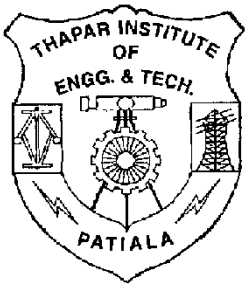
"VHDL IMPLEMENTATION OF REED – SOLOMON CODES"

The Thesis submitted in partial fulfillment of the requirements for the award of the degree of

MASTER OF ENGINEERING
IN
ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted By:
SANDEEP KAUR
Roll No-8044122

Under the guidance of
Mr. SANJAY SHARMA
(ASST.PROF. TIET, PATIALA)



DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

THAPAR INSTITUTE OF ENGINEERING & TECHNOLOGY (DEEMED UNIVERSITY), PATIALA – 147004.

YEAR 2006

ACKNOWLEDGEMENT

Words are often too less to reveal one's deep regards. An understanding of the work like this is never the outcome of the efforts of a single person. I take this opportunity to express my profound sense of gratitude and respect to all those who helped me through the duration of this thesis.

First, I would like to thank the Supreme Power, the GOD, who guided me to work on the right path of life. Without his grace, this would not have been possible. This work would not have been possible without the encouragement and able guidance of **'Mr. SANJAY SHARMA'**, Assistant Professor, TIET, PATIALA. Their enthusiasm and optimism made this experience both rewarding and enjoyable. Most of the novel ideas and solutions found in this thesis are the result of our numerous stimulating discussions. His feedback and editorial comments were also invaluable for the writing of this thesis. I am grateful to **Head of the Department Dr. R. S. KALER** and **Dr. A.K. CHATTERJEE** for providing the facilities for the completion of thesis.

I take pride of my self being daughter of ideal great parents whose everlasting desire, sacrifice, affectionate blessing and help without which it would have not been possible for me to complete my studies.

At last, I would like to thank all the members and employees of Electronics and Communication Department, TIET Patiala whose love and affection made this possible.

(SANDEEP KAUR)

ABSTRACT

Channel coding is used for providing reliable information through the transmission channel to the user .It is an important operation for the digital communication system transmitting digital information over a noisy channel. Channel coding can use either Automatic Repeat request or Forward Error Correction technique depending on the properties of the system or on the application in which the error correcting is to be introduced.

Error – control coding techniques are based on the addition of redundancy to the information message according to a prescribed rule thereby providing data a higher bit rate. This redundancy is exploited by decoder at the receiver end to decide which message bit was actually transmitted. The combined goal of the channel encoder and the decoder is to minimize the channel noise. Block codes and convolutional codes are two main methods to introduce error – correcting codes.

Reed – Solomon codes are an important sub – class of nonbinary BCH codes. These are cyclic codes and are very effectively used for the detection and correction of burst errors. Galois field arithmetic is used for encoding and decoding of Reed – Solomon codes. Galois field multipliers are used for encoding the information block. At the decoder, the syndrome of the received codeword is calculated using the generator polynomial to detect errors. Then to correct these errors, an error locator polynomial is calculated. From the error locator polynomial, the location of the error and its magnitude is obtained. Consequently a correct codeword is obtained.

Block lengths and symbol sizes can be readily adjusted to accommodate a wide range of message sizes. Reed – Solomon codes provides a wide range of code values that can be chosen to optimize performance.

VHDL implementation creates a flexible, fast method and high degree of parallelism for implementing the Reed – Solomon codes. The language can be used as an exchange medium between the chip vendors and the CAD tool users. Different chip vendors can provide VHDL description of their components to system designers. CAD tool users can use it to capture the behavior of the design. Also various digital modeling techniques such as finite state machine descriptions, algorithmic descriptions, and Boolean equations are modeled using the language.

The results constitute simulation of VHDL codes of different modules of the Reed – Solomon codes in MODELSIM SE 5.5c. The results demonstrate that the Reed – Solomon codes are very efficient for the detection and correction of burst errors.

TABLE OF CONTENTS

CHAPTER TITLE PAGE No.

LIST OF FIGURES

FIGURE No.	NAME OF FIGURE	PAGE No.

CHAPTER 1

INTRODUCTION

Digital communication system is used to transport an information bearing signal from the source to a user destination via a communication channel. The information signal is processed in a digital communication system to form discrete messages which makes the information more reliable for transmission. Channel coding is an important signal processing operation for the efficient transmission of digital information over the channel. It was introduced by Claude E. Shannon in 1948 by using the channel capacity as an important parameter for error - free transmission. In channel coding the number of symbols in the source encoded message is increased in a controlled manner in order to facilitate two basic objectives at the receiver: error detection and error correction.

Error detection and error correction to achieve good communication is also employed in electronic devices. It is used to reduce the level of noise and interferences in electronic medium. The amount of error detection and correction required and its effectiveness depends on the signal to noise ratio (SNR).

1.1 OVERVIEW

Every information signal has to be processed in a digital communication system before it is transmitted so that the user at the receiver end receives an error – free information.

A digital communication system has three basic signal processing operations: source coding, channel coding and modulation. A digital communication system is shown in the block diagram given below:

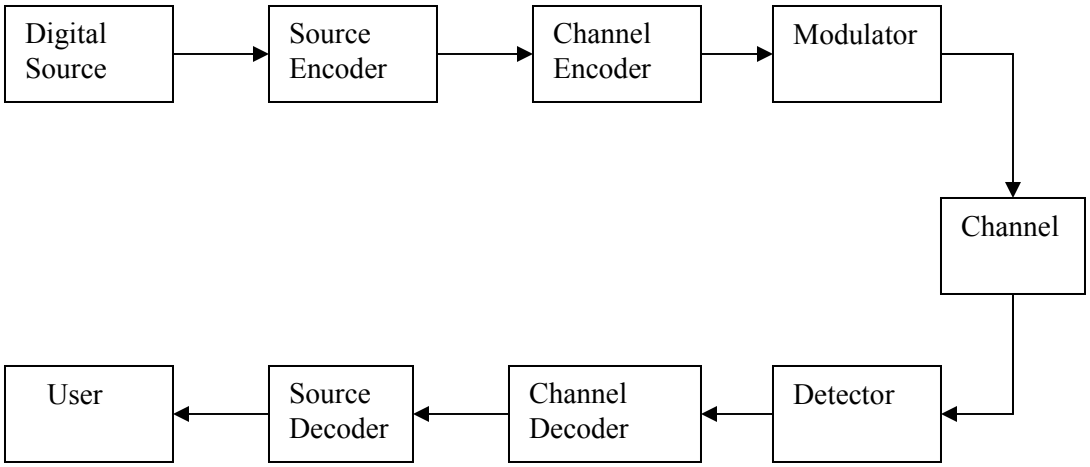


Figure-1.1: Block Diagram of a Digital Communication System

In source coding, the encoder maps the digital generated at the source output into another signal in digital form. The objective is to eliminate or reduce redundancy so as to provide an efficient representation of the source output. Since the source encoder mapping is one-to – one, the source decoder on the other end simply performs the inverse mapping, thereby delivers to the user a reproduction of the original digital source output. The primary benefit thus gained from the application of source coding is a reduced bandwidth requirement.

In channel coding, the objective for the encoder is to map the incoming digital signal into a channel input and for the decoder is to map the channel output into an output signal in such a way that the effect of channel noise is minimized. That is the combined role of the channel encoder and decoder is to provide for a reliable communication over a noisy channel. This provision is satisfied by introducing redundancy in a prescribed fashion in the channel encoder and exploiting it in the decoder to construct the original encoder input as accurately as possible. Thus in source coding, redundant bits are removed whereas in channel coding, redundancy is introduced in a controlled manner.

Then modulation is performed for the efficient transmission of the signal over the channel. Various digital modulation techniques could be applied for modulation such as Amplitude Shift Keying (ASK), Frequency- Shift Keying (FSK) or Phase – Shift Keying (PSK).

The addition of redundancy in the coded messages implies the need for increased transmission bandwidth. Moreover, the use of coding adds complexity to the system, especially for the implementation of decoding operations in the receiver. Thus bandwidth and system complexity has to be considered in the design trade – offs in the use of error - control coding to achieve acceptable error performance

Different errors correcting codes can be used depending on the properties of the system and the application in which the error correcting is to be introduced. Generally error – correcting codes have been classified into block codes and convolutional codes. The distinguishing feature for the classification is the presence or absence of memory in the encoders for the two codes. To generate a block code, the incoming information stream is divided into blocks and each block is processed individually by adding redundancy in accordance with a prescribed algorithm. The decoder processes each block individually and corrects errors by exploiting redundancy.
Many of the important block codes used for error – detection are cyclic codes. These are also called cyclic redundancy check codes.

In a convolutional code, the encoding operation may be viewed as the discrete – time convolution of the input sequence with the impulse response of the encoder. The duration of the impulse response equals the memory of the encoder. Accordingly, the encoder for a convolutional code operates on the incoming message sequence, using a "sliding window" equal in duration to its own memory. Hence in a convolutional code, unlike a block code where codewords are produced on a block- by – block basis, the channel encoder accepts message bits as continuous sequence and thereby generates a continuous sequence of encoded bits at a higher rate.

1.2 LITERATURE SURVEY

Channel coding is a widely used technique for the reliable transmission and reception of data. Generally systematic linear cyclic codes are used for channel coding. In 1948, Shannon introduced the linear block codes for complete correction of errors [1]. Cyclic codes were first discussed in a series of technical notes and reports written between 1957 and 1959 by Prange [2], [3], [4]. This led directly to the work published in March and September of 1960 by Bose and Ray-Chaudhuri the BCH codes. [5], [6], [7]. In 1959, Irving Reed and Gus Solomon described a new class of error-correcting codes called Reed-Solomon codes [8]. Originally Reed-Solomon codes were constructed and decoded through the use of finite field arithmetic [9], [10] which used nonsingular Vandermonde matrices [10]. In 1964 Singleton showed that this was the best possible error correction capability for any code of the same length and dimension [11]. Codes that achieve this "optimal" error correction capability are called Maximum Distance Separable (MDS). Reed-Solomon codes are by far the dominant members, both in number and utility, of the class of MDS codes. MDS codes have a number of interesting properties that lead to many practical consequences.

The generator polynomial construction for Reed-Solomon codes is the approach most commonly used today in the error control literature. This approach initially evolved independently from Reed-Solomon codes as a means for describing cyclic codes. Gorenstein and Zierler then generalized Bose and Ray-Chaudhuri's work to arbitrary Galois fields of size p^m, thus developing a new means for describing Reed and Solomon's "polynomial codes" [12]. It was described that vector c is a code word in the code defined by $g(x)$ if and only if its corresponding code polynomial $c(x)$ is a multiple of $g(x)$. So the information symbols could be easily mapped onto code words. All valid code polynomials are multiples of the generator polynomial. It follows that any valid code polynomial must have as roots the same $2t$ consecutive powers of α that form the roots of $g(x)$. This approach leads to a powerful and efficient set of decoding algorithms.

After the discovery of Reed-Solomon codes, a search began for an efficient decoding algorithm. In 1960, Reed and Solomon proposed a decoding algorithm based on the solution of sets of simultaneous equations [8].Though much more efficient than a look-up table, Reed and

Solomon's algorithm is still useful only for the smallest Reed-Solomon codes. In 1960 Peterson provided the first explicit description of a decoding algorithm for binary BCH codes [13], His "direct solution" algorithm is quite useful for correcting small numbers of errors but becomes computationally intractable as the number of errors increases. Peterson's algorithm was improved and extended to non - binary codes by Gorenstein and Zierler (1961) [12], Chien (1964) [14], and Forney (1965) [15]. These efforts were productive, but Reed-Solomon codes capable of correcting more than six or seven errors still could not be used in an efficient manner.

In 1967, Berlekamp demonstrated his efficient decoding algorithm for both non - binary BCH and Reed-Solomon codes [16], [17]. Berlekamp's algorithm allows for the efficient decoding of dozens of errors at a time using very powerful Reed-Solomon codes. In 1968 Massey showed that the BCH decoding problem is equivalent to the problem of synthesizing the shortest Linear Feedback Shift Register capable of generating a given sequence [18]. Massey then demonstrated a fast shift register-based decoding algorithm for BCH and Reed-Solomon codes that is equivalent to Berlekamp's algorithm. This shift register-based approach is now referred to as the Berlekamp-Massey algorithm.

In 1975 Sugiyama, Kasahara, Hirasawa, and Namekawa showed that Euclid's algorithm can also be used to efficiently decode BCH and Reed- Solomon codes [19]. Euclid's algorithm is a means for finding the greatest common divisor of a pair of integers. It can also be extended to more complex collections of objects, including certain sets of polynomials with coefficients from finite fields.

As mentioned above, Reed – Solomon codes are based on the finite fields so they can be extended or shortened. In this thesis Reed-Solomon codes used for decoding in the compact discs are encoded and decoded. The generator polynomial approach has been used for encoding and decoding of data.

1.3OBJECTIVE OF THESIS

The objectives of the thesis are:

- To analyze the important characteristics of various coding techniques that could be used for error control in a communication system for reliable transmission of digital information over the channel.
- To study the Galois Field Arithmetic on which the most important and powerful ideas of coding theory are based.
- To study the Reed – Solomon codes and the various methods used for encoding and decoding of the codes to achieve efficient detection and correction of the errors.
- Implementation of the Reed – Solomon codes in VHDL.
- Analysis of the simulation results of the Reed – Solomon encoder and decoder.

1.4 ORGANIZATION OF THESIS

- In chapter 2, an overview of various channel coding techniques i.e. Forward – Error Correcting technique and the Automatic Repeat reQuest and their characteristics are discussed. Also various error – correcting codes such as parity codes, checksum codes, Hamming codes, BCH codes, burst error correcting codes i.e. cyclic codes are discussed.

- In chapter 3, the important properties and construction of Galois fields is discussed. Algebraic operations such as addition, subtraction, multiplication and division in the finite fields are discussed thoroughly. Also various algorithms for division in a Galois field are discussed in this chapter.

- In chapter 4, the Reed – Solomon codes are thoroughly discussed. The formation of codewords using the generator polynomial for encoding the information message is described in this chapter. The algorithms which are used for decoding i.e. Berlekamp's algorithm, Euclid's algorithm, Chien search algorithm and Forney's algorithm are also discussed in detail.

- Chapter 5 gives an introduction to VHDL language, its advantages, design units and different levels of abstraction. Also a brief introduction to the hardware implementation of the Reed – Solomon codes is given in this chapter.

- Chapter 6 gives the simulation results of the implementation of the Reed – Solomon codes in MODELSIM SE 5.5c.

- Chapter 7 gives the important conclusion and future scope of this thesis.

Chapter 2

ERROR CONTROL CODING

2.1 INTRODUCTION

The designer of an efficient digital communication system faces the task of providing a system which is cost effective and gives the user a level of reliability. The information transmitted through the channel to the receiver is prone to errors. These errors could be controlled by using Error- Control Coding which provides a reliable transmission of data through the channel. In this chapter, a few error control coding techniques are discussed that rely on systematic addition of redundant symbols to the transmitted information. Using these techniques, two basic objectives at the receiver are facilitated: Error Detection and Error Correction.

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 [20].

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 codeword (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 can not 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.

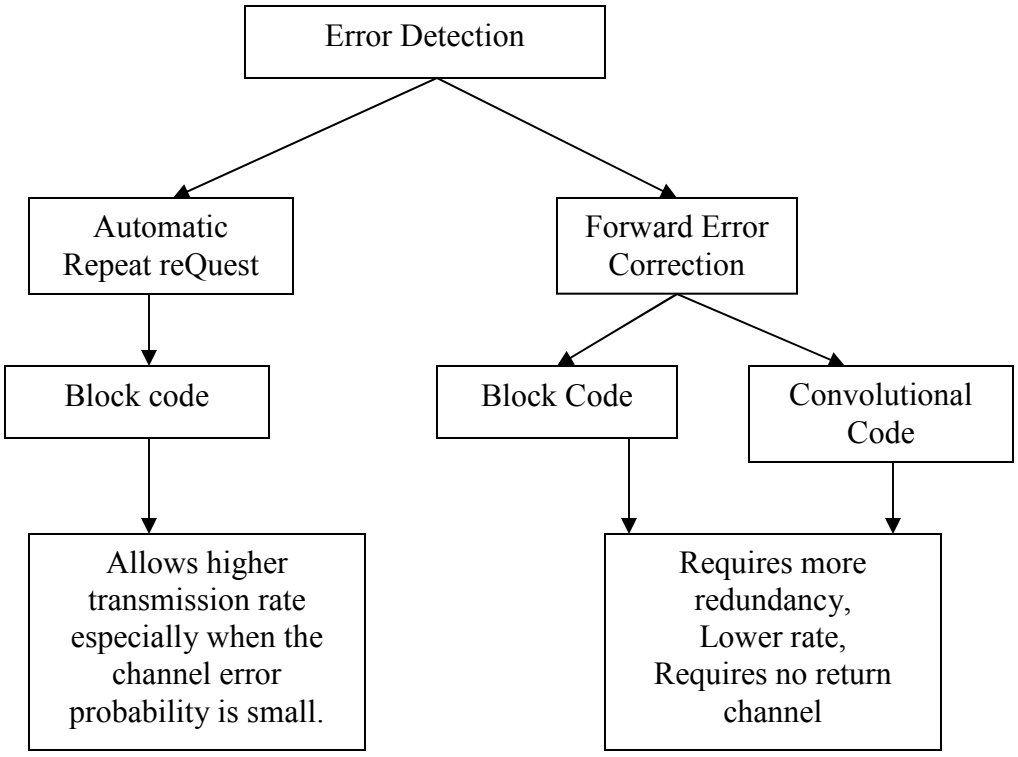


Figure-2.1: Error Detection and Correction

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 [1]. These error-correcting codes compensated for noise and other physical elements to allow for full data 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**

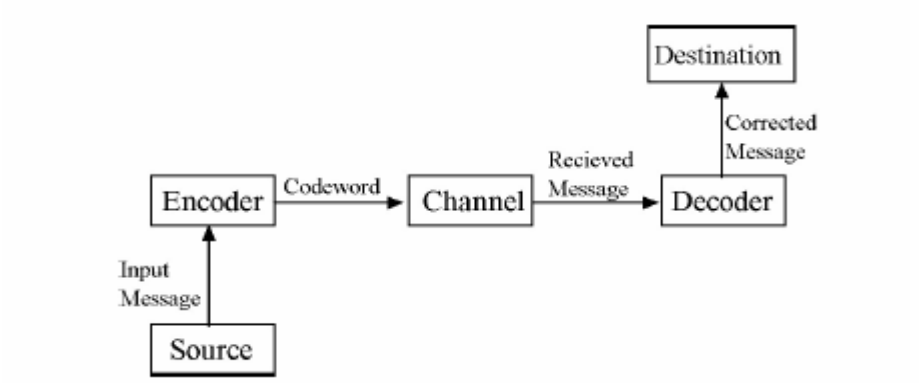


Figure-2.2: Concurrent Error Detection Implementation

Here the message generated by the source is passed to the encoder which adds redundant check bits, and turns the message into a codeword. 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 transmission, 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, depending 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 codewords (encoded data) is also a codeword. Sum here means adding two binary block bit wise XOR.
  - *Non-separable or Non-systematic*: if the check-bits are embedded within the codeword, and cannot be processed concurrently with the information, else is referred to as separable or systematic codes.
  - *Block codes*: if the codewords 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.
  - *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 corrupted data, without any further retransmissions.

- *Backward Error Correcting:* if the redundancy is enough only to detect errors and retransmission is required.

- The Hamming Distance of two codewords $x, y \in F^n$ denoted by $d(x, y)$, is the minimum number of bits in which they differ .i.e., the number of 1*s* 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 Codeword 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 codeword 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.

2.3.2.1 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.

2.3.2.2 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.

2.3.2.3 m-out-of-n Codes

In this scheme the codeword 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.

2.3.2.4 Berger Codes

Berger codes are systematic unidirectional error detecting codes. They can be considered as an extension of the parity codes. Parity codes have one check bit, which can be considered as the number of information bits of value 1 considered in modulo 2. On the other hand Berger codes have enough check bits to represent the count of the information bits having value 0. The number of check bits *(r)* required for *k*-bit information is given by

$$r = [\log_2(k-1)]$$

Of all the unidirectional error detecting codes that exist [21] suggests, m-out-of-n codes to be the most optimal. These codes however, are not of much application because of its non-separable nature. Amongst the separable codes in use, the Berger codes have been proven to be most optimal, requiring the smallest number of check bits [22].

The Berger Codes, however, are not optimal when only *t* unidirectional errors need to be detected instead of all unidirectional errors. For this reason a number of different modified Berger codes exist: Hao Dong introduced a code [23] that accepts slightly reduced error

detection capabilities, but does so using fewer check bits and smaller checker sizes. In this code the number of check bits is independent of the number of information bits.

Bose and Lin [24] have introduced their own variation on Berger codes and Bose [25] has further introduced a code that improves on the burst error detection capabilities of his previous code, where erroneous bit are expected to appear in groups.
Blaum [26] further improves on Bose-Lin Code. Favalli proposes an approach where code cost is reduced because of the graph theoretic optimization [27].

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 [1]**.** 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 increased 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 neighboring 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.

2.3.3.1 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 $(2^m - 1)/2$] there exists a binary BCH code with the following parameters:

Block length:	$n = 2^m - 1$
Number of message bits	$k \geq n - mt$
Minimum distance	$d_{\min} \geq 2t + 1$

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 codeword. 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.

2.3.3.2 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 $n = (p^m - 1)/(p - 1)$ 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 understanding of more complex correcting schemes. Here the codewords 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.

2.3.3.2 Burst Error Correcting Codes

The transmission channel could be memory less or it may be having some memory. If the channel is memory less then 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 neighbors 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 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 posses 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 [5], [14].

CHAPTER 3

GALOIS FIELD ARITHMETIC

3.1 INTRODUCTION

In chapter 2, various types of the error correcting codes were discussed. Burst errors are efficiently corrected by using cyclic codes. The Galois field or the Finite Fields are extensively used in the Error - Correcting Codes (ECC) using the Linear Block Codes. In this chapter these finite fields are discussed thoroughly. The Galois Field is a finite set of elements which has defined rules for arithmetic. These roots are not algebraically different from those used in the arithmetic with ordinary numbers but the only difference is that there is only a finite set of elements involved. They have been extensively used in Digital Signal Processing (DSP), Pseudo- Random Number Generation, Encryption and Decryption protocols in cryptography. The design of efficient multiplier, inverter and exponentiation circuits for Galois Field arithmetic is needed for these applications.

3.2 DEFINITION OF GALOIS FIELD

A Finite Field is a field with a finite field order (i.e., number of elements), also called a Galois field. The order of a finite field is always a prime or a power of a prime . For each prime power, there exists exactly one finite field GF(p^m).A Field is said to be infinite if it consists of infinite number of elements, for e.g. Set of real numbers, complex numbers etc. Finite field on the other hand consist of finite number of elements.

GF(p^m) is an extension field of the ground field *GF(p),* where m is a positive integer. For $p = 2$, *GF(2^m)* is an extension field of the ground field *GF(2)* of two elements (0,l). *GF(2^m)* is a vector space of dimension *m* over *GF(2)* and hence is represented using a basis of *m* linearly independent vectors. The finite field $GF(2^m)$ contains $(2^m - 1)$ non zero elements. All finite

fields contain a zero element and an element, called a generator or primitive element α, such that every non-zero element in the field can be expressed as a power of this element. The existence of this primitive element (of order 2^m-1) is asserted by the fact that the nonzero elements of GF(2^m) form a cyclic group.

Encoders and decoders for linear block codes over GF(2^m), such as Reed-Solomon codes, require arithmetic operations in GF(2^m). In addition, decoders for some codes over GF(2), such as BCH codes, require computations in extension fields GF(2^m). In GF(2^m) addition and subtraction are simply bitwise exclusive-or. Multiplication can be performed by several approaches, including bit serial, bit parallel (combinational), and software. Division requires the reciprocal of the divisor, which can be computed in hardware using several methods, including Euclid's algorithm, lookup tables, exponentiation, and subfield representations. With the exception of division, combinational circuits for Galois field arithmetic are straightforward. Fortunately, most decoding algorithms can be modified so that only a few divisions are needed, so fast methods for division are not essential.

.

3.3 PROPERTIES OF GALOIS FIELDS

- In GF(2^m) fields, there is always a primitive element α, such that you can express every element of GF(2^m) except zero as a power of α [28]. You can generate every field GF(2^m) using a primitive polynomial over GF(2), and the arithmetic performed in the GF(2^m) field is modulo this primitive polynomial.
- If α is a primitive element of GF(2^m), its conjugate α^{2^m} is also primitive elements of GF(2^m).
- If α is an element of order n in GF(2^m), all its conjugates have the same order n.
- If α, an element in GF(2^m), is a root of a polynomial *f(x)* over GF(2), then all the distinct conjugates of α, also elements in GF(2^m), are roots of *f(x)*.
- The 2^m - 1 nonzero elements of GF(2^m) form all the roots of $x^{2^m - 1} - 1 = 0$. The elements of GF(2^m) form all the roots of $x^{2^m} - x = 0$.

- Let α be in GF(2^m), $\phi(x)$ is the minimal polynomial of α over GF(2), if $\phi(x)$ is the polynomial of smallest degree over GF(2) such that $\phi(\alpha) = 0$.
- Every element α of GF (2^m) has a unique minimal polynomial over GF (2).
- The minimal polynomial $\phi(x)$ of a field element α is irreducible.
- An irreducible polynomial *f(x)* over GF(2) is the minimal polynomial of an element α in GF(p^m) if f(α) = 0.
- Let $\phi(x)$ be the minimal polynomial of α, If α is a root of *f(x)*, then *f(x)* is divisible by $\alpha(x)$.
- The minimal polynomial $\phi(x)$ of an element α in GF(2^m) divides $x^{2^m} - x$.
- The minimal polynomial $\phi(x)$, which is also irreducible, of an element α in GF(p^m) is

$$\phi(x) = \prod_{i=0}^{e-1} (x + \alpha^{2^m}) \qquad (3.1)$$

- Let *e* be the degree of a minimal polynomial $\phi(x)$ of an element α in GF(p^m), then *e* is the smallest integer such that $\alpha^{p^e} = \beta$, and $e \leq m$, *e* divides *m*.
- Any operation on elements of the field will preserve the closure property. Every element has an additive and multiplicative inverse (except for '0' which has no multiplicative inverse).

3.4 CONSTRUCTION OF GALOIS FIELDS

A Galois field GF (2^m) with primitive element α is generally represented as (0, 1, $\alpha, \alpha^2, \ldots\ldots\ldots \alpha^{2^k-2}$). The simplest example of a finite field is the binary field consisting of the elements (0, 1). Traditionally referred to as $GF(2)^2$, the operations in this field are defined as integer addition and multiplication reduced modulo 2. Larger fields can be created by extending GF(2) into vector space leading to finite fields of size 2^m. These are simple extensions of the base field GF(2) over *m* dimensions. The field GF(2^m) is thus defined as a field with 2^m elements each of which is a binary *m*-tuple. Using this definition, *m* bits of

binary data can be grouped and referred to it as an element of GF(2^m). This in turn allows applying the associated mathematical operations of the field to encode and decode data [10].

Let the primitive polynomial be $\phi(x)$, of degree m over GF(2^m). Now any i^{th} element of the field is given by

$$a_i(\alpha) = a_{i0} + a_{i1}\alpha + a_{i2}\alpha + ... + a_{im-1}\alpha^{m-1} \quad (3.2)$$

Hence all the elements of this field can be generated as powers of α. This is the polynomial representation of the field elements, and also assumes the leading coefficient of $\phi(x)$ to be equal to 1. **Figure 3.1** shows the Finite field generated by the primitive polynomial $1 + \alpha + \alpha^2 + \alpha^3 + \alpha^4 + \alpha^8$ represented as GF(2^8) or GF(256).

Power	Polynomial				Binary 4-tuple	decimal	index
0	0				(0 0 0 0)	0	-1
1	1				(1 0 0 0)	1	0
α^1		α			(0 1 0 0)	2	1
α^2			α^2		(0 0 1 0)	4	2
α^3				α^3	(0 0 0 1)	8	3
α^4	1	α			(1 1 0 0)	3	4
α^5		α	α^2		(0 1 1 0)	6	5
α^6			α^2	α^3	(0 0 1 1)	12	6
α^7	1	α		α^3	(1 1 0 1)	11	7
α^8	1		α^2		(1 0 1 0)	5	8
α^9		α		α^3	(0 1 0 1)	10	9
α^{10}	1	α	α^2		(1 1 1 0)	7	10
α^{11}		α	α^2	3	(0 1 1 1)	14	11
α^{12}	1	α	α^2	α^3	(1 1 1 1)	15	12
α^{13}	1		α^2	α^3	(1 0 1 1)	13	13
α^{14}	1			α^3	(1 0 0 1)	9	14

Figure 3.1: Representation of some elements in GF(2^8)

Note that here, the primitive polynomial is of degree 4, and the numeric value of α is considered purely arbitrary. Using the irreducibility property of the polynomial $\phi(x)$, this can be proven that this construction indeed produces a field.

The primitive polynomial is used in a simple iterative algorithm to generate all the elements of the field. Hence different polynomials will generate different fields. In the work [29], the author claims that even though there are numerous choices for the irreducible polynomials, the fields constructed are all isomorphic.

3.5 GALOIS FIELD ARITHMETIC

Galois Field Arithmetic (GFA) is very attractive in several ways. All the operations that generally result in overflow or underflow in traditional mathematics gets mapped on to a value inside the field because of modulo arithmetic followed in GFA, hence rounding issues also get automatically eliminated.

GF generally facilitates the representation of all elements with a finite length binary word. For e.g. in GF(2^m), all the operands are of m-bits, where m is always a number smaller than the conventional bus width of 32 or 64. This in turn introduces a huge amount of parallelism into the GFA operations. Further, we assume '*m*' to be a multiple of 8, or a power of 2, because of its inherent convenience for sub-word parallelism [30].

To study the GFA an introduction to the mathematical concepts of the trace and dual basis are necessary [31], [32].

Definition 1: The trace of an element β which belongs to *GF(2^m)* is defined as

$$Tr(\beta) = \sum_{k-0}^{m-1} (\beta)^{2^k} \qquad (3.3)$$

Definition 2: A basis $\{\mu_j\}$ in *GF(2^m)* is a set of m linearly independent elements in *GF(2^m),* where $0 \leq j \leq m-1$.

Definition 3: Two bases $\{\mu_j\}$ and $\{\lambda_k\}$ are the dual of one another if

$$Tr(\mu_j \lambda_k) = 1 \qquad \text{if } j = k$$
$$= 0 \qquad \text{if } j \neq k$$

The elements of *GF(2^m)* are usually expressed as powers of the primitive element α, where α is defined *as* the root of the primitive polynomial

$$g(x) = x^m + g_{m-1}x^{m-1} + g_{m-2}x^{m-2} + ... + g_1x + 1 \qquad (3.4)$$

where $f_i \in \{0,1\}$. Each element *z* of *GF(2^m)* can also be written in [31], [32], [33].

- the standard basis as $z = a_{m-1}\alpha^{m-1} + a_{m-2}\alpha^{m-2} + ... + a_2\alpha^2 + a_1\alpha^1 + a_0$
- the normal basis as $z = a_{m-1}\alpha^{2^{m-1}} + ... + a_2\alpha^2 + a_1\alpha^1 + a_0\alpha$
- the dual basis λ_k as $z = \sum_{k=0}^{m-1} z_k\lambda_k = \sum_{k=0}^{m-1} Tr(z\mu_k)\lambda_k$

where

$a_i = GF(2)$ and $z_k = Tr(z\mu_k)$ is the k-th coefficient of the dual basis.

The standard basis is commonly used in implementing algebraic RS decoders in hardware. Since multiplication is the most dominant arithmetic operation, the standard basis multiplier is often preferred for its lowest design complexity compared to the normal and dual based multipliers. It does not require basis conversion and thus can be more easily matched to any input/output system.

3.5.1 ADDITION/SUBTRACTION

Generally the field GF (2^m) represents a set of integers from zero to 2^m - 1. Addition and subtraction of elements of GF(2^m) are simple XOR operations of the two operands. Each of the elements in the GF is first represented as a corresponding polynomial. The addition or subtraction operation is then represented by the XOR operation of the coefficient of corresponding polynomials. However since the more complex operations are extensively used in RS encoding and decoding algorithms, the development of their hardware structures have received considerable attention.

Note that GFA does not distinguish between addition and subtraction operations; both are considered as XOR operations. Since both operations follow modulo arithmetic, the result always evaluates to a value within the field.

3.5.2 MULTIPLICATION

Multiplication operation over the Galois Field is a more complex operation than the addition/subtraction operation. It is also based on modulo arithmetic, but various approaches with varying complexities have been suggested and adopted till date. Majority of these approaches are described and evaluated in this subsection.

Approach 1

Let $A = a_0 + a_1\alpha + ... + a_{m-1}\alpha^{m-1}$ and $B = b_0 + b_1\alpha + ... + b_{m-1}\alpha^{m-1}$ be two elements from GF(2^m), then

$$A + B = C = c_0 + c_1\alpha + ... + c_{m-1}\alpha^{m-1} \quad (3.5)$$

This is the normal add operation and can be achieved by following the methodology described above. The multiplication over GF(2^m) is defined through

$$a(x) \otimes b(x) = a(x)b(x) \bmod f(x) \quad (3.6)$$

The first step in this computation is building of the partial products

$$P_i(x) = a(x)b_i x^i \qquad (0 \le i < m)$$

Next, all partial products $P_i(x)$ have to be added modulo 2. A partial result $C_p(x)$ is thus obtained.

$$C_p(x) = \sum P_i(x) \bmod 2 \quad (3.7)$$

$$C(x) = c_0 + c_1 x + ... + c_{2m-1}x^{2m-1} \quad (3.8)$$

This partial result, however is not a element of the field GF(2). Hence the highest order term x^h of $C_p(x)$ has to be reduced using the following property of the irreducible polynomial

$$x = f_0 + f_1 x + ... + f_{m-1}x^{m-1} \quad (3.9)$$

The substitution starts with the highest order term $c_{2m-2}x^{2m-2}$ from (3.9). (3.10) is used to produce a partial result C_p of one degree less than C_p :

$$C_p(x) = \left[c + c_1 x + ... + c_{2m-2} x^{2m-2}\right] + \left[f_0 x^{m-2} + f_1 x x^{m-2} + ... + f_{m-1} x^{m-1} x^{m-2}\right] (3.10)$$

This shifting operation is continued until $C_p(x)$ is of degree m-1, i.e. it becomes an element of GF(2^m). The result of a GF(2) multiplication is a (2m –1) bit word, which represents a degree (2m – 2) polynomial in extended form.

For m=8, the product is represented as follows:

Consider two operands in the GF(2^8),

$$Op1 = a_7\alpha^7 + a_6\alpha^6 + a_5\alpha^5 + a_4\alpha^4 + a_3\alpha^3 + a_2\alpha^2 + a_1\alpha^1 + a_0 \quad (3.11)$$

$$Op2 = b_7\alpha^7 + b_6\alpha^6 + b_5\alpha^5 + b_4\alpha^4 + b_3\alpha^3 + b_2\alpha^2 + b_1\alpha^1 + b_0 \quad (3.12)$$

the product can be represented as follows:

$$\Pr od(a) = Op1 \times Op2 = A_{14}\alpha^{14} + A_{13}\alpha^{13} + A_{12}\alpha^{12} + A_{11}\alpha^{11} + A_{10}\alpha^{10} + A_9\alpha^9 + A_8\alpha^8 + A_7\alpha^7 + A_6\alpha^6 + A_5\alpha^5 + A_4\alpha^4 + A_3\alpha^3 + A_2\alpha^2 + A_1\alpha^1 + A_0 \quad (3.13)$$

The product is a 15-bit word, containing the coefficients of the extended form polynomial *Prod(a)*. This word contains the coefficients of the extended form polynomial *Prod (a)*.

The extended form polynomial is reduced modulo *f(x)*. This is equivalent to calculating the remainder from the extended form polynomial divided by *f(x)*. The remainder in this case is a polynomial of degree 7 with 8 binary coefficients and therefore represents a valid element of GF(2^8).

Approach 2

When the field is restricted to a small number of finite elements, say GF(2^k), where $k \leq 16$, then approach suggested in [34]**,** based on logarithm tables can be adopted.

This approach employs two tables : gflog and gfilog.

For the GF(2^m - 1), table gflog consists of indexes from 1 to 2^m - 1, and maps these indexes to their corresponding logarithm in Galois Field. The table gfilog, on the other hand is defined for the indices 0 to 2^m -2, and maps these index to their respective inverse logarithm in the Galois Field.

A multiplication operation now consists of the following three steps:

- A gflog table lookup operation to obtain the log values of the input operands.
- The log values are added to obtain an intermediate result.
- Another table lookup, now from the gfilog table, yields the product of the two operands.

As mentioned earlier, GFA prevents overflow and rounding issues, hence the result of log operations, unlike regular logarithms result in a integer for any non-zero element in the field, leading to precise multiplication operations.

An effective implementation of this scheme requires one conditional, three lookup tables (two for logarithm lookups and one for inverse log lookup), an adder, and one modulo circuitry [34].

Approach 3

The work [30] proposes an alternative method for multiplication, but again based on table lookup. According to the author the multiplication operation can be performed on operands as a simple addition modulo operation, after converting them to, what the author calls as the 'multiplicative form' of the operand. The three steps involved in the computation are summarized below.

- The operands (which are originally in additive form) are converted to multiplicative form using the add2mul table.
- Addition modulo is then performed on the converted operands.
- The resultant value is then converted back to additive form using the mul2add table.

Figure 3.2 illustrates the working of this approach to calculate $c = a \times b$, and has been adopted from [30].

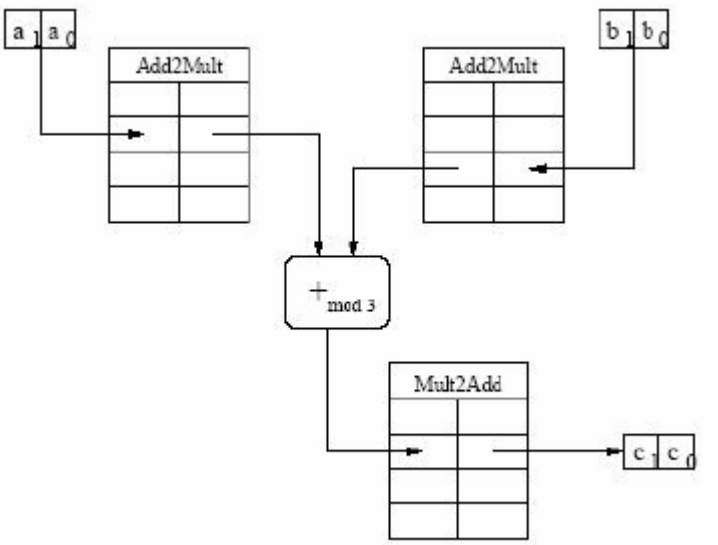


Figure- 3.2: Multiplication Operation

A division and bit-serial multiplication algorithm were presented by Hasan and Bhargava in 1992 [35] . Using the coordinates of supporting elements, division over $GF(q^m)$ is performed by solving a system of m linear equations over $GF(q)$ when the field elements are represented by polynomials. it is further shown that division can be performed with a lower order of computational complexity by solving a Wiener-Hopf equation of degree m. The discrete-time Wiener-Hopf equation is defined as a system of m linear inhomogeneous equations, with m unknowns [35]**.**

3.5.2.1 EXISTING MULTIPLICATION ARCHITECTURES

The multiplication circuits for GF(2^m) can be classified, based on their architecture implementations as: bit-serial, bit-parallel, hybrid, and digit-serial architecture. The bit-serial architectures [32] process one bit/clock cycle, are area - efficient and suitable for low-area applications. The bit-parallel architectures**,** process one word/clock cycle, are ideal for high-speed applications. The hybrid and the digit-serial [36] architectures have recently been proposed to provide to compensate the trade - offs of the bit-serial and bit-parallel architectures.

The hybrid multiplier architecture represents the field GF(2^m) as GF((2^n)^k). It uses bit parallel arithmetic in the sub-field GF(2^n) and serial processing for the extension field arithmetic. Thus, the multiplier processes input data at a rate of n-bits per clock cycle and yields output results at a rate of one every k clock cycles.

3.5.3 DIVISION

The division operation is said to be the more time and area consuming of all the operations on Galois fields. The general approach for performing division over GF(2^m)
is to multiply the inverse element of a divisor (say β) with the dividend (say γ). The real problem for performing this division operation is in finding an efficient computation algorithm for the multiplicative inverse element of the divisor β with the dividend γ:

$$\beta \div \gamma = \beta \times \gamma^{-1} \tag{3.14}$$

Two known algorithms for implementing this computation are:
1. Euclids algorithm
2. Continuous square algorithm

3.5.3.1 EUCLID'S ALGORITHM

The Euclidean algorithm determines the Greatest Common Divisor (GCD) of two integers (without requiring factorization). The Extended Euclidean algorithm is a slightly modified version of the same, which can compute both GCD and the multiplicative inverse of a element [19]

Given two integers a and b, the algorithm first computes the GCD of the two terms GCD*(a, b)*, as well as integers *x* and *y* such that *ax + by = gcd(a, b)*. The additional computation is not very costly, especially since the steps of Euclid's algorithm always deal with sums of multiples of *a* and *b*. The working of the extended version of the Euclid's algorithm is shown in Figure 3.3.

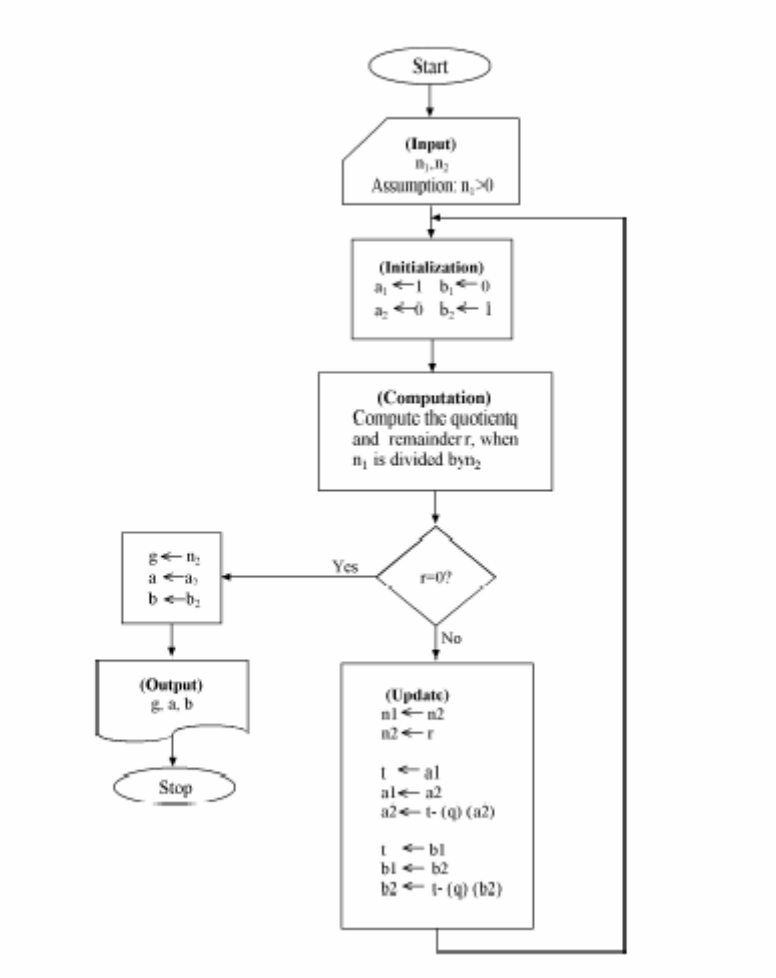


Figure 3.3: Flowchart of Extended Euclidean Algorithm

The equation *ax + by = gcd(a,b)* is of prime importance in finite field division where a and b are relatively prime, as the value of *x* will now represent the multiplicative inverse of *a modulo b*.

3.5.3.2 CONTINUOUS SQUARE ALGORITHM

The working of the continuous square algorithm is based on the fundamental property of GF(2^m) that

$$\gamma^{2^m} - 1 = 1 \qquad (3.15)$$

multiplying equation (3.5) with γ^{-1} gives:

$$\gamma^{-1} = \gamma^{2m-1} \times \gamma^{-1} = \gamma^{2m-2} \tag{3.16}$$

since

$$2^m - 2 = \sum_{i=1}^{m-1} 2^i \tag{3.17}$$

γ^{-1} can be computed successively by squaring and multiplication.

Another alternative approach to the above methodology is to follow the same steps as in *Approach 1* of multiplication, except that in *step 2*, subtract the two resulting log values, instead of the usual addition operation.

Chapter 4

REED SOLOMON CODES

In chapter 3, the Galois field arithmetic is discussed. It is noted that the Galois fields have the useful property that any operation on an element of the field will always result in another element of the field. An arithmetic operation that, in traditional mathematics, results in a value out with the field gets mapped back in to the field - it's a form of modulo arithmetic. In this chapter Reed Solomon encoding and decoding is discussed thoroughly. The RS encoder generates the codeword based on the message symbols. The parity symbols are computed by performing a polynomial division using GF algebra. The decoder processes the received code word to compute an estimate of the original message symbols.

4.1 INTRODUCTION

Reed-Solomon (RS) codes first appeared in technical literature in 1960. In 1960 Irving S. Reed and Gustave Solomon, staff members at MIT's Lincoln Laboratory, developed the Reed-Solomon code, which has become the most widely used algorithm for error correcting. The Reed-Solomon code is well understood, relatively easy to implement provides a good tolerance to error bursts and is compatible with binary transmission systems.

Since their introduction, they have seen widespread use in a variety of applications. These applications include interplanetary communications (e.g., the Voyager spacecraft), CD audio players, medical diagnosis military purposes and countless wired and wireless communications systems.

RS codes belong to the family known as *block codes*. This family is so named because the encoder processes a block of message symbols and then outputs a block of codeword symbols. To be specific, RS codes are non-binary systematic cyclic linear block codes. Non-binary codes work with symbols that consist of several bits. A common symbol size for non-binary

codes is 8 bits, or a byte. Non-binary codes such as RS are good at correcting burst errors because the correction of these codes is done on the symbol level. By working with symbols in the decoding process, these codes can correct a symbol with a burst of eight errors just as easily as they can correct a symbol with a single bit error. A systematic code generates codewords that contain the message symbols in unaltered form. The encoder applies a reversible mathematical function to the message symbols in order to generate the redundancy, or parity, symbols. The codeword is then formed by appending the parity symbols to the message symbols. The implementation of a code is simplified if it is systematic. A code is considered to be cyclic if a circular shift of any valid codeword also produces another valid codeword. Cyclic codes are popular because of the existence of efficient decoding techniques for them. Finally, a code is linear if the addition of any two valid codewords also results in another valid codeword.

RS codes are generally represented as an RS (n, k), with m-bit symbols, where

Block Length :	n
No. of Original Message symbols:	k
Number of Parity Digits:	n - k = 2t
Minimum Distance:	d = 2t + 1.

A codeword based on these parameters is shown diagrammatically in Figure 4.1

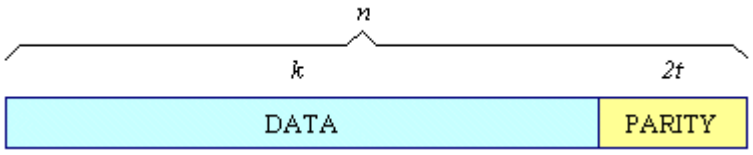


Figure 4.1: Reed Solomon Codeword

Such a code can correct up to (n-k)/2 or t symbol (each symbol is an element e of GF (2^m)) (i.e.) any t symbols corrupted in anyway (single- or multiple-bit errors) can still lead to recovery of the original message. RS codes are one of the most powerful burst - error correcting codes and have the highest code rate of all binary codes. They are particularly good at dealing with burst errors because, although a symbol may have all its bits in error, this counts as only one symbol error in terms of the correction capacity of the code.

A Reed Solomon protected communication or data transfer channel is as shown in Figure 4.2

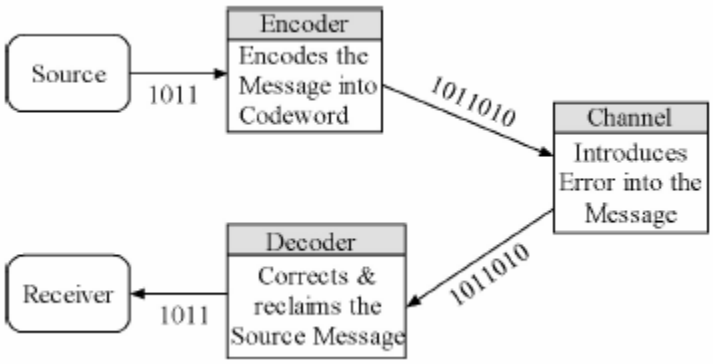


Figure 4.2: Reed Solomon Protected Channel

The RS encoder provided at the transmitter end encodes the input message into a codeword and transmits the same through the channel. Noise and other disturbances in the channel may disrupt and corrupt the codeword. This corrupted codeword arrives at the receiver end (decoder), where it gets checked and corrected message is passed on to the receiver. In case the channel induced error is greater than the error correcting capability of the decoder a decode failure can occur. Decoding failures are said to have occurred if a codeword is passed unchanged, a decoding failure, on the other hand will lead to a wrong message being given at the output.

Though mentioned in the description as non-binary, RS Codes can also be operated on binary data. This is done by representing each element in the GF (2^m) as a unique binary m-tuple [37]. For an (n, k) RS code a message of *km* bits is first divided into '*k*' *m*-bit bytes and is then encoded into *n*-byte codeword based on the RS encoding rule. By doing this, a RS code is expanded with symbols from GF (2^m) into a binary (*nm, km*) linear code, called a binary RS code.

The simulation results provided in [37] suggest that for a reasonably high value of t (≥ 8) and $n \geq 5t$, the probability of mis-decoding is much smaller than that of decoding failure, and hence, can be ignored.

A RS encoder takes a block of digital data and adds extra redundant bits, and converts it into a codeword. The Reed-Solomon decoder processes each block and attempts to correct errors that occur in transmission or storage. The number and type of errors that can be corrected depends on the characteristics of the RS code used.

In the next two sections a detailed insight into the encoding and decoding processes of these RS codes is provided.

4.2 RS ENCODING

Encoding is a process of converting a input message into a corresponding codeword, where each codeword $\in$ GF (2^m)). A RS codeword is generated by a polynomial g(x) of degree n-k with coefficient from GF (2m). According to [37] **,** for a t-error correcting RS code, this generator polynomial is given by:

$$g(x) = \prod_{i=1}^{2t} x + \alpha^i \tag{4.1}$$

where α is a primitive element in GF(2^m).

4.2.1 FORMING CODEWORDS

Let a message or data unit is represented in the polynomial form as

$$M(x) = m_{k-1}x^{k-1} + m_{k-2}x^{k-2} + \ldots\ldots\ldots\ldots\ldots + m_1 x + m_0 \tag{4.2}$$

and the codeword be represented as

$$C(x) = c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \ldots\ldots\ldots\ldots\ldots\ldots c_1 x + c_0 \tag{4.3}$$

represent the result of multiplication of the data unit with the generator polynomial. One important property of G(x) is that it exactly divides c (x), assume Q(x) and P(x) to be the corresponding quotient and remainder, hence the codeword looks like

$$C(x) = x^{n-k}M(x) + P(x) = Q(x)G(x) \tag{4.4}$$

Here P(x) is the polynomial which represents the check symbols.

Dividing by the generator polynomial and rewriting gives

$$\frac{x^{n-k}M(x)}{G(x)} = Q(x) - \frac{P(x)}{G(x)} \tag{4.5}$$

Q(x) can be identified as ratio and – *P(x)* as a remainder after division by *G(x)*. The idea is that by concatenating these parity symbols to the end of data symbols, a codeword is created which is exactly divisible by *g(x)*. So when the decoder receives the message block, it divides it with the RS generator polynomial. If the remainder is zero, then no errors are detected, else indicates the presence of errors.

The lowest degree term in $X^{n-k}M(x)$ is $m_0 x^{n-k}$, while *P(x)* is of degree at most *n-k-1*, it follows that the codeword is given by

$$C(x) = c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + ...c_1 x + c_0 \tag{4.6}$$

$$= m_{k-1} + m_{k-2} + ... + m_1 + m_0, -p_{n-k-1} - p_{n-k-2} ..... - p_1 - p_0 \tag{4.7}$$

and it consists of the data symbols followed by the parity-check symbols.

4.2.2 RS ENCODER

As previously mentioned, RS codes are systematic, so for encoding, the information symbols in the codeword are placed as the higher power coefficients. This requires that information symbols must be shifted from power level of *n-1* down to *n-k* and the remaining positions from power *n-k-1* to *0* be filled with zeros. Therefore any RS encoder design should effectively perform the following two operations, namely division and shifting. [28] suggests that both operations can be easily implemented using Linear-Feedback Shift Registers.

The parity symbols are computed by performing a polynomial division using GF algebra. The steps involved in this computation are as follows:

- Multiply the message symbols by X^{n-k}. This shifts the message symbols to the left to make space for the *n-k* parity symbols.
- Divide the message polynomial by the code generator polynomial using GF algebra.

- The parity symbols are the remainder of this division. These steps are accomplished in hardware using a shift register with feedback. The architecture for the encoder is shown in Figure 4.3

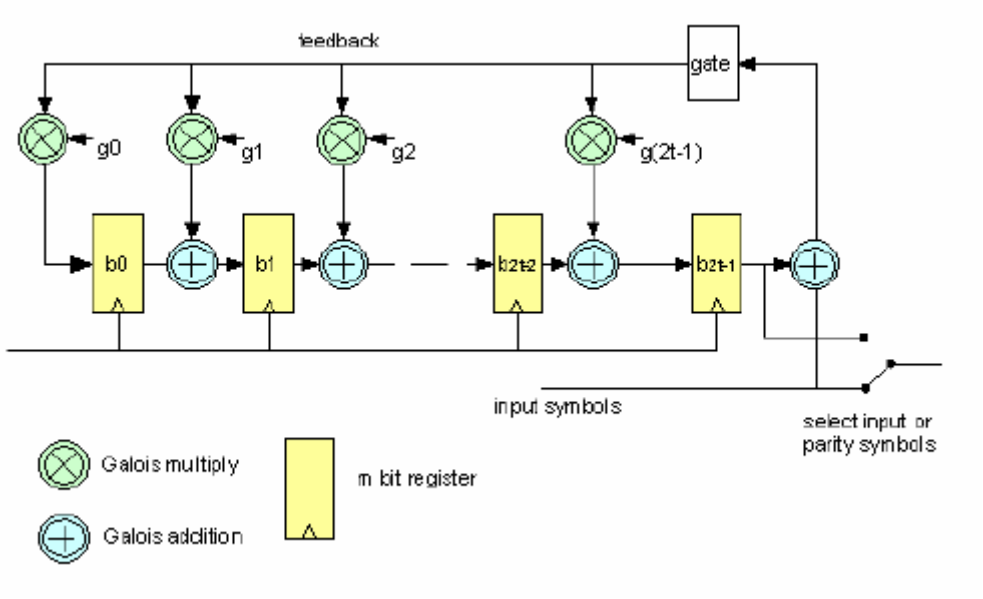


Figure 4.3- RS Encoder Circuitry

The encoder block diagram shows that one input to each multiplier is a constant field element, which is a coefficient of the polynomial *g(x)*. For a particular block, the information polynomial *M(x)* is given into the encoder symbol by symbol. These symbols appear at the output of the encoder after a desired latency, where control logic feeds it back through an adder to produce the related parity. This process continues until all of the *k* symbols of *M(x)* are input to the encoder. During this time, the control logic at the output enables only the input data path, while keeping the parity path disabled. With an output latency of about one clock cycle, the encoder outputs the last information symbol at *(k+1)th* clock pulse. Also, during the first *k* clock cycles, the feedback control logic feeds the adder output to the bus. After the last symbol has been input into the encoder (at the *kth* clock pulse), a wait period of at least *n-k* clock cycles occurs. During this waiting time, the feedback control logic disables the adder output from being fed back and supplies a constant zero symbol to the bus. Also, the output control logic disables the input data path and allows the encoder to output the parity symbols (*k+2th to n+1th* clock pulse). Hence, a new block can be started at the *n+1th* clock pulse [28].

4.3 RS DECODING

The channel of transmission, especially in critical applications like space, submarine, nuclear introduces a huge amount of noise into the information message. Thus the input codeword is received at the receiver end as codeword, *c(x)*, plus any channel induced errors, *e(x)*, say *r(x) = c(x) + e(x)*. This received polynomial *r(x)* is given by

$$R(x) = r_{n-1}x^{n-1} + r_{n-2}x^{n-2} + ...r_1x + r_0 \quad (4.8)$$

The decoding procedure for Reed- Solomon codes involves determining the locations and magnitudes of the errors in the received polynomial *r(x)*. Locations are those powers of *x* *(*x^2 *,*x^3*, and others)* in the received polynomials whose coefficients are in error. Magnitudes of the errors are symbols that are added to the corrupted symbol to find the original encoded symbol. These locations and magnitudes constitute the error polynomial. Also, if the decoder is built to support erasure decoding, then the erasure polynomial has to be found. An erasure is an error with a known location. Thus, only the magnitudes of the erasures have to be found for erasure decoding. A RS (*n, k*) code can successfully correct as many as *2t = n-k* erasures if no errors are present. With both errors and erasures present, the decoder can successfully decode if $n\text{-}k \geq 2t + e$, where *t* is the number of errors, and *e* is the number of erasures [28]**.**

Technically, error correcting can be as simple as inverting the erroneous values in the input message. But implementing the same involves complex design solutions, typically involving as much as ten times the resources, be it logic, memory or processor cycles, than that is required for the encoding process [38]. This is attributed by the complexity of the task involving the identification of the location of occurrence of errors in the received message.

A typical decoder follows the following stages in the decoding cycle, namely

1. Syndrome Calculation
2. Determine error-location polynomial
3. Solving the error locator polynomial - Chien search
4. Calculating the error Magnitude

5. Error Correction

Figure 4.4 shows the internal flow diagram for the functional level depiction of the RS decoder unit. Rest of this section deals with various functional level stages shown in the diagram.

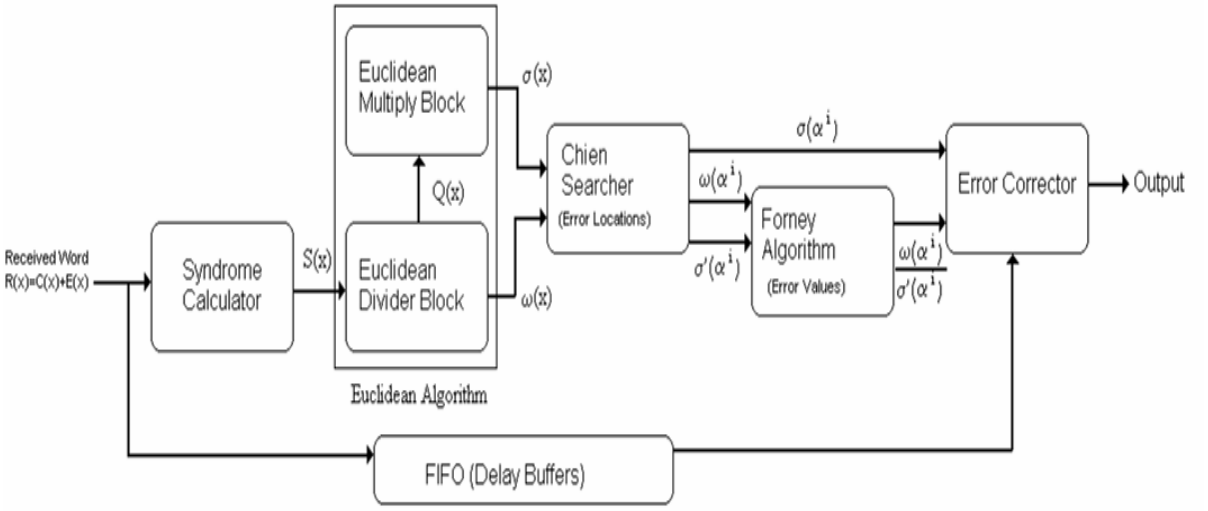


Figure- 4.4: Block Diagram of a RS Decoder

4.3.1 SYNDROME CALCULATION

The first step in decoding the received symbol is to determine the data syndrome. Here the input received symbols are divided by the generator polynomial. The result should be zero. The parity is placed in the codeword to ensure that code is exactly divisible by the generator polynomial. If there is a remainder, then there are errors. The remainder is called the syndrome.

The syndromes can then be calculated by substituting the *2t* roots of the generator polynomial *g(x)* into *R(x)*. The process is then iterated for other symbols and net *2t* syndromes are obtained as soon as the last parity symbol has been read in. The syndromes depend only on the errors, not on the underlying encoded data [38].

The syndrome polynomial is generally represented as

$$S(x) = S_0 + S_1 x + ... + S_{2t-1} x^{2t-1} = \sum_{j=0}^{n-1} r_j \alpha^{ij} \quad (4.9)$$

where α is the primitive element.

4.3.2 DETERMINATION OF ERROR-LOCATOR POLYNOMIAL

The next step, after the computing the syndrome polynomial is to calculate the error values and their respective locations. This stage involves the solving of the *2t* syndrome polynomials, formed in the previous stage. These polynomials have *v* unknowns, where *v* is the number of unknown errors prior to decoding. If the unknown locations are $i_1, i_2, ...i_v$, the error polynomial can be expressed as

$$E(x) = Y_1 x^{i_1} + Y_2 x^{i_2} + ... + Y_v x^{i_v} \quad (4.10)$$

where Y_l is the magnitude of the *l*th error at location i_l.

If X_l is the field element associated with the error location i_l, then the syndrome coefficients are given by

$$S_j = \sum_{i=1}^{v} Y_l X_l^i \quad (4.11)$$

for j =1,2,….,2t

where Y_l is the error value and X_l is the error location of the *l*th error symbol.

The expansion of (11) gives the following set of *2t* equations in the *v* unknown error locations $X_1, X_2, ...X_v$ and *v* unknown error magnitudes $Y_1, Y_2, ...Y_v$

$$\begin{aligned} S_1(x) &= Y_1 X_1 + Y_2 X_2 + ... + Y_V X_V \\ S_2(x) &= Y_1 X_1^2 + Y_2 X_2^2 + ... + Y_V X_V^2 \\ &. \\ &. \\ &. \\ S_{2t}(x) &= Y_1 X_1^{2t} + Y_2 X_2^{2t} + ...Y_v X_v^{2t} \end{aligned} \quad (4.12)$$

The above set of equations must have at least one solution because of the way the syndromes are defined. This solution is unique. Thus the decoder's task is to find the unknowns given the syndromes. This is equivalent to the problem in solving a system of non-linear equations.

Clearly, the direct solution of the system of nonlinear equations is too difficult for large values of *v*. Instead, intermediate variables can be computed using the syndrome coefficients S_j from which the error locations, $X_1, X_2, ..., X_v$, can be determined.

The error-locator polynomial is introduced as

$$\sigma(x) = \sigma_v x^v + \sigma_{v-1} x^{v-1} + ... + \sigma_1 x + 1 \qquad (4.13)$$

The polynomial is defined with roots at the error locations X_l^{-1} for *l=1,2,.....v*

The error location numbers X_l indicate errors at locations i_l for *l=1, 2, ...,v* This can be written as

$$\sigma(x) = (1 - xX_1)(1 - x\ X_2).....(1 - xX_v) \qquad (4.14)$$

where $X_l = \alpha^{i_l}$

to determine the coefficients of $\sigma(x)$ from the syndromes, equate equations (13) and (14) and multiply both sides by $Y_l X^{j+v}$ and set $x = X_l^{-1}$, i.e.,

$$Y_l X^{j+v}(\sigma(x)) = \sum_{l=1}^{v}(1 - xX_l) = \sigma_v x^v + \sigma_{v-1} x^{v-1} + ... + \sigma_1 X + 1 \qquad (4.15)$$

Then the left side becomes zero, giving

$$0 = Y_l\ X_l^{j+v}(1 + \sigma_1 X_l^{-1} + \sigma_2 X_l^{-2} + ... + \sigma_{v-1} X_l^{-(v-1)} + \sigma_v X_l^{-v}) \qquad (4.16)$$

or

$$Y_l(X_l^{j+v} + \sigma_1 X_l^{j+v-1} + ... + \sigma_v X_l^j) = 0 \qquad (4.17)$$

such an equation holds for each l and j. summing up these equations from *l*=1 to *l*=*v*, for each j, gives,

$$\sum_{l=1}^{v} Y_l(X_l^{j+v} + \sigma_1 X_l^{j+v-1} + ... + \sigma_v X_l^j) = 0 \qquad (4.18)$$

the individual sums seem to be syndromes and thus the equation becomes

$$\sigma_1 S_{j+v-1} + \sigma_2 S_{j+v+2} + ... + \sigma_v S_j = -S_{j+v} \tag{4.19}$$

where $j = 1, 2, \ldots\ldots, v$

This set of linear equations relates the syndromes to the coefficients of the error location polynomial $\sigma(x)$. It can also be expressed in the matrix form as

$$\begin{bmatrix} S_1 & S_2 & S_3 & .... & S_{V-1} & S_V \\ S_2 & S_3 & S_4 & .... & S_V & S_{V+1} \\ S_3 & S_4 & S_5 & .... & S_{V+1} & S_{V+2} \\ . & . & . & .... & . & . \\ . & . & . & .... & . & . \\ . & . & . & .... & . & . \\ S_V & S_{V+1} & S_{V+2} & .... & S_{2V-2} & S_{2V-1} \end{bmatrix} \begin{bmatrix} \sigma_V \\ \sigma_{V-1} \\ \sigma_{V-2} \\ \\ \\ \\ \sigma_1 \end{bmatrix} = \begin{bmatrix} -S_{V+1} \\ -S_{V+2} \\ -S_{V+3} \\ . \\ . \\ . \\ -S_{2V} \end{bmatrix} \tag{4.20}$$

The above system of equations has a unique solution for σ which can be obtained by inverting the matrix A, if A is nonsingular. The matrix A is nonsingular if $v \leq t$. Peterson's direct – solution algorithm solves for the error locator polynomial $\sigma(x)$ in equation (20) follows [39]**:** as a trial value, v is set to the error correction capability of the code t and the determinant of the matrix computed. If the determinant is non- zero, it can be shown that this is the correct value of v. otherwise, if it is zero, then the trial value of v is reduced by 1 and the process is repeated until a non- zero determinant is obtained. After the determinant has been obtained, the coefficients of $\sigma(x)$ are determined using the value of v in (20) by standard techniques of linear algebra.

From [8], for t = 1s, (20) can be written as

$$[S_2][\sigma_0] = [-S_1] \quad \text{and} \quad \sigma_0 = \frac{S_1}{S_2}$$

and then the error location is given as

$$\sigma(x) = \sigma_0 + x \tag{4.21}$$

solving the key equation for $t = 1$

$$\omega(x) = -(\sigma_0 + x)(S_1 + S_2 x) \bmod x^2 \tag{4.22}$$

where the error polynomial is

$\omega(x) = \omega_0$ and $\omega_0 = \sigma_0 S_1$

similarly for $t = 2$, (20) is reduced to

$$\begin{bmatrix} S_2 & S_3 \\ S_3 & S_4 \end{bmatrix} \begin{bmatrix} \sigma_1 \\ \sigma_0 \end{bmatrix} = \begin{bmatrix} -S_1 \\ -S_2 \end{bmatrix} \tag{4.23}$$

so the error location is given by

$$\sigma_0 = \frac{S_1 S_3 + (S_2)^2}{S_2 S_4 + (S_3)^2} \quad \text{and} \quad \sigma_1 = \frac{S_2 S_3 + S_1 S_4}{S_2 S_4 + (S_3)^2} \tag{4.24}$$

then the error location polynomial is written as

$\sigma(x) = \sigma_0 + \sigma_1 x + x^2$

solving the key equation for t=2 yields

$$\omega(x) = -(\sigma_0 + \sigma_1 x + x^2)(S_1 + S_2 x + S_3 x^2 + S_4 x^3) \bmod x^4 \tag{4.25}$$

the error value polynomial can be represented as

$$\omega(x) = \omega_0 + \omega_1 x \tag{4.26}$$

with $\omega_0 = \sigma_0 S_1$ and $\omega_1 = \sigma_0 S_2 + \sigma_1 S_1$

Peterson's direct solution is inefficient for codes with a large error correcting capability *t*. the number of computations necessary to convert a *v* by *v* matrix is directly proportional to v^3.

In most applications, designers often prefer to use codes that correct a large number of errors. Two very efficient decoding methods are discussed in the following sections: the Berelkamp-Massey algorithm and the Euclid's algorithm.

4.3.2.1 BERLEKAMP- MASSEY ALGORITHM

The Berlekamp-Massey algorithm relies on the fact that the matrix equation of (20) is not arbitrary in its form, rather, the matrix is highly structured. This structure is used to obtain the vector σ by a method that is conceptually more complicated but computationally much simpler [17].

If the vector σ is known, then the first row of the above matrix equation defines S_{v+1} in terms of $S_1, S_2, ...S_v$. The second row defines S_{v+2} in terms of $S_2, S_3, ...S_{v+1}$ and so forth. This sequential process can be summarized by the recursive relation

$$S_j = -\sum_{i=1}^{v} \sigma_i S_{j-1} \qquad j = v+1, \ldots\ldots\ldots, 2v \qquad (4.27)$$

For fixed σ , this is equivalent to the equation of an autoregressive filter. It can be implemented as a linear-feedback shift register with taps given by the coefficients of σ.
Using this argument, the problem has been reduced to the design of a linear- feedback shift register that will consequently generate the known sequences of syndromes. Many such shift registers exist, but it is desirable to find the smallest linear-feedback shift register with this property. This will give the least-weight error pattern with a polynomial $\sigma(x)$ of smallest degree *v*. The polynomial of smallest degree *v* is unique, since the $v \times v$ matrix of the original problem is invertible.

Any procedure for designing the autoregressive filter is also a method for solving the matrix equation for the σ vector. The procedure applies in any field and does not assume any special properties for the sequence $S_1, S_2, ...S_{2t}$. To design the required shift register, the shift register length *L* and feedback connection polynomial $\sigma(x)$ must be determined. $\sigma(x)$ has the form

$$\sigma(x) = \sigma_v x^v + \sigma_{v-1} x^{v-1} + ...\sigma_1 x + 1 \qquad (4.28)$$

where $\deg \sigma(x) \le L$.

The Berlekamp-Massey Algorithm uses the initial conditions
$\sigma^{(0)}(x) = 1, B^{(0)} = 1$, and $L_0 = 0$, to compute $\sigma^{(2t)}(x)$ as follows:

$$\Delta_r = \sum_{j=0}^{n-1} \sigma_j^{(r-1)} S_{r-j} \qquad (4.29)$$

$$L_r = \delta_r (r - L_{r-1}) + (1 - \delta_r) L_{r-1} \qquad (4.30)$$

$$\begin{bmatrix} \sigma^{(r)}(x) \\ B^{(r)}(x) \end{bmatrix} = \begin{bmatrix} 1 & -\Delta_r x \\ \Delta_r^{-1}\delta_r & (1-\delta_r)x \end{bmatrix} \begin{bmatrix} \sigma^{(r-1)}(x) \\ B^{(r-1)}(x) \end{bmatrix} \qquad (4.31)$$

for r = 1,……2t

$\delta_r = 1$, if both $\Delta_r \neq 0$ and $2L_{r-1} \le r-1$; and $\delta_r = 0$, otherwise.

At the end of the 2t iterations, the smallest – degree polynomial $\sigma^{(2t)}(x)$ with $\sigma_0^{2t} = 1$ satisfying the relation

$$S_r + \sum_{j=1}^{n-1} \sigma_j^{2t} S_{r-j} = 0 \tag{4.32}$$

where $r = L_{2t} + 1,.....,2t$ will be obtained.

Then if the error evaluation polynomial $\omega(x)$ is defined by the relation

$$S(x)\sigma(x) = \omega(x) \bmod x^{2t} \tag{4.33}$$

then $\omega(x)$ can be used to solve for he error magnitudes $Y_1,.....Y_v$.

4.3.2.2 EUCLID'S ALGORITHM

Euclid's algorithm is a recursive procedure for calculating the greatest common divisor (GCD) of two polynomials [19]. In a slightly expanded version, the algorithm will always produce the polynomials *a(x) and b(x)* satisfying

$$GC\,D\,[\,s(x),t(x)\,] = a(x)s(x) + b(x)t(x) \tag{4.34}$$

Euclid's algorithm uses the initial conditions

$$R^{(0)}(x) = x^{2t}, T^{(0)}(x) = \sum_{j=1}^{2t} S_j x^{j-1} \text{ , and} \tag{4.35}$$

$$A^{(0)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{4.36}$$

to compute $\sigma^{(2t)}(x)$ as follows:

$$Q^{(r)}(x) = \left[\frac{R^{(r)}(x)}{T^{(r)}(x)} \right] \tag{4.37}$$

$$A^{(r+1)}(x) = \begin{bmatrix} 1 & 0 \\ 0 & Q^{(r)}(x) \end{bmatrix} A^{(r)}(x) \tag{4.38}$$

$$\begin{bmatrix} R^{r+1}(x) \\ T^{(r+1)}(x) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & Q^{(r)}(x) \end{bmatrix} \begin{bmatrix} R^{r}(x) \\ T^{(r)}(x) \end{bmatrix} \tag{4.39}$$

The algorithm stops when the degree of $T^{(r)}$ is less than *t*.

At the end of the iteration, the error evaluator and error locator polynomials are found using

$$\omega(x) = \Delta^{-1} T^{(r)}(x) \tag{4.40}$$

$$\sigma(x) = \Delta^{-1} A_{22}^{(r)}(x) \tag{4.41}$$

respectively, where $\Delta = A_{22}^{(r)}(0)$ and A_{22} is the element of the matrix $A^{(r)}$ in the second row and second column.

This algorithm has been modified by Shao et al to avoid the computation of the inverse elements of the Galois field [40]**.** The modified Euclid's algorithm recursively finds the *i*th remainder $R_i(x)$ and the quantities $\gamma_i(x)$ and $\lambda_i(x)$ that satisfy the relation

$$\gamma_i(x)A(x) + \lambda_i(x)S(x) = R_i(x) \tag{4.42}$$

and stops when the degree of the remainder polynomial $R_i(x)$ is less than t**,** where

$A(x) = x^{2t}$ and $S(x) = \sum_{k=1}^{2t} S_k x^{2t-k}$

Using the initial conditions $R_0(x) = A(x)$, $Q_0(x) = S(x)$, $\lambda_0(x) = 0$, $\mu_0(x) = 1$, $\gamma_0(x) = 1$, $\eta_0(x) = 0$, it computes $R_i(x), \lambda_i(x)$ and $\gamma_i(x)$ as follows:

$$R_i(x) = \left[\sigma_{i-1} b_{i-1} R_{i-1}(x) + \hat{\sigma}_{i-1} a_{i-1} Q_{i-1}(x)\right] - x^{l_i - 1}\left[\sigma_{i-1} a_{i-1} Q_{i-1}(x) + \hat{\sigma}_{i-1} b_{i-1} R_{i-1}(x)\right] \tag{4.43}$$

$$\lambda_i(x) = \left[\sigma_{i-1} b_{i-1} \lambda_{i-1}(x) + \hat{\sigma}_{i-1} a_{i-1} \mu_{i-1}(x)\right] - x^{l_i - 1}\left[\sigma_{i-1} a_{i-1} \mu_{i-1}(x) + \hat{\sigma}_{i-1} b_{i-1} \lambda_{i-1}(x)\right] \tag{4.44}$$

$$\gamma_i(x) = \left[\sigma_{i-1} b_{i-1} \gamma_{i-1}(x) + \hat{\sigma}_{i-1} a_{i-1} \eta_{i-1}(x)\right] - x^{l_i - 1}\left[\sigma_{i-1} a_{i-1} \mu_{i-1}(x) + \hat{\sigma}_{i-1} b_{i-1} \gamma_{i-1}(x)\right] \tag{4.45}$$

$$Q_i(x) = \sigma_{i-1} Q_{i-1}(x) + \hat{\sigma}_{i-1} R_{i-1}(x) \tag{4.46}$$

$$\mu_i(x) = \sigma_{i-1} \mu_{i-1}(x) + \hat{\sigma}_{i-1} \lambda_{i-1}(x) \tag{4.47}$$

$$\eta_i(x) = \sigma_{i-1} \eta_{i-1}(x) + \hat{\sigma}_{i-1} \lambda_{i-1}(x) \tag{4.48}$$

where a_{i-1} and b_{i-1} are the leading coefficients of $R_{i-1}(x)$ and $Q_{i-1}(x)$ respectively, $l_{i-1} = \deg\left[R_{i-1}(x)\right] - \left[\deg(Q_{i-1}(x))\right]$, $\sigma_{i-1} = 1$ if $l_{i-1} \geq 0$ and $\sigma_{i-1} = 0$ if $l_{i-1} < 0$.

The iterations stop when deg $[R_i(x)]$ <*t,* after which the error locator polynomial $\sigma(x) = \lambda_i(x)$ and the error evaluator polynomial $\omega(x) = R_i(x)$.

Both the above mentioned algorithms have their own advantages and disadvantages, but generally Euclid's Algorithm is very easy to implement, while its counterpart Berlekamp-Massey Algorithm is more Hardware efficient.

4.4 SOLVING THE ERROR LOCATOR POLYNOMIAL-CHIEN SEARCH

Once the error locator $\sigma(x)$ and error evaluator $\omega(x)$ polynomials have been determined using the above techniques, the next step in the decoding process is to evaluate the error polynomial $\sigma(x)$, and obtain its roots. The roots thus obtained will now point to the error locations in the received message. RS decoding generally employs the Chien search scheme to implement the same.

A number 'n' is said to be a root of a polynomial, if the result of substitution of its value in the polynomial evaluates to zero. Chien Search is a brute force approach for guessing the roots, and adopts direct substitution of elements in the Galois field, until a specific *i* from *i*=0, 1, ….., n-1 is found such that $\sigma(\alpha^i) = 0$. In such a case α^i is said to be the root and the location of the error is evaluated as $\sigma(x)$. Then the number of zeros of the error locator polynomial $\sigma(x)$ are computed and are compared with the degree of the polynomial. If a match is found the error vector is updated and $\sigma(x)$ is evaluated in all symbol positions of the codeword [14]**.** A mismatch indicates the presence of more errors than can be corrected.

4.5 ERROR VALUE COMPUTATION- FORNEY ALGORITHM

Once the errors are located, the next step is to use the syndromes and the error polynomial roots to derive the error values. Forney Algorithm is generally used for this purpose. It is an efficient way of performing a matrix inversion, and involves two

main stages.

First the error evaluator polynomial $\omega(x)$ is calculated. This is done by convolving the syndromes with the error polynomial $\sigma(x)$ (from the Euclid's Algorithm result).

$$\omega(x) = S(x)\sigma(x) \bmod (x^p) \tag{4.49}$$

This calculation is carried out at each zero location, and the result thus arrived is then divided by the derivative of lambda. Each such calculation gives the error symbol at the corresponding location. The error magnitude at each error location x^i is given by

$$e_i = \frac{\omega(\alpha^i)}{\sigma'(\alpha^i)} \tag{4.50}$$

If the error symbol has any set bit, it means that the corresponding bit in the received symbol is at error, and must be inverted. To automate this correction process each of the received symbol is read again (from an intermediate store), and at each error location the received symbols XOR'ed with the error symbol. Thus the decoder corrects any errors as the received word is being read out from it.

In summary, the decoding algorithm works as follows:

Step 1: Calculate the syndromes according to Equation (4.9)

Step 2: Perform the Berlekamp-Massey or Euclid's algorithm to obtain the error locator polynornial $\sigma(x)$. Also find the error evaluator polynomial $\omega(x)$.

Step 3**:** Perform the Chien Search to find the roots of $\sigma(x)$.

Step 4**:** Find the magnitude of the error values using the Forney's Algorithm**.**

Step 5: Correct the received word *C (x) = E(x) + R (x)*

CHAPTER 5

VHSIC HARDWARE DESCRIPTION LANGUAGE

5.1 INTRODUCTION

Hardware description languages are especially useful to gain more control of parallel processes as well as to circumvent some of the idiosyncrasies of the higher level programming languages. The compilers often add latency to loops during compilation for implementation. This can be difficult to fix in the higher-level languages, though the solution may be quite obvious at the hardware description level. One particularly frustrating peculiarity is the implementation of multipliers. For all multiply commands, the complier requires three multipliers to be used, though typically one is sufficient. The compiler's multipliers also are intended for integers. For a fixed-point design, the decimal point must be moved after every multiply. This is much easier to implement at the hardware description level [41].

VHDL is a programming language that has been designed and optimized for describing the behavior of digital systems. VHDL has many features appropriate for describing the behavior of electronic components ranging from simple logic gates to complete microprocessors and custom chips. Features of VHDL allow electrical aspects of circuit behavior such as rise and fall times of signals, delays through gates, and functional operation to be precisely described. The resulting VHDL simulation models can then be used as building blocks in larger circuits using schematics, block diagrams or system-level VHDL descriptions for the purpose of simulation.

VHDL is also a general-purpose programming language: just as high-level programming languages allow complex design concepts to be expressed as computer programs, VHDL allows the behavior of complex electronic circuits to be captured into a design system for

automatic circuit synthesis or for system simulation. Like Pascal, C and C++, VHDL includes features useful for structured design techniques, and offers a rich set of control and data representation features. Unlike these other programming languages, VHDL provides features allowing concurrent events to be described. This is important because the hardware described using VHDL is inherently concurrent in its operation.

One of the most important applications of VHDL is to capture the performance specification for a circuit, in the form of what is commonly referred to as a test bench. Test benches are VHDL descriptions of circuit stimuli and corresponding expected outputs that verify the behavior of a circuit over time. Test benches should be an integral part of any VHDL project and should be created in tandem with other descriptions of the circuit.

One of the most compelling reasons for learning VHDL is its adoption as a standard in the electronic design community. Using a standard language such as VHDL virtually guarantees that the engineers will not have to throw away and recapture design concepts simply because the design entry method chosen is not supported in a newer generation of design tools. Using a standard language also means that the engineer is more likely to be able to take advantage of the most up-to-date design tools and that the users of the language will have access to a knowledge base of thousands of other engineers, many of whom are solving similar problems.

5.2 HISTORY OF VHDL

- 1981 - Initiated by US DoD to address hardware life-cycle crisis
- 1983-85 - Development of baseline language by Intermetrics, IBM and TI
- 1986 - All rights transferred to IEEE
- 1987 - Publication of IEEE Standard
- 1987 - Mil Std 454 requires comprehensive VHDL descriptions to be delivered with Asics
- 1994 - Revised standard (named VHDL 1076-1993)

Figure- 5.1: History of VHDL

5.3 ADVANTAGES OF VHDL

There are many reasons for choosing VHDL for design efforts. The most important feature of VHDL is that it improves productivity when used it is used in a structured, top-down approach to design. Real increases in productivity come later, when VHDL has been learnt to a good level and the user have accumulated a library of reusable VHDL components.

Productivity also increases when VHDL is used to enhance communication between team members and when an advantage of the more powerful tools for simulation and design verification that are available is taken. In addition, VHDL allows to design at a more abstract level. Instead of focusing on a gate-level implementation, the behavioral function of the design can be addressed.

It is easy to build and use libraries of commonly used VHDL modules. A design can be reused as many times as required in VHDL. Due to the benefits of reusable code, VHDL statements are generally written in ways that make them general purpose. Writing portable code becomes an automatic reflex.

Another important reason to use VHDL is the rapid pace of development in Electronic Design Automation (EDA) tools and in target technologies. Using a standard language such as VHDL greatly improves the chances of moving into more advanced tools (for example, from a basic low-cost simulator to a more advanced one) without having to re – enter the circuit descriptions. The ability to retarget circuits to new types of device targets (for example, ASICs, FPGAs, and complex PLDs) also improve by using a standard design entry method [41].

5.4 ENTITIES AND ARCHITECTURES

Every VHDL design description consists of at least one entity/architecture pair. An entity declaration describes the circuit as it appears from the "outside" - from the perspective of its input and output interfaces. Entity declaration is analogous to a block symbol on a schematic.

The second part of a minimal VHDL design description is the architecture declaration. Before simulation or synthesis can proceed, every referenced entity in a VHDL design description must be bound with a corresponding architecture. The architecture describes the actual function—or contents—of the entity to which it is bound. Using the schematic as a metaphor, the architecture could be said to be roughly analogous to a lower-level schematic referenced by the higher-level functional block symbol.

5.5 DATA TYPES

Like a high-level software programming language, VHDL allows data to be represented in terms of high-level data types. A data type is an abstract representation of stored data, which might be required in software languages. These data types might represent individual wires in a circuit, or they might represent collections of wires.

The **bit** data type has only two possible values: ‘1’ or ‘0’. A **bit vector** is simply an array of **bits.** Every data type in VHDL has a defined set of values, and a defined set of valid operations. Type checking is strict, so it is not possible, to directly assign the value of an **integer** data type to a **bit vector** data type. A type of functions called type conversion functions can be used in such a case for conversion to a required data type.

5.6 DESIGN UNITS

One concept unique to VHDL when compared to software programming languages and to its main rival, Verilog, is the concept of a design unit. Design units in VHDL (which may also be referred to as library units) are segments of VHDL code that can be compiled separately and stored in a library [41].

Two design units have already been introduced: the entity and the architecture. There are actually five types of design units in VHDL i.e. entities, architectures, packages, package bodies, and configurations. Entities and architectures are the only two design units that are compulsory in any VHDL design description. Packages and configurations are optional.

5.6.1 ENTITIES

A VHDL entity is a statement, indicated by the **entity** keyword, that defines the external specification of a circuit or sub-circuit. The minimum VHDL design description must include at least one entity and a corresponding architecture.

When an entity declaration is written, a unique name for that entity must be provided and a port list defining the input and output ports of the circuit. Each port in the port list must be given a name, direction or mode and a type. Optionally, special type of parameter list can also be included (called a generic list) that allows the engineer to pass additional information into an entity.

5.6.2 ARCHITECTURES

A VHDL architecture declaration is a statement (beginning with the **architecture** keyword) that describes the underlying function and/or structure of a circuit. Each of the architectures in the design must be associated by name with one entity in the design.

VHDL allows creating more than an alternate architecture for each entity. This feature is particularly useful for simulation and for project team environments in which the design of the system interfaces (expressed as entities) is performed by a different engineer than the lower-level architectural description of each component circuit, or when simply experimentation with different methods of description is wanted.

An architecture declaration consists of zero or more declarations of items such as intermediate signals, components that will be referenced in the architecture, local functions and procedures, and constants followed by a **begin** statement, a series of concurrent statements, and an **end** statement.

5.6.3 PACKAGES AND PACKAGE BODIES

A VHDL package declaration is identified by the **package** keyword, and is used to collect commonly used declarations for use globally among different design units. A package can be defined as a common storage area, used to store such things as type declarations, constants, and global subprograms. Items defined within a package can be made visible to any other design unit in the complete VHDL design, and they can be compiled into libraries for later re-use.

A package consists of two basic parts: a package declaration and an optional package body. Package declarations can contain the following types of statements:

- Type and subtype declarations
- Constant declarations
- Global signal declarations
- Function and procedure declarations
- Attribute specifications
- File declarations
- Component declarations
- Alias declarations
- Disconnect specifications
- Use clauses

Items appearing within a package declaration can be made visible to other design units by using a **use** statement.

If the package contains declarations of subprograms, such as functions or procedures, or it defines one or more deferred constants (constants whose value is not immediately given), then a package body is required in addition to the package declaration. A package body which is specified using the **package body** keyword combination must have the same name as its corresponding package declaration, but it can be located anywhere in the design, in the same or a different source file.

5.6.4 CONFIGURATIONS

The final type of design unit available in VHDL is called a configuration declaration. A configuration declaration is roughly analogous to a parts list for the design. A configuration declaration (identified with the **configuration** keyword) specifies which architectures are to be bound to which entities, and it allows the engineer to change how components are connected in the design description at the time of simulation.

Configuration declarations are always optional, no matter how complex a design description is created. In the absence of a configuration declaration, the VHDL standard specifies a set of rules that provide with a default configuration. In the case more than one architecture for an entity has been provided, the last architecture compiled takes precedence and gets bound to the entity.

5.7 LEVELS OF ABSTRACTION

VHDL supports many possible styles of design description. These styles differ primarily in how closely they relate to the underlying hardware. When we speak of the different styles of VHDL, we are really talking about the differing levels of abstraction possible using the language—behavior, dataflow, and structure.

The human language specification must be refined into a description that can actually be simulated. A design module written with a sequential description is one such expression of the design. These are important points in the **Behavior** level of abstraction.

After this initial simulation, the design must be further refined until the description can be efficiently used by a VHDL synthesis tool. Synthesis is a process of translating an abstract concept into a less-abstract form. The highest level of abstraction accepted by synthesis tools these days is the **Dataflow** level.

The **Structure** level of abstraction is used when small parts of circuitry are to be connected together to form bigger circuits. Physical information is the most basic level of all and is outside the scope of VHDL. This level involves actually specifying interconnects of the transistors on a chip, placing and routing macro cells within a gate array or FPGA, etc.

5.7.1 BEHAVIOR

The highest level of abstraction supported in VHDL is called the behavioral level of abstraction. When creating a behavioral description of a circuit, the circuits is described in terms of its operation over time. The concept of time is the critical distinction between behavioral descriptions of circuits and lower-level descriptions specifically descriptions created at the dataflow level of abstraction.

In a behavioral description, the concept of time can be expressed precisely, with actual delays between related events (such as the propagation delays within gates and on wires), or it may simply be an ordering of operations that are expressed sequentially (such as in a functional description of a flip-flop). While writing VHDL for input to synthesis tools, behavioral statements in VHDL can be used to imply that there are registers in your circuit. It is unlikely, however, that the synthesis tool will be capable of creating precisely the same behavior in actual circuitry as defined in the language. It is also unlikely that the synthesis tool will be capable of accepting and processing a very wide range of behavioral description styles.

5.7.2 DATAFLOW

In the dataflow level of abstraction, the circuit is described in terms of how data moves through the system. Registers are the most important part of most digital systems today, so in the dataflow level of abstraction it is described how the information is passed between registers in the circuit. The combinational logic portion of the circuit may also be described at a relatively high level (and let a synthesis tool figure out the detailed implementation in logic gates), but specifications about the placement and operation of registers in the complete circuit are given by the user. The dataflow level of abstraction is often called register transfer logic, or RTL.

There are some drawbacks to using a dataflow method of design in VHDL. There are no built-in registers in VHDL; the language was designed to be general-purpose, and VHDL's designers placed the emphasis on its behavioral aspects. For writing VHDL at the dataflow level of abstraction, first behavioral descriptions of the register elements which will be used in the design must be created or obtained. These elements must be provided in the form of components (using VHDL's hierarchy features) or in the form of subprograms (functions or procedures).

5.7.3 STRUCTURE

The third level of abstraction is structure. It is used to describe a circuit in terms of its components. Structure can be used to create a very low-level description of a circuit such as a transistor-level description or a very high-level description such as a block diagram.

In a gate-level description of a circuit, components such as basic logic gates and flip-flops might be connected in some logical structure to create the circuit. This is often called a net list. For a higher-level circuit — one in which the components being connected are larger functional blocks — structure might simply be used to segment the design description into manageable parts.

Structure-level VHDL features, such as components and configurations, are very useful for managing complexity. The use of components can dramatically improve the ability to re-use elements of the designs and they can make it possible to work using a top-down design approach.

5.8 OBJECTS

VHDL includes a number of language elements, collectively called objects that can be used to represent and store data in the system being described. Three basic types of objects that are used when entering a design description for synthesis or creating functional tests (in the form of a test bench) are signals, variables and constants. Each object that is declared has a specific data type (such as **bit** or **integer**) and a unique set of possible values.

5.8.1 SIGNALS

Signals are objects that are used to connect concurrent elements (such as components, processes and concurrent assignments), similar to the way that wires are used to connect components on a circuit board or in a schematic. Signals can be declared globally in an external package or locally within architecture, block or other declarative region.

5.8.2 VARIABLES

Variables are objects used to store intermediate values between sequential VHDL statements. Variables are only allowed in processes, procedures and functions, and they are always local to those functions. The 1076-1993 language standard adds a new type of global variable that has visibility between different processes and subprograms. Variables in VHDL are much like variables in a conventional software programming language. They immediately take on and store the value assigned to them and they can be used to simplify a complex calculation or sequence of logical operations.

5.9 HARDWARE IMPLEMENTATION OF RS CODES

Since its advent in 1960, efficient and practical Reed-Solomon (RS) decoder design has been an important research topic. RS decoders decode transmitted data using Reed-Solomon linear cyclic block codes to detect and correct errors. Applications include error correction in compact disk players, deep space communications, and military radios. Transistors or gates are fabricated in a 2 dimensional array on a die to form the standard base of an Application Specific Integrated Circuit (ASIC). The device is programmed by custom metal layers interconnecting nodes in the array. Traditionally, RS decoder design exploits (ASIC) technology to realize low processing times and small areas. Nevertheless, ASICs achieve these advantages at the expense of flexibility.

RS decoder design can also be implemented on the Field Programmable Gate Array (FPGA). This device is similar to the gate array, defined above, with the device shipped to the user with general-purpose metallization pre-fabricated, often with variable length segments or routing tracks. The device is programmed by turning on switches which make connections between circuit nodes and the metal routing tracks. The connection may be made by a transistor switch (which are controlled by a programmable memory element) or by an antifuse. The transistor switch may be controlled by an SRAM cell or an EPROM/EEPROM/Flash cell. Timing is generally not easily predictable. Some architectures employ dedicated logic and routing resources for optimizing high-speed functions such as carry chains, wide decodes, etc. Additional features such as programmable I/O blocks, storage registers, etc., may be included in these devices. Commercial, military, and space devices use a variety of programmable elements

General Purpose Processors (GPPs), Digital Signal Processors (DSPs), or hybrid ASICs can provide flexibility, but do so at the expense of processing time and area. FPGAs, however, promise to realize RS decoders with time and area performance similar to ASICs but with flexibility similar to GPPs.

This is intriguing as it opens the door to dynamically changing error control coding based on the current characteristics of the transmission channel. Indeed, FPGA-hosted RS decoders

already exist but little work has been reported on realizing FPGA-based dynamic error control or on simply characterizing the optimal FPGA implementation of RS decoders.

The finite field multipliers are the most numerous building blocks of RS decoders. There are a number a RS decoder design parameters which can be adjusted to reduce the resource requirement, increase speed, and make FPGA-based finite field decoders more feasible. Nevertheless, attempting to optimize the FPGA-based finite field multipliers in terms of both speed and area is an important step.

The dependence of RS decoders on finite field arithmetic implies that the efficient RS decoders are dependent on efficient finite field adders and multipliers. Addition is easy because of the finite fields. It equates to a bit-wise XOR of the *m*-tuple and is realized by an array of *m* XOR gates. The finite field multiplier is, by comparison, much more complicated and is the key to developing efficient finite field computational circuits. Multipliers can be classified into bit parallel and bit serial architectures. The bit parallel multipliers compute a result in one clock cycle but have generally a higher area requirement. The Bit serial multipliers compute a product in k clock cycles but have a lower area requirement.

A number of multipliers can be used for RS Decoders on FPGAs such as Massey-Omura Multiplier, Hasan – Bhargava Multiplier, LFSR Multiplier, Morri – Berlekamp Multiplier, Paar - Rosner Multiplier, Mastrovito Multiplier, Pipelined Combinational Multiplier [18], [35], [42].

The underlying architecture of the target FPGA must be taken into account for optimal performance. Specifically, optimal performance is dependent on single clock cycle throughput, registered datapaths, and maximum use of each logic element. So care must be taken about the design changes that decrease processing time does not increase resource usage such that any gains are nullified [42].

CHAPTER 6

SIMULATION RESULTS OF REED-SOLOMON CODES IN VHDL

In chapter 4, the Reed – Solomon codes and their encoding and decoding procedures were thoroughly discussed. In chapter 5, the basic features of VHDL were discussed. In this chapter the RS codes are implemented in VHDL and their simulation results are thoroughly discussed.

6.1 INTRODUCTION

Reed-Solomon (RS) codes have many communication and memory applications and are very powerful in dealing with burst errors, especially when two RS codes are 'interleaved' to control errors.

Using interleaving, the burst error channel is effectively transformed into an independent error channel for which many FEC coding techniques are applicable. An interleaver scrambles the encoded data stream in a deterministic manner by separating the successive bits transmitted over the channel as widely as possible. In the deinterleaver, the inverse operation is preformed and the received data is unscrambled so that the decoding operation may proceed properly. After deinterleaving, error bursts that occur on the channel are spread out in the data sequence to be decoded, thereby spanning many codewords. The combination of interleaving and FEC thus provides an effective means of combating the effect of error bursts.

As discussed earlier, the RS codes, as with any block code used in a systematic context, can be "shortened" to an arbitrary extent. RS codes defined over the field GF(256) can be said to have a natural length of 256 (the original approach) or 255 (the generator polynomial approach). The compact disc system is able to use RS codes of length 32 and 28 symbols, while still retaining

the 8-bit symbol structure of a length-255 code. The shorter code word length keeps the implementation of the interleaver simple, while the 8-bit symbols provide protection against error bursts and provide a good match for the 16-bit samples taken from the analog music source.

In this thesis, a (32, 28) RS code has been implemented in VHDL. The encoding and decoding of the (32, 28) RS code using various modules and their simulation results are given in this chapter.

6.2 THE RS ENCODER

The encoding process has been described in section 4.2**,** so now the VHDL module of the encoder is considered here.

The encoder has been designed using behavioral style of modeling.

6.2.1 RESULTS FOR ENTITY RS ENCODER

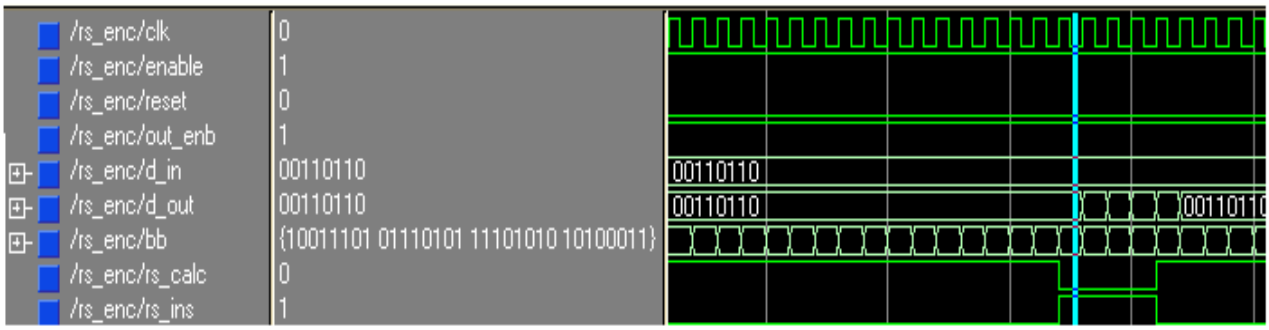


Figure-6.1: Simulation Results of Entity Encoder

6.3 MODULES IN REED – SOLOMON DECODER

The Decoder consists of the following parts:

- Syndrome Calculator
- Euclid's Algorithm- for calculating the error locator polynomial

- Chien's Search Algorithm- for calculating the location of errors.
- Forney's Algorithm – for calculating the magnitude of errors.
- Error Corrector – for obtaining the corrected codeword.

The complete description of these components has been given in **section 4.3.** These components were implemented in VHDL using the behavioral modeling.

The Euclid's block is divided into two parts:

- Euclid's Multiplication Block and
- Euclid's Division Block.

Separate entities are designed to implement these blocks.

The simulation results obtained for entities are given below:

6.3.1 RESULTS FOR THE ENTITY SYNDROME CALCULATOR

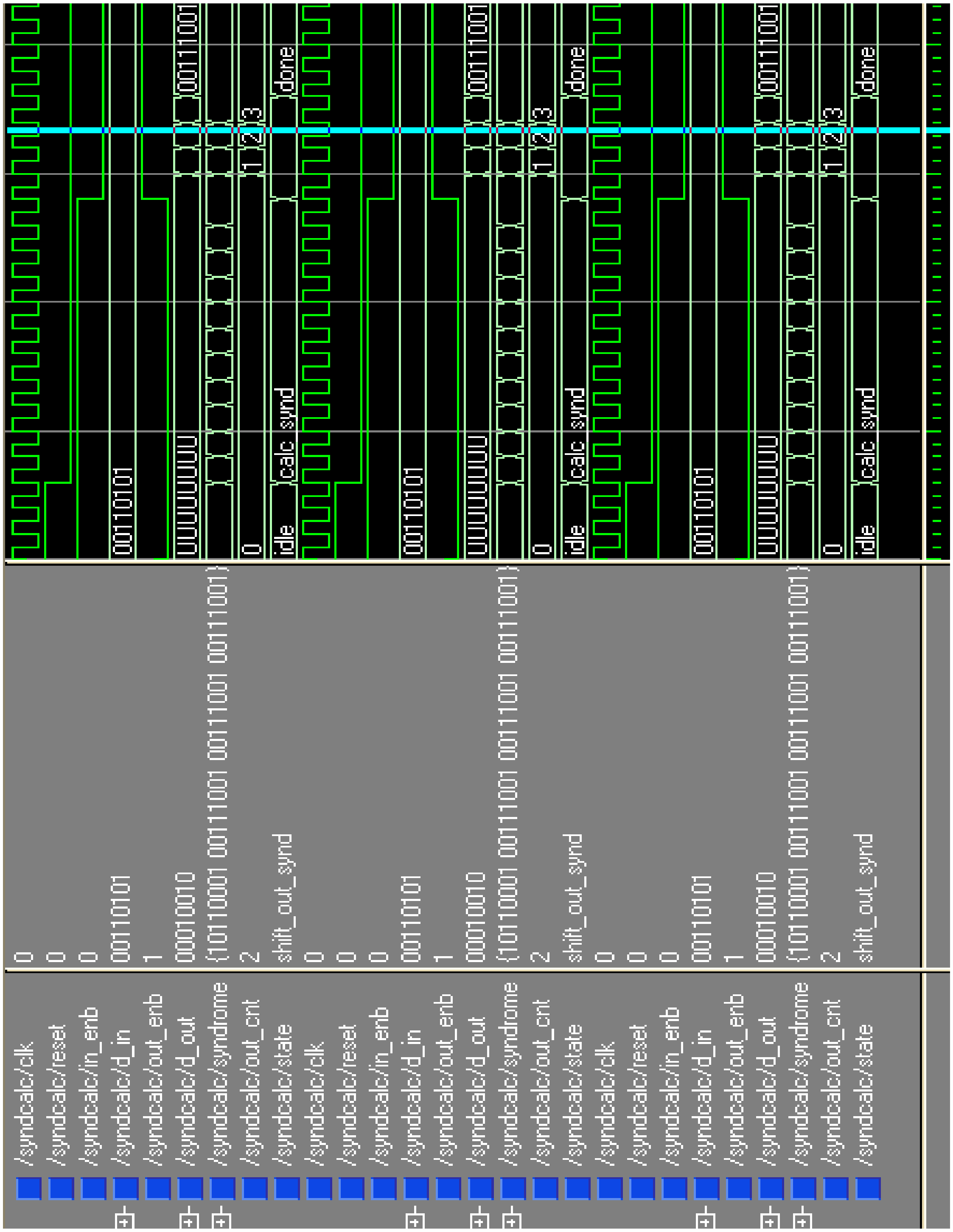


Figure- 6.2: Simulation Results of Syndrome Calculator

6.3.2 RESULTS FOR ENTITY EUCLID'S MULTIPLICATION BLOCK

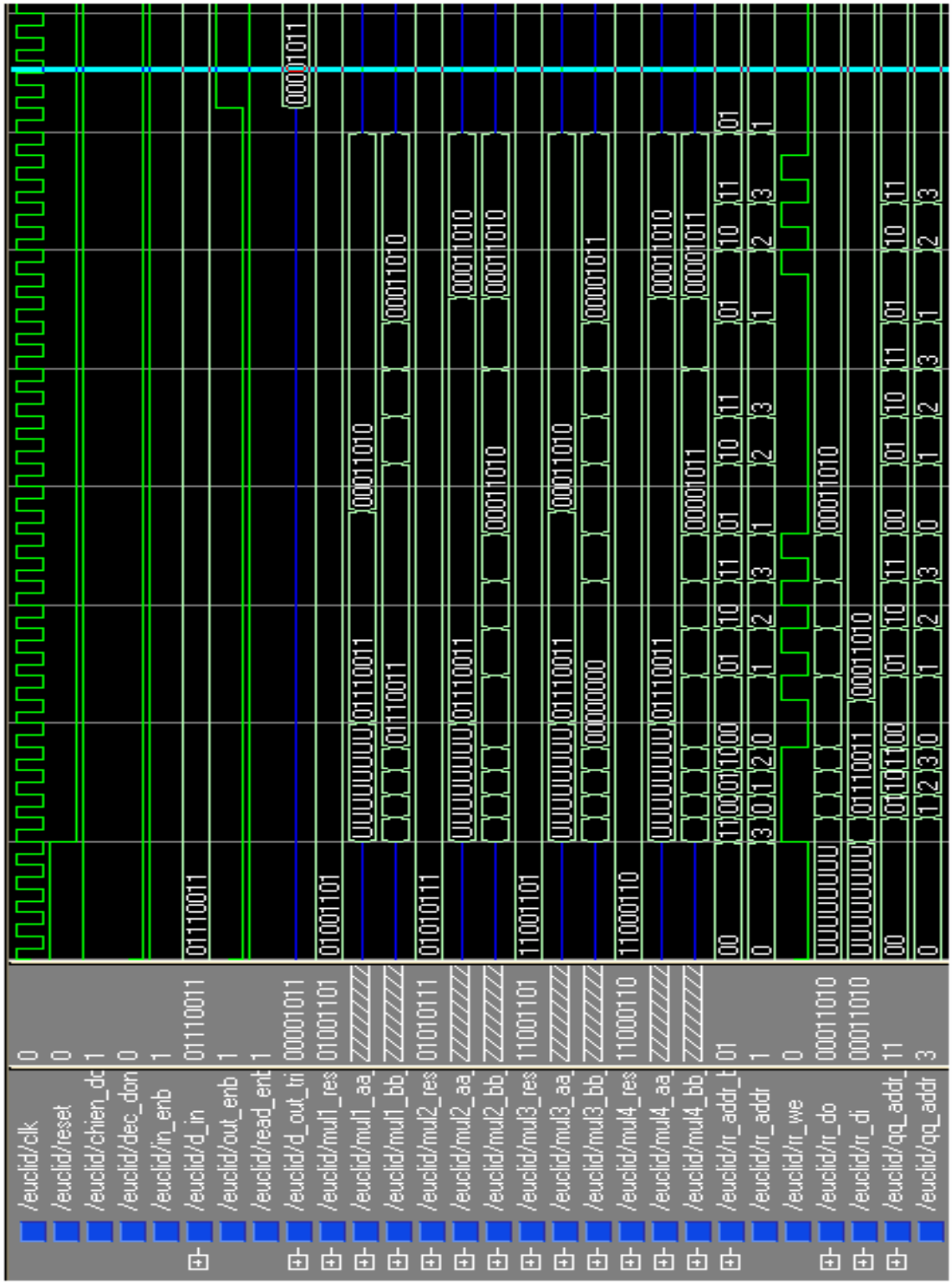


Figure- 6.3(a): Simulation Results of Euclid's Multiplication Block

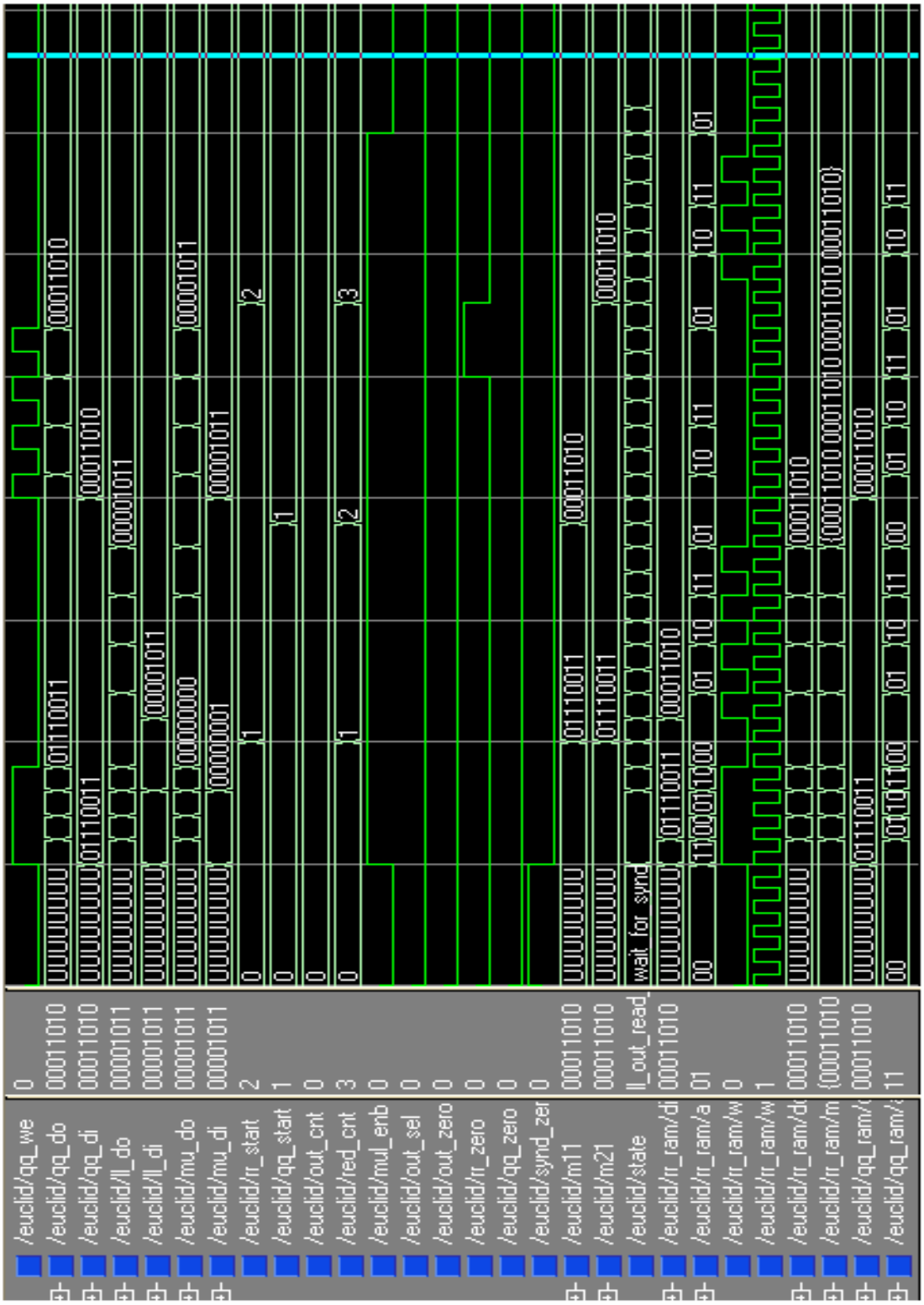


Figure- 6.3(b): Simulation Results of Euclid's Multiplication Block

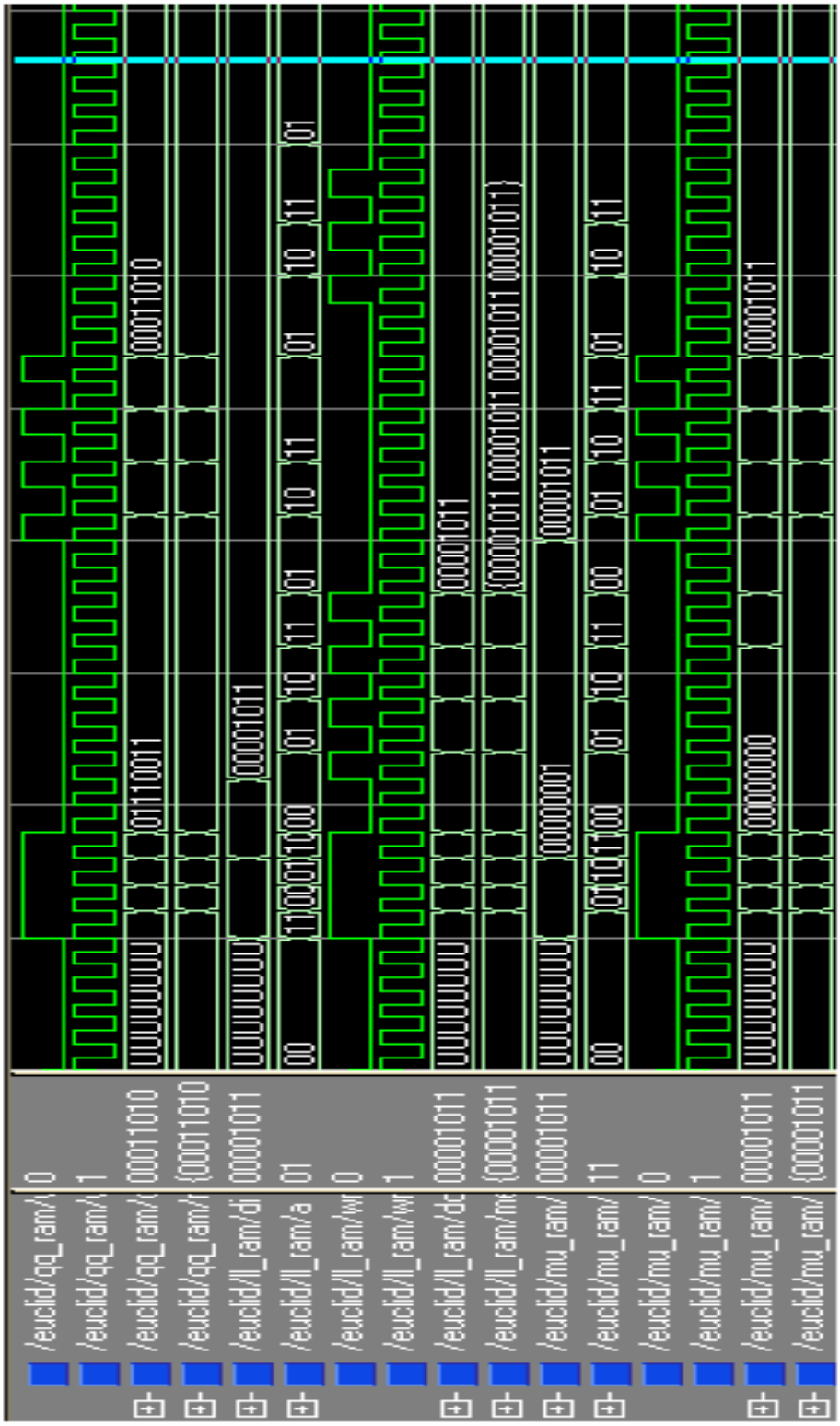


Figure- 6.3(c): Simulation Results of Euclid's Multiplication Block

6.3.3 RESULTS FOR THE ENTITY EUCLID'S DIVISION BLOCK

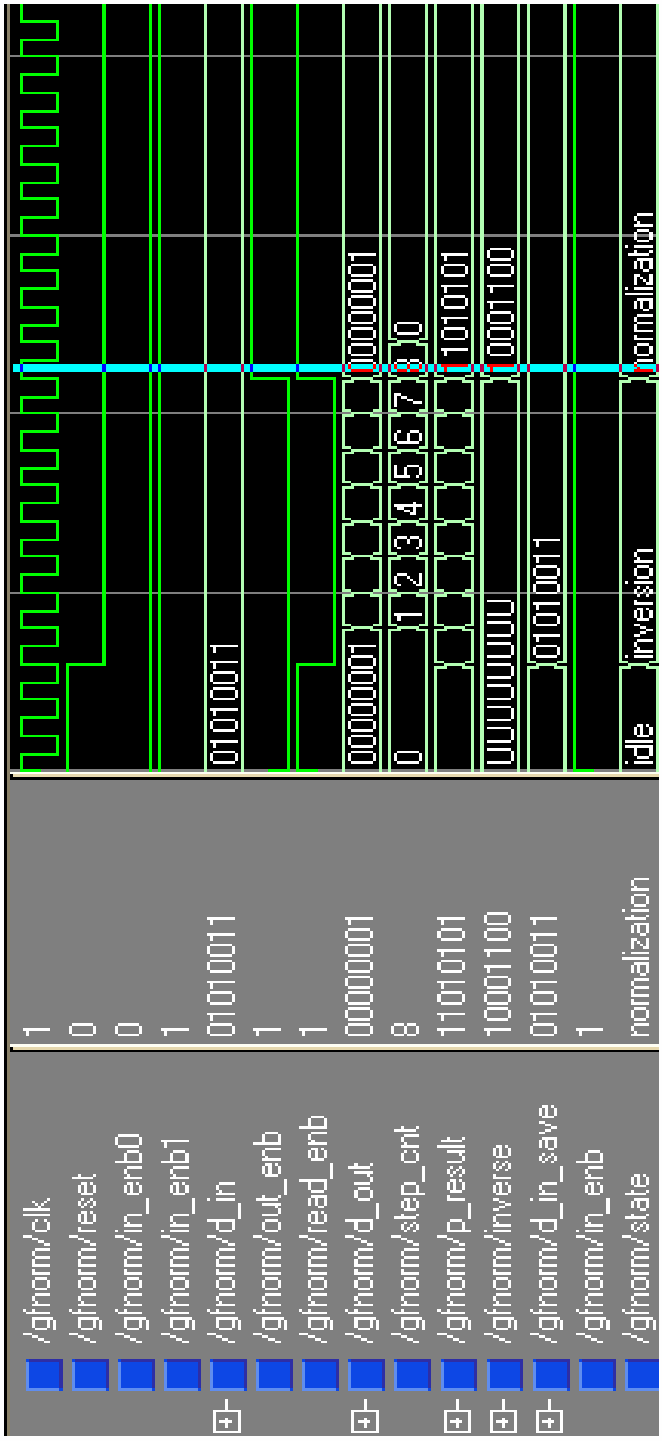


Figure- 6.4(a): Simulation Results of Euclid's Division Block

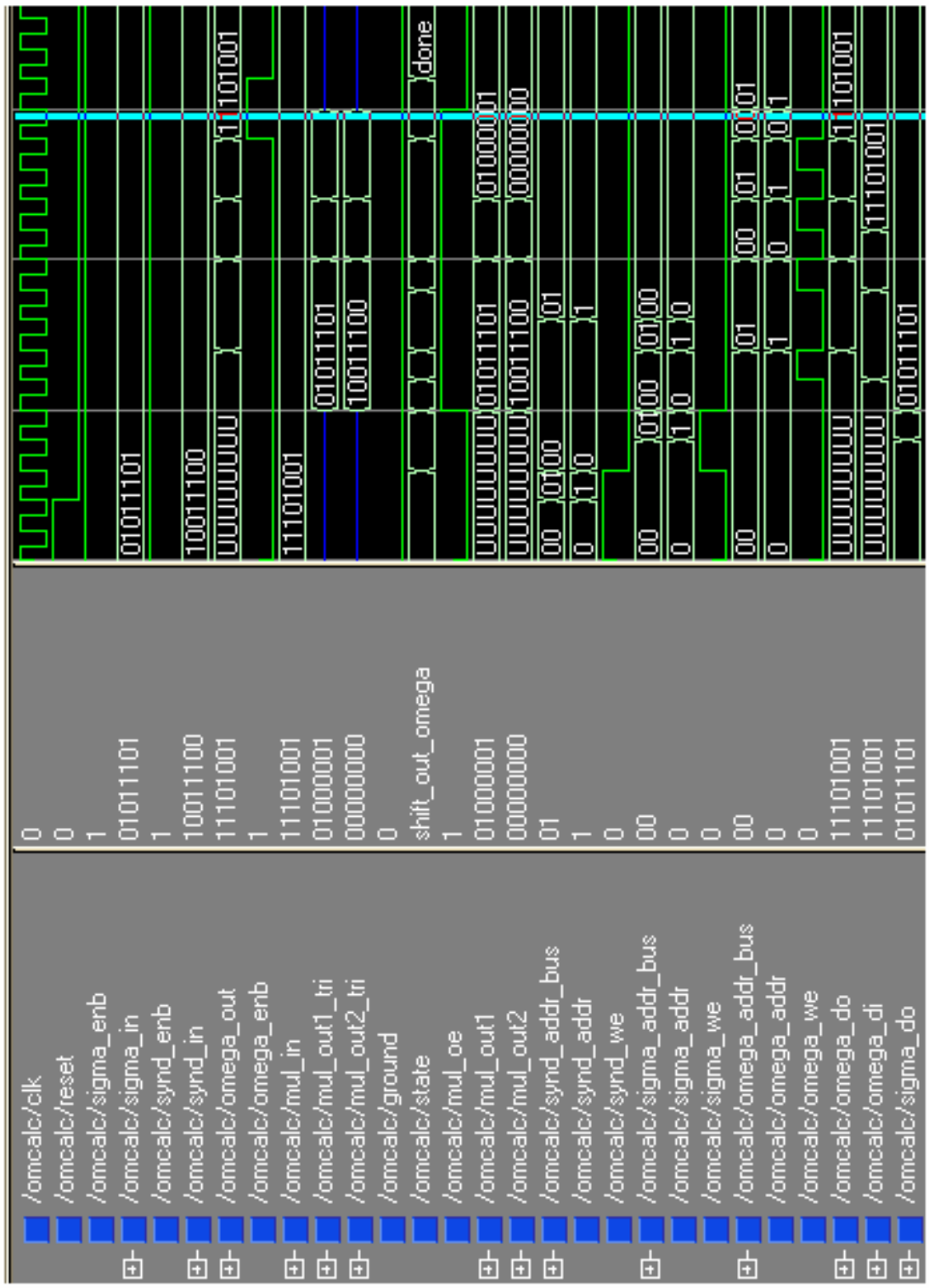


Figure- 6.4(b): Simulation Results of Euclid's Division Block

6.3.4 RESULTS FOR THE ENTITY CHIEN SEARCH AND FORNEY'S ALGORITHM

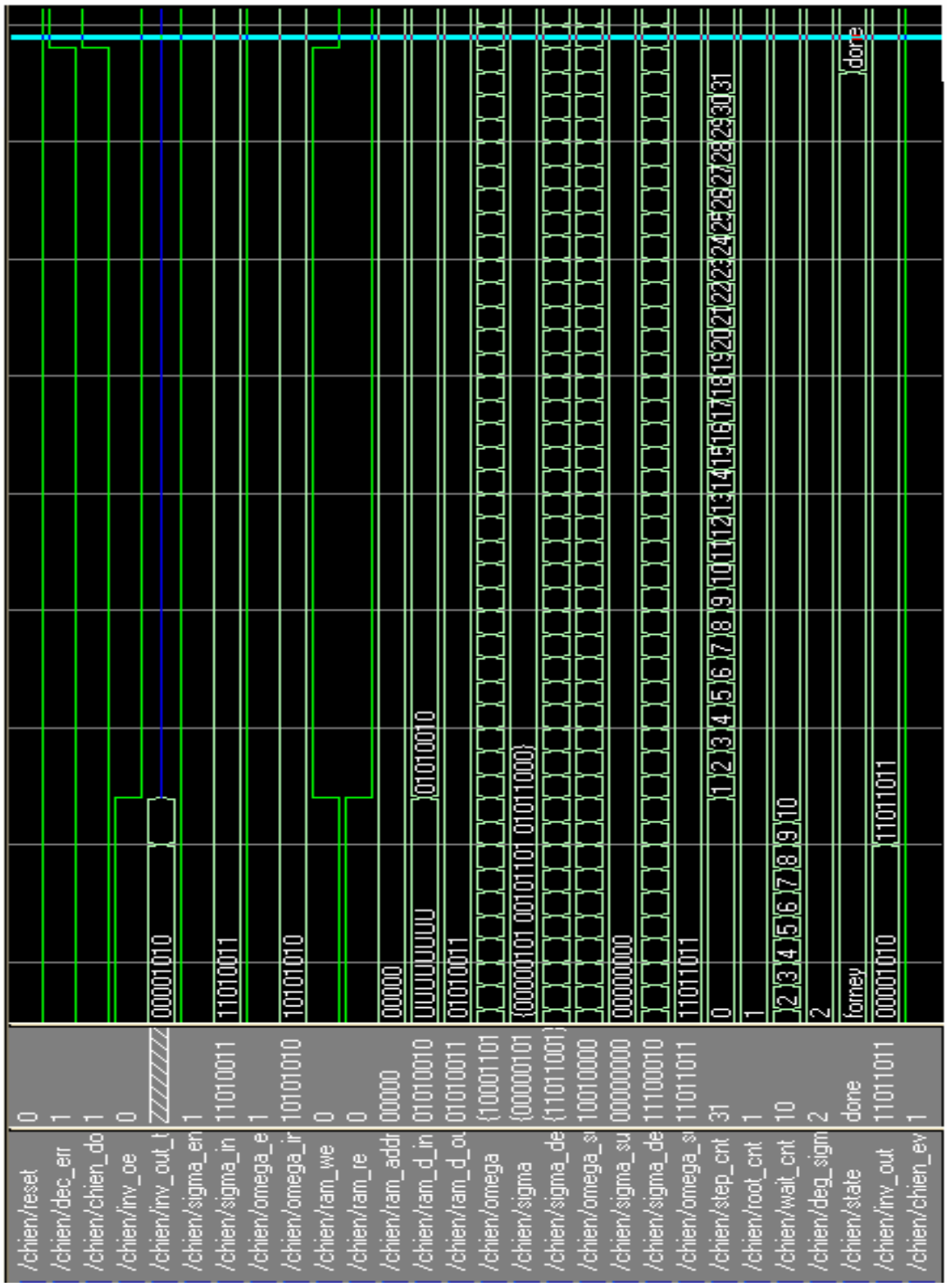


Figure- 6.5: Simulation Results of Chien search and Forney Algorithm

6.3.5 RESULTS FOR THE ENTITY FIFO DELAY BUFFERS

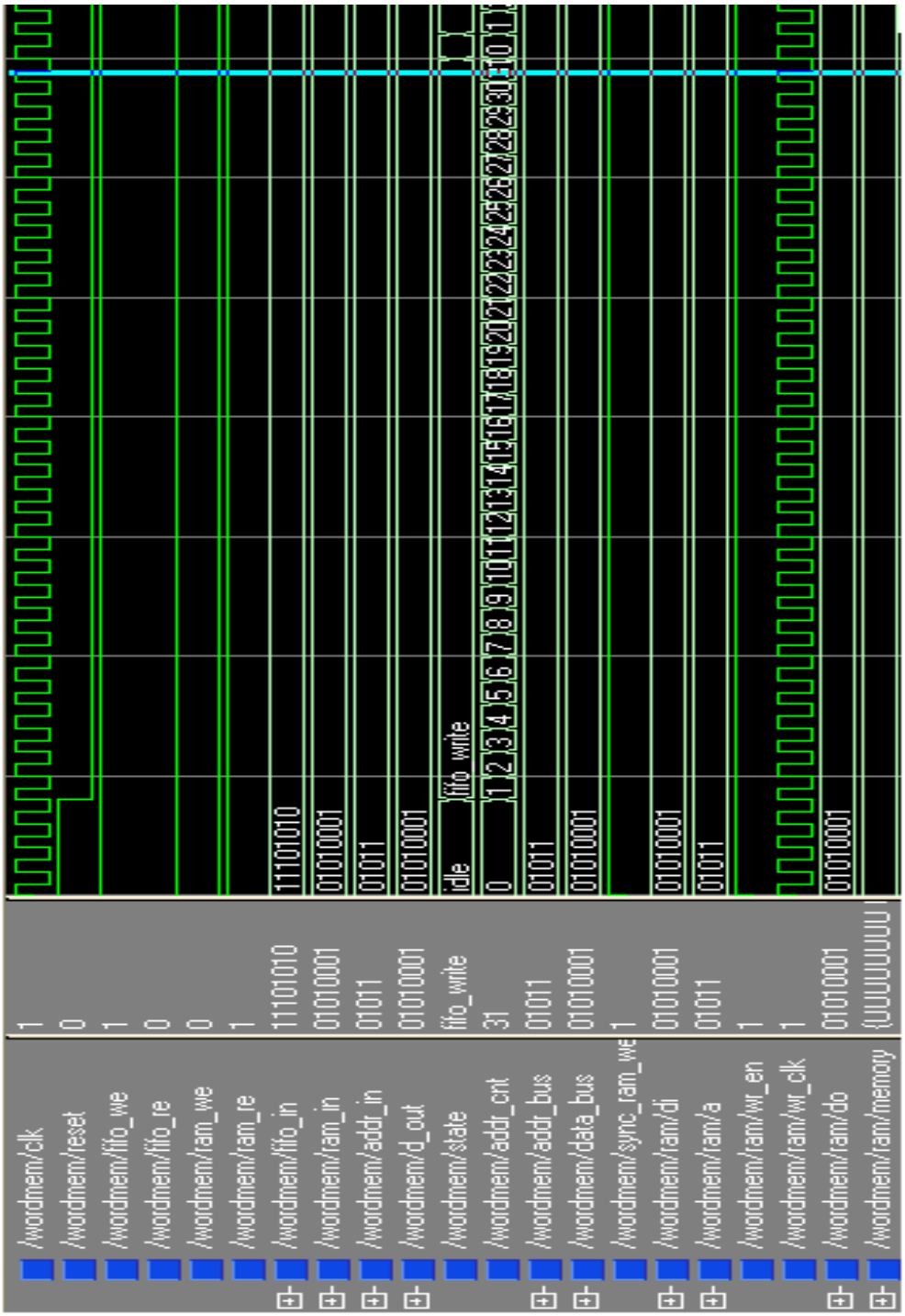


Figure- 6.6: Simulation Result of FIFO Delay Buffers

6.4 THE RS DECODER

The RS decoder is implemented using all the entities implemented above. Structural style of modeling is used for mapping various components of the decoder to this entity. The complete simulation results after port - mapping the components to the ports of the decoder entity are shown below:

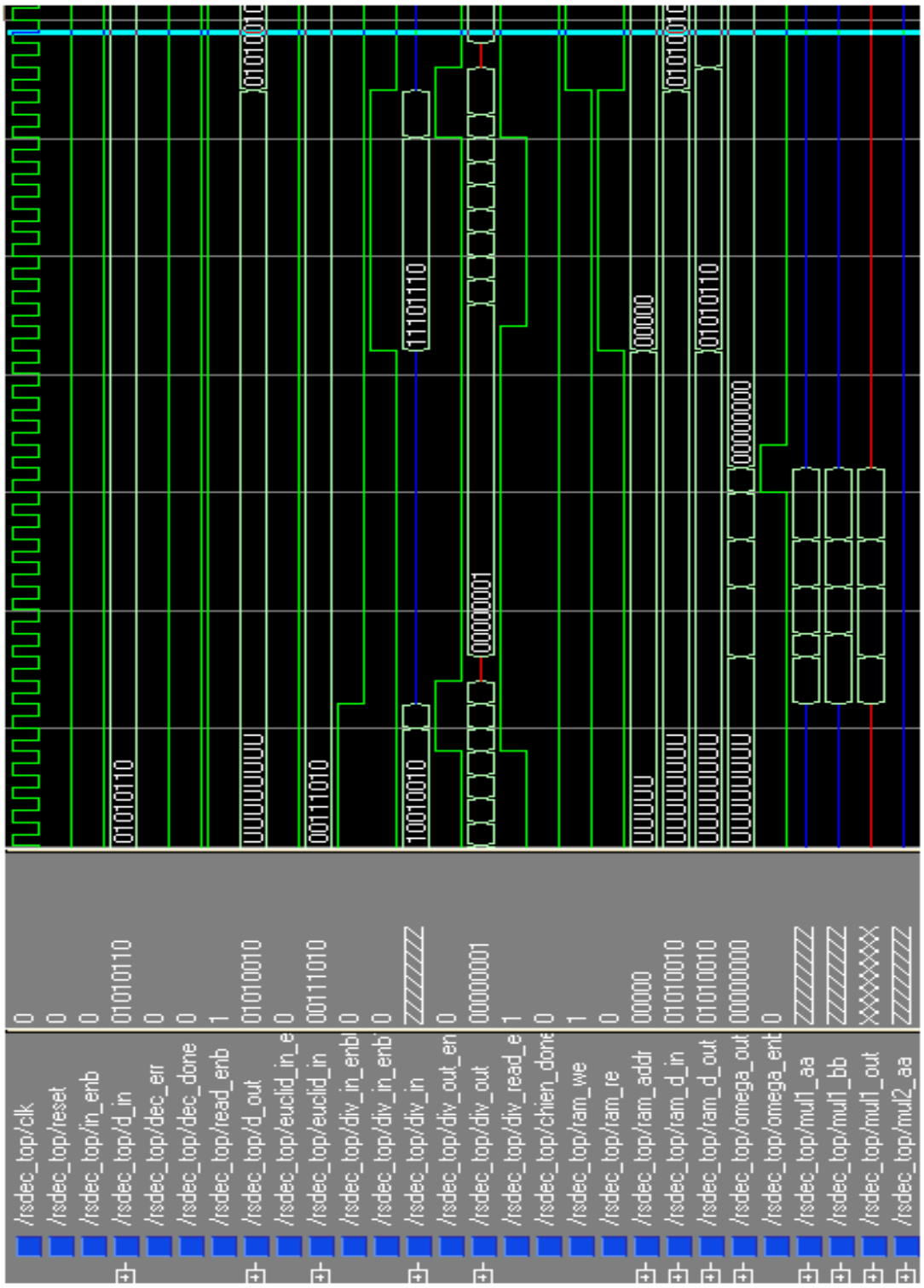


Figure- 6.7(a): Simulation Results of Decoder

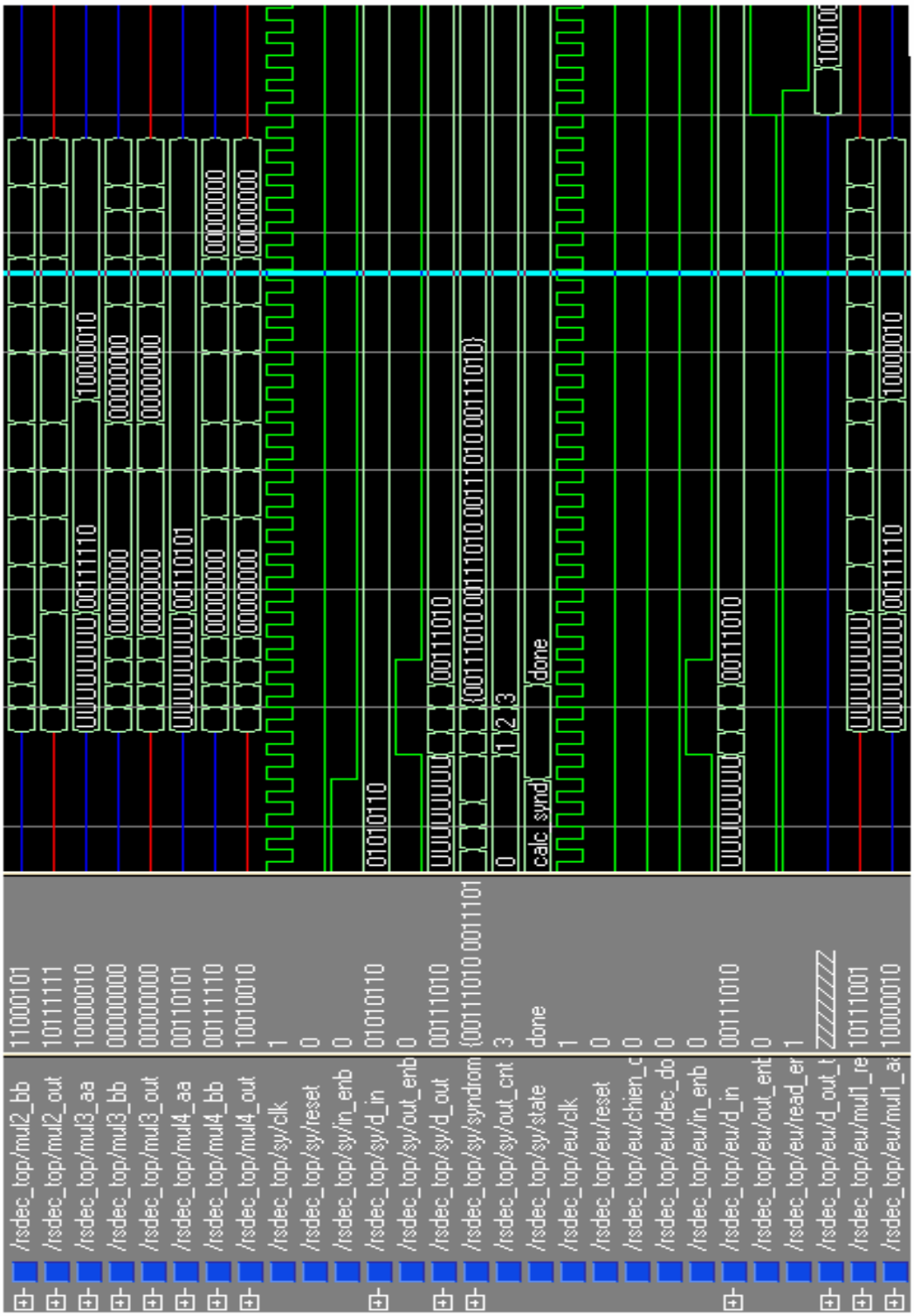


Figure- 6.7(b): Simulation Results of Decoder Using Entity Syndrome Calculator

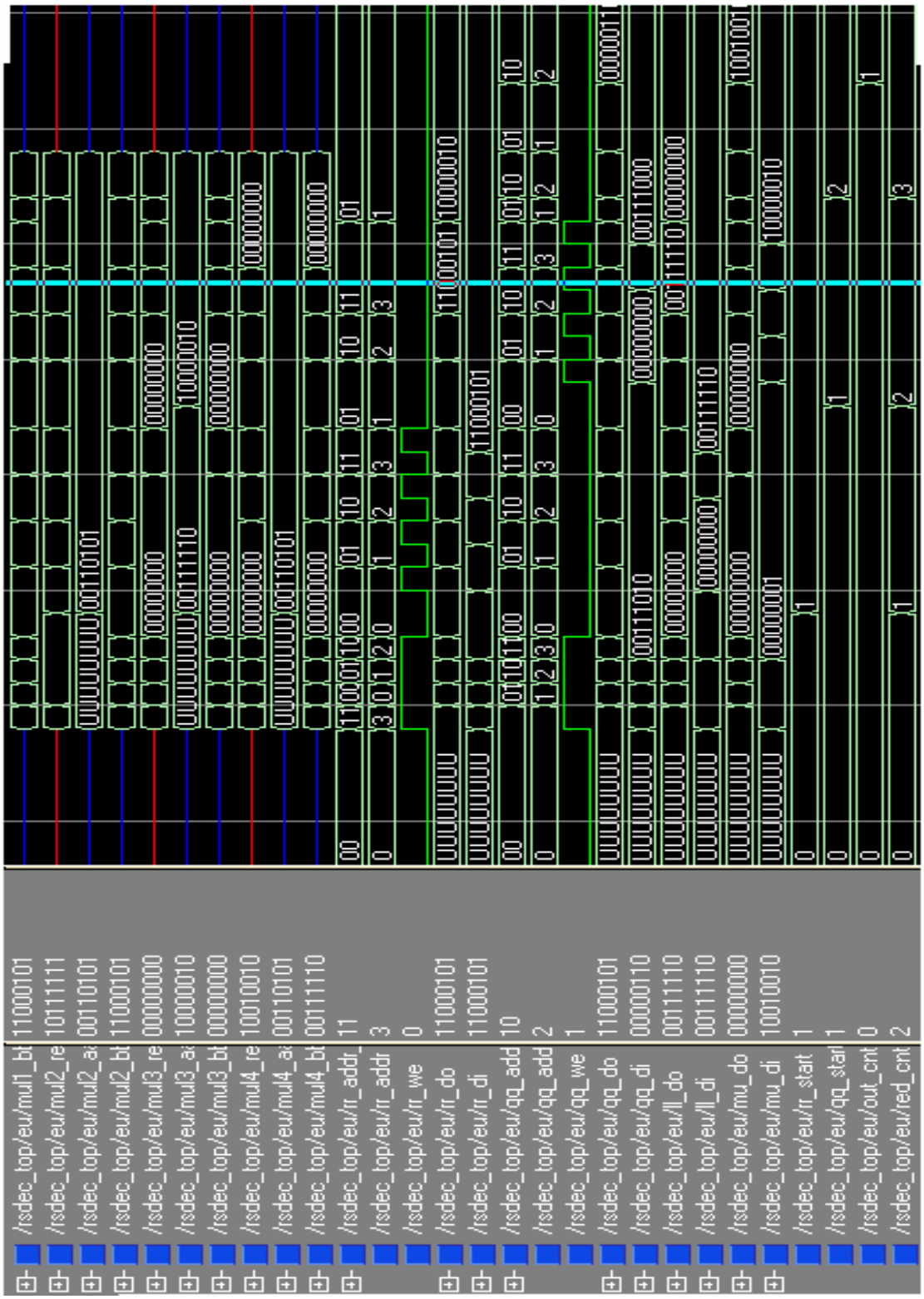


Figure- 6.7(c): Simulation Results Of Decoder Using Entity Euclid Multiplication Block

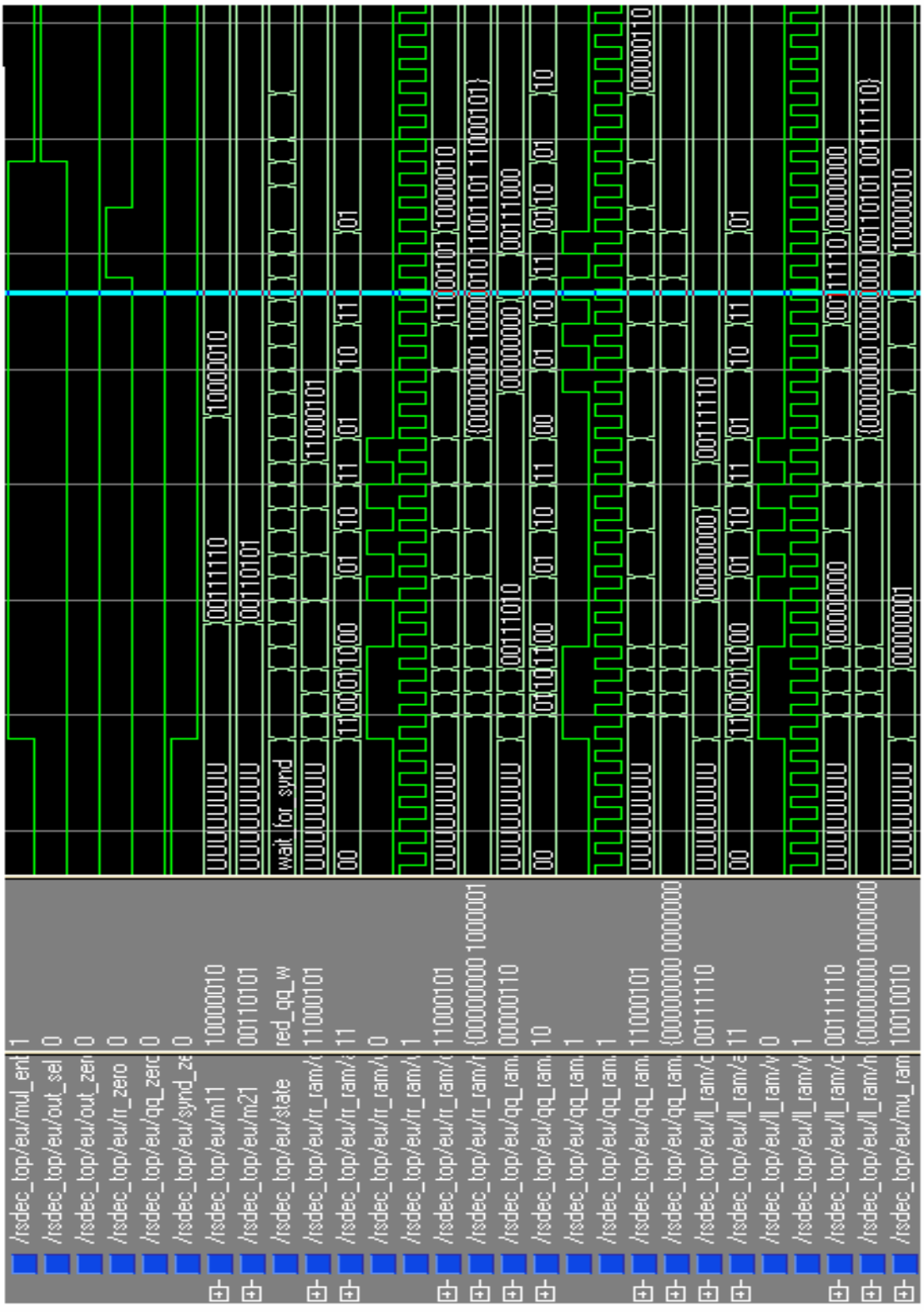


Figure-6.7(d): Simulation Results of Decoder Using Entity Euclid Multiplication Block

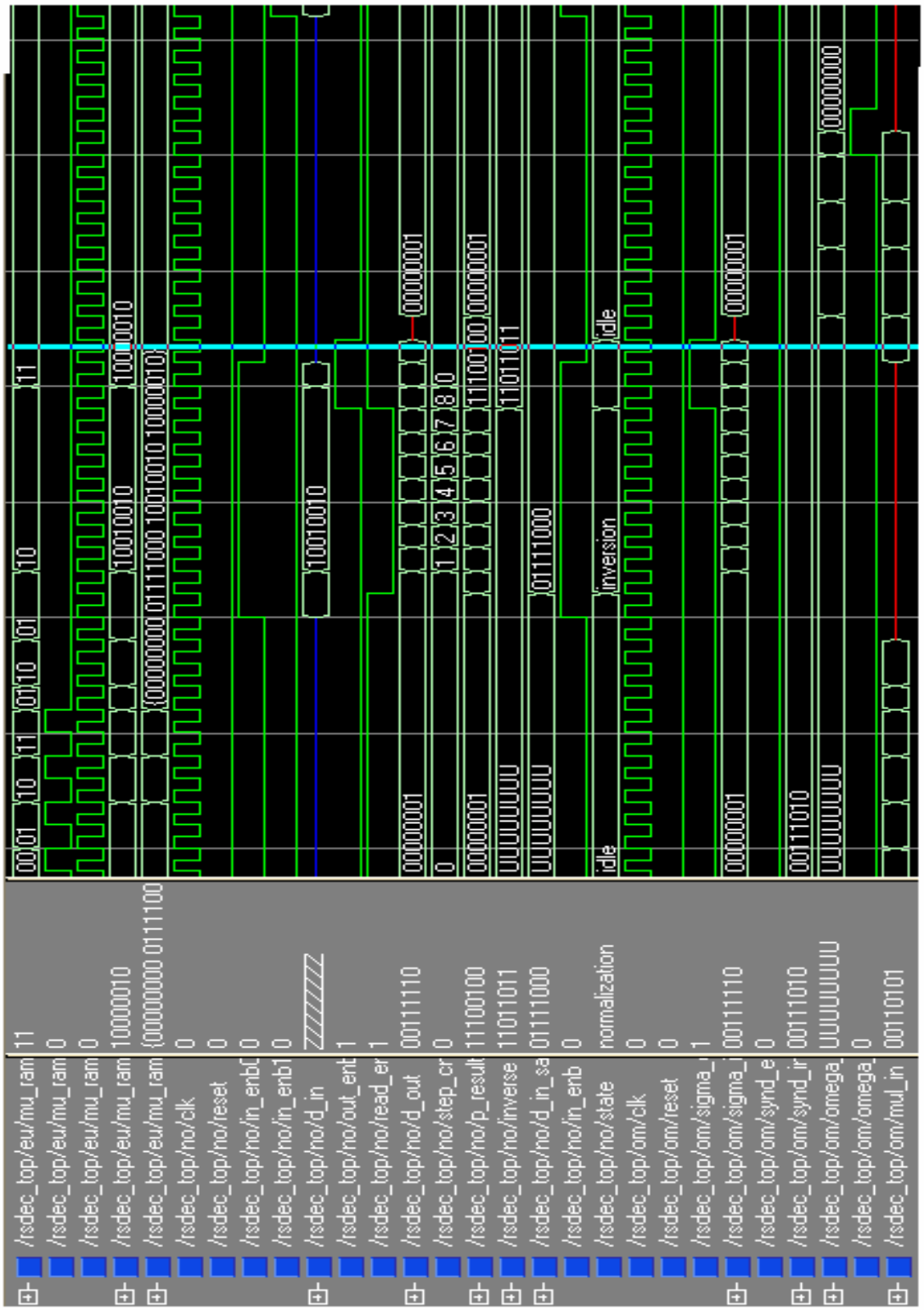


Figure- 6.7(e): Simulation Results of Decoder Using Euclid Division Block

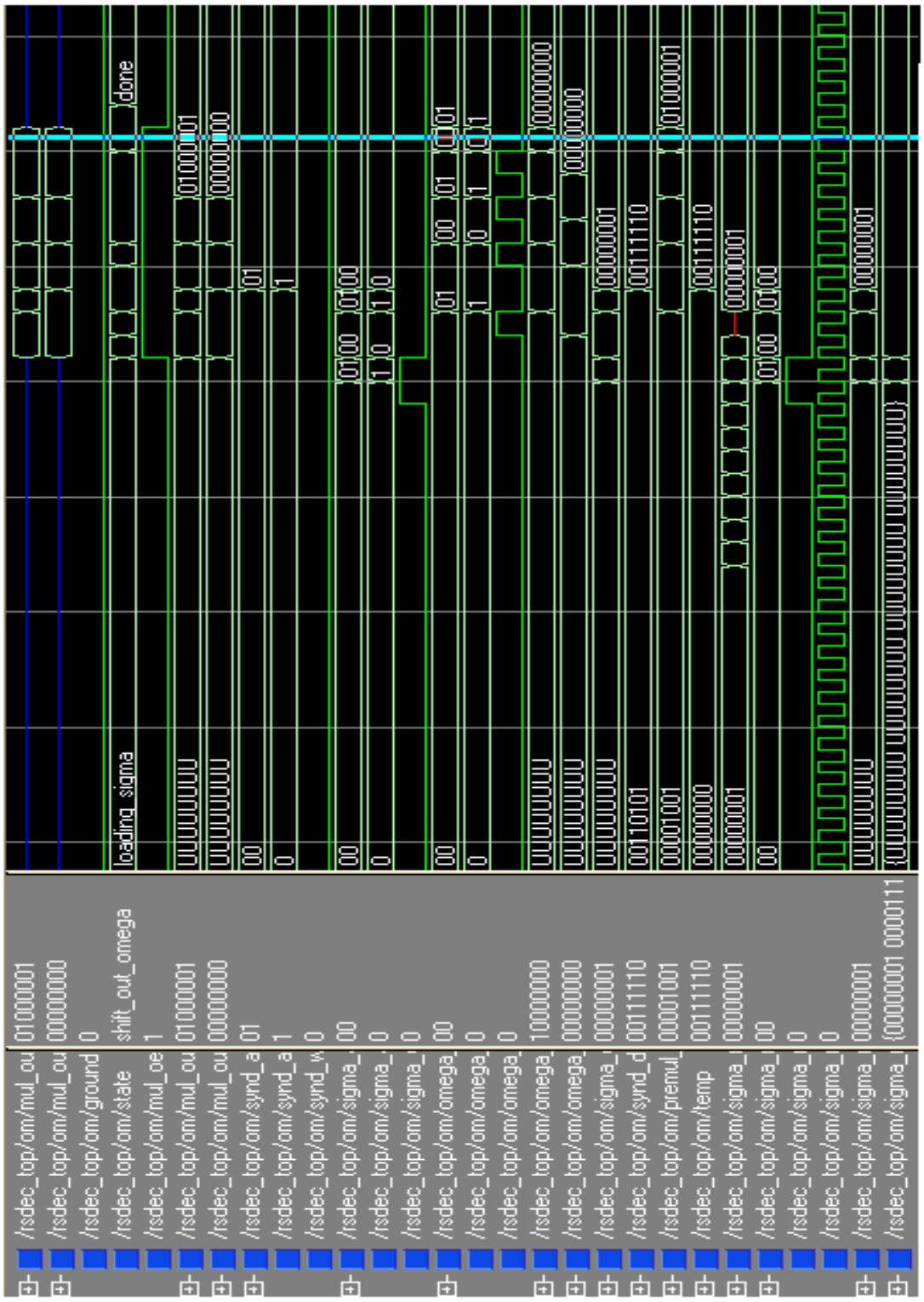


Figure- 6.7(f): Simulation Results of Decoder Using Euclid Division Block

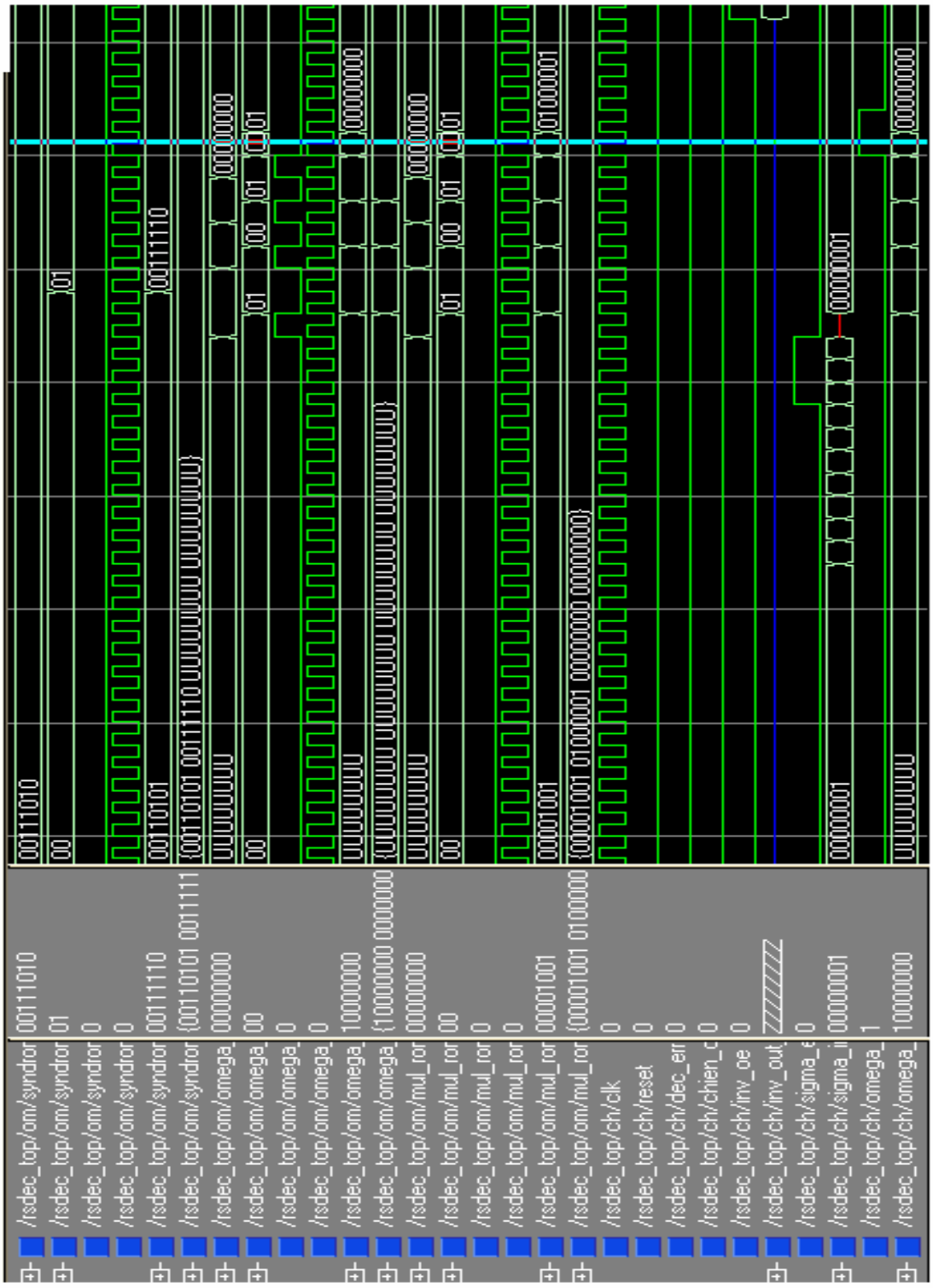


Figure- 6.7(g): Simulation Results of Decoder Using Euclid Division Block

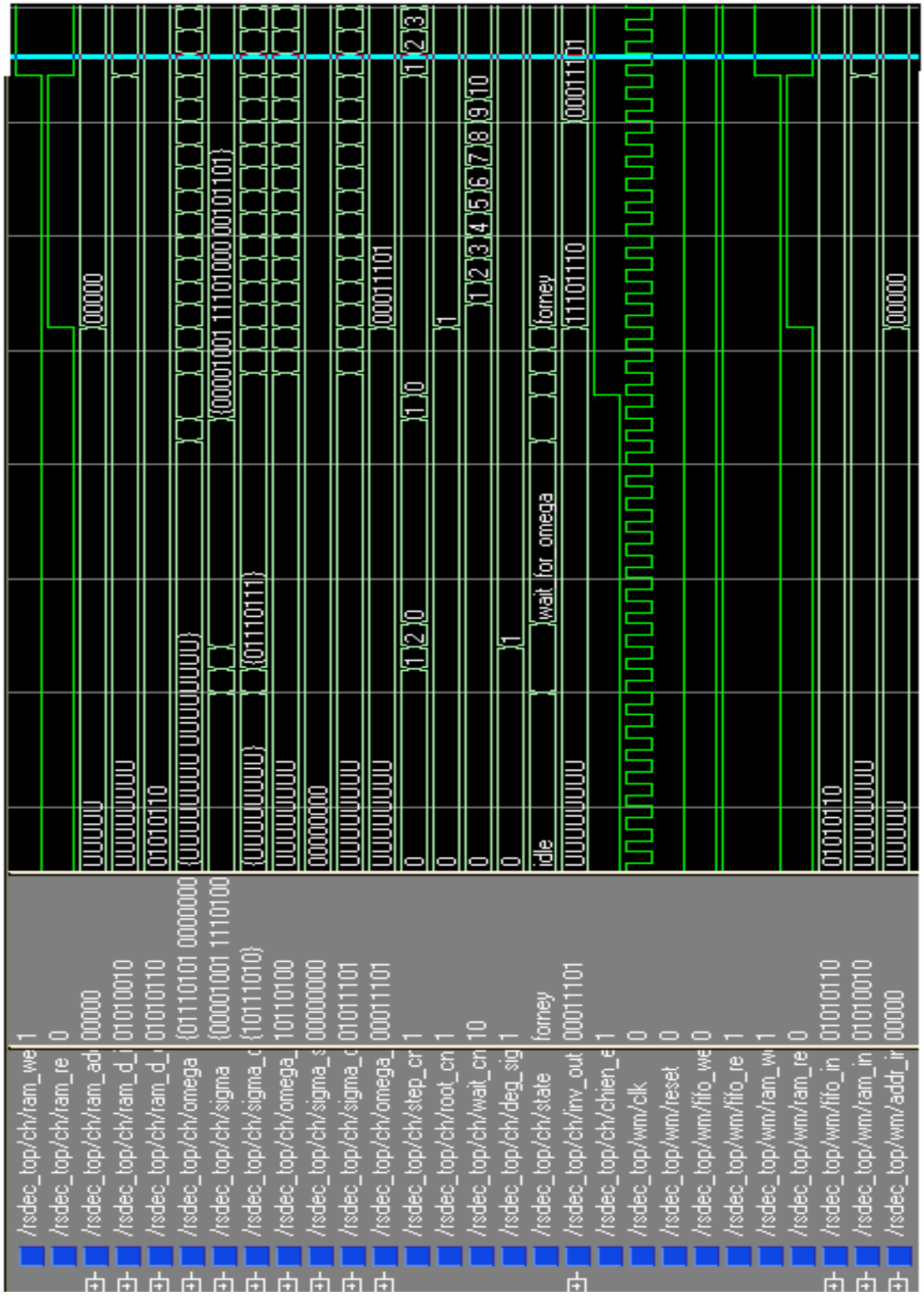


Figure- 6.7(h): Simulation Results of Decoder Using Chien Search and Forney's Algorithm

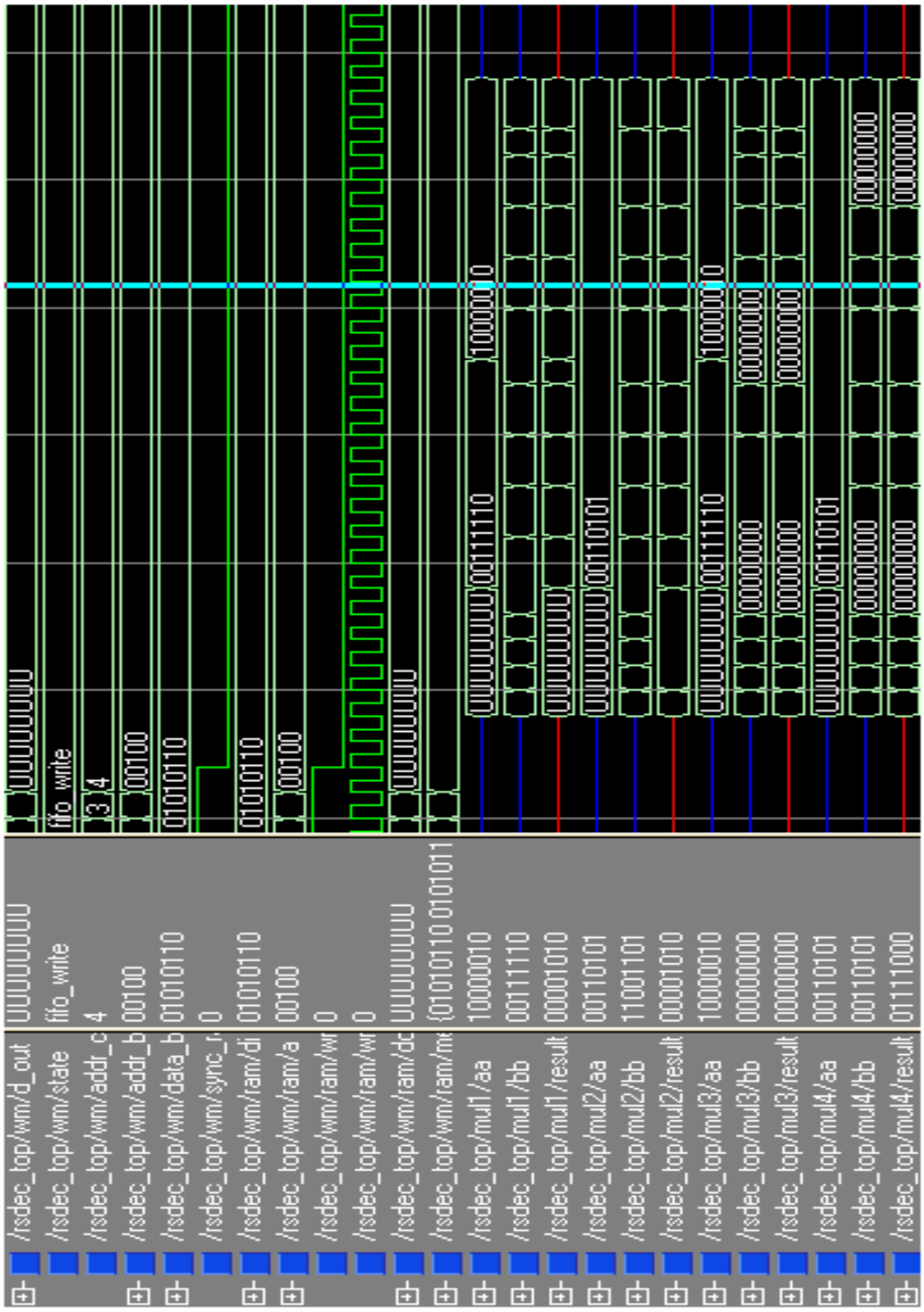


Figure- 6.7(i): Simulation Results of Decoder Using Entity FIFO Delay Buffer

CHAPTER 7
CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

In this thesis, error detection and correction techniques have been used which are essential for reliable communication over a noisy channel. The effect of errors occurring during transmission is reduced by adding redundancy to the data prior to transmission. The redundancy is used to enable a decoder in the receiver to detect and correct errors. Cyclic Linear block codes are used efficiently for error detection and correction. The encoder splits the incoming data stream into blocks and processes each block individually by adding redundancy in accordance with a prescribed algorithm. Likewise, the decoder processes each block individually and it corrects errors by exploiting the redundancy present in the received data. An important advantage of cyclic codes is that they are easy to encode. Also they posses a well defined mathematical structure which has lead to very efficient decoding schemes for them.

Galois finite fields are used for the processing of linear block codes. In each Galois field there exists a finite element α, and all other nonzero elements of the field are represented as powers of this primitive element α. The symbols used in the block codes are the elements of a finite Galois field. Due to modulo arithmetic followed in the finite fields, the resultant of any algebraic operation is mapped into the field and hence rounding issues are conveniently solved. So the addition, subtraction, multiplication or division of two codewords is again a valid codeword inside the field.

Reed – Solomon codes are one of the most powerful and efficient nonbinary error – correcting codes for detecting and correcting burst errors. An irreducible generator polynomial is used for generating the encoded data called codeword. All encoded data symbols are elements of the Galois field defined by the parameters of the application and the properties of the system. The encoder is implemented using linear shift registers with feedback. The decoder checks the

received data for any errors by calculating the syndrome of the codeword. If an error is detected, the process of correction begins by locating the errors first. Generally Euclid's Algorithm is used to calculate the error – locator polynomial it is very easy to implement, while its counterpart Berlekamp - Massey Algorithm is more hardware efficient. The precise location of the errors is calculated by using Chien search algorithm. Then magnitude of the errors is calculated using Forney's algorithm. The magnitude of the error is added to the received codeword to obtain a correct codeword. Hence the using the Reed – Solomon codes, burst errors can be effectively corrected.

Reed – Solomon codes are efficiently used for compact discs to correct the bursts which might occur due to scratches or fingerprints on the discs.

7.2 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 fibers see increased use in high-speed multiprocessors, Reed-Solomon codes can be used there as well.

Convolutional encoding with Viterbi decoding has been in use for many years in commercial satellite systems. A characteristic of the Viterbi decoding process is that as the power density ratio (E_b / N_0) becomes increasingly smaller, the uncorrectable errors that are passed through the system are clumped together. Although the distribution of errors caused on the link is Gaussian, the uncorrectable error distribution at the Viterbi decoder output tends to occur in

bursts. The obvious conclusion is to concatenate the Viterbi and Reed - Solomon codes. The resulting scheme, called Reed-Solomon/ Viterbi concatenated coding offers high reliability and modest complexity. These two desirable features make Reed-Solomon codes an apparent choice for deep space probes. They have been efficiently used in the image communication system of NASA's *Voyager* mission.

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

REFERENCES

[1] C.E. Shannon, “A Mathematical Theory Of Communication ”, Bell System Technology Journal, volume 27, pp. 379-423, 623-656, 1948.S

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

[3] E. Prange, "Some Cyclic Error-Correcting Codes with Simple Decoding Algorithms," Air Force Cambridge Research Center-TN-58-156, Cambridge, Mass., April 1958.

[4] E. Prange, "The Use of Coset Equivalence in the Analysis and Decoding of Group Codes," Air Force Cambridge Research Center-TR-59-164, Cambridge, Mass., 1959.

[5] R. C. Bose and D. K. Ray-Chaudhuri, "On a Class of Error Correcting Binary Group Codes," Information and Control, Volume 3, pp. 68-79, March 1960.

[6] R. C. Bose and D. K. Ray-Chaudhuri, "Further Results on Error Correcting Binary Group Codes," Information and Control, Volume 3, pp. 279-290, September 1960.

[7] A. Hocquenghem, "Codes correcteurs d'erreurs," Chiffres, Volume 2, pp. 147-156, 1959

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

[9] R. J. McEliece, Finite Fields for Computer Scientists and Engineers, Boston: Kluwer Academic, 1987

[10] S. B. Wicker, Error Control Systems for Digital Communication and Storage, N.J.: Prentice-Hall, 1994.

[11] R. C. Singleton, "Maximum Distance Q-nary Codes," IEEE Transactions on Information Theory, Volume IT-10, pp. 116-118,1964.

[12] D. Gorenstein and N. Zierler, "A Class of Error Correcting Codes in pm Symbols," Journal of the Society of Industrial and Applied Mathematics, Volume 9, pp. 207-214, June 1961.

[13] W. W. Peterson, "Encoding and Error-Correction Procedures for the Bose-Chaudhuri Codes," IRE Transactions on Information Theory, Volume IT-6, pp. 459-470, September 1960.

[14] 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.

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

[16] E. R. Berlekamp, "Nonbinary BCH Decoding," paper presented at the 1967 International Symposium on Information Theory, San Remo, Italy.

[17] E. R. Berlekamp, Algebraic Coding Theory, New York: McGraw-Hill, 1968.

[18] J. L. Massey, "Shift Register Synthesis and BCH Decoding," IEEE Transactions on Information Theory, Volume IT-15, Number 1, pp. 122- 127, January 1969.

[19] Y. Sugiyama, Y. Kasahara, S. Hirasawa, and T. Namekawa, "A Method for Solving Key Equation for Goppa Codes," Information and Control, Volume 27, pp. 87-99, 1975.

[20] Tommy Oberg, “Modulation, Detection and Coding”, John Wiley and Sons, 2001.

[21] N K Jha, “Separable Codes For Detecting Unidirectional Errors”, IEEE Trans. on Computer-Aided Design, v8, 571 - 574, 1989.

[22] J E Smith, “On Separable Unordered Codes”, IEEE Trans. Computers, C33, 741 - 743, 1984.

[23] H Dong, “Modified Berger Codes For Detection of Unidirectional Errors”, IEEE Trans. Comput, C33, 572 - 575, 1984.

[24] M. Nicolaidis, “Design for Soft-Error Robustness to Rescue Deep Submicron Scaling”, White Paper, iRoC Technologies, 2000.

[25] Lin Shu, “An Introduction To Error-Correcting Codes”, Prentice Hall, Inc., 1970.

[26] B Bose, “Burst Unidirectional Error Detecting Codes”, IEEE Trans. Comput, C35, 350 - 353, 1989.

[27] Favalli M., Metra C., “Optimization Of Error Detecting Codes For The Detection Of Crosstalk Originated Errors”, Design Automation and Test in Europe, pp 290-296, March 2001.

[28] Syed Shahzad Shah, Saqib Yaqub, and Faisal Suleman, Self-correcting codes conquer noise Part 2: Reed-Solomon codecs, Chameleon Logics, (Part 1: Viterbi Codecs), 2001.

[29] Ling-Pei Kung, “Introduction To Error Correcting Codes”, NSDL Scout Report for Math, Engineering, and Technology, 2003.

[30] Raghav Bhaskar, Pradeep K. Dubey, Vijay Kumar, Atri Rudra, “Efficient Galois Field Arithmetic On SIMD Architectures”, ACM Symp. On ParallelAlgorithms and Architectures, pp 256-257, 2003.

[31] T.K. Truong, L.J. Deutsch, I.S. Reed, I.S. Hsu, K. Wang, and CS. Yeh, 'sThe VLSI Design of a Reed-SoIomon Encoder Using Berlekamp's Bit-Senal Multiplier Algorithm," Third Caltech Conf. on VLSI, pp. 303-329, 1983.

[32] I.S. Hsu, I.S. Reed, T.K. Truong, K. Wang, CS. Yeh, and L.J. Deutsch, "The VLSI Implementation of a Reed-Solomon Encoder Using Berlekamp's Bit-Serial Multiplier Algorithm," IEEE Trans. Comput., vol. (3-33, no. 10, pp. 906-911, Oct. 1984.

[33] C.C. Wang, T.K Truong, H.M. Shao, L.J. Deutsch, J.K. Omura, and I.S. Reed,"VLSI Architectures for Comput ing Multiplications and Inverses in GF(2m)' IEEE Trans. Comput., vol. G34, no. 8, pp. 709-716, Aug. 1985.

[34] James S.Plank, A Tutorial on Reed-Solomon Coding for Fault- Tolerance in RAID- like Systems, Technical Report CS-96-332.

[35] M .A. Hasan and V. K. Bhargava, "Division and Bit-Serial Multiplication over GF(2m)," IEE Proc., part E, vol. 139, no. 3, pp. 230-236, May 1992.

[36] G. Orlando and C. Paar, “A Super-Serial Galois Fields Multiplier For Fpgas And Its Application To Public-Key Algorithms”, Proc. of the 7th Annual IEEE Symposium on Field Programmable Computing Machines, FCCM’99, Napa Valley, California, pp. 232-239, April. 1999.

[37] S. B. Wicker and V. K. Bhargava, “Reed-Solomon Codes And Their Applications”, New Tork, IEEE Press, 1994.

[38] Joel Sylvester, “Reed Solomon Codes”, Elektrobit , January 2001.

[39] R.E Blahut, “Theory and Practice of Error Control Codes”, Reading, Mass: Addison Wesley, 1984.

[40] H.M. Shao, T.K. Truong, L.J. Deutsch, J. Yuen and LS. Reed, "A VLSI Design of a Pipeline Reed-Solomon Decoder," IEEE Trans. Comput., vol. C-34, no. 5, pp. 393-403, May 1985.

[41] Peter J. Ashenden, “The Designer’s Guide to VHDL”, Second Edition, Morgan Kaufmann Publishers, 2004.

[42] E. D. Mastrovito. “VLSI Architectures for Computations in Galois Fields”. Link¨oping University, Dept. Electr. Eng., Link¨oping, Sweden, 1991.

List of Publications

(1) Sandeep Kaur, Harpreet Singh, " Variable Sped Control Of AC Machines Using DSP", in the proceedings of National Conference On Electronic Circuits And Communication Systems (ECCS – 06) held on 9th February, 2006 At Thapar Institute of Engineering and Technology, Patiala (Punjab).

(2) Sandeep Kaur, Sanjay Sharma, "Color Plane Interpolation Using Inter- Channel Correlation", in the proceedings of National Conference On Electronic Circuits And Communication Systems (ECCS – 06) held on 9th February, 2006 At Thapar Institute of Engineering and Technology, Patiala (Punjab).

(3) Sandeep Kaur, Sanjay Sharma, "Implementation Of Reed – Solomon codes in VHDL ", in the proceedings of National Conference On Wireless and Embedded Systems (WNES – 06) to be held on 28th July, 2006 at Chitkara Institute of Engineering and Technology, Rajpura, Patiala (Punjab).