is always a number different from zero in the field, it is always one in the binary codes. In a non – binary code the error can take many values and the magnitude of the error has to be determined to correct the error. Some of the non – binary codes are discussed in the following sections.

Bose – Chaudhuri – Hocquenqhem (BCH) Codes

BCH codes are the most important and powerful classes of linear block codes which are cyclic codes with a wide variety of parameters. The most common BCH codes are characterized as follows. Specifically, for any positive integer m (equal to or greater than 3) and t [less than ($2m$-1)/2] there exists a binary BCH code with the following parameters:

Block length: $n=2m-1$

Number of message bits $k\geq n-mt$

Minimum distance $dmin\geq 2t+1$

Where m is the number of parity bits and t is number of errors that can be corrected.

Each BCH code is a t – error correcting code in that it can detect and correct up to t random errors per code word. The Hamming single – error correcting codes can be described as BCH codes. The BCH codes offer flexibility in the choice of code parameters, block length and code rate.

Hamming Single Error – Correcting Codes

Hamming codes can also be defined over the non – binary field. The parity check matrix is designed by setting its columns as the vectors of GF $(p)m$ whose first non – zero element equals one. There are $n=(pm-1)/(p-1)$ such vectors and any pair of these is linearly independent. If p = 3 and r = 3, then a Hamming single error – correcting code is generated.

These codes represent a primitive and trivial error correcting scheme, which is key to the un-distending of more complex correcting schemes. Here the code words have a minimum Hamming distance 3 (i.e. d = 3), so that one error can be corrected, two errors detected. For enabling error correction, other than the error detection, the location of the error must also be identified. So for one-bit correction on an n-bit frame, if there is an error in one out of the total n bit positions, it must be identified otherwise it must be stated that there is no error. Once located, the correction is trivial: the bit is inverted.

Hamming codes have the advantage of requiring fewest possible check bits for their code lengths, but suffer from the disadvantage that, whenever more than single error occurs, it is wrongly interpreted as a single-error, because each non-zero syndrome is matched with one of the single-error events. Thus, it is inefficient in handling burst errors.

Burst Error Correcting Codes

The transmission channel could be memory less or it may be having some memory. If the channel is memory less than the errors may be independent and identically distributed. Sometimes it is possible that the channel errors exhibit some kind of memory. The most common example of this is burst errors. If a particular symbol is in error, then the chances are good that its immediate neighbours are also wrong. Burst errors occur for instance in mobile communications due to fading and in magnetic recording due to media defects. Burst errors can be converted to independent errors by the use of an interleaver.

A burst error can also be viewed as another type of random error pattern and be handled ac-accordingly. But some schemes are particularly well suited to dealing with burst errors. Cyclic codes represent one such class of codes. Most of the linear block codes are either

cyclic or are closely related to the cyclic codes. An advantage of cyclic codes over most other codes is that they are easy to encode. Furthermore, cyclic codes possess a well-defined mathematical structure called the Galois Field, which has led to the development of a very efficient decoding schemes for them.

Reed Solomon codes represent the most important sub-class of the cyclic codes.

Turbo codes

It is a convolutional code. Encoding is simple convolutional encoding. It is defined by (n, k, l) turbo code where n is the number of input bits, k is the number of output bits and 1 is the memory of the encoder. Decoding is done in two stages. First one is soft decoding stage then a hard-decoding stage. It has very good error correcting capability i.e. coding gain. The main drawback is that it has low coding rate and high latency. Hence it is not suitable for many applications. But in case of satellite communication as the latency due to the distance itself is so high this additional latency is null able. Hence it is mainly used in satellite communication

.

Low Density Parity Check Codes

Low density parity check code is a linear block code. The message block is transformed into a code block by multiplying it with a transform matrix.

Low density as the name implies low density of the transform matrix.
That means number of 1s in the transform matrix is less. It is the best code as far as the coding gain is concerned but encoder and decoder design is complex. Mainly used in digital video broadcasting.

2.4 Types of error correction coding

There are two major types of error correction schemes and they are as follows

Forward Error Correction (FEC)

In FEC method parity bits or symbols are added to the message bits or symbols at the transmitter. These parity symbols or bits are used to detect and correct the errors at the receiver.

Automatic Repeat request (ARQ)

Error control for a two-way system can be accomplished by error detection and retransmission called ARQ. In this scheme, when error is detected at the receiver a request is sent to the transmitter to retransmit the message.

After being introduced to both the approaches, one should choose whether which approach is to be used. Automatic repeat request is easier but if the error occurs much frequently, then retransmission at that frequency will particularly reduce the effective rate of data transmission. However, in some cases retransmission may not be feasible to us. In those cases, Forward Error correction would be more suitable, as Forward Error Correction involves additional information during transmission along with the actual data. It also reduces the effective data rate which is independent of rate of error. Hence, if error occurs less frequently then Automatic request approach is followed keeping in mind that retransmission is feasible.

2.4.1 Forward Error Correction (FEC)

Classification of Forward Error Correction Codes

There are several ways of classifying the forward error correction codes as per different characteristics.

• Linear vs. Non-linear- Linear codes are those in which the sum of any two valid code words is also a valid code word. In case of non-linear code, the above statement is not always true.

• Cyclic vs. Non-Cyclic- Cyclic code word are those in which shifting of any valid code word is also a valid code word. In case of non-circular code word, the above statement is not always true.

• Systematic vs. Non-systematic- Systematic codes are those in which the actual information appears unaltered in the encoded data and redundant bits are added for detection and correction of error. In non-systematic code the actual message does not appear in its original form in the code rather there exists one mapping method from the data word to code word and vice versa.

• Block vs. convolutional- The block codes are those in which one block of message is transformed into on block of code. In this case no memory is required. In case of convolutional code, a sequence of message is converted into a sequence of code. Hence encoder requires memory as present code is combination of present and past message.

• Binary vs. Non binary- Binary codes are those in which error detection and correction is done on binary information i.e. on bits. Hence after the error is located, correction means only flipping the bit found in error. In non-binary code error detection and corrections are done on symbols, symbols may be binary though.

Hence both the error location and magnitude is required to correct the symbol in error.
Out of the various block codes, Reed Solomon code is one. These are non-binary, systematic, linear block error correcting codes with wide range of applications in the field of digital communications. These codes are used to correct errors in devices such as CD’s, DVDs, etc.., wireless communications, many digital subscriber lines such as ADSL, HDSL etc…

2.4.2 Comparison between some Forward Error Detection Techniques

After getting familiarized with classification and properties of forward error correction codes some of the error detection and correction codes are explained and compared in this section.

• **Hamming Code** - In a hamming encoder parity bit are inserted into the message bits. These parity bits are decided so as to impose a fixed parity on different combinations of data and parity bits. In decoder those combinations are checked for that fixed parity. Accordingly, decoder parity bits are set. Binary equivalent of this combination decides the location of the error. Then that particular bit is flipped to correct the data. Hamming code is a single error correction code. Double errors can be detected if no correction is attempted.

• **Low Density Parity check code (LDPC)** - Low density parity check code is a linear block code. The message block is transformed into a code block by multiplying it with a transform matrix. Low density in the name implies low density of the transform matrix. That means number of 1s in the transform matrix is less. It is the best code as far as the coding gain is concerned but encoder and decoder design are complex. Mainly used in Digital Video Broadcasting.

• **Turbo Code** - It is a convolutional code. Encoding is simple convolutional encoding. It is defined by (n, k, l) turbo code where n is the number of input bits, k is the number of output bits and l is the memory of the encoder. Decoding is done in two stages. First one is soft decoding stage then a hard-decoding stage. It has very good error correcting capability i.e. coding gain. The main drawback is that it has low coding rate and high latency. Hence it is not suitable for many applications. But in case of satellite communication as the latency due to the distance itself is so high this additional latency is negligible. Hence it is used mainly in satellite communication.

• **Reed Solomon Code** - Reed Solomon code is a linear cyclic systematic non-binary block code. In the encoder Redundant symbols are generated using a generator polynomial and appended to the message symbols. In decoder error location and magnitude are calculated using the same generator polynomial. Then the correction is applied on the received code. Reed Solomon code has less coding gain as compared to LDPC and turbo codes. But it has very high coding rate and low complexity. Hence it is suitable for many applications including storage and transmission.

2.5 Mathematical Theorem's

This chapter is to get a thorough knowledge of the groups and the fields used in mathematics that will also be used further for RS encoding & decoding.

2.5.1 Groups

The most fundamental algebraic structure is a group. Following definitions elaborate it all.

Definition 4.1: A group (G; *) is a set G with a binary operation *: G*G→ G satisfying the following 3 axioms:

Associativity: For all a; b; c ϵ G: (a * b * c) = (a * b) * c = a * (b * c).

Identity element: There is an element e ϵ G such that for all a ϵ G: e * a = a * e = a.

Inverse element: For each a ϵ G, there is an element b ϵ G such that a * b = b * a = e, where e is the identity element.

As is stated, a group is simply a set of elements having a neutral and inverses, noted a^{-1} or -a depending on the situation. A group is said to be commutative, or Abelian, if for a; b ϵ G we have ab = be.

2.5.2 Fields

Fields are mathematical structures behaving with the same rules as usual arithmetic and close to our everyday intuition. A field is a set where operations like addition, subtraction, multiplication or division subject to the laws of commutativity, distributivity, associativity and other "usual" properties.

Definition 4.5: A field is a set F or F with two operations + and. such that:

(F, +) is a commutative group; (F*,.) where F* = F \ {0}, is a commutative group; the distributive law holds.

Notice that a field is a ring, by definition, with the additional property that every element has an inverse under multiplication. Some common examples of fields are R, C and Q. Indeed, every axiom is straightforward to verify. However, the set of integers Z is not a field. Indeed for (Z*, .) to be a group, any of its element should have an inverse under multiplication. This is clearly not the case since no integers have an inverse in this set except -1 and 1.

Moreover, it should be noted that no flat assumptions are made about operations like subtraction or division. These two can respectively be seen as undoing the operation, i.e. a + (-b) and a.b-1. Other familiar properties are not assumed by default but easily proved. A field is finite when it contains a finite number of elements, referred to as the size of a field and Fq denotes a field of size q. This notation turns out to be unambiguous because all fields of the same size are isomorphic (identical via a renaming of elements). One of the most common finite fields used in coding theory is the binary field encountered before.

Theorem4.3: Z= nZ is a field if and only if n is prime.

However, they are by no means the only finite fields. They are the simplest and sufficient to illustrate and understand how arithmetic in fields can be done. The key thing to remember is that because of the field axioms, elements can be unambiguously added, subtracted, multiplied and divided, in opposition to previous algebraic structures covered. By taking the alphabet as a field, say F, then Fn is a vector space (which is itself a group). This vector space structure is practical and will help us further explaining finite field and Reed Solomon Code.

2.5.3 Galois Fields

In order to understand the encoding & decoding principles of Reed Solomon code, one should have a thorough knowledge of finite fields known as Galois fields. For any prime number p there exists a Galois field GF(p) which contains exactly p elements. It is quite possible to extend GF (p) to an extension of p^m elements making it GF (p^m) where m is a non-zero positive integer.

Besides the numbers 0 and 1, there are additional unique elements in the extension field that will be represented with a new symbol α. Each nonzero element in GF (2^m) can be represented by a power of α. An infinite set of elements, F, is formed by starting with the elements {0, 1, α}, and generating additional elements by progressively multiplying the last entry by α, which yields the following:

$$F = \{0,1, \alpha, \alpha^2, \ldots, \alpha^j,\} = \{0, \alpha^0, \alpha^1, \alpha^2, \ldots, \alpha^j, \ldots\} \qquad (2.5)$$

To obtain the finite set of elements of GF (2^m) from F, a condition must be imposed on F so that it may contain only 2^melements and is closed under multiplication.

The condition that closes the set of field elements under multiplication is characterized by the irreducible polynomial shown below:

$(\alpha^{(2m-1)}) + 1 = 0$ (2.6)

Or $\alpha^{(2m-1)} = 1 = \alpha^0$ (2.7)

Each of the 2^melements of the finite field, GF(2^m), can be represented as a distinct polynomial of degree m - 1 or less. The degree of a polynomial is the value of its highest-order exponent. We denote each of the nonzero elements of GF(2^m) as a polynomial, $a_i(X)$, where at least one of the m coefficients of $a_i(X)$ is nonzero. One of the benefits of using extension field elements $\{\alpha^i\}$ in place of binary elements is the compact notation that facilitates the mathematical representation of non-binary encoding and decoding processes. Addition of two elements of the finite field is then defined as the modulo-2 sum of each of the polynomial coefficients of like powers.

$\alpha^i + \alpha^j = (a_{i,0} + a_{j,0}) + (a_{i,1} + a_{j,1}) X + \ldots + (a_{i,\, m-1} + a_{j,\, m-1}) X_{m-1}$ (2.8)

2.5.4 Primitive Polynomial

A class of polynomials called primitive polynomials is of interest because such functions define the finite fields GF(2^m) that in turn are needed to define R-S codes. The following condition is necessary and sufficient to guarantee that a polynomial is primitive. An irreducible polynomial f(X) of degree m is said to be primitive if the smallest positive integer n for which f(X) divides X^n+1 is n = 2m - 1. Note that the statement A divides B means that A divided into B yields nonzero quotient and a zero remainder. Polynomials will usually be shown low order to high order. Sometimes, it is convenient to follow the reverse format.

Table 2-4: Primitive polynomial for different rate RS codes

m in GF(2^m)	**(n,k)**	**Parity (2t=n-k)**	**Code rate**	**Primitive polynomial**
3	(7,3)	4	3/7=0.428	X^3+X+1
4	(15,11)	4	11/15=0.733	X^4+X+1
5	(31,27)	4	27/31=0.871	X^5+X^2+1
6	(63,55)	8	55/63=0873	X^6+X+1
7	(127,111)	8	111/127=0.874	X^7+X^3+1
8	(255,239)	16	239/255=0.937	X^8+X^7+X^2+X+1
8	(255,223)	32	223/255=0.875	X^8+X^7+X^2+X+1

Example: Verify whether the polynomial is primitive or not. $f(X) = 1 + X + X^4$

We can verify whether this degree m = 4 polynomial is primitive by determining whether it divides $X^n + 1 = X^{(2m-1)} = X^{15} + 1$, but does not divide $X^n + 1$, for values of n in the range of $1 \leq n < 15$. It is easy to verify that $1 + X + X^4$ divides $X^{15} + 1$, and after repeated computations it can be verified that $1 + X + X^4$ will not divide $X^n+1$for any n in the range of $1 \leq n < 15$. Therefore, $1 + X + X^4$ is a primitive polynomial.

2.5.5 Constructing the Galois field

Consider an example of (15,11) RS Codes. Primitive polynomial to construct GF (24) is X4+X+1…. (4)

All the non-zero elements of the Galois field can be constructed by using the fact that the primitive element is a root of the field generator polynomial, so that

$p(\alpha) = 0.$ (2.9)

Thus, for GF (16) with the field generator polynomial shown in (4), we can write:

$\alpha 4 + \alpha + 1 = 0$ (2.10)

or

$\alpha 4 = \alpha + 1$ (remembering that + and - are the same in a Galois field).

Multiplying by α at each stage, using α + 1 to substitute for α4 and adding the resulting terms can be used to obtain the complete field as shown in Table. This shows the field element values in both index and polynomial forms along with the binary and decimal short-hand versions of the polynomial representation.

If the process shown in Table is continued beyond 14, it is found that 15 = 0, 16 = 1 ...so that the sequence repeats with all the values remaining valid field elements.

Table 2-5: Galois field elements for (15, 11) RS codes.

Index form	Polynomial form	Binary form	Decimal form
0	0	0000	0
α^0	1	0001	1
α^1	α^1	0010	2
α^2	α^2	0100	4
α^3	α^3	1000	8
α^4	$\alpha+1$	0011	3
α^5	$\alpha^2+\alpha$	0110	6

α^6	$\alpha^3+\alpha^2$	1100	12
α^7	$\alpha^3+\alpha+1$	1011	11
α^8	α^2+1	0101	5
α^9	$\alpha^3+\alpha$	1010	10
α^{10}	$\alpha^2+\alpha+1$	0111	7
α^{11}	$\alpha^3+\alpha^2+\alpha$	1110	14
α^{12}	$\alpha^3+\alpha^2+\alpha+1$	1111	15
α^{13}	$\alpha^3+\alpha^2+1$	1101	13
α^{14}	α^3+1	1001	9

2.5.6 Galois field addition and subtraction

When we add two field elements, we add the two polynomials:

$$(a_{m-1}x^{m-1}+....+a_1x^1+a_0x^0)+(b_{m-1}x^{m-1}+....+b_1x^1+b_0x^0)=c_{m-1}x^{m-1}+....+c_1x^1+c_0x^0$$

Where

$c_i = a_i + b_i$ for $i = 0$ to $m-1$.

However, the coefficients can only take the values 0 and 1, so

$c_i = 0$ for $a_i = b_i$

$c_i = 1$ for $a_i \neq b_i$

Thus, two Galois field elements are added by modulo-two addition of the coefficients, or in binary form, producing the bit-by-bit exclusive-OR function of the two binary numbers.

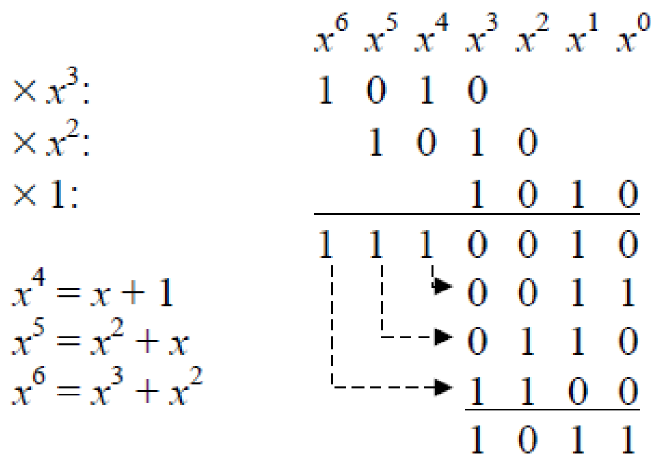
2.5.7 Galois field multiplication and division

Galois field (GF) multiplication is shown using an example.

Example: 13x10

13 in GF (16) – 1101

10 in GF (16) – 1010



1011 in GF (16) is 11.

So, 13 x 10 =11

Another method (Logarithmic Method): Taking the index of α of the numbers to be multiplied and adding them and taking mod15. The α to the power of the number so obtained gives the product of the numbers.

<u>Example</u>: 10= α^9 and 13= α^{13}

10 x 13 = $\alpha^{(9+13)\ \%15} = \alpha^{22\%15} = \alpha^7$=11

A slight disadvantage of the logarithmic method is that field element 0 cannot be represented in index form. The method therefore has to sense the presence of zero values and force the result accordingly.

The logarithmic technique can also be used for division:

$11 \div 10 = \alpha^7 \div \alpha^9 = \alpha^{7-9\%15} = \alpha^{13} = 13$

However, division of two field elements is often accomplished by multiplying by the inverse of the divisor.

The inverse of a field element is defined as the element value that when multiplied by the field element produces a value of 1 (= 0). It is therefore possible to tabulate the inverse values of the field elements using Table.

Example: Inverse of 10: 10=α^9

$10^{-1} = \alpha^{15-9} = \alpha^6 = 12.$

Chapter 3

DESIGN METHODOLOGY

3.1 Reed Solomon code

The Reed Solomon codes are non-binary cyclic systematic block codes consisting of ‘n’ symbols of which ‘k’ information symbols and ‘n-k’ parity symbols. Each symbol is of ‘m’ bits hence making ‘n = 2^m-1’. All symbols belong to the GF (2^m). All basic mathematical operations result in GF (2^m) elements.

It has a minimum hamming distance of ‘n-k-1’ hence it is called Maximum Distance Separable codes. It can detect up to ‘n-k’ number of symbols and correct a burst error of up to (n-k)/2 symbols.

An (n, k) Reed-Solomon code is constructed by forming the code generator polynomial g(x), consisting of n-k=2t factors, the roots of which are consecutive elements of the Galois field. Choosing consecutive elements ensures that the distance properties of the code are maximized. Thus the code generator polynomial takes the form:

$g(x) = (x - \alpha^{b})(x - \alpha^{b+1}) \ldots (x - \alpha^{b+2t-1})$

CCSDS Standard

$$g(x) = \prod_{j=n-t}^{j=n+t-1} (x - \alpha^{i(j)\%n}) \tag{3.1}$$

Where α^i is the primitive element chosen such that $\alpha^i \varepsilon$ GF(2^n) and not an element in the GF(2^{n-1}) or its subsets.

Table 3-1: Primitive Polynomial for different rates

m	Primitive element	Generator Polynomial
3	α^3	$X^4+4X^3+2X^2+4X+1$
4	α^7	$X^4+10X^3+5X^2+10X+1$
5	α^3	$X^4+5X^3+13X^2+5X+1$
6	α^5	$X^8+6X^7+29X^6+34X^5+3X^4+34X^3+29X^2+6X+1$
7	α^9	$X^{16}+109X^{15}+85X^{14}+99X^{13}+79X^{12}+95X^{11}+103X^{10}+91X^9+4X^8+91X^7+103X^6+95X^5+79X^4+99X^3+85X^2+109X+1$
8	α^{11}	$X^{16}+109X^{15}+85X^{14}+99X^{13}+79X^{12}+95X^{11}+103X^{10}+91X^9+4X^8+91X^7+103X^6+95X^5+79X^4+99X^3+85X^2+109X+1$

8	α^{11}	$X^{32}+91X^{31}+127X^{30}+86X^{29}+16X^{28}+30X^{27}+13X^{26}+235X^{25}+97X^{24}+165X^{23}+8X^{22}+42X^{21}+54X^{20}+86X^{19}+171X^{18}+32X^{17}+113X^{16}+32X^{15}+171X^{14}+86X^{13}+54^{12}+42X^{11}+8X^{10}+165X^{9}+97X^{8}+235X^{7}+13X^{6}+30X^{5}+16X^{4}+86X^{3}+127X^{2}+91X+1$

3.1.1 The message polynomial

The k information symbols that form the message to be encoded as one block can be represented by a polynomial M(x) of order k-1, so that:

$M(x) = M_{k-1}\ x_{k-1}+.... + M_1x + M_0$

where each of the coefficients M_{k-1}, ..., M_1, M_0 is an m-bit message symbol, that is, an element of GF(2^m). M_{k-1} is the first symbol of the message.

3.1.2 Forming the code word

For a (15, 11) code, the block length is 15 symbols, 11 of which are information symbols and the remaining 4 are parity words. Because t=2, the code can correct errors in up to 2 symbols in a block. Substituting for n in: n = 2^{m-1} gives the value of m as 4, so each symbol consists of a 4-bit word and the code is based on the Galois field with 16 elements. The example will use the field generator polynomial as shown in equation below, so that the arithmetic for the code will be based on the Galois field shown in Table. The code generator polynomial for correcting up to 2 error words requires 4 consecutive elements of the field as roots, so we can choose:

$$g(x) = (x+\alpha^0)(x+\alpha^1)(x+\alpha^2)(x+\alpha^3) = (x+1)(x+2)(x+4)(x+8)$$

To encode the message, the message polynomial is first multiplied by x^n-k and the result divided by the generator polynomial, g(x). Division by g(x) produces a quotient q(x) and a remainder r(x), where r(x) is of degree up to n-k-1. Thus:

$$\frac{M(x)\times x^{n-k}}{g(x)} = q(x)+\frac{r(x)}{g(x)} \tag{3.2}$$

Having produced r(x) by division, the transmitted code word T(x) can then be formed by combining M(x) and r(x) as follows:

$$\begin{aligned} T(x) &= M(x)\times x^{n-k} + r(x) \\ &= M_{k-1}x^{n-1} + ... + M_0x^{n-k} + r_{n-k-1}x^{n-k-1} + ... + r_0 \end{aligned} \tag{3.3}$$

This shows that the code word is produced in the required systematic form.

We can now choose a message consisting of eleven 4-bit symbols for our (15, 11) code, for example, the values 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 which we wish to encode. These values are represented by a message polynomial:

$x^{10} + 2\ x^{9} + 3\ x^{8} + 4\ x^{7} + 5\ x^{6} + 6\ x^{5} + 7\ x^{4} + 8\ x^{3} + 9\ x^{2} + 10x + 11$

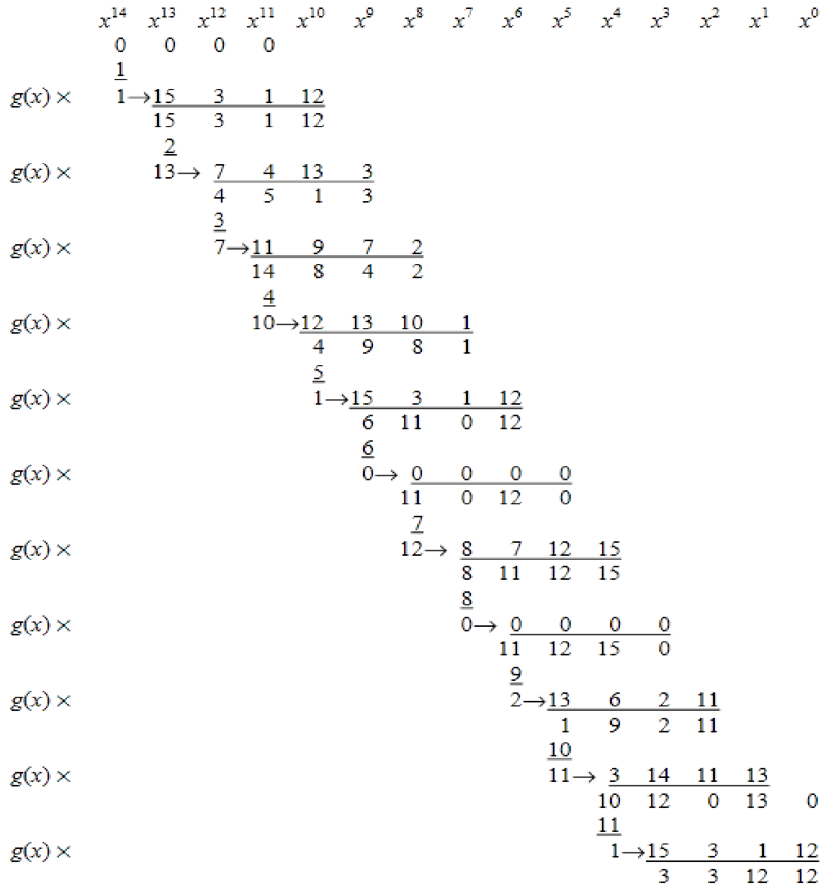
The message polynomial is then multiplied by x^4 to give:

$x^{14} + 2\ x^{13} + 3\ x^{12} + 4\ x^{11} + 5\ x^{10} + 6\ x^{9} + 7\ x^{8} + 8\ x^{7} + 9\ x^{6} + 10x^{5} + 11x^{4}$

This polynomial is then divided by the code generator polynomial to produce the four parity symbols as a remainder. This can be accomplished in columns as a long division process as shown before, except that in this case, the coefficients of the polynomials are field elements of GF (16) instead of binary values, so the process is more complicated.

3.1.3 Pipelined version

Hardware encoders usually operate on pipelined data, so the division calculation is made in a slightly altered form using the message bits one at a time as they are presented:



```
          x14  x13  x12  x11  x10  x9   x8   x7   x6   x5   x4   x3   x2   x1   x0
           0    0    0    0
           1
g(x) ×     1→  15    3    1   12
               15    3    1   12
                2
g(x) ×         13→   7    4   13    3
                     4    5    1    3
                     3
g(x) ×               7→  11    9    7    2
                         14    8    4    2
                          4
g(x) ×                   10→  12   13   10    1
                               4    9    8    1
                               5
g(x) ×                         1→  15    3    1   12
                                    6   11    0   12
                                    6
g(x) ×                              0→    0    0    0    0
                                         11    0   12    0
                                          7
g(x) ×                                   12→   8    7   12   15
                                               8   11   12   15
                                               8
g(x) ×                                         0→    0    0    0    0
                                                    11   12   15    0
                                                     9
g(x) ×                                               2→  13    6    2   11
                                                          1    9    2   11
                                                         10
g(x) ×                                                   11→   3   14   11   13
                                                              10   12    0   13    0
                                                              11
g(x) ×                                                         1→  15    3    1   12
                                                                    3    3   12   12
```

Figure 3-2:Long division

The division produces the remainder: r(x) = $3x^3 + 3x^2 + 12x + 12$. The quotient, q(x), produced as the left-hand column of multiplying values is not required and is discarded. The encoded message polynomial T(x) is then: $x^{14} + 2x^{13} + 3x^{12} + 4x^{11} + 5x^{10} + 6x^9 + 7x^8 + 8x^7 + 9x^6 + 10x^5 + 11x^4 + 3x^3 + 3x^2 + 12x + 12$ or, written more simply: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 3, 3, 12, 12.

3.2 Reed-Solomon decoding techniques

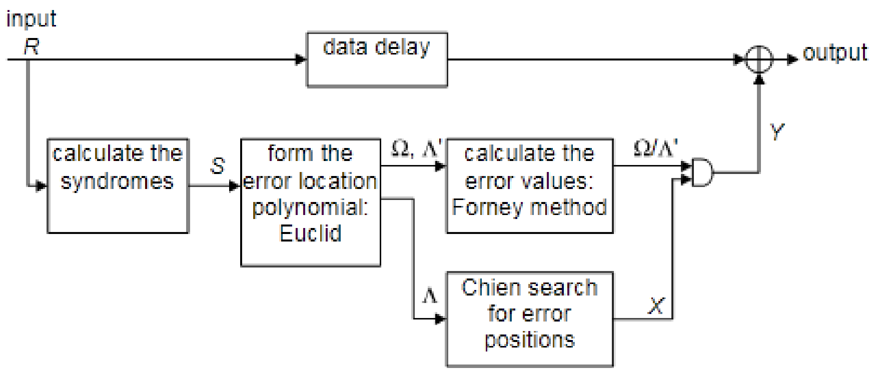

input
R
data delay
output
Y
calculate the syndromes
S
form the error location polynomial: Euclid
Ω, Λ'
calculate the error values: Forney method
Ω/Λ'
Λ
Chien search for error positions
X

Figure 3-3: Decoder Block Diagram

The first process is to calculate the syndrome values from the incoming code word. These are then used to find the coefficients of the error locator polynomial $\Lambda_1 .... \Lambda_v$ and the error value polynomial $\Omega_0 .... \Omega_{v-1}$ using the Berlekamp algorithm. 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 code word, it is necessary to store the message until the results of the calculation are available. Then, to correct the errors, each error value is added (modulo 2) to the symbol at the appropriate location in the received code word**.**

3.2.1 Calculating the syndromes

Previously it was shown that the transmitted code word is always divisible by the generator polynomial without remainder and that this property extends to the individual factors of the generator polynomial. Therefore, the first step in the decoding process is to divide the received polynomial by each of the factors $(x + \alpha^i)$ of the generator polynomial, equation (6). This produces a quotient and a remainder, that is:

$$\frac{R(x)}{x+\alpha^i} = Q_i(x) + \frac{S_i}{x+\alpha^i} \qquad for \quad b \le i \le b+2t-1 \tag{3.4}$$

Where b is chosen to match the set of consecutive factors in (6). The remainders Si resulting from these divisions are known as the syndromes and, for b=0, can be written as S_0 .... S_{2t-1}. Rearranging the above equation produces:

$$S_i = Q_i(x)\times(x+\alpha^i)+R(x) \tag{3.5}$$

so that when x = α^i this reduces to:

$$\begin{aligned} S_i &= R(\alpha^i) \\ &= R_{n-1}(\alpha^i)^{n-1} + R_{n-2}(\alpha^i)^{n-2} + \ldots. + R_1\alpha^i + R_0 \end{aligned} \tag{3.6}$$

3.2.2 The error locator polynomial

The next step is to introduce the error locator polynomial. This turns out to be one of the more confusing steps in Reed-Solomon decoding because the literature defines two different, but related, expressions as the error locator polynomial. One form, often denoted σ(x), is constructed to have the error locators X_1 .... X_v as its roots, that is, v factors of the form $(x+X_j)$ for j= 1 to v. When expanded, these factors produce a polynomial of degree v with coefficients σ_1 ....σ_v:

$$\begin{aligned} \sigma(x) &= (x+X_1)(x+X_2)\ldots.(x+X_v) \\ &= x^v + \sigma_1 x^{v-1} + \ldots. + \sigma_{v-1}x + \sigma_v \end{aligned} \tag{3.7}$$

The alternative form is usually denoted Λ(x). This is constructed to have v factors of the form $(1+X_jx)$ and therefore has the inversesX_1^{-1} .... , $X_v^{-1}$of the v error locators as its roots. When expanded, these factors produce a polynomial of degree v with coefficients Λ_1 ....Λ_v:

$$\begin{aligned} \Lambda(x) &= (1+X_1x)(1+X_2x)\ldots.(1+X_vx) \\ &= 1 + \Lambda_1 x + \ldots. + \Lambda_{v-1}x^{v-1} + \Lambda_v x^v \end{aligned} \tag{3.8}$$

However, it turns out that

$$\sigma(x) = x^v \times \Lambda\left(\frac{1}{x}\right) \tag{3.9}$$

So the coefficients σ_1.... σ_v are the same as Λ_1.... Λ_v.

3.2.3 Berlekamp Algorithm

Berlekamp algorithm consists of a series of steps based on improving an approximation to the error locator polynomial Λ(x) using a correction polynomial C(x) and the syndrome values S_0 ....S_{2t-1} as inputs. It also requires a step parameter K and a parameter L which tracks the order of the equations.

Initially we set: K = 1, L = 0, (x) = 1 and C(x) = x.

Then each step consists of first calculating an error value e using:

$$e = S_0 + \sum_{i=1}^{L} \Lambda_i S_{K-1-i} \tag{3.10}$$

So initially e = S_0. Then, provided that e is non-zero, we produce Λ*(x), a new approximation to the error locator polynomial, given by:

$$\Lambda^*(x) = \Lambda(x) + e \times C(x). \tag{3.11}$$

If 2L < K, we set L = K - L and form a new correction polynomial from: C(x) = (x)) e. If e is zero, these calculations are omitted. Then we produce a new C(x) by multiplying the old correction polynomial by x, replace Λ(x) by Λ*(x) and increment K. The next step starts with the new values of K, L, Λ(x) and C(x) until K > 2t in which case Λ(x) is the required error locator polynomial

3.2.4 Error Magnitude Polynomial

The error magnitude polynomial can be written as:

$$\Omega(x) = \Omega_{v-1}x^{v-1} + ... + \Omega_1 x + \Omega_0 \tag{3.12}$$

The key equation then is written as:

$$\Omega(x) = [S(x)\ \Lambda(x)] \bmod x^{2t} \tag{3.13}$$

Any terms of degree x^{2t} or higher in the product are ignore, so that:

$$\Omega_0 = S_b$$

$$\Omega_1 = S_{b+1} + S_b\Lambda_1$$

$$\vdots$$

$$\Omega_{v\text{-}1} = S_{b+v\text{-}1} + S_{b+v\text{-}2}\Lambda_1 + \ldots. + S_b\Lambda_{v\text{-}1}$$

3.2.5 Solving the error locator polynomial - the Chien search

Having calculated the coefficient values, Λ_1 .... Λ_v , of the error locator polynomial, it is now possible to find its roots. If the polynomial is written in the form:

$$\Lambda(x) = X_1(x + X_1^{-1})\, X_2(x + X_2^{-1}) \ldots.$$

then clearly the function value will be zero if x = X_1^{-1}, X_2^{-1}, ... , that is:

$$x = \alpha^{-e_1}, \alpha^{-e_2}, \ldots.. \qquad (3.14)$$

The roots, and hence the values of X_1 .... Xv, are found by trial and error, known as the Chien search, in which all the possible values of the roots (the field values i, 0 i n-1) are substituted into error locator polynomial and the results evaluated. If the expression reduces to zero, then that value of x is a root and identifies the error position. Since the first symbol of the code word corresponds to the xn-1 term, the search begins with the value$\alpha^{-(n-1)}$ (=α^1), then $\alpha^{-(n-2)}$ (=α^2), and continues to 0, which corresponds to the last symbol of the code word.

3.2.6 Calculating the error values -The Forney algorithm

This is an alternative means of calculating the error value Y_j having established the error locator polynomial Λ(x) and the error value polynomial σ(x). Methods of calculating the error values Y_1 ....Y_v based on Forney's algorithm are more efficient than the direct method of solving the syndrome equations

$$Y_j = X_j^{1-b} \frac{\Omega(X_j^{-1})}{\Lambda'(X_j^{-1})} \qquad (3.15)$$

Where $\Lambda'(X_j^{-1})$ is the derivative of (x) for x = X_j^{-1}. When b=1, the X_j^{1-b}term disappears, so the formula is often quoted in the literature as simply Ω/Λ', which gives the wrong results for b=0 and other values. It should be noted that the above equation only gives valid results for symbol positions containing an error. If the calculation is made at other positions, the result is generally non-zero and invalid. The Chien search is therefore still needed to identify the error positions.

Chapter 4

FPGA IMPLEMENTATION OF RS CODEC

4.1 Introduction

Field Programmable Gate Array (FPGA) can be defined as the user programmable integrated circuit consisting of a set of logic blocks, which can be interconnected by general routing resources. The user, without the use of fabrication facility can directly configure Field Programmable devices.

FPGA consists of an array of uncommitted elements that can be interconnected in a general way. Xilinx Company introduced FPGA in 1985. Since then many different FPGAs have been developed by a number of companies: ACTEL, ALTERA, Quick logic, Algotronix, Advanced Micro Devices (AMD), concurrent logic and cross point solutions among others. Logic circuits are implemented in the FPGA by partitioning the logic blocks and then interconnecting the blocks as required via the switches.

The structure and content of a logic block are called its architecture. Some FPGA logic blocks are as simple as two input NAND gates. Other blocks have more complex structure such as multiplexed or look up tables. Most logic blocks also contain some type of flip-flop to aid in the implementation of sequential circuits.

A field programmable device is a device in which the final logic structure can be directly configured by the end user, without use of an integrated circuit fabrication facility. In context of implementing logic circuits, superior speed performance can be obtained with a mask programmable chip because connection with in device can be hardwired during manufacture.

4.2 ACTEL FPGAs

Space applications require radiation tolerant FPGAs. ACTEL is the leading provider of FPGA for high reliability space applications. ACTEL provides anti-fuse-based FPGAs. These devices use metal-to-metal anti-fuse connection for configuration and include built-in TMR on all registers anti-fuse connection are immune radiation the correct state for each bit. In addition to radiation tolerance, anti-fuse device offers low power consumption and high operating frequencies.

4.3 ACTEL Axcelerator RTAX2000 FPGA Features

Axcelerator AX2000 FPGA offers high performance at densities of up to two million equivalent system gates AX2000 has several system level features such as embedded SRAM (with complete FIFO control logic), PLCs, Segment able clocks, and chip-wide highway routing.

AX2000 has the following features:

- Protected against Single Event Upsets (SEU) and Single Event Transients (SET). (otherwise occur due to heavy ion radiation in space).
- Immune to radiation induced configuration upsets.
- Military temperature range of -55ºC to 125ºC.
- 288 K bits of embedded SRAM/FIFO with SEU performance < 10^-10 upsets/bit-day when used with the ACTEL provided Error Detection and Correction (EDAC) Soft IP and background SRAM scrubber.
- Based upon 0.15 µm, seven-layer-metal CMOS anti-fuse process technology offers a level of performance equivalent in ASIC technology.
- Bank-selectable I/O’s, 8 banks per chip.
- Single-ended I/O standard: LVTTL, LVCMOS, 3.3vPCI, and 3.3vPCI-X.
- Clock trees-dedicated clock drivers.

4.4 Software tools used

Libero Integrated Design Environment (IDE) is ACTEL’s comprehensive software toolset for designing with all ACTEL FPGAs. From design, synthesis and simulation, through floor planning, place and route, timing constraints and analysis, power analysis, and program file generation, Libero IDE manages the entire design flow quickly and efficiently. Libero’s second-generation smart design provides an efficient methodology for creating complete simple and complex processor-based System on Chip (SOC) designs with ease.

4.5 Libero IDE 9.1 Software Features

- Powerful project and design flow management.
- Full suite of integrated design entry tools and methodologies.
- “user-defined block” creation flow for design reuse.
- View draws schematic capture.
- ACTEL cell libraries.
- Synplify/Synplify PRO AEsynthesis fully optimizes ACTEL FPGA device performance and area utilization.
- Simplify DSP AE performs high-level DSP optimization with in a Simulink Environment.
- Modelsim VHDL or Verilog behavioural, post-synthesis and post layout simulation capability.
- Designer physical design implementation, floor planning, physical constraints, and layout.
- Timing and power-driven place and route.
- Smart time environment for timing constraints management and analysis.
- Interface to Flash Pro and Silicon sculptor programming software. Supported on windows and Linux operating system.

4.6 Hardware estimate of Generalised RS Encoder:

Target Part	:	ax2000_cqfp256-s
Combinational Cells	:	253 of 21504(1%)
Sequential Cells	:	128 of 10752 (1%)
Total Cells	:	381 of 32256 (2%)
Clock Buffers	:	2
IO Cells	:	20

RAM/ROM Usage Summary

Block RAM	:	0 of 22 (0%)

4.7 Reed- Solomon Encoder

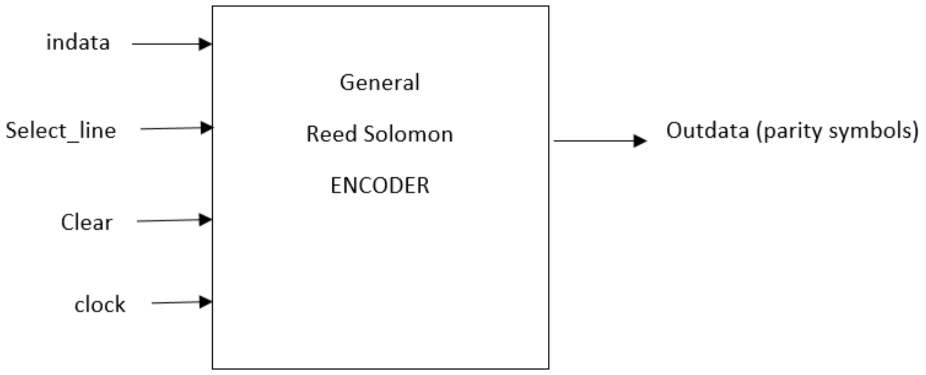


Figure 4-1: Top Module of General RS Encoder

The general block diagram of the RS encoder is shown here. The inputs are Data slot which is kept high when we are giving input. Message input is the information symbols that are to be encoded. Select line is used to select any one of the encoders that is shown in the table. An 8-bit message input is taken for every rising edge of the clock. Encode out is the parity symbols obtained from the process of encoding. Each time we have 16 symbol output of which only the number of parities defined for given select line is taken out and rest will be zero.

Table 4-2: Select lines for selection of different rate RS codes

Select Lines	(n,k)	Parity= (n-k)
000	(7,3)	4
001	(15,11)	4
010	(31,27)	4
011	(63,55)	8
100	(127,111)	16
101	(255,239)	16
110	(255,223)	32

4.7.1 Pipelined Division:

The output obtained from the multiplier is fed to this block for polynomial or pipelined division. In the above figure the 1st array of D-flip flops is used for pipelined division.

This works with the clock (blue colour) input. The 2nd array of D-flip-flops are used to latch the parity symbols upon generation of the stop (red colour) signal. This array of flip flops is working with stop signal as the clock input. Here whenever a new block of information symbols are fed a signal (shown in Orange colour) is used to reset the 1st array of flip-flops to zero.

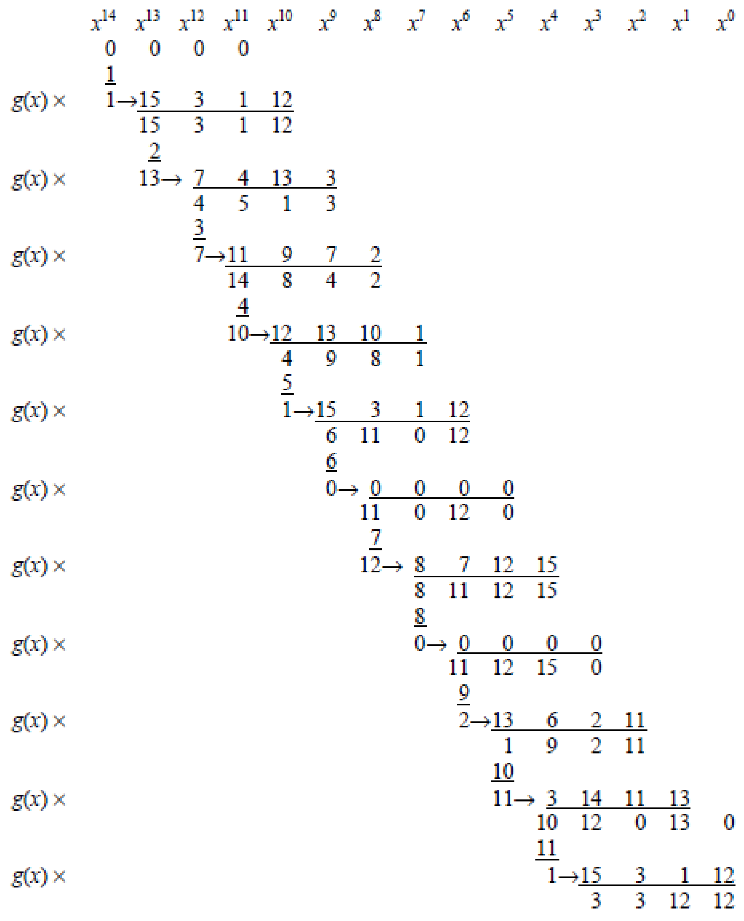


Figure 4-3: Pipelined Division

4.7.2 General Multiplier:

The above figure shows the multipliers for different rates. The input message symbol is or XORed with the output of the D-flipflop shown in the next figure.

This is given to all multipliers. Each multiplier gives 32 symbols output of which all except the (n-k) symbols are zero. The number of bits each symbols is made of differ from each multiplier. But here we have considered 8-bit by appending zeros to the symbol. The output of the 7-multipliers is fed to the 7:1 multiplexer. Depending on the Select_line input its corresponding multiplier output is chosen.

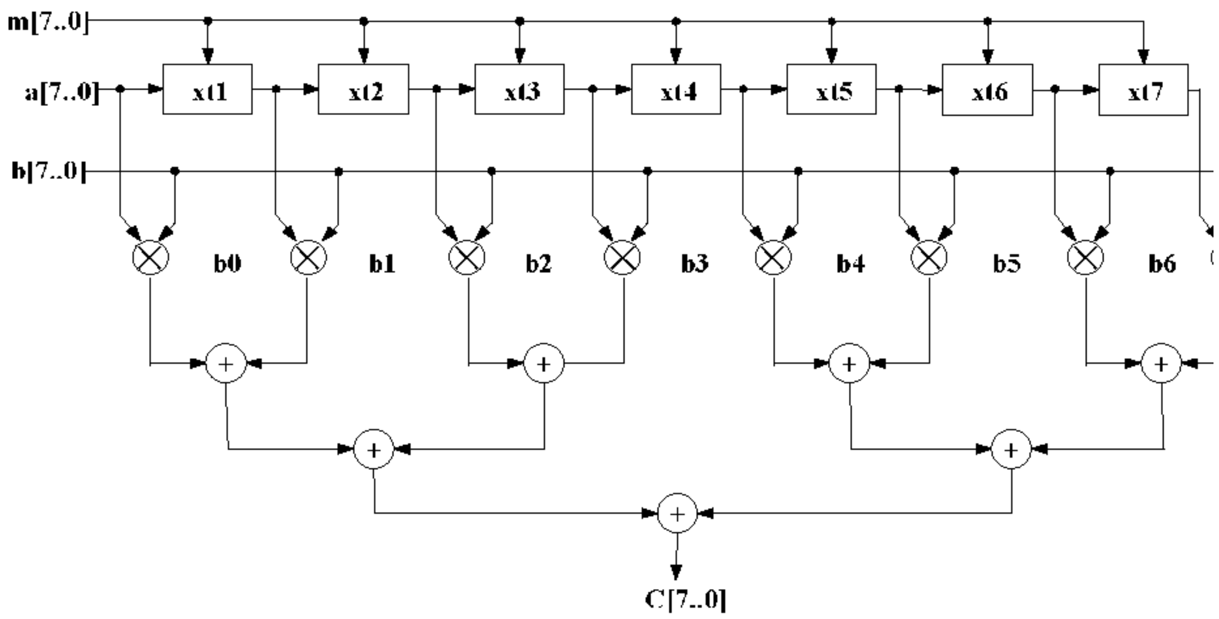


Figure 4-4:General Multiplier

Control signals generation

The logic to generate start, stop and reset signals is shown. Data slot is made high as and when we start giving the information symbols and made low after a block of information is entered. The logic circuit to generate control signal requires 2 shifts of the Data slot so we shift the input message as well before giving it to the multiplier.

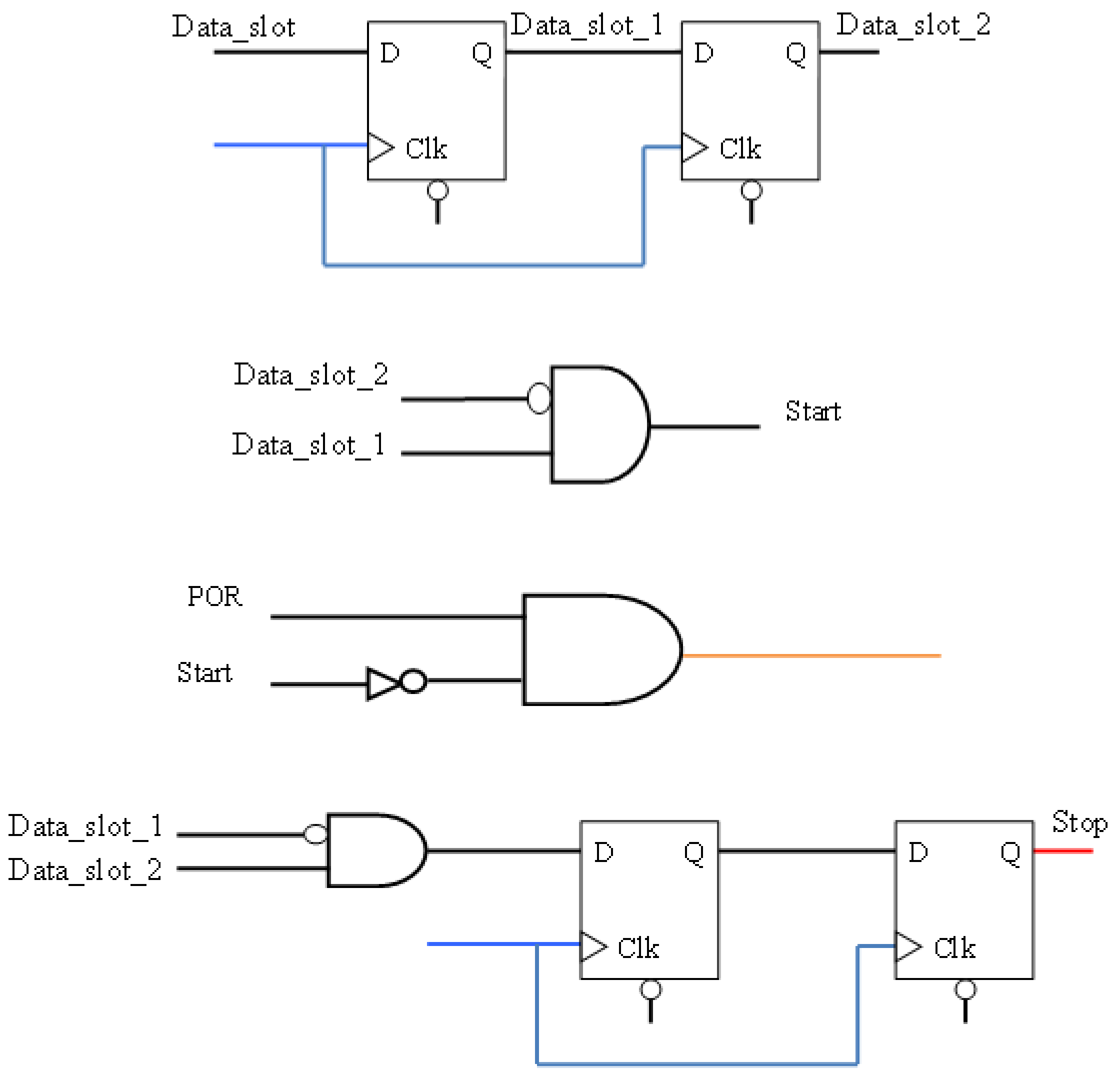


Figure 4-5: Control Signals for RS Encoder

4.7.3 Encoder hardware

General Arrangement

The pipelined calculation is performed using the conventional encoder circuit shown in Figure 4.1.

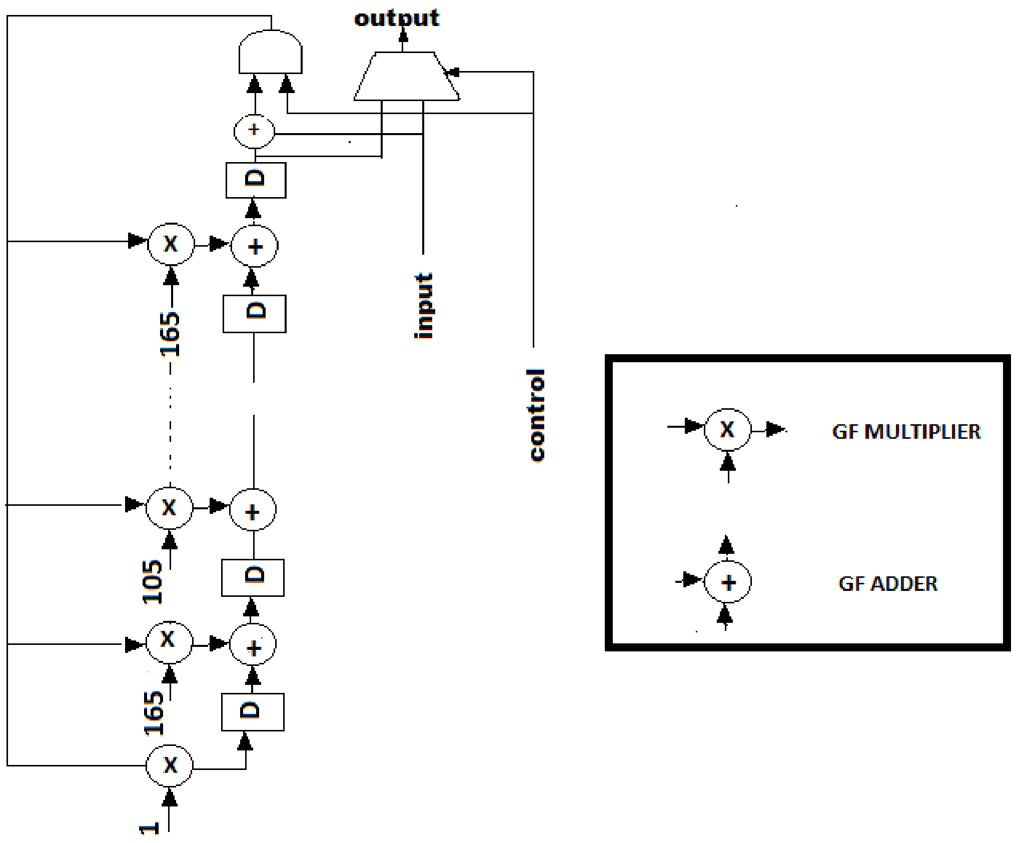


Figure 4-6: Encoder hardware RS (255,239)

- This is the hardware implementation of pipeline division which is being used to generate the parity.
- The main components are Galois field multipliers, shift registers, Galois adders and multiplexers.
- During the message input period, the control line is 1, the AND gate is enabled and the input is sent as output.

- During the parity period, the control line is 0, the AND gate is disabled and the parity generated is shifted to the output using shift registers.

4.7.4 Galois field adders

The adders of Figure perform bit-by-bit addition modulo-2 of 8-bit numbers and each consists of eight 2-input exclusive-OR gates. The multipliers, however, can be implemented in a number of different ways.

4.7.5 Galois field constant multipliers

Since each of these units is multiplying by a constant value, there are two approach, one is using dedicated logic multiplier circuit and other approach would be to use a full multiplier and to fix one input. We used dedicated logic multiplier circuit which is explained in the below section.

4.1.5.1 Dedicated logic constant multipliers

For the logic circuit approach, we can work out the required functionality by using a general polynomial representation of the input signal

$a7\alpha_7 + a6\alpha_6 + a5\alpha_5 + a4\ \alpha_4 + a3\alpha_3 + a2\alpha_2 + a1\alpha + a0$.

This is then multiplied by the polynomials represented by the values 1, 165, 105, 27, 159, 104, 152, 101, 76 from GF table. This involves producing a shifted version of the input for each non-zero coefficient of the multiplying polynomial. Where the shifted versions produce values in the α14, α13, α12, α11, α10, α9, α8 columns, the 8-bit equivalents (from GF table) are substituted. The bit values in each of the α7, α6, α5, α4, α3, α2, α1 and α0 columns are then added to give the required input bit contributions for each output bit. The detail work is shown:

e.g.

Finding equation of multiplier “104”.

104 (in binary) = 01101000.

	α^{14}	α^{13}	α^{12}	α^{11}	α^{10}	α^{9}	α^{8}	α^{7}	α^{6}	α^{5}	α^{4}	α^{3}	α^{2}	α^{1}	α^{0}
0															
1		a7	a6	a5	a4	a3	a2	a1	a0						
1			a7	a6	a5	a4	a3	a2	a1	a0					
0															
1					a7	a6	a5	a4	a3	a2	a1	a0			
0		a7	a6+a7	a5+a6	a4+a5+a7	a4+a3+a6	a2+a3+a5								
0		↓ ↓ ↓ ↓ ↓	↓ ↓ ↓ ↓ ↓	↓ ↓ ↓ ↓ ↓	↓ ↓ ↓ ↓ ↓	↓ ↓ ↓ ↓ ↓	↓ →	a2+a3+a5	0	0	0	0	a2+a3+a5	a2+a3+a5	a2+a3+a5
0		↓ ↓ ↓ ↓ ↓	↓ ↓ ↓ ↓ ↓	↓ ↓ ↓ ↓ ↓	↓ ↓ ↓ ↓ ↓	↓ →	→	a4+a3+a6	0	0	0	a4+a3+a6	0	0	a4+a3+a6
		↓ ↓ ↓ ↓ ↓	↓ ↓ ↓ ↓ ↓	↓ ↓ ↓ ↓ ↓	↓ →→	→	→	a4+a5+a7	0	0	a4+a5+a7	0	a4+a5+a7	0	a4+a5+a7
		↓ ↓ ↓	↓ ↓ ↓	↓ →→	→→	→	→	a5+a6	0	a5+a6	0	a5+a6	a5+a6	0	a5+a6
		↓ ↓ ↓	↓ →→	→→	→→	→	→	a6+a7	a6+a7	0	a6+a7	a6+a7	a6+a7	0	a6+a7
		→→	→→	→→	→→	→	→	0	0	a7	a7	a7	a7	0	a7

Considering for 8-bit input from a0 to a7.

Solving for the column (α^7): 0utput7:

O7=a1+a2+a4+a2+a3+a5+a4+a3+a6+a4+a5+a7+a5+a6+a7.

(Because all Addition are XOR addition therefore 1) XOR addition of same number result in zero. 2) XOR addition of different number are retained.)

Therefore, O7 = a1+a4+a5+a6.

Similarly, all the columns are solved to get the respective outputs, which are as follows:

O0=a2+a5+a6+a7;

O1=a2+a3+a5;

O2=a2+a3+a4+a5+a7;

O3=a0+a3+a4+a5+a6;

O4=a1+a4+a5+a6+a7;

O5=a0+a2+a5+a6+a7;

O6=a0+a1+a3+a6+a7;

Similarly finding the equation for the other multipliers:

For multiplier “1”:

o0=a0;

o1=a1;

o2=a2;

o3=a3;

o4=a4;

o5=a5;

o6=a6;

o7=a7;

For multiplier “27”:	For multiplier “76”	For multipler “101”
o0=a0+a4+a7;	o0=a2+a3+a4+a6+a7;	o0=a0+a2+a6+a7;
o1=a0+a1+a4+a5+a7;	o1=a2+a5+a6;	o1=a1+a2+a3+a6;
o2=a1+a2+a4+a5+a6+a7;	o2=a0+a2+a4;	o2=a0+a3+a4+a6;
o3=a0+a2+a3+a5+a6+a7;	o3=a0+a1+a3+a5;	o3=a1+a4+a5+a7;
o4=a0+a1+a3+a4+a6+a7;	o4=a1+a2+a4+a6;	o4=a2+a5+a6;
o5=a1+a2+a4+a5+a7;	o5=a2+a3+a5+a7;	o5=a0+a3+a6+a7;
o6=a2+a3+a5+a6;	o6=a0+a3+a4+a6;	o6=a0+a1+a4+a7;
o7=a3+a6;	o7=a1+a2+a3+a5+a6;	o7=a1+a5+a6+a7;

For multiplier “104”

o0=a2+a5+a6+a7;

o1=a2+a3+a5;

o2=a2+a3+a4+a5+a7;

o3=a0+a3+a4+a5+a6;

o4=a1+a4+a5+a6+a7;

o5=a0+a2+a5+a6+a7;

o6=a0+a1+a3+a6+a7;

o7=a1+a4+a5+a6;

For multiplier “105”

o0=a0+a2+a5+a6+a7;

o1=a1+a2+a3+a5;

o2=a3+a4+a5+a7;

o3=a0+a4+a5+a6;

o4=a1+a5+a6+a7;

o5=a0+a2+a6+a7;

o6=a0+a1+a3+a7;

o7=a1+a4+a5+a6;

For multiplier “152”

o0=a1+a2+a3+a5+a6;

o1=a1+a4+a5+a7;

o2=a1+a3;

o3=a0+a2+a4;

o4=a0+a1+a3+a5;

o5=a1+a2+a4+a6;

o6=a2+a3+a5+a7;

o7=a0+a1+a2+a4+a5;

For multiplier “159”

o0=a0+a1+a2+a6;

o1=a3+a6+a7;

o2=a0+a1+a2+a4+a6;

o3=a1+a2+a3+a5+a7;

o4=a2+a3+a4+a6;

o5=a0+a3+a4+a5+a7;

o6=a1+a4+a5+a6;

o7=a0+a1+a5+a7;

For multiplier “165”

o0=a0+a1+a2+a6;

o1=a3+a6+a7;

o2=a0+a1+a2+a4+a6;

o3=a1+a2+a3+a5+a7;

o4=a2+a3+a4+a6;

o5=a0+a3+a4+a5+a7;

o6=a1+a4+a5+a6;

o7=a0+a1+a5+a7;

4.8 ENCODER INTERLEAVER

4.8.1 Introduction to Interleaver

The allowable values of interleaving are I=1, 2, 3, 4, 5, and 8, out of which the presented work is done for I =1, 5, 8. However I=1 is same as the absence of interleaving. The interleaving depth shall normally be fixed on a physical channel for a mission. Interleaving is accomplished in a functional way described in the Figure given below:

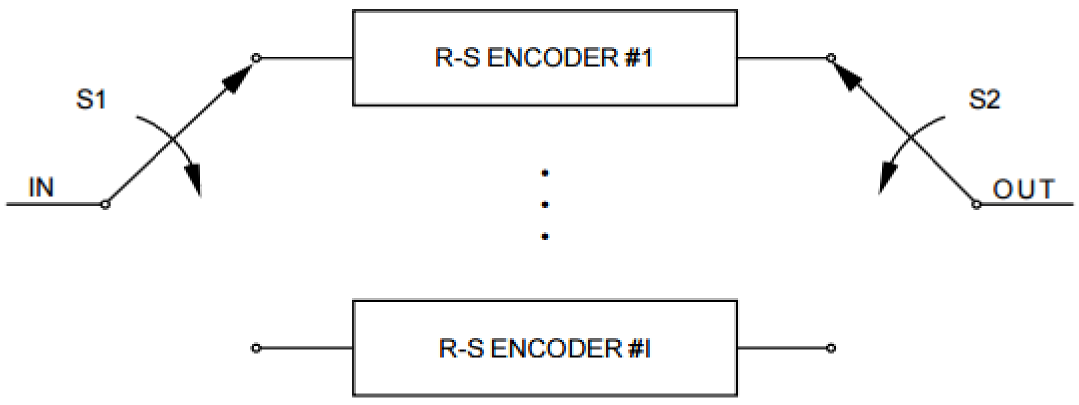


Figure 4-7:Interlever block diagram

Data bits to be encoded into a single Reed-Solomon Encoder block which enter at the port labeled “IN”.

Switches S1 and S2 are synchronized in such a way that it advanced from encoder to encoder in the sequence 1,2...I, 1,2...I….. spending one R-S symbol time (8 bits) in each position.

One code block will be formed from kI R-S symbols entering “IN”. In this functional representation, a space of 2EI R-S symbols in duration is required between each entering set of kI R-S information symbols.

Due to the action of S1, each encoder accepts k of these symbols, each symbol spaced I symbols apart (in the original stream). These k symbols are passed directly to the output of each encoder. The synchronized action of S2 reassembles the symbols at the port labeled “OUT” in the same way as they entered at “IN”.

If, for I=8, the original symbol stream is

$d_{11} \dots d_{18}\, d_{21} \dots d_{28} \dots d_{k1} \dots d_{k8}$ [2E × 8 spaces]

Then the output is the same sequence with the [2E × 5 spaces] filled by the [2E × 5] check symbols as shown below:

$$P_{11} \ldots\ldots P_{18} \ldots\ldots P_2E_1 \ldots\ldots P_2E_8 \qquad (4.1)$$

Where

$$d_{1i} \ldots\ldots\ldots d_{2i} \ldots\ldots d_{ki} \qquad P_{1i} \ldots P_2E_i \qquad (4.2)$$

is the R-S code word produced by the ith encoder. If q virtual fill symbols are used in each code word, then replace k by (k - q) in the above.

With this method of interleaving, the original k_I consecutive information symbols that entered the encoder appear unchanged at the output of the encoder with 2EI R-S check symbols appended.

4.8.2 Hardware implementation of RS- Interleaver

RS-Interleaver is implemented in a manner functionally described in the following diagram:

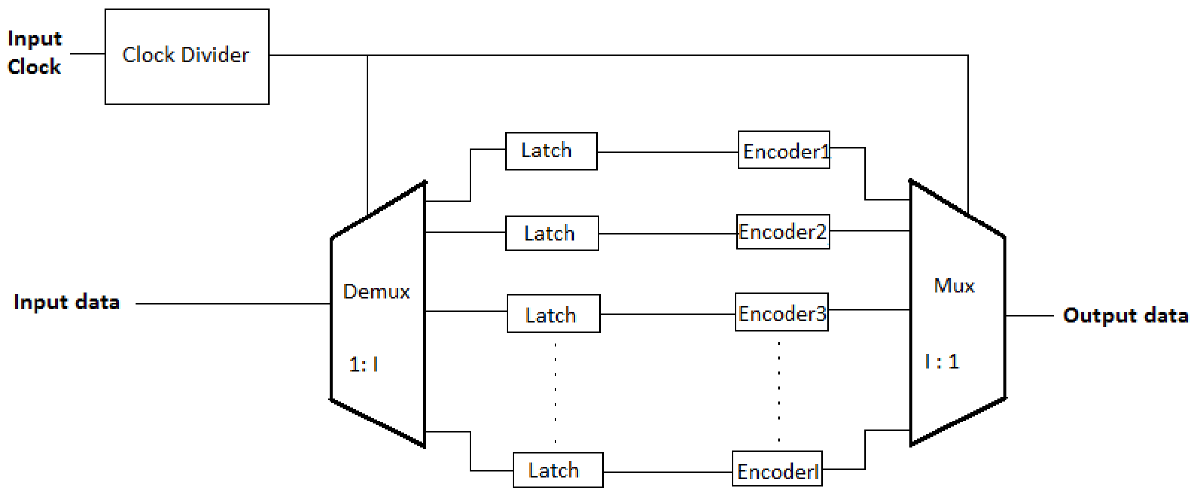


Figure 4-8: Working of Interleaver

- Interleaving depth, I =1 is equivalent to the absence of Interleaver.
- For Interleaver depth I = 5, 8 we use “clock divider” at the input section to accomplish the required Interleaver function.
- At the input section, Demux is used followed by latching of the input which is then fed to the encoder blocks correspondingly.
- At the output section, output from the encoders fed to input of the Mux to get output of inter-leaver process in order required.

- Note: clock divider is used as control in both Mux and Demux.

4.8.3 Advantage of Interleaver

- Errors typically occur in bursts rather than independently. If the number of errors within a code word exceeds the error-correcting code's capability, it fails to recover the original code word. Interleaving ameliorates this problem by shuffling source symbols across several code words, thereby creating a more uniform distribution of errors. Therefore, interleaving is wide-ly used for burst error-correction

4.8.4 Disadvantage of Interleaver

- Use of interleaving techniques increases total delay. This is because the entire interleaved block must be received before the packets can be decoded. Also Interleavers hide the structure of errors.

\

4.9 REED SOLOMON DECODER

Once the data is encoded and transmitted, it has to be received and decoded. The first step in this de-coding process is the calculation of syndrome values from the received message. These syndrome values are used to find the error locator polynomial $\Lambda_1 \ldots \Lambda_v$ and the error magnitude polynomial $\Omega_0 \ldots . \Omega_{v-1}$ through Euclidean algorithm. Now, the locations where errors exit are identified by Chien search method. The error values are corrected using the Forney's algorithm. Finally, the Forney's output is added (galois field addition of modulo 2) to the received message to get the error free transmitted message.

Errors can be added to the coded message polynomial, T(x), in the form of an error polynomial, E(x). Thus, the received polynomial, R(x), is given by:

$$R(x) = T(x) + E(x) \tag{4.3}$$

Where

$$E(x) = En-1xn-1 + ... + E1x + E0 \tag{4.4}$$

and each of the coefficients En-1 .... E0 is an m-bit error value, represented by an element of GF(2m), with the positions of the errors in the code word being determined by the degree of x for that term. Clearly, if more than t = (n-k)/2 of the E values are non-zero, then the correction capacity of the code is exceeded and the errors are not correctable.

4.9.1 The syndromes

Calculating the syndromes

It was shown that the transmitted code word is always divisible by the generator polynomial without remainder and that this property extends to the individual factors of the generator polynomial. Therefore, the first step in the decoding process is to divide the received polynomial by each of the factors (x + αi) of the generator polynomial. This produces a quotient and a remainder, that is:

$$\frac{R(x)}{x+\alpha^i} = Q_i(x) + \frac{S_i}{x+\alpha^i} \qquad for \quad b \le i \le b+2t-1 \tag{4.5}$$

where b is chosen to match the set of consecutive factors. The remainders Si resulting from these di-visions are known as the syndromes and, for b=0, can be written as S0 .... S2t-1.

Rearranging produces:

$$Si = Qi(x) * (x + \alpha i) + R(x) \qquad (4.6)$$

So that when x=αi this reduces to:

$$Si = R(\alpha i)$$

$$=Rn-1(\alpha i)n-1 + Rn-2(\alpha i)n-2 + \ldots + R1(\alpha i) + R0 \qquad (4.7)$$

where the coefficients Rn-1 .... R0 are the symbols of the received code word. This means that each of the syndrome values can also be obtained by substituting x = αi in the received polynomial, as an alternative to the division of R(x) by (x + αi) to form the remainder.

Error Locator Polynomial

The next step is to introduce the error locator polynomial. This turns out to be one of the more con-fusing steps in Reed-Solomon decoding because the literature defines two different, but related, expressions as the error locator polynomial. One form, often denoted σ(x), is constructed to have the error locators X1 .... Xv as its roots, that is, v factors of the form (x+Xj) for j= 1 to v. When expanded, these factors produce a polynomial of degree v with coefficients σ1 .... σv:

$$\sigma(x) = (x + X1)(x + X2)\ldots(x + Xv) = xv + \sigma 1xv-1 + \ldots + \sigma v-1x + \sigma v$$

The alternative form is usually denoted σ(x). This is constructed to have v factors of the form

(1+Xjx) and therefore has the **inverses** X1-1, ... , Xv-1 of the v error locators as its roots. When expanded, these factors produce a polynomial of degree v with coefficients σ1 .... σv:

$$\Lambda(x) = (1 + X1x)(1 + X2x)\ldots\ldots(1 + Xvx) = 1 + \Lambda 1x + \ldots + \Lambda v-1xv-1 + \Lambda vxv$$

However, it turns out that

$$\sigma(x) = x^{v} \times \Lambda\left(\frac{1}{x}\right) \qquad (4.8)$$

so the coefficients σ1 .... σv are the same as Λ1 .... Λv.

The Euclidean algorithm

Another efficient technique for obtaining the coefficients of the error location polynomial is based on Euclid's method for finding the highest common factor of two numbers. This uses the relationship between the errors and the syndromes expressed in the form of an equation based on polynomials.

This is also often referred to as the fundamental or key equation and requires two new polynomials, the syndrome and error magnitude polynomials, to be introduced.

The syndrome polynomial

For use in the key equation, the syndrome polynomial is defined as:

S(x) = Sb+2t-1x2t-1 + …. + Sb-1x + Sb (4.9)

where the coefficients are the 2t syndrome values calculated from the received code word using equation, or its equivalent for other values of b.

The error magnitude polynomial

The error magnitude polynomial can be written as:

Ω(x) = Ωv-1xv-1 + ... + Ω1x + Ω0 (4.10)

This is sometimes referred to as the error value or error evaluator polynomial.

The key equation

The key equation can then be written as:

Ω(x) = [S(x) Ω(x)] mod (x^2t)

where S(x) is the syndrome polynomial and Λ(x) is the error locator polynomial. Any terms of degree x2t or higher in the product are ignored, so that

$$\Omega_0 = S_b$$

$$\Omega_1 = S_{b+1} + S_b\Lambda_1$$

$$\vdots$$

$$\Omega_{v\text{-}1} = S_{b+v\text{-}1} + S_{b+v\text{-}2}\Lambda_1 + \ldots. + S_b\Lambda_{v\text{-}1} \qquad (4.11)$$

Applying Euclid's method to the key equation

Euclid's method can find the highest common factor d of two elements a and b, such that:

ua + vb = d .... (4.12)

where u and v are coefficients produced by the algorithm.

The product of S(x), which has degree 2t-1, and Λ(x), which has degree v, will have degree 2t+v-1. (4.13)

So, the product can be expressed as:

S(x) × Λ(x) = F(x) × x2t + Ω(x) (4.14)

in which the terms of x^2t and above are represented by the F(x) term and the remaining part is represented by Ω(x). This can be rearranged as:

Λ(x) × S(x) + F(x) × x2t = Ω(x) (4.15)

so that the known terms S(x) and x^2t correspond to the a and b terms. The algorithm then consists of dividing by S(x) to produce a remainder. S(x) then becomes the dividend and the remainder be-comes the divisor to produce a new remainder. This process is continued until the degree of the remainder becomes less than t. At this point, both the remainder Λ(x) and the multiplying factor Ω (x) are available as terms in the calculation.

Solving the error locator polynomial - the Chien search

Having calculated the coefficient values, Λ1 .... Λv, of the error locator polynomial, it is no possible to find its roots. If the polynomial is written in the form:

Λ(x) = $X_1(x + X_1^{-1})X_2(x + X_2^{-1})$ then clearly the function value will be zero if x =X_1^{-1},X_2^{-1} .... , that is:

$$x = \alpha^{-e}, \alpha^{-2e} \ldots\ldots\ldots\ldots\ldots\ldots$$

The roots, and hence the values of $X_1 \ldots X_v$ are found by trial and error, known as the Chien search, in which all the possible values of the roots (the field values α_i, $0 \leq i \leq n-1$) are substituted into (x) equation and the results evaluated. If the expression reduces to zero, then that value of x is a root and identifies the error position. Since the first symbol of the code word corresponds to the x_{n-1} term, the search begins with the value $\alpha_{-(n-1)}$ (=α_1), then $\alpha_{-(n-2)}$ (=α_2), and continues to α_0, which corresponds to the last symbol of the code word.

Calculating the error values by Forney algorithm

This is to calculate the error value Yj having established the error locator polynomial Λ(x) and the error value polynomial Ω(x). The algorithm makes use of the derivative of the error locator polynomial.

The derivative of the error locator polynomial

For a polynomial f(x) given by:

$$f(x) = 1 + f_1x + f_2x^2 + ... + f_vx^v$$

the derivative is given by:

$$f'(x) = f_1 + 2f_2x + ... + vf_vx^{v-1}$$

However, for the error locator polynomial $\Lambda(x)$, for $x = X_{j-1}$, the derivative reduces to:

$$\Lambda(X_{j-1}) = \Lambda_1 + \Lambda_3X_{j-2} + \Lambda_5X_{j-4} + ...$$

which amounts to setting even-powered terms of the error locator polynomial to zero and dividing through by $x = X_{j-1}$.

Forney's equation for the error magnitude

Methods of calculating the error values Y_1 .... Y_v based on Forney's algorithm are efficient. According to Forney's algorithm, the error value is given by:

$$Y_j = X_j^{1-b} \frac{\Omega(X_j^{-1})}{\Lambda'(X_j^{-1})} \qquad (4.16)$$

where (X_{j-1}) is the derivative of (x) for $x = X_j^{-1}$. When b=1, the X_j^{1-b}term disappears, so the formula is often quoted in the literature as simply $\Omega/\widehat{\Lambda}$, which gives the wrong results for b=0 and other values.

It should be noted that equation only gives valid results for symbol positions containing an error. If the calculation is made at other positions, the result is generally non-zero and invalid. The Chien search is therefore still needed to identify the error positions.

Error correction

Having located the symbols containing errors, identified by Xj, and calculated the values Yj of those errors, the errors can be corrected by adding the error polynomial E(x) to the received poly-nominal R(x). It should be remembered that conventionally the highest degree term of the received polynomial corresponds to the first symbol of the received code word.

Flow diagram for decoder

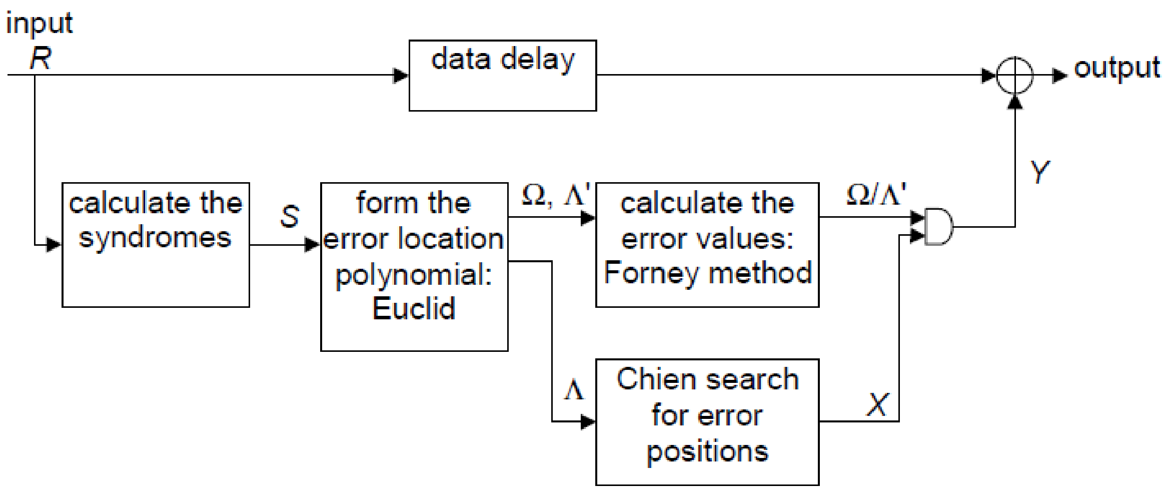

input
R
data delay
output
Y
calculate the syndromes
S
form the error location polynomial: Euclid
Ω, Λ'
calculate the error values: Forney method
Ω/Λ'
Λ
Chien search for error positions
X

Figure 4-9: Flow diagram for decoder

The Flow diagram of the RS decoder has the following steps:

- Generate the Galois field table using the given field generator polynomial, Also generate the Inverse Galois field table.
- Calculate the syndrome (using pipeline method)
- To obtain error locator and error magnitude polynomial using Extended Euclidean method.
- To obtain the error locations using the error locator polynomial by Chien search method
- To correct the errors and to obtain the correct values using Forney method.
- The Forney method output value is added (Galois addition) to the input data to get the final corrected output.

The Decoder receives the input data, performs the following steps mentioned above to detect and correct any errors if any.

4.9.2 Algorithm of RS decoder through a worked example – RS (15,11)

Let's consider RS (15,11) example.

Let the Input to Encoder be 1, 2….11

The output of encoder along with parity is 1,2,3,4,5,6,7,8,9,10,11,3,3,12,12

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

Now this is transmitted and received by the decoder. There might be errors introduced during trans-mission.

Consider couple of errors in the transmission.

The transmitted data is 1,2,3,4,5,11,7,8,9,10,11,3,**1**,12,12

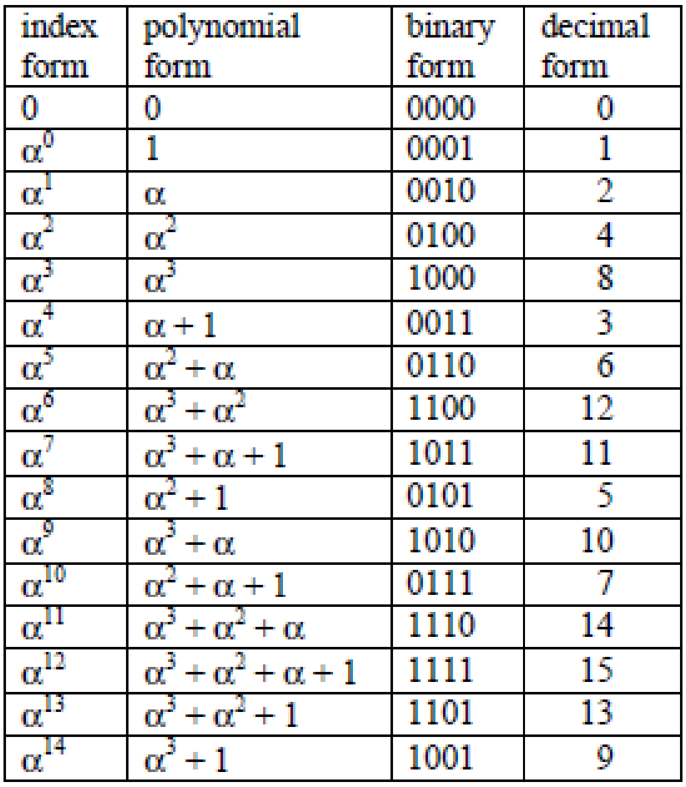
This is received by the decoder and let’s see how the decoder works and how it detects and corrects the errors.

Galois Field Table generation and Inverse Galois Field table generation

The generation of Galois field table is explained in the previous sections. So for this given example,

The Galois field table is as follows:

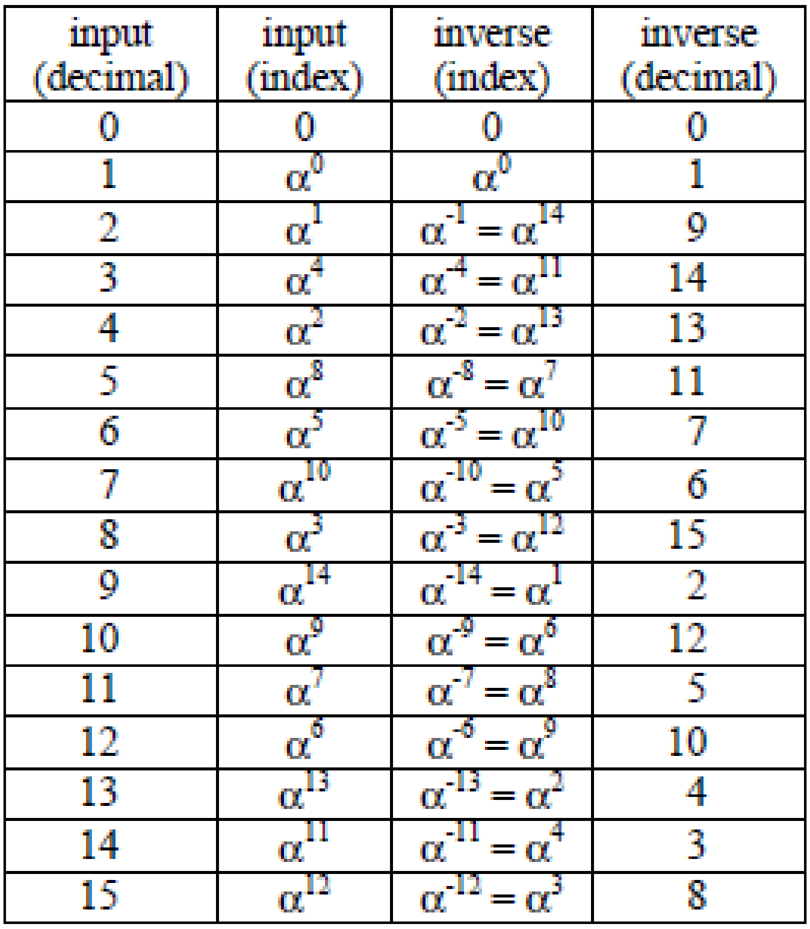
Table 4. 10 Galois field table for primitive polynomial x4+x+1



index form	polynomial form	binary form	decimal form
0	0	0000	0
α^0	1	0001	1
α^1	α	0010	2
α^2	α^2	0100	4
α^3	α^3	1000	8
α^4	$\alpha+1$	0011	3
α^5	$\alpha^2+\alpha$	0110	6
α^6	$\alpha^3+\alpha^2$	1100	12
α^7	$\alpha^3+\alpha+1$	1011	11
α^8	α^2+1	0101	5
α^9	$\alpha^3+\alpha$	1010	10
α^{10}	$\alpha^2+\alpha+1$	0111	7
α^{11}	$\alpha^3+\alpha^2+\alpha$	1110	14
α^{12}	$\alpha^3+\alpha^2+\alpha+1$	1111	15
α^{13}	$\alpha^3+\alpha^2+1$	1101	13
α^{14}	α^3+1	1001	9

The inverse Galois field table is as follows

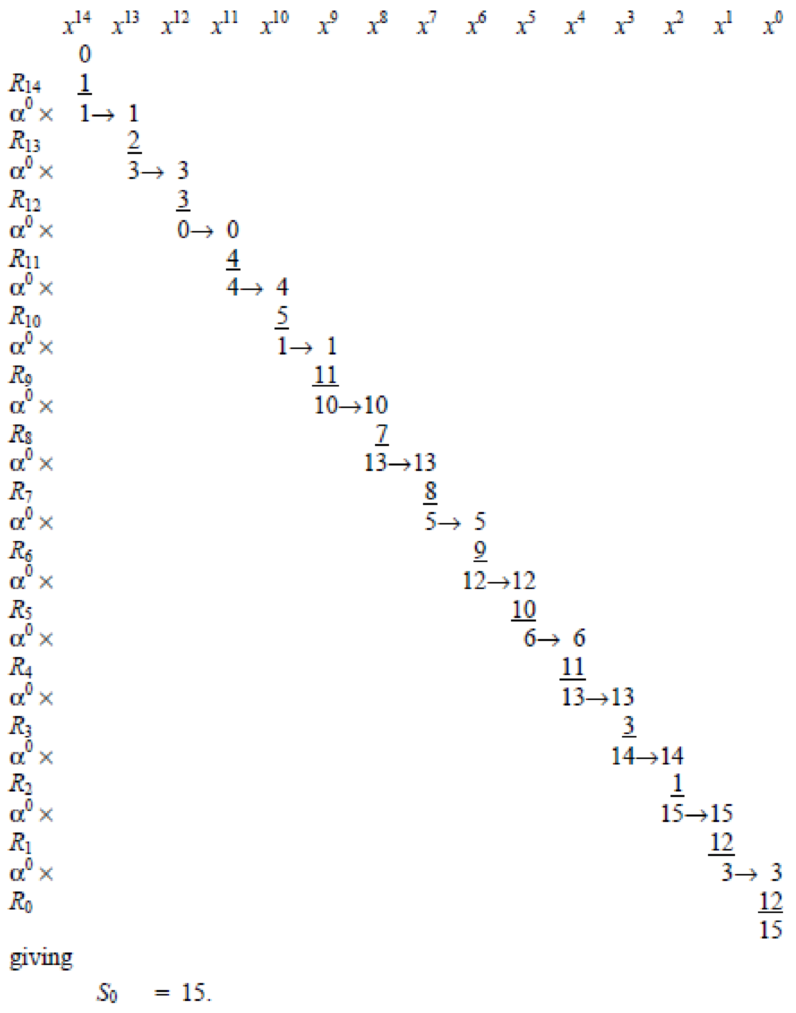
Table 4. 11 Inverse Galois Field table



input (decimal)	input (index)	inverse (index)	inverse (decimal)
0	0	0	0
1	α^0	α^0	1
2	α^1	$\alpha^{-1} = \alpha^{14}$	9
3	α^4	$\alpha^{-4} = \alpha^{11}$	14
4	α^2	$\alpha^{-2} = \alpha^{13}$	13
5	α^8	$\alpha^{-8} = \alpha^7$	11
6	α^5	$\alpha^{-5} = \alpha^{10}$	7
7	α^{10}	$\alpha^{-10} = \alpha^5$	6
8	α^3	$\alpha^{-3} = \alpha^{12}$	15
9	α^{14}	$\alpha^{-14} = \alpha^1$	2
10	α^9	$\alpha^{-9} = \alpha^6$	12
11	α^7	$\alpha^{-7} = \alpha^8$	5
12	α^6	$\alpha^{-6} = \alpha^9$	10
13	α^{13}	$\alpha^{-13} = \alpha^2$	4
14	α^{11}	$\alpha^{-11} = \alpha^4$	3
15	α^{12}	$\alpha^{-12} = \alpha^3$	8

4.9.3 Syndrome Calculation

For syndrome calculation, the pipelined approach is more suitable.



	x^{14}	x^{13}	x^{12}	x^{11}	x^{10}	x^{9}	x^{8}	x^{7}	x^{6}	x^{5}	x^{4}	x^{3}	x^{2}	x^{1}	x^{0}
	0														
R_{14}	$\underline{1}$														
$\alpha^0 \times$	$1 \rightarrow$	1													
R_{13}		$\underline{2}$													
$\alpha^0 \times$		$3 \rightarrow$	3												
R_{12}			$\underline{3}$												
$\alpha^0 \times$			$0 \rightarrow$	0											
R_{11}				$\underline{4}$											
$\alpha^0 \times$				$4 \rightarrow$	4										
R_{10}					$\underline{5}$										
$\alpha^0 \times$					$1 \rightarrow$	1									
R_9						$\underline{11}$									
$\alpha^0 \times$						$10 \rightarrow$	10								
R_8							$\underline{7}$								
$\alpha^0 \times$							$13 \rightarrow$	13							
R_7								$\underline{8}$							
$\alpha^0 \times$								$5 \rightarrow$	5						
R_6									$\underline{9}$						
$\alpha^0 \times$									$12 \rightarrow$	12					
R_5										$\underline{10}$					
$\alpha^0 \times$										$6 \rightarrow$	6				
R_4											$\underline{11}$				
$\alpha^0 \times$											$13 \rightarrow$	13			
R_3												$\underline{3}$			
$\alpha^0 \times$												$14 \rightarrow$	14		
R_2													$\underline{1}$		
$\alpha^0 \times$													$15 \rightarrow$	15	
R_1														$\underline{12}$	
$\alpha^0 \times$														$3 \rightarrow$	3
R_0															$\underline{12}$
															15

giving

$S_0 = 15.$

All the addition and multiplication are Galois field addition and multiplication.

Initially we take α0 and get S0, then we consider α1, α2, α3.

So we get

S1=3

S2=4

S3=12

So, the syndrome equation becomes $S(x) = 12x^3 + 4x^2 + 3x + 15$

4.9.4 Euclidean Algorithm

In this method, we obtain the error locator polynomial and error magnitude polynomial.

The Euclidean algorithm is used give the gcd of two numbers.

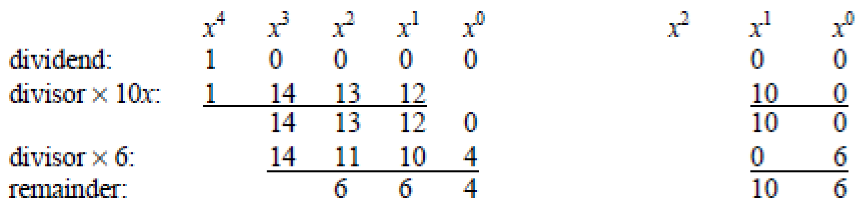
The Euclidean algorithm uses two facts

- gcd (0, a) =a
- gcd (a, b) =gcd (b, r)

where r=remainder when a is divided by b

The first step of the algorithm is to divide x2t (in this case x4) by S(x).

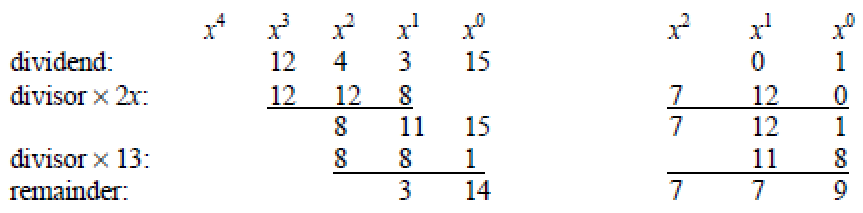
This involves multiplying S(x) × 10x (10 = 1/12) and subtracting (adding), followed by S(x) × 6 (6 = 14/12) and subtracting. This gives the remainder 6x2 + 6x + 4. In the right-hand process, the initial value 1 is multiplied by the same values used in the division process and added to an initial sum value of zero. So, the right-hand calculation produces 0 + 1 × (10x + 6) = 10x + 6.



	x^4	x^3	x^2	x^1	x^0	x^2	x^1	x^0
dividend:	1	0	0	0	0		0	0
divisor × 10x:	1	14	13	12			10	0
		14	13	12	0		10	0
divisor × 6:		14	11	10	4		0	6
remainder:			6	6	4		10	6

Having completed the first division, the degree of the remainder is not less than t (= 2), so we do a new division using the previous divisor as the dividend and the remainder as the divisor, that is, di-viding S(x) by the remainder 6x2 + 6x + 4. First the remainder is multiplied by 2x (2 = 12/6) and subtracted, then multiplied by 13 (13 = 8/6) and subtracted to produce the remainder 3x + 14. At the right-hand side, the previous initial value (1) becomes the initial sum value and the previous result (10x + 6) is multiplied by the values used in the division process. This produces 1 + (10x + 6)

× (2x + 13) = $7x^2 + 7x + 9$.



	x^4	x^3	x^2	x^1	x^0	x^2	x^1	x^0
dividend:		12	4	3	15		0	1
divisor × 2x:		12	12	8		7	12	0
			8	11	15	7	12	1
divisor × 13:			8	8	1		11	8
remainder:				3	14	7	7	9

In general, the process would continue repeating the steps described above, but now the degree of the remainder (= 1) is less than t (= 2) so the process is complete. The two results $7x^2 + 7x + 9$ and 3x + 14 are in fact γ×Λ(x) and γ×Ω(x), respectively, where in this case the constant factor γ=9. So, dividing through by 9 gives the polynomials in the defined forms:

$\Lambda(x) = 14x^2 + 14x + 1$ and $\Omega(x) = 6x + 15$.

Chien Search Method

This method is used to solve the error locator polynomial and to get the error positions. This is more by trial of all combinations. Where ever the sum is equal to zero, then error exists at that position.

To try the first position in the code word, corresponding to ej=14, we need to substitute α-14 into the error locator polynomial:

$$
\begin{aligned}
\Lambda(x) &= 14x^2 + 14x + 1\\
\Lambda(\alpha^{-14}) &= 14(\alpha^{-14})^2 + 14(\alpha^{-14}) + 1\\
&= 14(\alpha^1)^2 + 14(\alpha^1) + 1\\
&= \alpha^{11}\,\alpha^2 + \alpha^{11}\,\alpha^1 + \alpha^0\\
&= \alpha^{13} + \alpha^{12} + \alpha^0\\
&= 13 + 15 + 1\\
&= 3
\end{aligned}
$$

and the non-zero result shows that the first position does not contain an error.

For subsequent positions, the power of α to be substituted will advance by one for the x term and by two for the term. Substitute x as α-13, get sum. Do this till α0 and get the sum.

The following table contains the sum of all the corresponding x=αi values.

Table 4. 12 Terms in Chien search

x	x^2 term	x term	unity	sum
α^{-14}	α^{13}	α^{12}	1	3
α^{-13}	α^{0}	α^{13}	1	13
α^{-12}	α^{2}	α^{14}	1	12
α^{-11}	α^{4}	α^{0}	1	3
α^{-10}	α^{6}	α^{1}	1	15
α^{-9}	α^{8}	α^{2}	1	0
α^{-8}	α^{10}	α^{3}	1	14
α^{-7}	α^{12}	α^{4}	1	13
α^{-6}	α^{14}	α^{5}	1	14
α^{-5}	α^{1}	α^{6}	1	15
α^{-4}	α^{3}	α^{7}	1	2
α^{-3}	α^{5}	α^{8}	1	2
α^{-2}	α^{7}	α^{9}	1	0
α^{-1}	α^{9}	α^{10}	1	12
α^{0}	α^{11}	α^{11}	1	1

We see that the sum is zero at positions corresponding to 6th and 13th symbols (x_9 and x_2 terms). So there exist errors at those locations. This gives the solution to the error locator polynomial.

Forney Method

This method corrects the error value at the positions where the errors exist. These positions are obtained by the Chien Search method.

The Forney method consists of calculating the quotient of two polynomials, Ω(x) and Λ'(x), the derivative of Λ(x), for x = Xj-1. The derivative is obtained by setting even powers of x to zero in:

$$\Lambda(x) = 14x^2 + 14x + 1$$

and dividing by x, so that:

$$\Lambda'(X_j^{-1}.) = 14\, X_j^{-1}.\, /\, X_j^{-1}. = 14.$$

$$Y_j = X_j \frac{6X_j^{-1} + 15}{14} \qquad (4.17)$$

Knowing the positions of the errors as the 6th (x9 term) and 13th (x2 term) the error values can be calculated for Xj = α9 as:

$$Y_j = \alpha^9 \frac{6\alpha^{-9} + 15}{14} = 13 \qquad (4.18)$$

And for Xj = α2

$$Y_j = \alpha^2 \frac{6\alpha^{-2} + 15}{14} = 2 \qquad (4.19)$$

So, the output at the end of Forney method, the error vector is 0,0,0,0,0,13,0,0,0,0,0,0,2,0,0

Correction of Errors

The error values and positions can be formed into an error vector and added to the received code word to produce the corrected message:

x^{14}	x^{13}	x^{12}	x^{11}	x^{10}	x^9	x^8	x^7	x^6	x^5	x^4	x^3	x^2	x^1	x^0
0	0	0	0	0	13	0	0	0	0	0	0	2	0	0
$\underline{1}$	$\underline{2}$	$\underline{3}$	$\underline{4}$	$\underline{5}$	$\underline{11}$	$\underline{7}$	$\underline{8}$	$\underline{9}$	$\underline{10}$	$\underline{11}$	$\underline{3}$	$\underline{1}$	$\underline{12}$	$\underline{12}$
1	2	3	4	5	6	7	8	9	10	11	3	3	12	12

So, when the error vector is added to the data received by the decoder, we observe that the error gets corrected to the original output of encoded data. Hence error detection and correction take place

Chapter 5

RESULTS AND DISCUSSIONS

The hardware implementation RS encoder, convolutional encoder and Viterbi decoder were designed in the previous chapters. The results and waveform for the same are shown below.

The frequency of operation for the hardware encoders and decoder is given below:

- RS Encoder- 104 MHz

5.1 GENERAL REED-SOLOMON ENCODER(VHDL)

5.1.1 (n, k) = (15,11) RS code

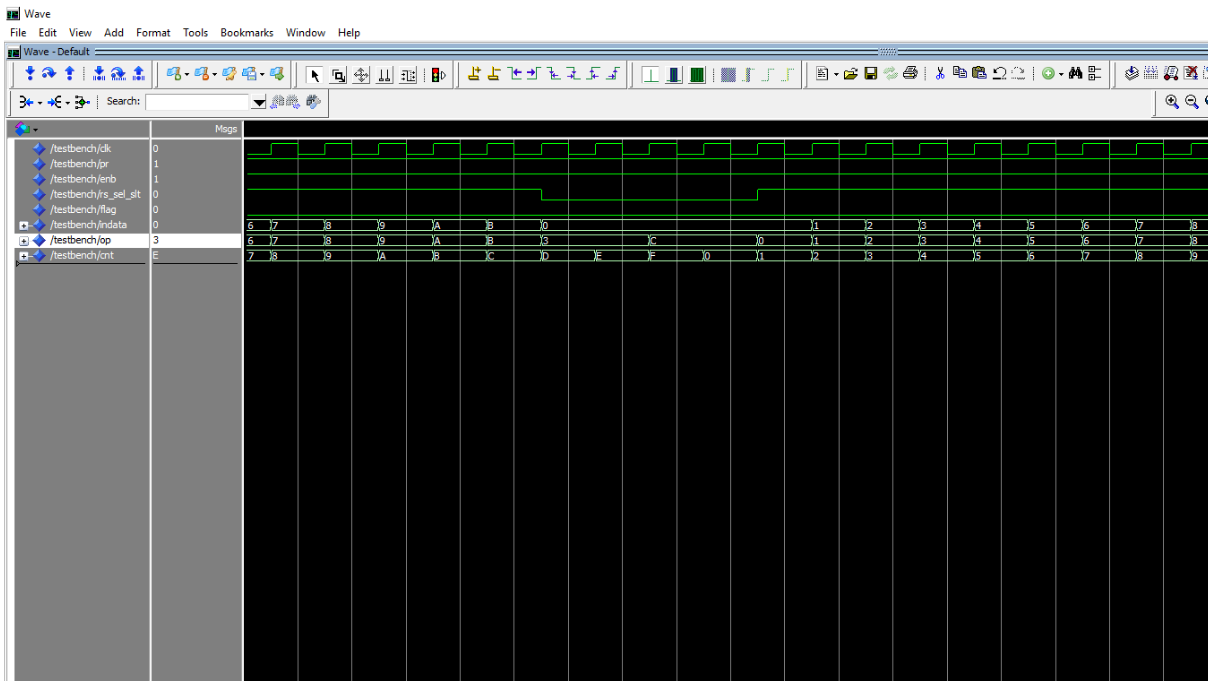


Figure 6-1: Parity symbols for (15,11)

- m=4 => 4-bit symbol
- (15,11) code
- Generator Polynomial = $x^4+10x^3+5x^2+10x+1$
- 11 information symbols – 1,2,3,4,5,6,7,8,9,10,11
- Parity – 3, 2, 10, 5

5.1.2 (n, k) = (255,239) RS codes

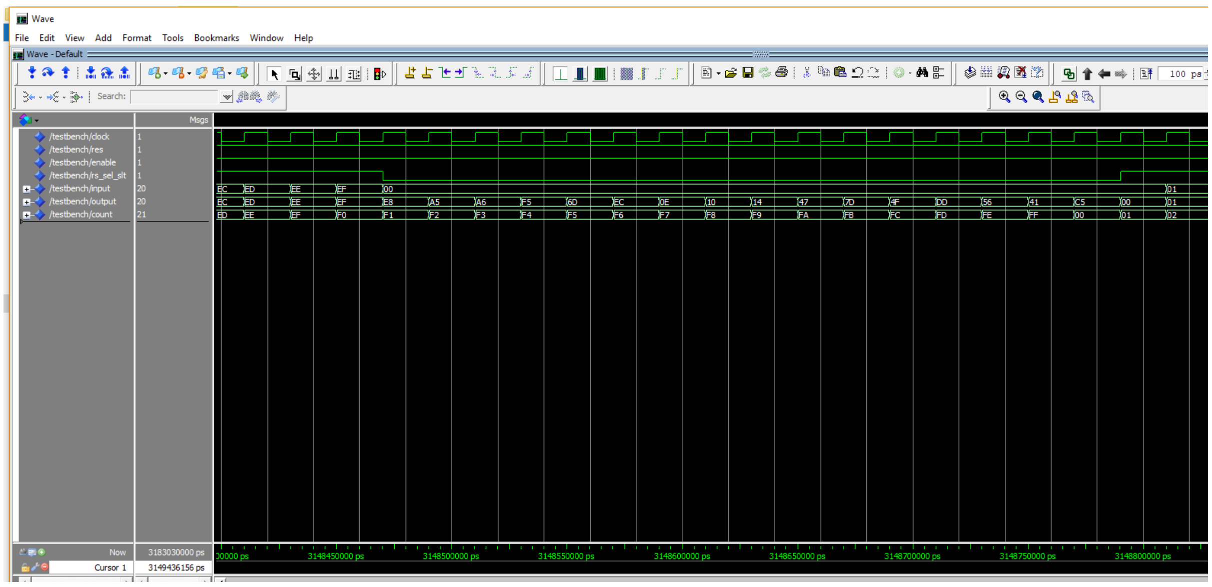
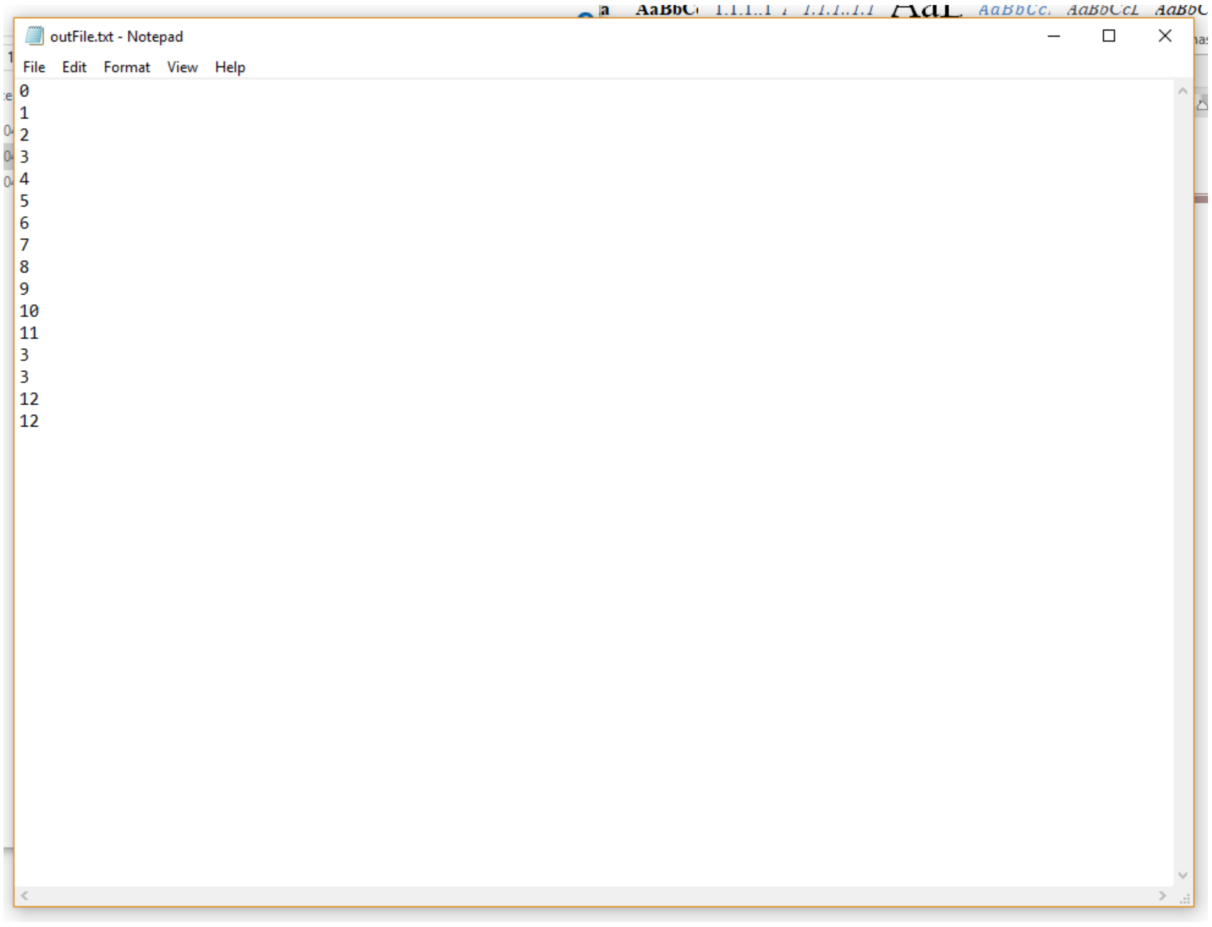


Figure 6-2: Parity symbols for (255,239)

- m=8 => 8-bit symbol
- (255,239) code
- Generator Polynomial = $x^{16}+165x^{15}+105x^{14}+127x^{13}+159x^{12}+104x^{11}+152x^{10}+101x^{9}+74x^{8}+101x^{7}+152x^{6}+104x^{5}+159x^{4}+127x^{3}+105x^{2}+165x+1$
- 239 information symbols – 1 to 239
- Parity – 232,165,166,245,109,236,14,16,20,71,125,19,221,86,65,197

5.2 REED SOLOMON ENCODER(SOFTWARE)

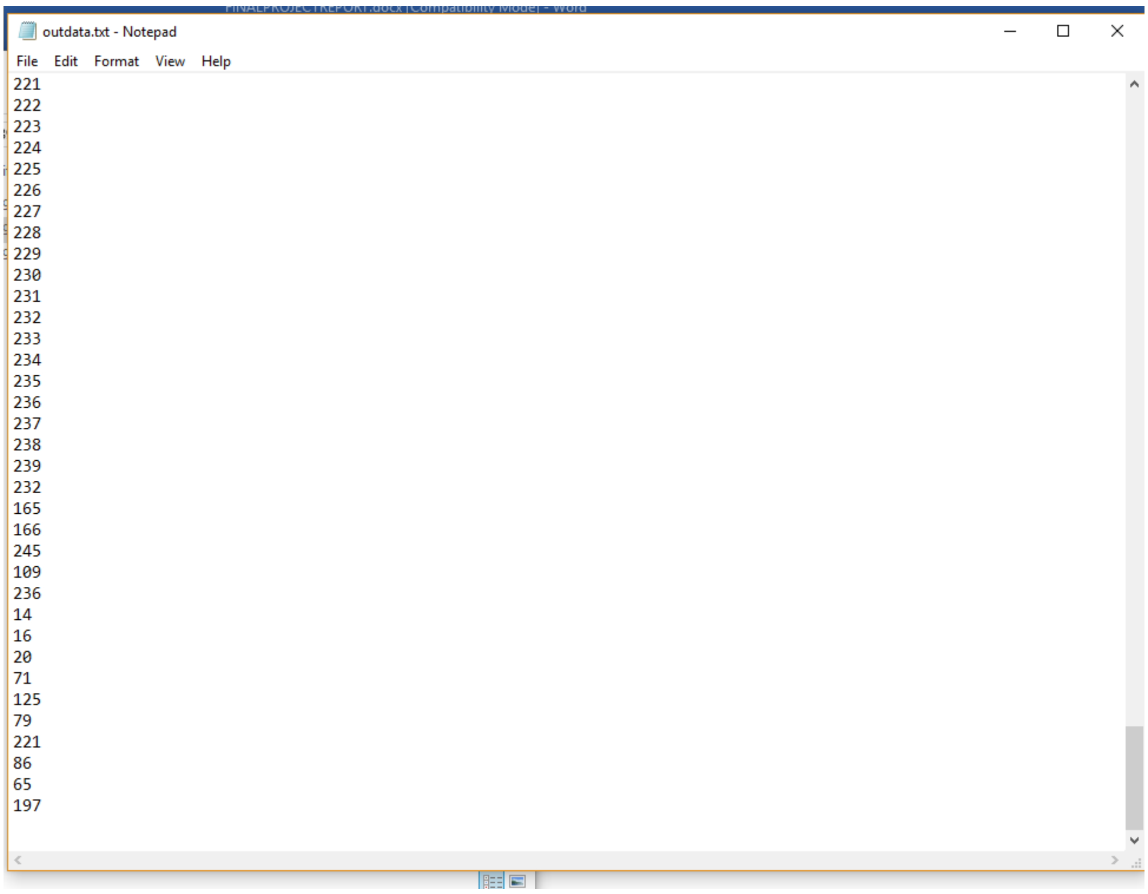
5.2.1 (n, k) = (15,11) RS code



```
0
1
2
3
4
5
6
7
8
9
10
11
3
3
12
12
```

Figure 6-3: Encoder output

5.2.2 (n, k) = (255,239) RS codes

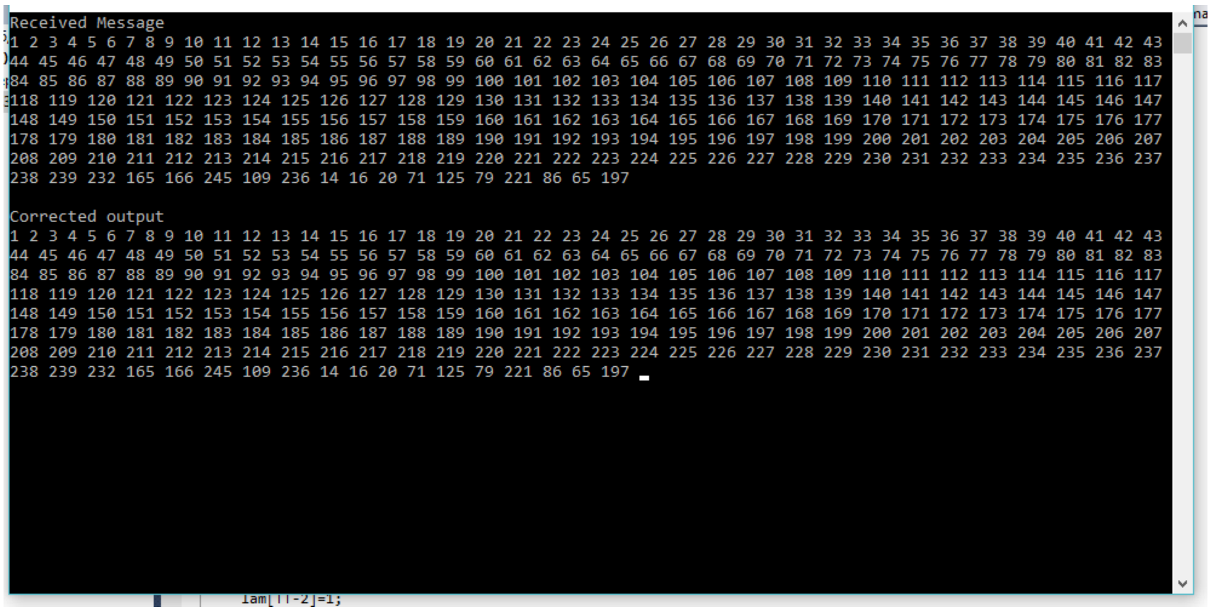


```
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
232
165
166
245
109
236
14
16
20
71
125
79
221
86
65
197
```

Figure 6-4:Encoder output

5.3 REED-SOLMON DECODER (SOFTWARE)

5.3.1 RS Decoder (255,239)

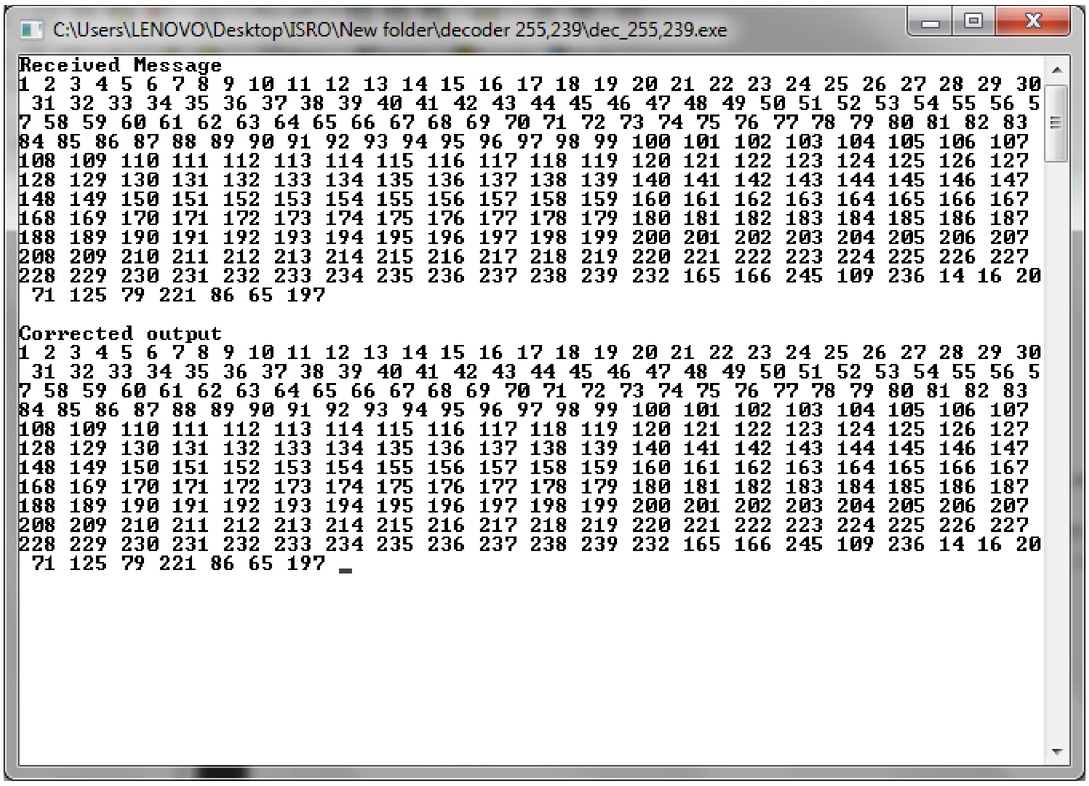


```
Received Message
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117
118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147
148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177
178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207
208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237
238 239 232 165 166 245 109 236 14 16 20 71 125 79 221 86 65 197

Corrected output
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117
118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147
148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177
178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207
208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237
238 239 232 165 166 245 109 236 14 16 20 71 125 79 221 86 65 197
```

Figure 6-5: Decoder output RS(255,239)

The encoded data is stored in a file. This data is given as input to the decoder. The de-coder detects if any error exists, it finds the location of the error and corrects it. The corrected output is again written back into a file.

CASE I) When no error in the Input data



C:\Users\LENOVO\Desktop\ISRO\New folder\decoder 255,239\dec_255,239.exe

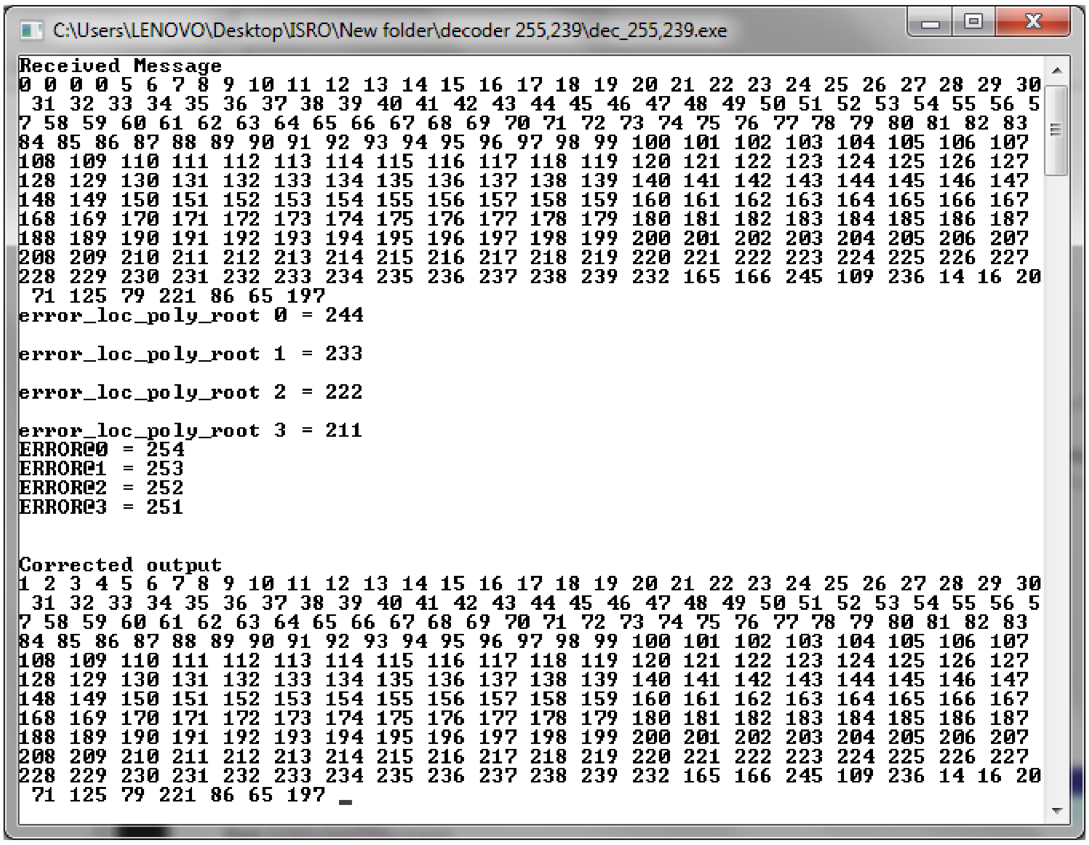
```
Received Message
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 5
7 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127
128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147
148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167
168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187
188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207
208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227
228 229 230 231 232 233 234 235 236 237 238 239 232 165 166 245 109 236 14 16 20
 71 125 79 221 86 65 197

Corrected output
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 5
7 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127
128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147
148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167
168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187
188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207
208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227
228 229 230 231 232 233 234 235 236 237 238 239 232 165 166 245 109 236 14 16 20
 71 125 79 221 86 65 197 _
```

Figure 6-6: Decoder output RS(255,239) No Error.

The decoded output is the same as encoded input.

CASE II) When errors are less than (n-k)/2 (that is 8)

~When the first 4 encoded data is corrupted.


C:\Users\LENOVO\Desktop\ISRO\New folder\decoder 255,239\dec_255,239.exe

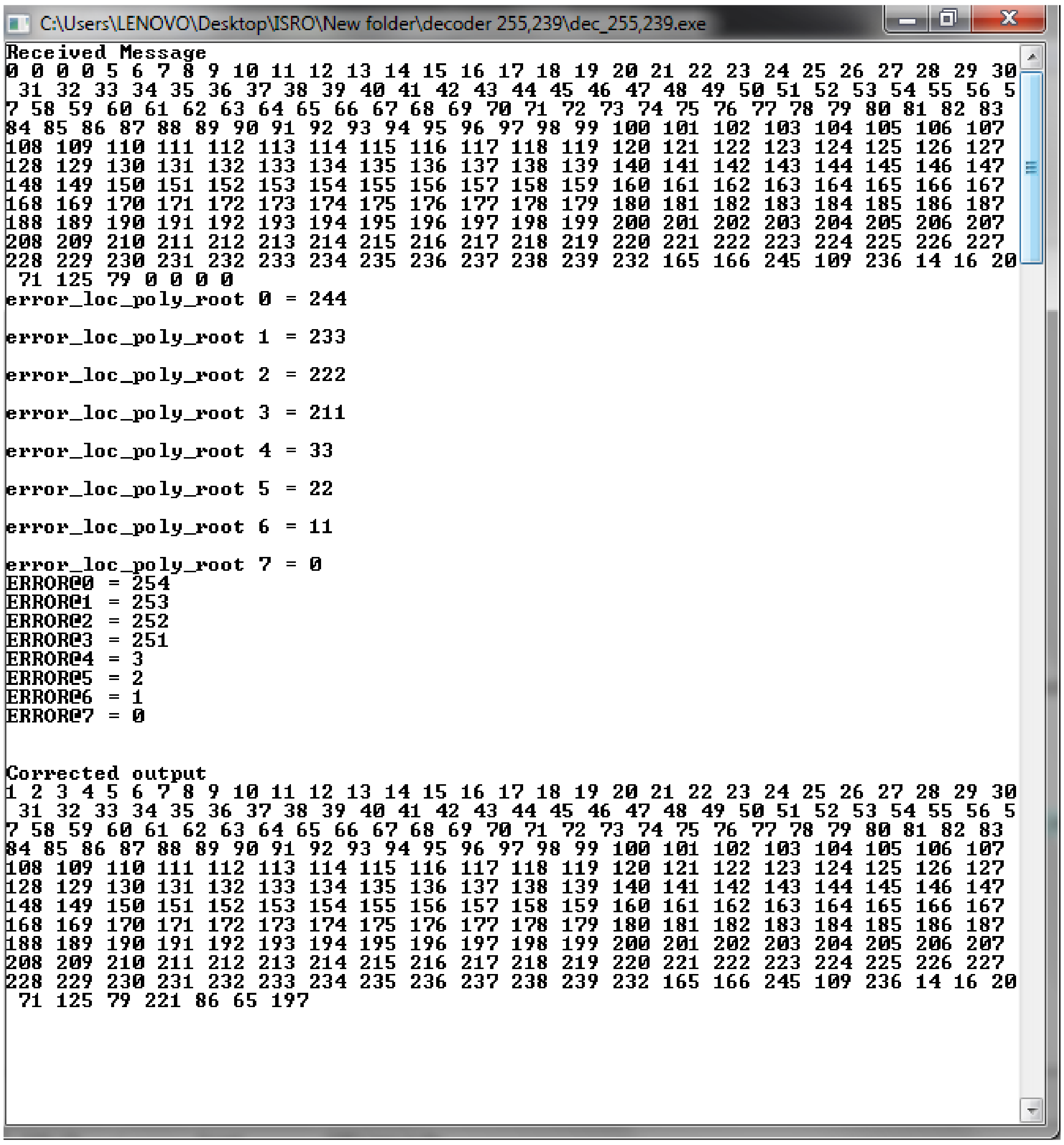
```
Received Message
0 0 0 0 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 5
7 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127
128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147
148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167
168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187
188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207
208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227
228 229 230 231 232 233 234 235 236 237 238 239 232 165 166 245 109 236 14 16 20
 71 125 79 221 86 65 197
error_loc_poly_root 0 = 244

error_loc_poly_root 1 = 233

error_loc_poly_root 2 = 222

error_loc_poly_root 3 = 211
ERROR@0 = 254
ERROR@1 = 253
ERROR@2 = 252
ERROR@3 = 251


Corrected output
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 5
7 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127
128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147
148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167
168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187
188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207
208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227
228 229 230 231 232 233 234 235 236 237 238 239 232 165 166 245 109 236 14 16 20
 71 125 79 221 86 65 197 _
```

Figure 6-7: Decoder output RS (255,239) 4 or less error.

The first 4 corrupted symbols are detected and corrected.

CASE III) When no errors is equal to (n-k)/2 (that is 8)

~ When the first 4 encoded data and the last 4 parity symbols are corrupted.



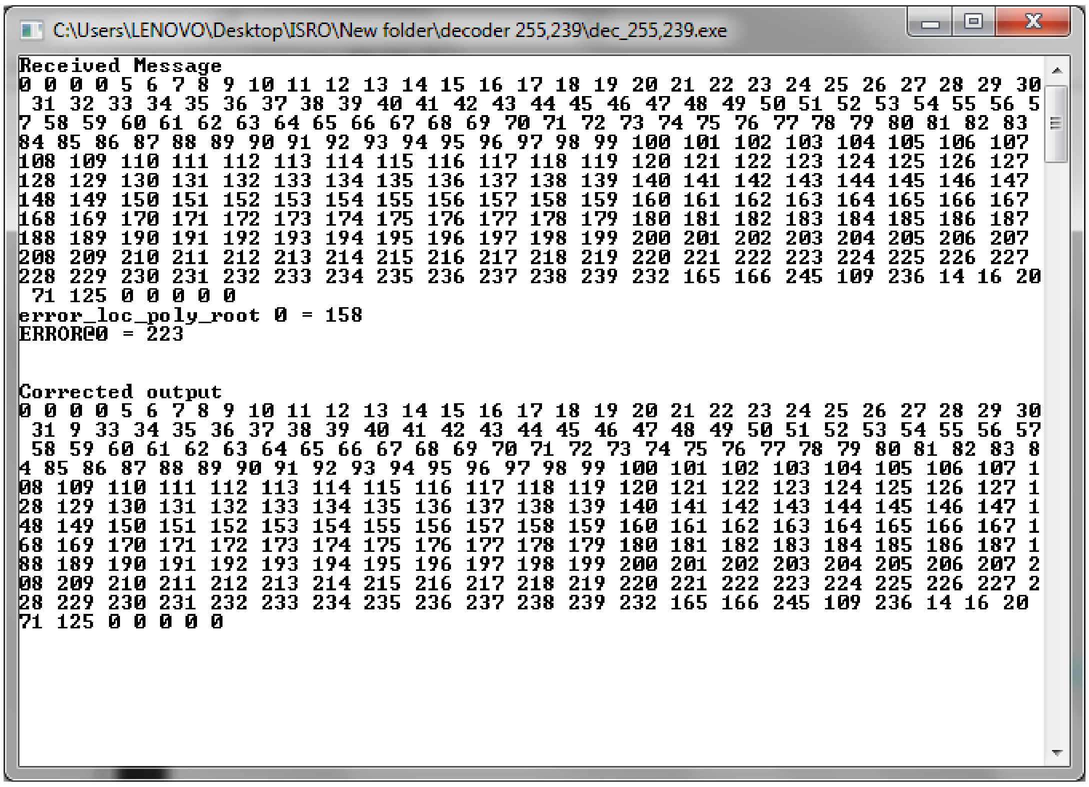
C:\Users\LENOVO\Desktop\ISRO\New folder\decoder 255,239\dec_255,239.exe

```
Received Message
0 0 0 0 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 5
7 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127
128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147
148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167
168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187
188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207
208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227
228 229 230 231 232 233 234 235 236 237 238 239 232 165 166 245 109 236 14 16 20
 71 125 79 0 0 0 0
error_loc_poly_root 0 = 244

error_loc_poly_root 1 = 233

error_loc_poly_root 2 = 222

error_loc_poly_root 3 = 211

error_loc_poly_root 4 = 33

error_loc_poly_root 5 = 22

error_loc_poly_root 6 = 11

error_loc_poly_root 7 = 0
ERROR@0  = 254
ERROR@1  = 253
ERROR@2  = 252
ERROR@3  = 251
ERROR@4  = 3
ERROR@5  = 2
ERROR@6  = 1
ERROR@7  = 0


Corrected output
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 5
7 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127
128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147
148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167
168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187
188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207
208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227
228 229 230 231 232 233 234 235 236 237 238 239 232 165 166 245 109 236 14 16 20
 71 125 79 221 86 65 197
```

Figure 6-8: Decoder output RS (255,239) 4 or more error.

CASE IV) When no of errors is greater than (n-k)/2 (that is 8)

~When the first 4 encoded input and the last 5 parity symbols are corrupted.

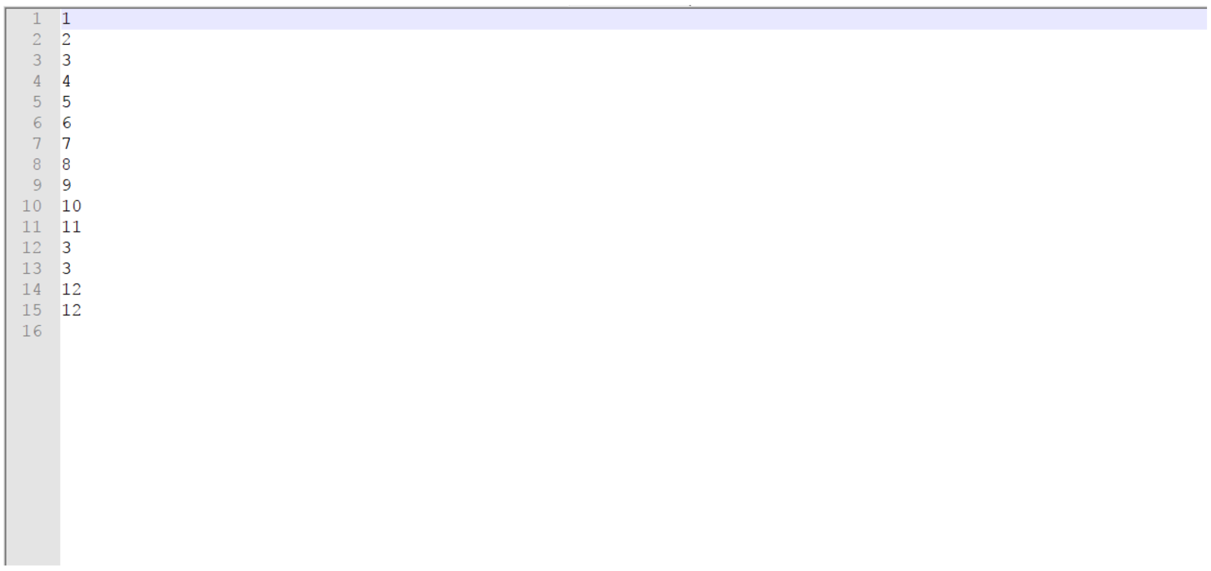

C:\Users\LENOVO\Desktop\ISRO\New folder\decoder 255,239\dec_255,239.exe

```
Received Message
0 0 0 0 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 5
7 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127
128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147
148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167
168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187
188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207
208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227
228 229 230 231 232 233 234 235 236 237 238 239 232 165 166 245 109 236 14 16 20
 71 125 0 0 0 0 0
error_loc_poly_root 0 = 158
ERROR@0 = 223


Corrected output
0 0 0 0 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 31 9 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 8
4 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 1
08 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 1
28 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 1
48 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 1
68 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 1
88 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 2
08 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 2
28 229 230 231 232 233 234 235 236 237 238 239 232 165 166 245 109 236 14 16 20
71 125 0 0 0 0 0
```

Figure 6-9: Decoder output RS(255,239) 8 error.

Conclusion: The RS (255,239) Decoder can detect upto 16 errors, but it can correct up to only 8 errors. This is verified by the above cases for different number of errors in the encoded input to the decoder.

5.3.2 RS Decoder (15,11)



```
1
2
3
4
5
6
7
8
9
10
11
3
3
12
12
```

Figure 6-10: Decoder output RS (15,11)

Conclusion: The RS (15,11) Decoder can detect upto 4 errors, but it can correct up to only 8 errors. This is verified by the above cases for different number of errors in the encoded input to the decoder.

CONCLUSION AND FUTURE SCOPE

CONCLUSION

Design, Construction and C/FPGA Implementation of RS (255,239) encoder and RS (15,11) was successfully done and the output was also verified using the Software Implementation of RS (255,239) Decoder.

The propagation delay in VHDL post layout simulation for encoder and was within the acceptable range of 2ns. The estimated frequency is seen to be much higher than the requested frequency. This ensures that there is a positive slack as seen from the hardware estimate section. It was found that if the error is in parity symbols even then the decoder is able to detect the output and it is of no matter to the decoder that in which symbols the error is present. The decoder first corrects the symbols and then removes the redundant parity symbols from the code word and produces the original input code word.

FUTURE SCOPE

The CD player is just one of the many commercial, mass applications of the Reed-Solomon codes. The commercial world is becoming increasingly mobile, while simultaneously demanding reliable, rapid access to sales, marketing, and accounting information. Unfortunately, the mobile channel is a problematic environment with deep fades an ever-present phenomenon. Reed-Solomon codes are the best solution to this problem. There is no other error control system that can match their reliability performance in the mobile environment.

The optical channel provides another set of problems altogether. Shot noise and a dispersive, noisy medium plague line-of-sight optical system, creating noise bursts that are best handled by Reed-Solomon codes. As optical fibres see increased use in high-speed multiprocessors, Reed-Solomon codes can be used there as well.

Reed-Solomon codes will continue to be used to force communication system performance ever closer to the line drawn by Shannon.

REFERENCE

1. CCSDS TM synchronization and channel coding CCSDS 130.1 1-G-2 November,2012

2. BBC R&D white paper – Reed Solomon error correction by C.K.P Clarke

3. The Art of error correction by Robert h Morelos-Zaragoza

4. Error Control Coding: Fundamentals and Applications by Shu Lin and Daniel J. Costello, Jr.

5. E. Prange, "Cyclic Error-Correcting Codes in Two Symbols," Air Force Cam-bridge Research Center-TN-57-103, Cambridge, Mass., September 1957.

6. I. S. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields," SI AM Journal of Applied Mathematics, Volume 8, pp. 300-304, 1960.

7. R. T. Chien, "Cyclic Decoding Procedure for the Bose- Chaudhuri-Hocquenghem Codes," IEEE Transactions on Information Theory, Volume IT-10, pp. 357-363, October 1964.

8. G. D. Forney, "On Decoding BCH Codes" IEEE Transactions on Information Theory, Volume IT-11, pp. 549-557, October 1965.

9. Lin Shu, “An Introduction to Error-Correcting Codes”, Prentice Hall, Inc., 1970.

1. B Bose, “Burst Unidirectional Error Detecting Codes”, IEEE Trans. Compute, C35,350 -353, 1989.