University of Massachusetts - Amherst
ScholarWorks@UMass Amherst

Masters Theses 1896 - February 2014 | Dissertations and Theses

January 2008

Energy Efficient Adaptive Reed-Solomon Decoding System

Jonathan D. Allen
University of Massachusetts - Amherst, jallen@ecs.umass.edu

Follow this and additional works at: http://scholarworks.umass.edu/theses

Allen, Jonathan D., "Energy Efficient Adaptive Reed-Solomon Decoding System" (2008). *Masters Theses 1896 - February 2014*. Paper 91.
http://scholarworks.umass.edu/theses/91

This Open Access is brought to you for free and open access by the Dissertations and Theses at ScholarWorks@UMass Amherst. It has been accepted for inclusion in Masters Theses 1896 - February 2014 by an authorized administrator of ScholarWorks@UMass Amherst. For more information, please contact scholarworks@library.umass.edu.

ENERGY EFFICIENT ADAPTIVE REED-SOLOMON DECODING SYSTEM

A Thesis Presented

by

JONATHAN D. ALLEN

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL AND COMPUTER ENGINEERING

February 2008

Department of Electrical and Computer Engineering

ENERGY EFFICIENT ADAPTIVE REED-SOLOMON DECODING SYSTEM

A Thesis Presented

by

JONATHAN D. ALLEN

Approved as to style and content by:

_____
Russell Tessier, Chair

_____
Dennis Goeckel, Member

_____
Marinos Vouvakis, Member

_____
C.V. Hollot, Department Head
Department of Electrical and Computer Engineering

CONTENTS

LIST OF TABLES vi

LIST OF FIGURES vii

CHAPTER

1. INTRODUCTION 1

2. BACKGROUND 12

2.1. ECC and RS introduction 12

2.1.1. Reed-Solomon Codes 14

2.1.2. Galois Fields 15

2.1.3. Reed-Solomon Encoding Algorithm 17

2.1.4. Reed-Solomon Decoding 23

2.2. Energy Consumption in FPGAs 31

2.3. Circuit Level Energy Reduction Methods 32

2.3.1. Pipelining 32

2.3.2. Clock Gating 34

2.3.3. Memory Access Reduction Techniques 37

2.4. Dynamic Reconfiguration 38

3. RELATED WORK 40

3.1. Previous RS Works 40

3.1.1. A Low-Power Reed-Solomon Decoder for STM-16 Optical Communications 40

3.1.2. Design of a Reed-Solomon Decoder using Partial Reconfiguration of XILINX FPGAs – A Case Study 41

3.1.3. Architecture for Decoding Adaptive Reed-Solomon Codes with Variable Block Length 42

3.1.4. A Reed-Solomon Decoder with Efficient Recursive Cell Architecture for DVD Applications 42
3.2. Previous FPGA Energy Reduction Works 43
3.2.1. The Impact of Pipelining on Energy per Operation in Field-Programmable Gate Arrays 43
3.2.2. Energy Efficient Signal Processing Using FPGAs 44
3.3. An Adaptive Errors-and-Erasures Reed-Solomon Decoder 45
3.4. Differences from Previous Work 48
4. IMPLEMENTATION 50
4.1. Channel Fading Model 50
4.1.1. Goals and Requirements 51
4.1.2. Simulation Flow 52
4.1.3. Model Details 54
4.1.4. Experiments 56
4.2. Hardware Optimizations 58
4.2.1. Recoding 59
4.2.2. Pipelining 63
4.2.3. Memory Optimizations 70
4.2.4. Clock Gating 72
5. CIRCUIT LEVEL OPTIMIZATIONS RESULTS AND ANALYSIS 76
5.1. Introduction 76
5.1.1. Previous Work 76
5.1.2. Quartus Synthesis Power Optimization 77
5.2. Recoding 79
5.3. Pipelining 82
5.3.1. Galois Field Multipliers 82
5.3.2. Small-Scale Pipelining 83
5.3.3. Global Pipelining 84

5.4. Memory Optimizations 85
5.5. Clock Gating 87
5.6. Summary 91
6. RECONFIGURATION RATE ANALYSIS AND RESULTS 92
6.1. New Reconfiguration Table 92
6.2. CER Analysis 93
6.3. Energy Efficiency Results 95
6.4. Decoding Rate Results 97
6.5. Summary 98
REFERENCES 99

LIST OF TABLES

2.1. Elements of GF(2^3) shown in three different representations.....16

2.2. Roots of Key Polynomials 30

3.1. Decoder Configurations, from [7].....47

4.1. Clock Gating Parameters 74

5.1. Results Generated from Designs Developed in [7] using Quartus II, v7.1.....77

5.2. Previous Work with Quartus Automated Power Optimization Results.....78

5.3. K239 Unit-by-unit Power Results.....78

5.4. Recoded Design Results, provides a new baseline for the following optimizations ..79

5.5. Functional Unit Energy Breakdown for Previous Work and Recoded.....80

5.6. Cycle Counts for Decoding a Codeword 81

5.7. Pipelined Galois Field Multiplier Results.....82

5.8. MEA Unit Comparison.....83

5.9. Small-scale Pipelining Results.....84

5.10. Global Pipelining Results 85

5.11. Clock Cycles per codeword before and after global pipelining.....85

5.12. Power Consumption results of Memory Buffering 20,400 bit memory 86

5.13. Power Consumption results of Memory Buffering 2040 bit memory 86

5.14. Results of Memory Optimizations.....87

5.15. Final Results after Clock Gating.....89

5.16. Final Results, in Energy (J) per MB of Message Data Reduction in relation to Recoded baseline values (Table 5.4.) 90

6.1. Configuration Table.....92

LIST OF FIGURES

1.1. Example of Glitching 8

2.1. Typical Communication Scheme 13

2.2. A General Reed-Solomon Encoder 20

2.3. Rayleigh Fading Channel 22

2.4. General Reed-Solomon Decoder Architecture 25

2.5. Clock Gating circuit 34

2.6. Memory Buffering 37

2.7. Activity of Memory, original vs. buffered 38

3.1. From [7], Architecture of an Adaptive errors-and-erasures Reed-Solomon Decoding System 46

4.1. SNR due to Shadowing vs. Shadowing and Fading 52

4.2. SNR variance during operation 57

4.3. System Block Diagram 58

4.4. a) New MEA Structure b) Previous MEA structure [7]. 61

4.5. Error Correction Unit 63

4.6. MEA unit with 3-stage (above), and 5-stage (below) pipelining 66

4.7. Time breakdown of the decoding process for an example of K=239 decoding 68

4.8. Timeline illustrating decoding of a codeword 68

4.9. Pipelining of Decoder Circuitry 70

4.10. Example of Memory Buffering Logic 71

4.11. Clock Gating Logic 74

4.12. Global Clocking Scheme 75

5.1. K239 Unit by Unit Energy Consumption Breakdowns 80

5.2. Global Clocking Scheme 88

5.3. Full Incremental Energy per Operation Results Breakdown 89

6.1. Graph of CER vs. Codewords per Reconfiguration.....94

6.2. Energy per Megabit vs. Codewords per Reconfiguration.....96

6.3. Reconfiguration Rate vs. Decode Rate 97

CHAPTER 1

INTRODUCTION

In recent years the continued rise of portable data-devices such as cell phones, PDAs, and laptops has driven enormous growth in the area of wireless communications. Whenever data is sent over a wireless channel, it is subject to degradation due to multipath fading and noise. Depending on the amount of degradation, the effect can be a loss or corruption of the original data during transfer. In order to alleviate this problem and ensure the reliable transfer of data, the typical solution has been the use of an error correction coding scheme. This work will detail the implementation of a *low-energy* error correction coding (ECC) scheme, based on the widely used Reed-Solomon algorithm, which will be implemented using a field programmable gate array (FPGA) device. FPGAs have been adopted for use in wireless communication and digital signal processing (DSP) applications due to their ease of use when compared to traditional DSP microcontrollers, high performance characteristics, and inherent configurability. Despite these benefits, much work remains to be done in order for these devices to truly be adopted for use in wireless devices, as current FPGAs are not naturally low-energy devices. This work will examine how to leverage the specialization and configurability of these devices in order to achieve low energy consumption characteristics while maintaining high levels of performance. This work is primarily aimed at systems which already include an FPGA for computation, as the performance of our algorithm implemented on an FPGA will not be able to outperform an ASIC implementation in terms of energy consumption. However, there are situations where incorporating a Reed-

Solomon decoding system into the functionality of an FPGA based system may be desirable, and the configurability of the FPGA allows for the processing unit to perform multiple operations simultaneously. In this case, it is much more desirable to implement the decoder in the pre-existing FPGA as opposed to incorporating a Reed-Solomon ASIC into the design.

The main contribution of this work is the development and analysis of an FPGA based Adaptive Reed-Solomon errors-and-erasures decoding system which is optimized to minimize energy consumption characteristics. This work is based on an earlier project [15][7], which has been modified to ensure low-energy operation through the use of several circuit-level energy optimization techniques and in addition to a new analysis of a scheduling approach for dynamic reconfiguration. The end result of the energy optimization is a reduction in system energy consumption of more than 70% compared to previous work.

The origin of ECC schemes dates back to the work of Shannon in 1948. His work [1][3][11] demonstrated that by properly encoding information before transferring over a lossy channel, the errors which are introduced in the channel can be reduced to any desired level without a severe decrease in transmission rate. Since then researchers have developed various error correction schemes. One of the most widely used of these schemes is Reed-Solomon coding [1][2]. Reed-Solomon coding has been used in systems ranging from CD players [4][23] (to correct errors introduced by dust in the optical drive) to NASA's wireless deep-space communications [24]. Reed-Solomon coding is what's known as a block coding scheme, under which fixed length blocks of data are encoded with a fixed amount of parity information. Other ECC schemes include Viterbi coding

[5], along with more recent schemes such as Turbocoding [6]. Traditionally, these algorithms have been implemented using DSP microcontrollers, which are based on microprocessors but specialized to allow for better signal processing performance. Recent work [25][26]has examined using FPGAs for these applications, as they provide similar performance characteristics while allowing for a much more customized design, with simplified and quicker development. FPGAs are customizable logic devices, which have seen more and more use in consumer and industrial electronics in recent years, are an alternative to traditional microproccessors and DSP devices. FPGAs are attractive devices to developers in need of specific solutions, as their configurability allows designers to tailor the device to provide the specific functionality necessary for a particular application. In addition, because the functionality is coded into the hardware of the device, mapping an application to an FPGA usually allows for an optimized, high-speed implementation, and allows to designer to customize the application at a much lower level than if he or she was using a general-purpose microprocessor.

Historically, the designers of ECC systems have focused on providing the best possible performance while maintaining the desired quality of service (QoS). Typically the limiting factor in these systems has been the data transfer rate, as all communication channels pose restrictions on their maximum bandwidth. However, with the continued growth of wireless systems, battery-powered devices of all types, power and energy consumption have become increasingly important design constraints.

The majority of mobile devices are no longer stand-alone devices. Increasingly, these devices are required to have extensive connectivity options, which means a need for wireless communications. If one examines a recently developed device such as Apple's

Iphone[27], we can see the need for not just one, but multiple wireless communication methods, from BlueTooth to 802.11 b/g wireless to cellular phone service and data transfer. These devices are constrained by the amount of energy which can be contained in their batteries, which introduces new design challenges for developers of these devices.

The work detailed in this document includes the development and analysis of a low-energy ECC decoding system, implemented on an Altera FPGA. It draws on previous works on ECC coding, and low power and energy FPGA design. The starting point for this is the work of Lilian Atieno [7], who developed an FPGA based adaptive errors-and-erasures Reed-Solomon decoding system. Her system was designed to adapt dynamically to changes in the noise level of the communication channel in order to provide maximum data-rates and reduced power consumption when compared to a static implementation of the decoding circuitry. If the channel is noisy, leading to increased error rates, a larger, more power-hungry decoder is swapped into the FPGA in order to keep the error rate below the required level. When there is less noise in the channel, a smaller, faster, and less power-hungry decoder is swapped in. The main metric of success for this previous work was decoding speed, with the secondary metric being power consumption.

Building on this previous work, this project adapts the previous design to minimize the amount of energy required to decode a particular amount of message data. Several modifications are made to both the structure of the design at the circuit level, and to the overall system functionality. An additional contribution of this work is the development of a more accurate channel noise and fading model, to get a better understanding of the real-world performance characteristics that could be expected.

The methods used in this work fall into two basic categories, application specific optimizations, and application independent, circuit level optimizations.

The application specific methods used during this work include:

1) Efficient Implementation of Application Primitives

The Reed-Solomon decoding application requires a multitude of specialized functional units to decode and correct errors in message data coming from a noisy channel. The fundamental units are Galois Field multipliers and Galois Field adders, which perform the most basic operations within the decoder. Larger units include the syndrome generation unit, the syndrome expansion unit, and the modified Euclidean algorithm block. Some of these units can be implemented several different ways in the FPGA hardware, so it is important that care be taken to ensure not only that the most efficient structure is used (in terms of energy consumption), but also that the desired structures are mapped as expected to the FPGA fabric. The development of a pipelined Galois field multiplier will be specifically documented in Chapter 4, along with the comparison of different structural implementations of the modified Euclidean algorithm block.

2) Adaptive System Design and Scheduling

The previous system makes use of a reconfiguration scheme designed to allow the functionality of the decoder to adapt to changing channel conditions in order to maintain the maximum possible decoding rate. For this work, the goal was to adapt to changing channel conditions to ensure the lowest energy consumption possible while maintaining a

fixed minimum codeword error rate (CER). Several changes to the overall system functionality were made. The previous system used several RS decoders in parallel, while maintaining the desired CER and decoding rate. This approach is inherently wasteful in terms of energy, since the result from only one decoder is ultimately used. Given that we are primarily concerned with energy, only single decoder versions are considered. This has a positive side-effect of reducing the total number of reconfigurations needed, as there are only seven different configurations as opposed to the previous system's twenty. The negative side effect is a slight reduction in decoding speed. This will be described in detail in Chapter 4.

The second change from the previous work is a more accurate channel model to evaluate the system's overall performance. The previous approach was designed to operate in a Rayleigh fading channel environment, with average SNRs varying from around 13 to 21dB. However, the model used to evaluate the system performance was not time-dependant, allowing for unrealistic changes in signal quality in short amounts of time. In order to better evaluate the performance of the system in a real-world situation, a time-dependant Rayleigh fading channel model was developed, which has several benefits over the previous model. Time dependency allows for evaluation of optimal reconfiguration rates answering the question, "How long should we wait before evaluating whether to reconfigure the system?" In addition, we evaluate the effects of differing reconfiguration rates on both the energy consumption characteristics and error-rate performance. The new model is used to answer several questions: first, what is the effect of the rate of reconfiguration on energy consumption, and second, what is the

effect of varying the rate of reconfiguration on the codeword error rate? This analysis is described in detail in Chapter 4, with final results shown in Chapter 5.

As mentioned above, application independent optimization methods were also used to reduce decoder energy consumption in the FPGA device. These methods are applicable to any design mapped to an FPGA, and have been shown in previous work to reduce energy consumption characteristics. Each of these methods was applied to the decoder circuitry at the highest level, and represent design choices which can be made by the designer of the application in order to reduce dynamic energy consumption. This is in contrast to algorithmic modifications which are automatically performed at lower levels by the CAD software used in FPGA development. It has been shown that higher level optimizations lead to the greatest possible benefit.

The methods used are detailed briefly below, and in full detail in Chapter 2.

1) Pipelining

Pipelining has been shown to reduce energy consumption in digital circuits, including FPGAs [8]. Pipelining allows for lower energy consumption by reducing the propagation of glitches through the circuitry. Glitches are defined as spurious transitions in the circuitry caused by timing mismatches. Figure 1.1 below illustrates how mismatched logic delays can cause spurious transitions.

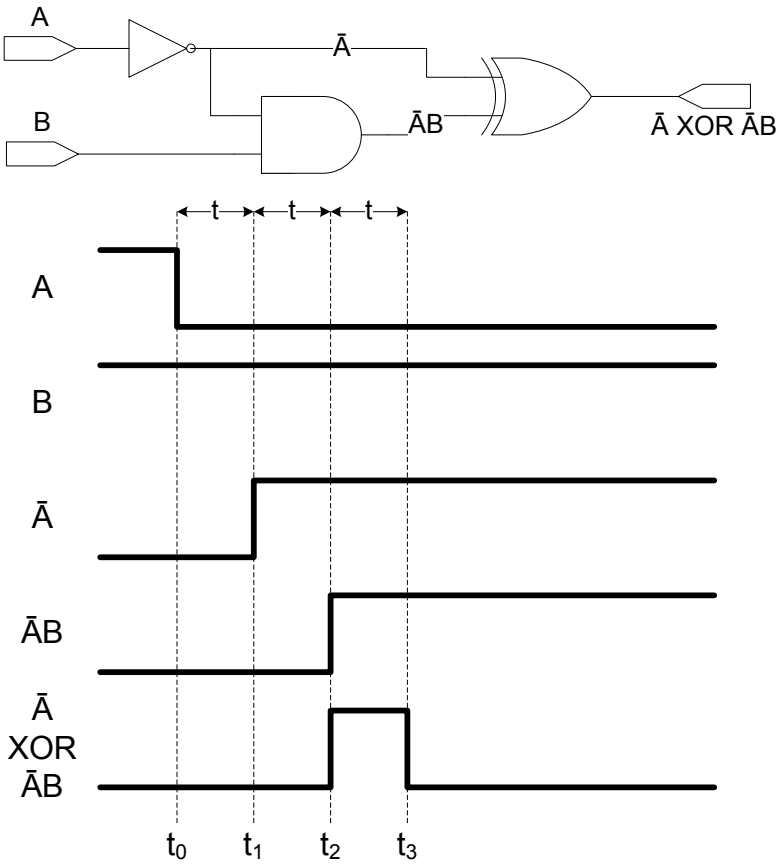


Figure 1.1. Example of Glitching

Assuming all of the gates have a delay of t, at t_0, input A goes from high to low. After one time unit, A inverse reflects this change. After another time unit, inverse A has caused a change in the output, even though the output of the AND gate hasn't propagated. After another time unit, the correct result is shown at the output. However, as the output went high briefly, this is an example of a glitch cause by mismatched logic depths, and in fact this glitch caused two separate spurious signal transitions.

As each signal transition in a digital device dissipates energy, minimizing the amount of unneeded transitions due to differing logic timing characteristics and reducing the distance these glitches are allowed to propagate is important in reducing energy consumption. If the output of the above circuit was fed onto a long communication line in the FPGA, the amount of energy dissipated by the glitch could be very large.

Pipelining is accomplished by inserting registers throughout the design, which effectively cuts off the propagation of glitches beyond the register. FPGAs have a configurable internal communication network made up of many long, high-capacitance wires, which dissipate significant amounts of energy. Considering their energy dissipation characteristics, reducing the number of transitions on these lines is paramount. In addition, pipelining allows for the hardware resources to be better utilized by allowing for greater levels of parallelism to be built into the application. This improves energy characteristics by preventing logic from being idle and thus dissipating energy without a purpose.

2) Clock Gating

Clock gating of digital circuits is another technique which has been shown to reduce energy consumption both in ASICs [9] and FPGAs [10]. The essential idea is that some parts of a design may not be needed for part of its operation, i.e. this portion of the circuit on this clock cycle generates an output value which is not needed by another portion of the circuit. If this is the case, these design features can have their clock suppressed (gated) so that the clock is not propagated to them when their results are unneeded, ensuring that they do not dissipate energy. Given that the RS system contains many individual functional blocks which are not needed at all times, clock gating these units so that they are only active when needed provides the opportunity to save large amounts of energy.

3) Efficient Structuring of Embedded Memories

As memory units dissipate energy on every read or write operation, reducing the overall number of accesses reduces the overall energy consumption of the design. A method was developed to combine data into large blocks for each read and write operation to reduce the number of required memory accesses. This technique allows for a more efficient use of clock gating for internal FPGA memory blocks and allows the memory to be inactive for a larger percentage of the time. While this can increase the energy required to perform an individual read or write, by greatly reducing the number of necessary reads and writes, energy consumption can be reduced. In many ways, these units work as small caches, preloading the data which will be needed for the next several clock cycles in order to allow the memory to maintain a lower activity rate.

Overall, applying these circuit level techniques resulted in a net reduction in the energy required to decode a megabit of data by 70%. The specific areas where each optimization was performed are detailed in Chapter 4, while the numerical benefits are shown in Chapter 5.

The rest of this document is structured as follows. Chapter 2 provides background information on the Reed-Solomon algorithm, a discussion of the sources of energy consumption in FPGA circuitry, and a detailed look at the energy reduction techniques which were used in this work.

Chapter 3 details related works in FPGA energy reduction and Reed-Solomon decoder implementations.

Chapter 4 details where the aforementioned techniques were used in the development of the decoding system, while also describing in detail the methodology that was used for these techniques, along with a detailed description of the new channel model, and how it was used to evaluate the performance of the system.

Chapter 5 provides numerical results for each individual optimization technique, while also providing overall system performance data.

Chapter 6 provides numerical results for the reconfiguration scheduling and analysis part of this work.

CHAPTER 2

BACKGROUND

2.1) ECC and RS introduction

All methods of digital communication are subject to some sort of noise or interference, whether the medium of communication is a physical link or a wireless one. In physical systems, noise can be introduced by the electromagnetic fields generated by the surrounding circuitry and components, by errors in data storage, or even physical phenomena such as a particle of dust getting in the way of a laser beam reading data from an optical storage device. In wireless channels, errors can be created by interference from other wireless signals, interference caused by the signal passing through a building, or fading caused by differing propagational paths of the wireless signal. The main challenge in digital communications has become how to deal with these unavoidable errors in an efficient way, so as to prevent data loss without causing undue overhead.

The work of Shannon [11] demonstrated that even though communication channels are subject to noise and errors, if some amount of redundancy is encoded into the signal, errors can be accounted for and corrected at the receiving end. This is the fundamental principle of error-correction coding schemes, and has led to the development of various encoding schemes, including Reed-Solomon coding. A typical communication scheme that meets this criterion can be modeled as seen in Figure 2.1.

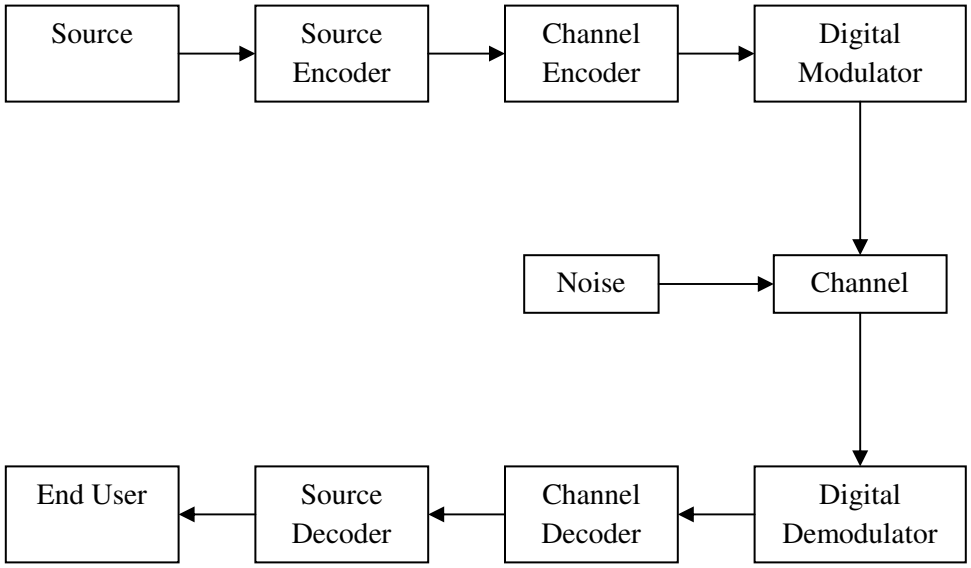


Figure 2.1. Typical Communication Scheme

The source represents the origin of the data to be sent, and can be a physical storage device such as a CD, DVD, or Magneto-Optical recording device with the data already in digital form, or an analog signal such as a voice or music sample. The source is first encoded into digital form if it wasn't already in such a form. The output of the source encoder must be a sequence of binary digits representing the data. How this encoding is performed is determined by the needs of the system, and is irrelevant to the communication methodology.

The data stream is then sent through the channel encoder, where the ECC encoding is performed. This unit takes the incoming data and adds redundancy via whichever ECC scheme is in use. In this work, a Reed-Solomon encoding device is used.

The encoded data is then sent to the digital modulator, which takes the digital signal and creates an analog waveform to be transmitted over the communication channel.

As the analog signal is transmitted over the channel, it is subject to noise, which distorts the original signal so that what is received at the other end is not identical to the signal which was sent. Noise is the source of the errors, which need to be corrected.

When the signal is received at the end of the channel, it is demodulated back into digital form by the demodulator. This process is the exact inverse of the modulation process, and the end result is again a stream of binary data. However, as mentioned above, the signal has changed due to noise during transmission, and so in most cases, the binary sequence output from the demodulator is not identical to the one which was originally presented to the modulator.

This sequence is then fed into the channel decoder, which attempts to decode the signal in a way that recovers all of the original data, correcting any errors which were introduced during transmission. This is made possible by the redundancy which was added to the signal during encoding. In this work, the decoding system is the main application of interest, and represents a Reed-Solomon decoding system.

After the errors have been corrected to the decoder's best ability, the data is transformed into the required format, for example, in a phone conversation, the binary stream is converted back into an analog signal to be output to the receiver's speaker.

2.1.1) Reed-Solomon Codes

Reed-Solomon codes were first introduced by Irving Reeds and Gus Solomon in 1960, in a paper entitled "Polynomial codes over certain finite fields" [2]. Since their

inception, RS codes have been one of the most widely used ECC schemes, mainly because the coding scheme allows for efficient correction of both burst and random errors. Reed-Solomon coding is known as a non-linear, block based coding scheme. RS is a block scheme because it encodes blocks of a specific amount of data individually, as opposed to operating on the entire data stream as a whole. RS codes are based on finite field arithmetic, known as Galois fields. These fields are mathematical constructs in which any operation on one data element results in another element in a constrained field. The general operation can be described as follows; a predetermined sized block of data (k bytes) is encoded so that the result is a data block of size n, where n>k. This size n block contains the k original data bytes, along with n-k parity bytes, representing the redundancy in the signal, for transmission over the noisy channel. Within the block, the RS algorithm works on multiple bits of data at a time, typically a byte. Each byte is a *symbol*, and the nature of the RS algorithm allows for the correction of whole symbols, as opposed to correcting individual bits. This means that the RS decoder can correct a symbol with 8 bit errors as well as a symbol with 1 bit in error. This is the particular characteristic which allows RS codes to be effective at correcting burst errors in addition to random errors.

2.1.2) Galois Fields

As mentioned above, the RS coding scheme uses abstract mathematical constructs known as Galois Fields. Each field contains a finite number of elements, and operations on elements in the field can only produce a result within the same field. The benefits of this kind of arithmetic include not having to deal with overflows and carries. Galois fields

are defined as $GF(X^Y)$, where X^Y equals the total number of elements in the field. For RS codes, X must be a prime positive integer, and Y must be an integer greater than or equal to 3. Y also determines the number of bits operated on simultaneously, so in the case of our RS system, Y will be equal to 8. X will be defined as 2 as this is a common value, and lends itself well to digital implementations.

As an example, Table 2.1 shows the elements of the Galois field $GF(2^3)$. Elements of the Galois field are generated from the 'primitive polynomial' p(x), in this case, $p(x)=1 + x + x^3$. When doing calculations in digital circuitry, the elements of each table entry are typically represented by bit values, instead of polynomial or power representations.

Power Representation	Polynomial Representation	3-Tuple Representation
$-\infty$	0	0 0 0
0	1	0 0 1
α	α	0 1 0
α^2	α^2	1 0 0
α^3	$\alpha + 1$	0 1 1
α^4	$\alpha^2 + \alpha$	1 1 0
α^5	$\alpha^2 + \alpha + 1$	1 1 1
α^6	$\alpha^2 + 1$	1 0 1

Table 2.1. Elements of $GF(2^3)$ shown in three different representations

Arithmetic operations performed within a Galois field are performed differently than when using typical arithmetic. The two operations used in the RS system are GF adds and GF multiplications. GF addition is performed in binary systems by XORing the corresponding bits of the codeword, which represent the coefficients of the polynomial. For example:

$$\alpha^6 + \alpha^4 = [101] + [110] = [011] = \alpha^3$$

GF multiplication is performed by adding the indices of the polynomial, for example:

$$\alpha^3 * \alpha^2 = \alpha^{2+3} = \alpha^5$$

In binary form, this operation is a modulo 2 sum of partial products, and which requires specialized multiplier circuitry. The circuitry will be described in detail in Chapter 4.

2.1.3) Reed-Solomon Encoding Algorithm

This section will provide an overview of the Reed-Solomon encoding and decoding algorithms, focusing on the mathematical description, while Chapter 3 will provide a look at how the decoding algorithm was previously implemented in hardware.

As mentioned above, Reed-Solomon codes operate on GF of the order $q=p^m$, where m is a positive integer greater than or equal to 3. Typically, the value of p is 2, and a typical value of q is 256. For this example, the assumed values will be p=2 and q=8, because the math becomes very complex as q scales upward. Our experimental implementation utilizes a q value of 256. Each GF is generated from a primitive polynomial of $p(x)=1 + x + x^3$.

The three columns in Table 2.1 illustrate different ways of representing the same data. If this particular GF was to be implemented in circuitry, 3-tuple representation

would be used, given its binary representation. All of the operations in the RS algorithm operate within this constrained field, meaning that any operation on data within the field will result in another entry within the field.

The encoding process is accomplished by taking in a k-bit block of data, and generating n-k parity bits to append to the original data for transfer. RS encoding makes use of a generator polynomial. The encoder generates the parity symbols by dividing the data by the generator polynomial, with the remainder being the parity bits.

An example of RS encoding follows. For this example, the code used is RS(7,3) operating on the GF(2^3), the elements of which are shown in Table 2.1. This implies that the encoder operates on 3-bit symbols, 3 of which will be used to generate 4 3-bit parity values, for a total message length of 7 3-bit symbols. A summary of the parameters is shown below.

$n=7,\ k=3$

$t=(7-3)/2=2$

N represents the total codeword length in symbols, while K represents the number of data symbols in each codeword. T represents the error correcting capability of the coding scheme. In this case, an errors-only RS(7,3) decoder can correct 2 erroneous symbols in the codeword, while and errors-and-erasures version of the same decoder can correct 2t=4 erroneous symbols. The difference between an errors-only and an errors and erasures decoder will be discussed in Chapter 3.

Suppose that the message below is to be encoded:

$$u_binary=[011,011,010]$$

As one can see from examining Table 2.1., this data can be represented in both polynomial form, as:

$$u(x) = \alpha^3 x^2 + \alpha^3 x^1 + \alpha^1 x$$

and in power form, as:

$$u = \alpha^3 \alpha^3 \alpha^1$$

The value of x in the polynomial form represents the position of the symbol in the block.

To determine the parity bits of the signal, a *generator polynomial* is used. It's general form is:

$$g(x) = (x + \alpha^0)(x + \alpha^1) + \ldots\ldots + (x + \alpha^{2t-1})$$

Given that for this example, t=2, the generator polynomial used is:

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

This can be expanded to $g(x) = \alpha^6 + \alpha^5 x + \alpha^5 x^2 + \alpha^2 x^3 + x^4$, so the coefficients of g are $g_0 = \alpha^6$, $g_1 = \alpha^5$, $g_2 = \alpha^5$, $g_3 = \alpha^2$. A simplified architecture for the encoder is illustrated below.

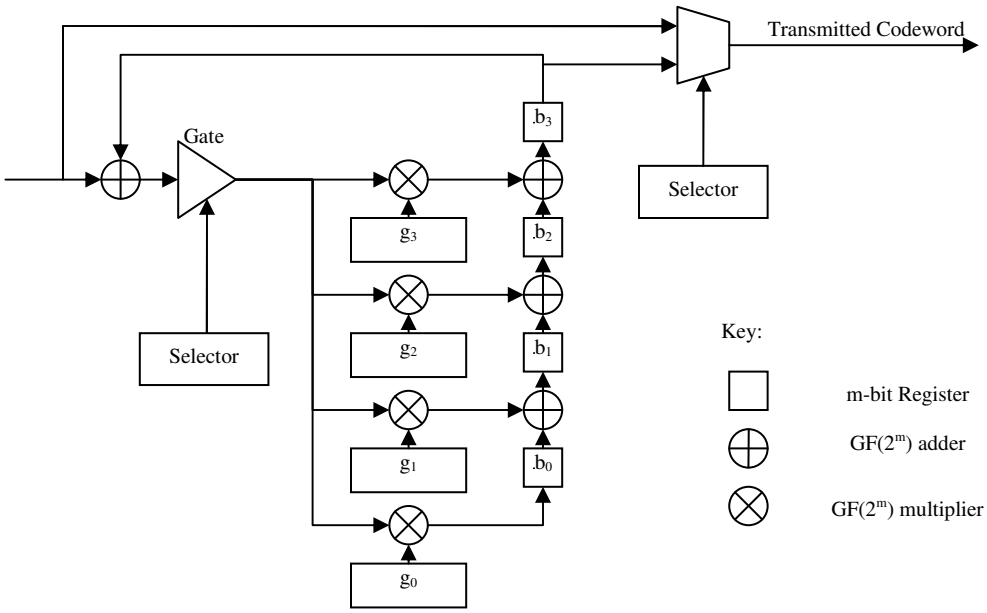


Figure 2.2. A general Reed-Solomon Encoder

The encoder uses the roots of g(x), along with a selection signal. This signal ensures that for the first m clock cycles, the input data is propagated to the output, followed by the propagation of the calculated parity symbols. An example using GF(2^3) is detailed below.

- Clock Cycle 1: The first message symbol 011 (or α^3) is sent into the encoder. The symbol is multiplied by each of the generator coefficients, and added to the previous data in registers b_0,b_1,b_2,b_3, which in this case, since this is the initial cycle, are all equal to 000. The resulting register values are:
    - $b_0 = \alpha^3 * \alpha^6 = \alpha^9$, which simplifies to α^2

- $b_1 = \alpha^3 * \alpha^5 + 000 = \alpha^8$, which simplifies to α^1
- $b_2 = \alpha^3 * \alpha^5 + 000 = \alpha^1$
- $b_3 = \alpha^3 * \alpha^2 + 000 = \alpha^5$

- Clock Cycle 2: The second message symbol, 011 (α^3) is sent into the decoder. It is XORed with the value of b_3, which is 111 (α^5) resulting in 101 (α^2). This value is multiplied by the generator coefficients, resulting in register values of:
  - $b_0 = \alpha^2 * \alpha^6 = \alpha^9 = \alpha^1$
  - $b_1 = \alpha^2 * \alpha^5 + \alpha^2 = \alpha^6$
  - $b_2 = \alpha^2 * \alpha^5 + \alpha^1 = \alpha^3$
  - $b_3 = \alpha^2 * \alpha^2 + \alpha^1 = \alpha^2$
- Clock Cycle 3: The third message symbol, 010 (α^1) is fed into the decoder. It is XORed with the value of b_3, α^2, resulting in α^4. The end results in the registers are:
  - $b_0 = \alpha^4 * \alpha^6 = \alpha^3$
  - $b_1 = \alpha^4 * \alpha^5 + \alpha^1 = \alpha^4$
  - $b_2 = \alpha^4 * \alpha^5 + \alpha^6 = \alpha^0$
  - $b_3 = \alpha^4 * \alpha^2 + \alpha^3 = \alpha^4$
- Clock Cycle 4-7: As the counter is now equal to 4, the data in registers b_0, b_1, and b_2 contain the parity data to be appended to the signal. The gate is disconnected, and the data is allowed to propagate out of the circuit.

The final message sent to the channel is [011,011,010,110,001,110,011]. The first 3 tuples are the original data, and the trailing four are the parity symbols. This binary string is modulated into an analog form. A typical modulation scheme is the Binary Phase Shift Key (BPSK) modulator, which transforms the data into a waveform, with 1s becoming -1s, and 0s becoming +1s. This signal is transmitted over the channel, and is subject to noise in the form of Rayleigh channel fading, and additive Gaussian white noise (AGWN).

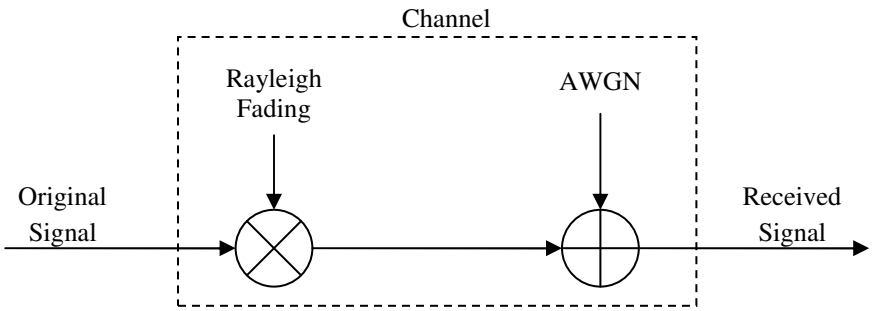


Figure 2.3. Rayleigh Fading Channel

The above diagram illustrates a typical Rayleigh fading channel. Rayleigh fading occurs because of the nature of a wireless transmitting environment. Signals in an environment such as this scatter off of physical objects such as walls and the result is that there are multiple paths from the transmitter to the receiver, resulting in different amounts of signal power coming to the antenna from different directions. In addition, electromagnetic interference also affects the signal while in transit. The resulting effect can be described mathematically as:

$$y_r = y_s * f + n$$

where y_s represents the signal as sent, y_r represents the signal as received, f represents the Rayleigh fading gain of the channel, and n represents the AWGN during transmission. Typically these parameters change over the course of the transmission of the codeword, so this can be better modeled on a bit by bit basis as:

$$y_r^i = y_s^i * f^i + n^i$$

where i stands for the ith bit of the transmitted sequence.

2.1.4) Reed-Solomon Decoding

When the signal from the encoder is modulated and passed through the channel, it is subject to both Rayleigh fading and AGWN, and thus the signal is not the clean -1 and +1 signs when it is received. The demodulator takes in this analog signal, and outputs floating point estimations of each bit's value. There are two ways to perform the decoding of this modified data, hard-decision decoding, and soft-decision decoding. Hard decision decoding yields an error-only RS decoder, and functions by determining that any signal received which is below 0 becomes a -1, and above zero becomes a +1. While this is usually correct, in the case where a large amount of noise was injected into the signal, these hard decisions may be incorrect. Consider for example a symbol which is received and demodulated into the values {.0675, -.0238, -.8905}. Using hard decision demodulation, this would become {+1, -1, -1}. However, the second bit is so close to 0 that it could conceivably have been either a +1 or -1 originally. When using hard-decision demodulation, the decoder has no way of knowing that this bit is unreliable, information which could aid in the decoding process.

Soft decision demodulation uses an *erasure generator* to signal the decoder when particular symbols are unreliable. The decoder still receives streams of the "most likely" symbols, but also receives a stream of flags indicating when a particular symbol is unreliable. The erasure generator takes in a symbol at a time, and generates two possible values, the most likely symbol (MLS), and 2nd MLS. The second MLS is determined by negating the bit with the lowest absolute amplitude, as this is the least reliable bit. It then calculates the difference between the two different symbols, the MLS and the 2nd MLS, and compares this to a pre-set threshold value. The actual function to determine whether to assert the erasure flag is detailed below.

The receiver receives y, representing the amplitudes of the received data from the channel. It then calculates the most likely symbol, or MLS, based on the fact that y was received. The possible symbols are denoted as s_0, s_1, $\ldots s_{n-1}$. This function is denoted as:

$$max_v f(y^j \backslash s_v)$$

The erasure flag is asserted for a particular symbol if and only if:

$$\frac{max_v f(y^j \backslash s_v)}{\sum_{i=0}^{n-1} f(y \backslash s_i)} = 1 - T$$

where s_v represents the MLS, and the bottom term represents the total conditional probability of s_v given that y^j was received.

The principal benefit of using a soft decision, errors-and-erasures version of an RS decoder is that the amount of errors that can be corrected per codeword is increased. The amount of errors that can be corrected by a hard decision decoder is t, while a soft-decision decoder can correct 2t erroneous symbols per codeword.

Figure 2.4 shows the general structure of an errors-and-erasures RS decoder. This decoder receives the stream of estimated data and a stream of erasure flags from the erasure generator, and attempts to correct any errors.

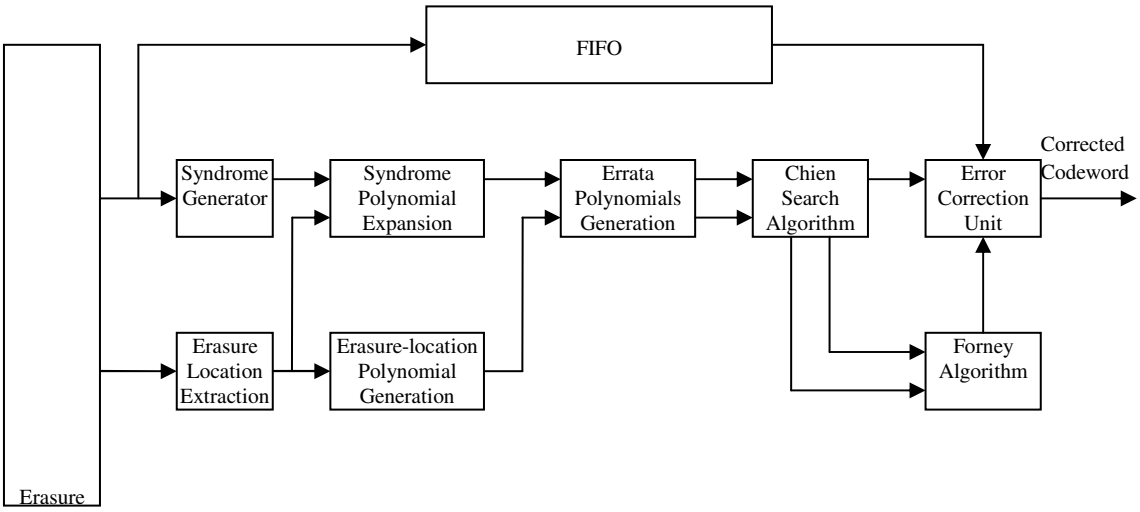


Figure 2.4. General Reed-Solomon Decoder Architecture

This next section will contain an example of RS soft-decision decoding, following from the encoding example. Each block's operation will be discussed along with the example.

1) Syndrome Generation Block

The function of the syndrome generation block is to divide the received codeword by the generator polynomial. As mentioned earlier, any valid codeword will be exactly divisible by the generator polynomial. If there is a remainder, one can assume that there are errors in the codeword. The typical method of performing this division is to substitute

all of the 2t roots of the generator poly into the received polynomial, generating 2t syndrome coefficients. This is known as the syndrome polynomial. If all of the coefficients are zero, then there are no errors in the codeword, and error correction can be bypassed.

Given the example from the encoding section, the received codeword should be

$u(x)$ = [011,011,010,110,001,110,011].

Let us assume that instead, the received sequence is:

$u(x)$ = [011,**010**,**100**, 110,001,110,011].

The polynomial representation of this sequence is:

$$u(x) = \alpha^3x^6 + \boldsymbol{\alpha^1x^5} + \boldsymbol{\alpha^2x^4} + \alpha^4x^3 + \alpha^0x^2 + \alpha^4x + \alpha^3$$

Two errors were introduced during the transmission of the signal over the channel, one in the second symbol, and one in the third. The syndrome generation unit substitutes all of the roots of the generator polynomial into the above equation, resulting in:

$$s(x) = \alpha^4x^3 + \alpha^2x^2 + \alpha^6x + \alpha^5$$

2) Erasure Location Extraction

This block receives the stream of erasure flags from the erasure generator, and expands them into a polynomial for use in calculating the locations of errors in the codeword. As erasure generation is not 100% accurate, let us assume that only the second symbol was flagged as being unreliable, and the third symbol, which also has an error, was missed.

When the second symbol arrives at the syndrome generation block, an erasure flag arrives at the erasure location extraction block. This block then performs the following calculation:

$$t = \alpha^{n-1}$$

$$t = \alpha^{7-2}$$

$$t = \alpha^{5}$$

The resulting polynomial is the sum of all of the results for every erasure flag plus 1. Since in this case there is only a single erasure flag, the resultant polynomial is:

$$t(x) = 1 + \alpha^5 x = \alpha^0 + \alpha^5 x$$

3) Syndrome Polynomial Expansion Block

This block receives the syndrome polynomial s(x) and the erasure location polynomial t(x) from the preceding blocks. Its job is to multiply these two polynomials together to generate the modified syndrome polynomial, T(x). Continuing with the example, the received vectors were:

$$s(x) = \alpha^4 x^3 + \alpha^2 x^2 + \alpha^6 x + \alpha^5$$

$$t(x) = \alpha^0 + \alpha^5 x$$

T(x) is defined as:

$$T(x) = t(x)s(x) mod\ x^{2t}$$

$$T(x) = (\alpha^0 + \alpha^5 x)(\alpha^4 x^3 + \alpha^2 x^2 + \alpha^6 x + \alpha^5)\ mod\ x^4$$

$$T(x) = \alpha^5 + \alpha^4 x + \alpha^1 x^2 + \alpha^5 x^3$$

4) Erasure-Location Polynomial Generator Block

This block calculates the erasure location polynomial in parallel with the above syndrome expansion block. This block expands the erasure location polynomial. Given that the current example has only one erasure, the poly remains unchanged, but if for example both of the erroneous symbols in the example had been flagged, then t(x) would have been:

$$t(x) = (1 + \alpha^6 x)(1 + \alpha^5 x)$$

And in this case, this would need to be expanded by multiplying out the factors. In this case, the polynomial is unchanged and is forwarded as is,

$$D(x) = 1 + \alpha^5 x = \alpha^0 + \alpha^5 x$$

5) Errata Polynomials Generation Block

The job of this block is two create two key polynomials which will help to identify the location and magnitude of the errors in the codeword. The two polynomials are the errata-locator-polynomial, $\Psi(x)$, and the errata-magnitude-polynomial $\Omega(x)$. The inputs to this block are the modified syndrome polynomial, T(x), and the erasure-location-polynomial, D(x). There are two methods of computing these polynomials, the Berlekamp-Massey algorithm[28], or the Modified-Euclidean algorithm (MEA)[29]. In this work, the MEA algorithm will be used.

The MEA algorithm is a recursive algorithm which operates on 4 polynomials, R, Q, L, and U. They are initialized as follows:

- R is initialized to x^{2t}
- L is initialized to 0
- Q is initialized with T(x)

- U is initialized with D(x)

The equations used to update the polynomial are:

$$R_i(x) = [\sigma_{i-1}b_{i-1}R_{i-1}(x) + \sigma`_{i-1}a_{i-1}Q_{i-1}(x)] - x^{|l_i-1|}[\sigma_{i-1}a_{i-1}Q_{i-1}(x) + \sigma`_{i-1}b_{i-1}R_{i-1}]$$

$$L_i(x) = [\sigma_{i-1}b_{i-1}L_{i-1}(x) + \sigma`_{i-1}a_{i-1}U_{i-1}(x)] - x^{|l_i-1|}[\sigma_{i-1}a_{i-1}U_{i-1}(x) + \sigma`_{i-1}b_{i-1}L_{i-1}]$$

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

$$U_i(x) = \sigma_{i-1}U_{i-1}(x) + \sigma`_{i-1}L_{i-1}(x)$$

where a_{i-1} and b_{i-1} are the leading coefficients of $R_{i-1}(x)$ and $Q_{i-1}(x)$, $l_{i-1} = deg(R_{i-1}(x)) - deg(Q_{i-1}(x))$, where $deg(y)$ signifies the degree of y, and $\sigma_{i-1} = 1$ if $l_{i-1} \geq 0$. $\sigma`_{i-1}$ is the opposite of $\sigma_{i-1.}$

The number of iterations needed depends on the number of errors which were not flagged by the erasure generator. It can be seen that the more accurate the erasure generator, the better the performance of this block. The computation stops when the degree of $R_i(x)$ is less than the degree of $L_i(x)$. When this occurs, the value of $L_i(x)$ is output as the error-locator polynomial, $\Psi(x)$, and the value of $R_i(x)$ is output as the error-magnitude polynomial, $\Omega(x)$. Continuing with the example, the initial values are:

$$R_0(x) = x^4$$

$$Q_0(x) = T(x) = \alpha^5 + \alpha^4x + \alpha^1x^2 + \alpha^5x^3$$

$$L_0(x) = 0$$

$$U_0(x) = D(x) = \alpha^0 + \alpha^5x$$

The end result of the MEA calculation results in:

$$\Psi(x) = \alpha^3x^2 + \alpha^1x + \alpha^1$$

$$\Omega(x) = \alpha^2x + \alpha^6$$

6) Chien Search Block

The job of the Chien-Search block is to take the error-location ($\Psi(x)$) and error magnitude ($\Omega(x)$) polynomials, and evaluate them across all of the possible values in the GF(2^m). In addition, the Chien-Search block creates and evaluates the derivative of $\Psi(x)$, $\Psi`(x)$, which is the odd terms of $\Psi(x)$.

When the result of an evaluation of $\Psi(x)$ equals 0, it indicates that there is an error in the (n-i)th symbol in the codeword. These three sets of evaluations are passed on to the Forney Algorithm and Error-Correction Block. Continuing the example, the result of these calculations yields:

	$\Psi(x)$	$\Psi`(x)$	$\Omega(x)$
α^0	α^3	α^1	α^0
α^1	α^0	α^2	α^4
α^2	0	α^3	α^3
α^3	0	α^4	α^1
α^4	α^3	α^5	0
α^5	α^1	α^6	α^2
α^6	α^0	α^0	α^5

Table 2.2. Roots of Key Polynomials

These results indicate an error at location 7-2=5 and 7-3=4 in the received codeword, which is correct.

7) Forney Algorithm and Error-Correction Block

This block is responsible for evaluating the magnitude of each error indicated by the Chien-Search block and performing the correction to the original received codeword.

It receives the evaluations of Ψ(x), Ψ`(x), and Ω(x), along with the original codeword from the FIFO. The magnitude of the error in location l is determined by the equation:

$$\hat{e}(\alpha^l) = \Omega(\alpha^l)/\Psi`(\alpha^l)$$

A polynomial ê(x) is formed by combining the error locations (as powers of x) with the error magnitudes (as powers of α). The codeword is corrected by combining this polynomial with the original codeword polynomial, u(x), as follows:

$$\hat{c}(x) = \hat{e}(x) + u(x)$$

The result, ĉ(x), is the corrected codeword. Following the example, errors are in location 5 and 4, corresponding to α^2 and α^3. The error magnitudes are calculated as:

$$\hat{e}(\alpha^2) = \Omega(\alpha^2)/\Psi`(\alpha^2) = \alpha^3/\alpha^3 = \alpha^0$$

$$\hat{e}(\alpha^3) = \Omega(\alpha^3)/\Psi`(\alpha^3) = \alpha^1/\alpha^4 = \alpha^4$$

And thus the error vector $\hat{e}(x)$ is:

$$\hat{e}(x) = \alpha^0 x^5 + \alpha^4 x^4$$

This vector is combined with $u(x)$, the original received codeword, as follows:

$$\hat{c}(x) = \hat{e}(x) + r(x)$$

$$\hat{c}(x) = (\alpha^0 x^5 + \alpha^4 x^4) + (\alpha^3 x^6 + \boldsymbol{\alpha^1 x^5} + \boldsymbol{\alpha^2 x^4} + \alpha^4 x^3 + \alpha^0 x^2 + \alpha^4 x + \alpha^3)$$

$$\hat{c}(x) = \alpha^3 x^6 + \alpha^3 x^5 + \alpha^1 x^4 + \alpha^4 x^3 + \alpha^0 x^2 + \alpha^4 x + \alpha^3$$

$$\hat{c} = [011,011,010,110,001,110,011]$$

This creates the original codeword, as all errors have been corrected.

2.2) Energy Consumption in FPGAs

There are two distinct types of energy consumption in FPGAs, static and dynamic. Static energy is consumed by all parts of the FPGA, whether active or not, as

long as the device is on. Static energy is consumed at a roughly constant rate. The main source of static energy consumption is the SRAM used to store the configuration of the device. Static energy consumption can be viewed as the cost of having the FPGA device "on", and is roughly design independent, meaning that the opportunities for reducing static energy consumption available to the designer are limited.

The second type of energy consumption in FPGAs is dynamic energy consumption. This is caused by signal transitions in the circuitry as the application performs work. Dynamic energy consumption is the cost of the device performing calculations. The amount of energy consumed is governed by the capacitance on a particular signaling line, and each transition will generally dissipate the same amount of energy. There are two types of signal transitions, transitions necessary for calculations, and spurious transitions caused by path-delay differences in the logic circuits, which are commonly referred to as glitches.

In general, reducing the length (and thus the capacitance) of a signal line, or reducing the number of transitions across a line, whether required or spurious, will reduce dynamic energy consumption. This work focuses on high level techniques which reduce the total number of signal transitions.

2.3) Circuit Level Energy Reduction Methods

2.3.1) Pipelining

The impact of pipelining has previously been examined for a variety of different devices, and has been found to be effective at reducing energy consumption in digital

circuits[30]. Most recently, Wilton et al did an analysis of the effects of pipelining on energy consumption in FPGA circuits [8]. The conclusion of this study was that pipelining is an effective method of reducing dynamic energy consumption in FPGAs. Pipelining is especially applicable to FPGAs because a) the registers used for pipelining are embedded in the FPGA fabric in every logic element, so the cost of using them is minimal, and b) because the communication lines on FPGAs tend to be longer and have higher capacitance on average than those in a custom ASIC, signal transitions require a significant amount of energy. It is necessary to minimize spurious transitions on these lines, one of the beneficial effects of pipelining.

One effect of pipelining is to split the logic into discrete sections, separated by registers. By splitting the logic up, it is possible to selectively de-activate sections of logic by using clock gating, another method of energy reduction. The granularity of the pipelining determines the size of the logical register-to-register sections which can be clock gated, and thus the two methods have significant interaction.

The methodology used in this work to reduce energy via pipelining is as follows. Pipelining was performed by hand, to evaluate differing amounts of pipelining while maintaining identical logical functionality, differing only in latency. The initial designs are examined for areas which could potentially be pipelined. Examples of such areas include areas of large logic depths between registers. As we are attempting to minimize the amount of logic through which a glitch can propagate, it is desirable to separate functional units by inserting pipeline registers between them.

Although pipeline registers can reduce the propagation of glitches across logic and interconnect, and thus reduce energy consumption, the additional registers also

dissipate some amount of energy. Thus it is not always the case that additional pipelining will be effective in reducing energy, as a balance must be found between the energy saved from reduced glitching, and the energy consumed by additional registers. It was observed in preliminary work that there is a degree of pipelining which provides optimal energy per operation results. The goal is to find the optimal amount of pipelining to achieve minimal energy-per-operation performance.

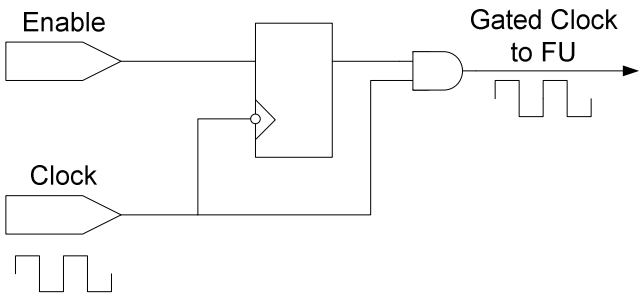


Figure 2.5. Clock Gating circuit

2.3.2) Clock Gating

Clock gating is a technique which has been shown to be effective for reducing power and energy consumption for all types of digital circuits [9] [10]. As illustrated in Figure 2.5, in its simplest form, clock gating is achieved by ANDing the clock signal to a particular element with an enable signal, so that when the enable is low, the combined signal is always forced low, effectively halting any clocked operations in any logic controlled by that clock signal. This is done to reduce unnecessary transitions in the logic, thus reducing dynamic energy consumption. Typically, we gate computational elements

when the results of their computation is unneeded, meaning the results have no impact on a current or future output signal. Clock gating is particularly applicable to FPGA circuits because the ability to enable and disable the clock is typically built into the logic elements in the FPGA, so the cost of using them is minimal. Preventing a transition on an interconnect line is particularly beneficial for FPGAs since interconnect capacitance is quite high compared to ASICs.

Despite these advantages, clock gating has significant tradeoffs. While energy may be saved by preventing unnecessary computation, additional energy will be consumed by the gating logic, and the designer must make sure that the energy saved exceeds the additional energy of the gating logic. In addition, the generation of clock gating control signals can sometimes introduce additional levels of logic, reducing the maximum operational frequency of the design.

Our methodology for applying clock gating is as follows. Given a design which has previously been pipelined, we have the option of gating a register, effectively cutting off any computation driven by the output of that register. We begin by examining the application for areas whose computation will not always be needed. For example, in a pipelined ALU, there are separate functional units for every operation, AND, OR, ADD/SUB, etc. Each particular unit is only needed when a particular operation is needed. Thus, it makes sense to gate portions of the functional unit based on the value of the required operation. In addition, given a pipelined architecture, it is often possible to know which units will be needed a cycle or more ahead of time. If this is the case, it is possible to perform cycle-ahead gating.

Once areas have been identified for possible clock gating, the designer must weigh the energy consumption of the logic needed to generate the enable signal against the energy which can be saved when the functional logic is gated. This depends not only on the amount of logic which is gated, but the percentage of the time that the logic will actually be used, which often depends on the incoming data. Often, it is a matter of running simulations based on expected data to determine the viability of reducing energy consumption via gating a particular section of logic.

To gate a clock, the designer needs to provide a control signal which allows logic to operate only when it is needed. In the ALU example, a simple solution is to use a combination of operand signals to create a gating signal. This control signal is then connected with the clock to an AND gate, so that when the enable signal is low, the clock is forced to remain low.

One final note is that it can be seen from the above discussion that we can only gate contiguous sections of logic, separated by registers. The granularity of the possible gating thus depends on the degree of pipelining applied to the circuit.

The end goal of clock gating the Reed-Solomon decoder design is to ensure that the various functional units only receive a clock when they are currently processing data. Because each of the units in the decoder takes a different amount of time to perform its operations, there is significant downtime among some of the functional units in the design.

2.3.3. Memory Access Reduction Techniques

Clock gating of embedded memory units is of particular interest in FPGAs, due to their high energy consumption rates when compared to general logic circuitry. Rather than clock gating an entire memory, it is possible to re-format a memory unit and insert small buffers before and/or after an embedded memory to allow it to be gated for larger periods of time. This is accomplished by reconfiguring the memory to have a bus size which is a multiple of the original width, reducing the overall number of elements, keeping the same overall size constant. The buffers then combine data from two or more contiguous writes into one data point, and similarly read a large data point and then provide each unit of the data to the logic separately, one after the other. Figure 2.6 and 2.7 below illustrate this concept.

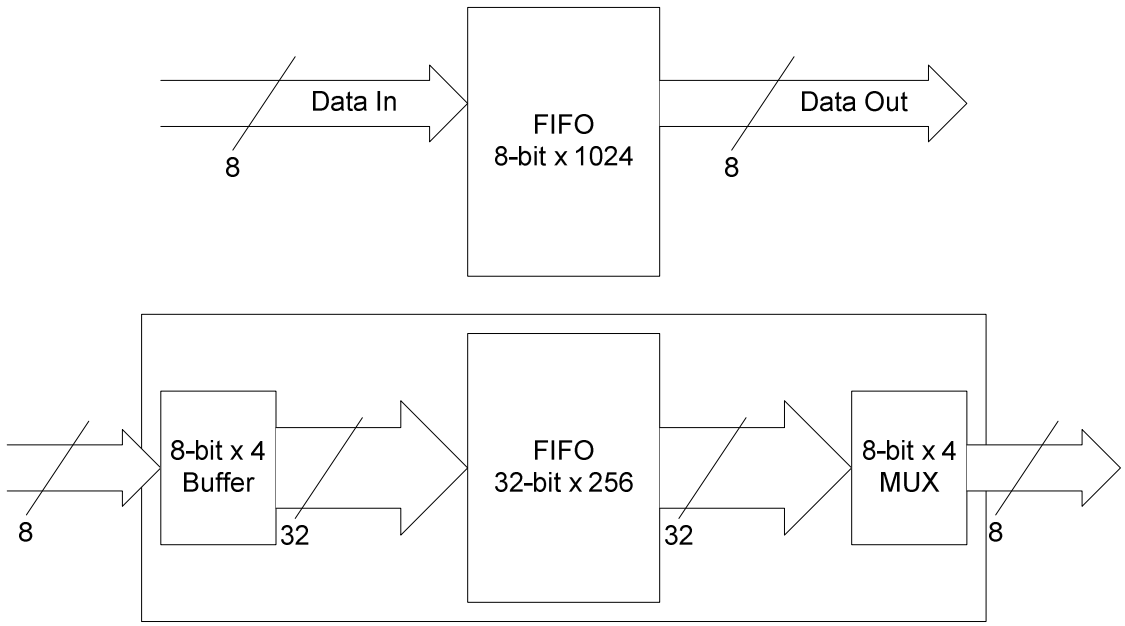


Figure 2.6. Memory Buffering

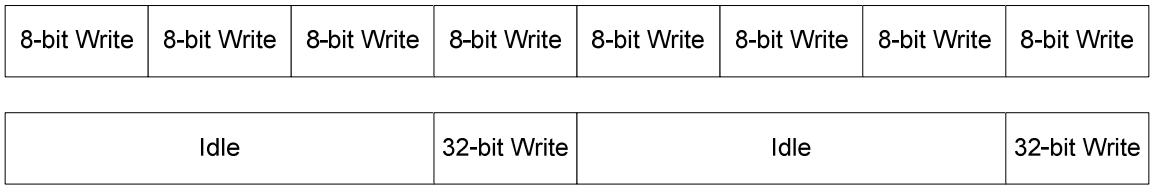


Figure 2.7. Activity of Memory, original vs. buffered

Figure 2.6 illustrates the structural differences between a typical memory setup (above), and the buffered setup (below). The buffered setup collects four 8-bit data points before performing each 32-bit write to the embedded memory block. On the read side, one 32-bit data point is read every four clock cycles, and 8-bits are presented to the output every cycle. Figure 2.7 illustrates the activity of the embedded memory under the typical and buffered schemes. From the activity diagram, it is evident that the activity of the memory can be reduced by 75% by buffering.

This method allows the memory to be deactivated for half or more of the time it would have previously been active, reducing energy consumption significantly. While some extra energy will be consumed by the buffers, it is typically much less than the energy saved by deactivating the memory. A caveat of this method is that it is only possible when the data will be written and read in order, otherwise this method is not applicable.

2.4. Dynamic Reconfiguration

One of the other methods used in this work is the concept of dynamic reconfiguration. Dynamic reconfiguration refers to the technique of changing the

functionality of a component during operation, to achieve a specific goal such as increased performance, reduced power consumption, increased speed, etc. Previous works have shown dynamic reconfiguration of FPGA based applications to be effective for many goals, including reducing the size of the necessary hardware component [12], for the support of concurrent applications[32], and directly related to this work, to reduce power consumption and increase performance [15]. This particular work will examine dynamic reconfiguration for energy efficiency, essentially attempting to minimize the amount of energy needed to decode a certain amount of data, and reconfiguring as channel conditions allow in order to swap in a more efficient decoder. The methodology of this process will be examined in Chapter 4.

CHAPTER 3

RELATED WORK

3.1. Previous RS works

This section will contain an overview of previous works in the area of Reed-Solomon decoders. As mentioned in the introduction, this system is designed to provide Reed-Solomon decoder functionality for a system which is already FPGA based, as the performance and energy consumption characteristics of an FPGA based RS decoder are unlikely to better an ASIC implementation.

3.1.1. A Low-Power Reed-Solomon Decoder for STM-16 Optical Communications

This paper [21] describes a low-power ASIC implementation of a Reed-Solomon (255,239) decoder, designed for submarine communications. It is included here to illustrate the current performance levels of ASIC implementations of the Reed-Solomon algorithm. The design implements a novel syndrome calculation unit, along with a modified Berlekamp-Massey algorithm as opposed to an implementation of the MEA or EA algorithms to solve the key equations. The chip was implemented using .25um CMOS standard cells. The resulting performance characteristics are a sustained 2.5Gbps throughput with a CER of 10^{-4}, and the entire chip consumes 68.5mW of power. Calculating the energy-per-codeword value from these characteristics, it is clear that each bit requires approximately 2.74×10^{-11} J to process. Comparing this to the previous work, which required approximately 1.25×10^{-9} J to process a bit, counting dynamic power only, it is unlikely that any FPGA implementation of an RS decoder will be able to beat the

ASIC in terms of energy consumption. This is why this work is aimed at systems already containing an FPGA for processing, in which case the RS decoder can be added to the existing FPGA code instead of requiring an external RS ASIC.

3.1.2. Design of a Reed-Solomon Decoder using Partial Reconfiguration of XILINX FPGAs – A Case Study

This paper [12] uses a Reed-Solomon coder and decoder to test a design methodology aimed at allowing for partial run-time reconfiguration of applications. The design uses both static modules, and so called pRTR modules, which are the partial run-time reconfigurable parts of the design. The design works by maintaining the same overall structure by loading pRTR units as needed into the same physical location. It makes use of a static CLB interface macro to handle communications between modules. The seven pRTR modules encompassing the RS encoder and decoder are: RS coder, RS decoder, syndrome calculation, error locations, error locator polynomial, error magnitudes, and error corrections. The design allows for the system to be implemented on a small FPGA by swapping in and out the modules as they are needed.

Unlike the above approach, the approach used in this work makes use of full dynamic reconfiguration. In addition, the reconfiguration is used to adapt to changing channel conditions, not allow for implementation of the design on area-limited devices.

3.1.3. Architecture for Decoding Adaptive Reed-Solomon Codes with Variable Block Length

This work [13] describes the implementation of an adaptive RS decoding system on an Altera APEX20KE FPGA. The system adapts to allow for varying block lengths between 13 and 255, while maintaining error correction capabilities of up to 10 erroneous bytes in a codeword. The goal of this work is to maintain the needed CER by varying the amount of redundancy in the symbol. With this design, the value of t can be varied on a codeword to codeword basis. The design makes use of a multiplexed MEA unit, which allows for pipelined operation of the design. The resultant data rate achieved is 240Mbps, with a resource utilization of approximately 17,000 LUTs.

The main difference between this and the work described in this document is that the system in this work does not vary the block length, it varies the value of K. This allows for less communication between the encoder and decoder, which is desirable in a real-world system. As the goal of this work was to reduce energy consumption over speed, the multiplexed MEA structure is not desirable. Lastly, our system is able to correct more errors given that it implements erasures. The use of erasures allows for a reduction on the load of the MEA unit, and thus a faster operation of this part of the algorithm.

3.1.4. A Reed-Solomon Decoder with Efficient Recursive Cell Architecture for DVD Applications

This paper presents an errors-only RS(208,192) decoder implemented on an Altera FLEX10KE200 FPGA [14]. The goal of this work was to examine and design an

efficient MEA architecture which would reduce the time to compute MEA by 32% compared to standard architectures. The design makes use of a reduced number of MEA cells, which are multiplexed and used recursively. The design makes use of the number of MEA cells needed so that the computation is not limited by the MEA block. This means that the computation must be performed in n clock cycles, as this is the number of clock cycles between codewords. For the RS(208,192) decoder, this needs only one MEA cell. However, if the architecture was used on larger decoders, such as RS(255,223), it would require 4 MEA cells. The architecture achieves a decoding speed of 20Mbps.

In the proposed work, again, the decoder can correct more errors by using an errors-and-erasures approach. This allows for a reduction in the necessary processing using the MEA unit, and thus one MEA unit can be used recursively, which also reduces overall energy consumption compared to the above work.

3.2. Previous FPGA Energy Reduction Works

This section will highlight previous works involving reduction of FPGA energy consumption.

3.2.1. The Impact of Pipelining on Energy per Operation in Field-Programmable Gate Arrays

This work by Wilton [8] examines the impact of pipelining on energy consumption for FPGA designs. The study used 4 benchmarks, 64-bit Integer Array Multiplication, Triple DES encryption, 8-tap FIR filter, and a CORDIC circuit. These designs were implemented with varying degrees of pipelining ranging from one or 2

levels to the maximum possible amount of pipelining, a register after every LUT. The result of the work demonstrates that pipelining can reduce the overall energy-per-operation values across all of the benchmarks, by as much as 75%. However, with some benchmarks, there is a reduction in benefits as more and more pipeline stages are introduced, suggesting that there is a particular amount of pipelining at which the best energy performance can be achieved.

An approach similar to the one used above, although less exhaustive, was used in this work to examine exactly how much pipelining is beneficial in the design.

3.2.2. Energy-Efficient Signal Processing Using FPGAs

In this work by Choi et al, [10], algorithmic level energy optimizations were examined for their impacts of energy dissipation in several FPGA applications. The applications studied were the Fast Fourier Transform (FFT) and Matrix Multiplication. The methods used to reduce energy were Architecture Selection, Module Disabling, Algorithm Selection, Pipelining, and Parallel Processing. Module disabling is essentially implemented by using clock gating to restrict the clock from propagating to sections of the logic when no result is needed. One of the main uses of this technique in this work is to clock gate the memories when they are not in use. The authors point out through simulations that an embedded FPGA memory block will dissipate approximately 10% of the energy when it is disabled than it would if enabled.

This technique was adopted with great success in this work given the large number of embedded memory units required by the design. In addition, clock gating, as mentioned in Chapter 2, was expanded to include any functional unit which can be switched off for any amount of time.

3.3. An Adaptive Reed-Solomon Errors-and-Erasures Decoder

This section will provide a detailed description of the Reed-Solomon errors-and-erasures decoder system developed by Lilian Atieno as part of her masters thesis [15]. The resultant work was presented at the Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Monterey, CA, February 2006 [7], and can be referred to for more in depth analysis of her work. The work described in this document uses this previous work as a baseline to improve upon.

For this previous work, an FPGA based adaptive errors-and-erasures Reed-Solomon (255,k) decoding system was developed. An FPGA was used for this work because it allows for dynamic reconfiguration during run-time, and also allows for high levels of parallelism and an efficient implementation of the design. The system makes use of a multi-decoder scheme, under which multiple decoders operate in parallel to allow for more accurate decoding of data. The system makes use of the reconfigurability inherent to the FPGA device by swapping in decodes of differing K values and thresholds as channel conditions dictate.

The adaptive algorithm operates on two levels. First, it attempts to adapt to small changes in the SNR value of the channel by changing the number of active decoders between 1 and 3 without changing the K value. In this case, each decoder has a different

threshold value. Secondly, if larger variations in SNR occur, the decoder sends a request to the encoder to modify the K value to add or subtract from the amount of included redundancy in the signal, and changes the decoder to match. A diagram of the adaptive system is shown in Figure 3.1.

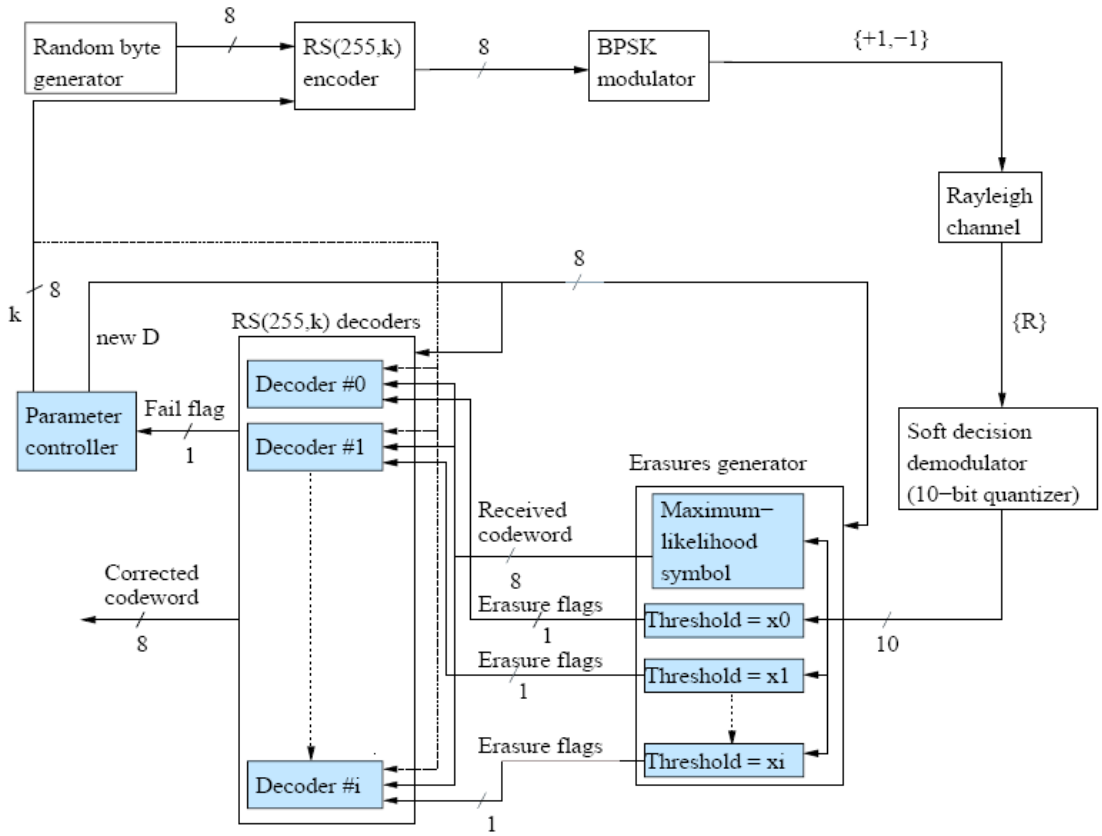


Figure 3.1. From [7], Architecture of an Adaptive errors-and-erasures Reed-Solomon Decoding System

The system aims to maintain a CER of better than 10^{-4}, while allowing for the maximum possible decode rate, as channel conditions dictate. The thresholds for the various erasure generators were determined through Matlab simulations, and the values

were chosen so that the required CER of 10^{-4} could be maintained under all circumstances. The table of decoder configurations is shown below.

k	D	T	SNR (dB)	LUTs	FFs
239	1	0.15	19.6-20.0	7,056	1,921
239	2	0.15, 0.28	19.4-19.6	11,644	2,925
239	3	0.15, 0.28, 0.32	19.2-19.4	16,459	3,922
237	1	0.20	19.0-19.2	7,608	2,207
237	2	0.08, 0.22	18.8-19.0	12,699	3,119
237	3	0.04, 0.22, 0.35	18.6-18.8	17,967	4,197
233	1	0.21	17.6-18.6	8,711	2,251
233	2	0.04, 0.21	17.4-17.6	14,553	3,503
233	3	0.04, 0.18, 0.26	17.2-17.4	20,544	4,744
229	1	0.21	16.4-17.2	9,710	2,475
229	2	0.21, 0.27	16.2-16.4	16,550	3,960
225	1	0.21	15.6-16.2	11,007	2,736
225	2	0.04, 0.22	15.4-15.6	18,580	4,344
225	3	0.04, 0.22, 0.30	15.2-15.4	26,652	5,935
221	1	0.21	14.8-15.2	12,102	2,962
221	2	0.04, 0.21	14.6-14.8	20,301	4,732
221	3	0.04, 0.21, 0.30	14.4-14.6	28,821	6,421
217	1	0.21	14.0-14.4	13,155	3,186
217	2	0.04, 0.21	13.8-14.0	22,175	5,116
217	3	0.10, 0.24, 0.34	13.6-13.8	31,567	6,965

Table 3.1. From [7], Decoder Configurations

As mentioned above, the system was designed to adapt to changing channel conditions. A set of experiments were performed to evaluate the effectiveness of this reconfiguration scheme given simulated channel characteristics. The results illustrate a 14% increase in decoding speed over a non-reconfigurable decoder with a K value fixed at 217.

3.4. Differences from Previous Work

There are several important differences between the previous work described in section 3.3 and the work done for this thesis, which will be described in detail in Chapter 4. These differences will be highlighted briefly here.

As mentioned above, the previous work made use of several multi-decoder implementations, in which several decoders were implemented in parallel. This was done in order to maintain the required CER while allowing for increased decoding speed. However, in terms of energy, it is extremely inefficient to have multiple decoders running at the same time when only one of their outputs is utilized. For this work, only single decoder implementations will be examined, with a slight reduction in decoding speed being the result. The new table (similar to Table 3.1 above) resulting from this change will be shown in Chapter 4.

The algorithm which will control the reconfiguration scheduling will be simplified as a result of the simplified single decoder system, changing configurations in order to maintain the required CER while attempting to use the most energy efficient decoder.

The channel model in the previous work was a very basic model, and part of this work is to evaluate the system using a more accurate model. The previous model assumed that the change in SNR between reconfiguration windows was essentially random, while also assuming that there was very little variation during the time a particular decoder was in operation. In order to get a more accurate assessment of the system performance characteristics, the new model is time-dependant, meaning that the SNR at any given point is related to the previous SNR values. In addition, we are not

assuming that the channel conditions remain static during the operation of a particular decoder. The analyses which were preformed include analysis of CER and energy consumption using different rates of reconfiguration, and will be detailed in the next section, Chapter 4.

CHAPTER 4

IMPLEMENTATION

This chapter contains two contributions. In section 4.1, the development of a new communication channel fading model will be discussed, including the reasoning behind changing the model from the previous version. In section 4.2, the process of performing hardware optimizations will be discussed in detail.

4.1. Channel Fading Model

A minor contribution of this work was the development of an accurate channel model to answer important questions regarding system-level performance characteristics under real-world constraints. The channel model used in [7] was a *non*-time-dependant Rayleigh fading channel model. While the channel model represented a Rayleigh channel, there was no correlation between consecutive samples. Thus, unrealistic variations in SNR could occur in very short amounts of time. This is not a realistic model, and while useful for general analysis, a more accurate time-dependant model is needed.

The non-time-dependent model made the assumption that the SNR would remain static between reconfigurations. This leads to an unrealistic representation of a Rayleigh fading channel, as it assumes the channel spends long periods of time in a relatively stable state while a particular decoder is operating, and then changes suddenly when we examine whether to reconfigure or not. Because of these reasons, a Rayleigh fading channel was developed which more accurately represents a real wireless environment.

4.1.1. Goals and Requirements

The main goal of applying a more accurate channel fading model is to accurately assess system characteristics relating to the time between reconfigurations of the adaptive decoding system. Specifically, the effects of reconfiguration on system energy consumption and CER are considered.

The most important requirement of this new model is that it be time dependant, so that each sample depends on the previous sample. In addition, it is important to model channel fading and shadowing as two distinct processes. The reasoning behind this requirement is that a decoding system can measure the average channel shadowing over time, but cannot measure the channel fading, as the changes due to fading occur too rapidly and vary greatly over short intervals of time. The system's decisions on how to reconfigure would thus be based on the channel shadowing measured over time, while the performance in terms of CER would be determined with regard to the cumulative effects of both shadowing and channel fading.

Finally, the model should represent a channel with an average SNR of approximately 16.8dB, with a range of SNR values (with regard to shadowing) from about 13dB to about 20.5-21dB, as this is the range of SNR values used by the original decoder. In reality, this range could be adjusted based on the required CER performance and other parameters, but for this work the previous assignments will be used for evaluation purposes.

Figure 4.1 illustrates how the SNR in the channel varies with regard to shadowing alone vs. shadowing and fading. With the inclusion of fading, the SNR varies wildly compared to the results of shadowing alone. This variation can sometimes be as much as 40dB from top to bottom.

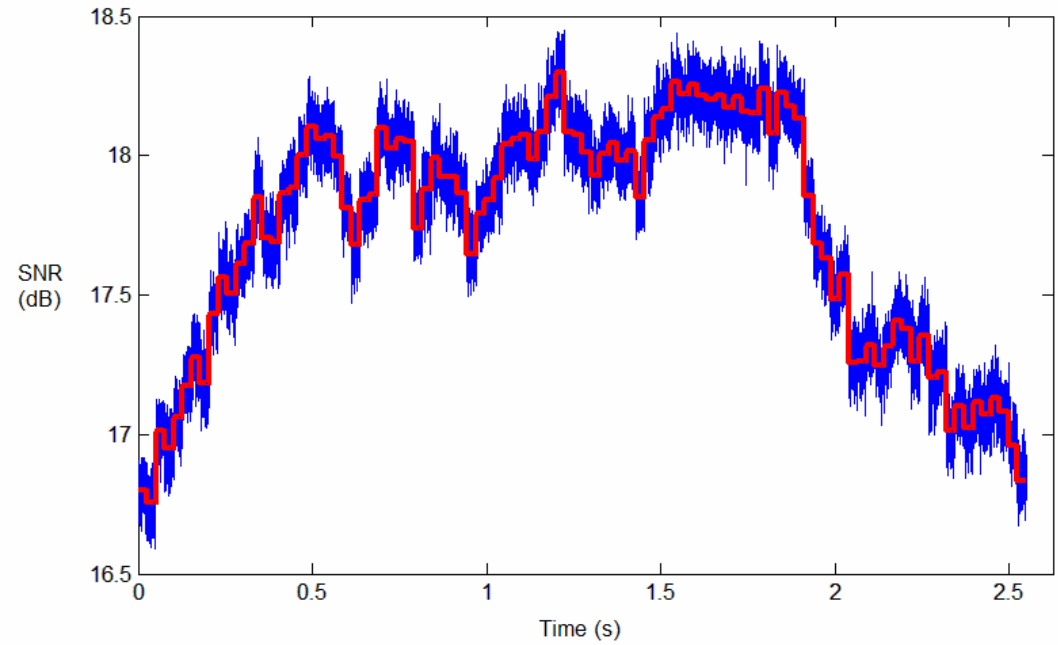


Figure 4.1. SNR due to Shadowing (red) vs Shadowing and Fading (blue).

4.1.2 Simulation Flow

In order to simulate the process of the message data being transferred over the noisy channel, we perform several steps. These steps are outlined below and discussed in more detail in subsequent sections. The purpose of simulation is to determine appropriate decoder parameters for later implementation in hardware.

1. An initial data block of size K bytes is randomly generated. This represents the message data. This data block is then encoded using the RS algorithm, resulting in a 255 byte-encoded message.

2. The encoded message is modulated using BPSK modulation, where each '0' bit becomes a +1, and each '1' it becomes a -1. This generates a stream of +1 and -1 values of length 2040.

3. The stream of BPSK modulated values is affected by shadowing and fading as it passes through the noisy channel. The details of this process are described in section 4.1.3. The result is a stream of values ranging from about -2 to 2.

4. To determine if the codeword will be decoded properly, we simulate the soft-decision demodulation process which is performed by the erasure generator. This process is described in detail in section 2.1.4. The end result is a number of erasure flags representing suspected errors. By comparing the received data stream to the original stream, we can quickly determine how many errors were introduced. We can then determine if this codeword would be decoded properly by comparing the number of erasures and unflagged errors to the error correction capacity of the particular decoder using the equation:

$$N\text{-}K \geq (\text{number of erasures}) + 2*(\text{number of unflagged errors})$$

5. If this equation holds true, then the decoder will properly decode the message, and no error has occurred. If not, then the decoder will be unable to correct the message. By simulating over a large number of codewords, we can determine the effective CER.

4.1.3. Model Details

The new channel model developed for this dissertation has two distinct parts. The first part models the shadowing. For the following equations, the basic parameters are SNR_mean = 16.8dB and ρ=0.99999. The variable ρ is determined by the time between samples and the expected relative velocity of the two nodes, which communicate via the wireless channel. Parameter N(x,y) is a Gaussian random variable, with mean x and variance y.

The algorithm for determining the SNR with respect to shadowing is:

1. Generate values of x_i, where $x_{i+1} = \rho x_i + N(0,1-\rho^2)$, beginning with an initial x_0 value of 0.
2. Generate SNR_i values via the equation SNR_i = SNR_mean + x_i. Each SNR_i value represents the channel as seen by the decoder over the course of a single codeword, measured in decibels (dB).

The resulting series of SNR_i values are representative of what the decoder measures during operation. The average of these values over time is used by the decoder to determine how to reconfigure the system in response to channel variations.

The second part of the channel model determines signal noise variations due to channel fading. These variations occur at a much greater frequency than the variations due to shadowing, and are assumed for this work to not be accurately measurable with regards to the decoding system.

Channel fading is performed as follows:

1. For each symbol in the codeword, generate a fading variable, φ.

   The fading variable φ is created via the following equations where C is a constant:

   - $\varphi = \text{sqrt}(\varphi_r^2 + \varphi_i^2)$
   - $\varphi_r = C + N(0,(1-C^2)/2)$. This is the real part of the fading.
   - $\varphi_i = N(0,(1-C^2)/2)$. This is the imaginary part of the fading.

2. This affects the BPSK modulated transmission of the symbol via:

   - $R_i = \varphi * \pm 1 + N(0,\sigma^2)$

   where ± 1 is the original value of the bit sent over the channel by the BPSK modulator, and $\sigma^2 = 1/(2*SNR_a)$. SNR_a is SNR_i in absolute notation, which is calculated via the equation: $SNR_a = (10)^{SNR_i/10}$. The R_i values represent the floating point values received by the decoder after the message data has been impacted by the channel fading during transmission. One can see that in this equation, we have the impact due to both channel shadowing (the random variable has variance equal the SNR due to shadowing), and channel fading, which is represented by the φ variable, on the original sequence of +1/-1 values representing each bit of the message.

Part of the benefit of the above model is that it can be changed from Rayleigh to Rician or anything in between very easily. A Rayleigh channel is representative of a situation without line-of-sight communications, whereas a Rician channel represents a situation where line-of-sight communication is possible. If the variable C is set to zero, the model is Rayleigh, whereas if C is set to approximately sqrt(0.8), the model is Rician. Setting the value of C to a smaller number creates more variance in the fading variable by increasing the variance of the random variables which determine φ.

The new model was coded as a C program to allow for simulation, which is described in the next section.

4.1.4) Experiments

Several avenues of investigation were pursued with the new channel model. First and foremost, an examination of the effect of time-between-reconfiguration on system energy consumption was performed. Since the focus of this work is on minimizing the energy consumption characteristics of the system while maintaining constant CER, there was a desire to evaluate how lengthening or shortening the time between system reconfiguration would affect the overall energy consumption.

The second part of the reconfiguration analysis was driven by a desire to examine how the CER would vary with regard to the reconfiguration rate. Conceptually, the longer the time between reconfigurations, the more the chance of the SNR varying to a point where the SNR is outside the range that the currently instantiated decoder was designed to operate in. If the SNR rises above the range for the decoder, it is not a problem, since that simply implies fewer errors on average than the maximum the

decoder was designed for. If, on the other hand, the SNR falls below the decoder's designated range, the CER will suffer as the decoder will be unable to correct all of the errors. Figure 4.2 below illustrates how the SNR can vary in between reconfigurations.

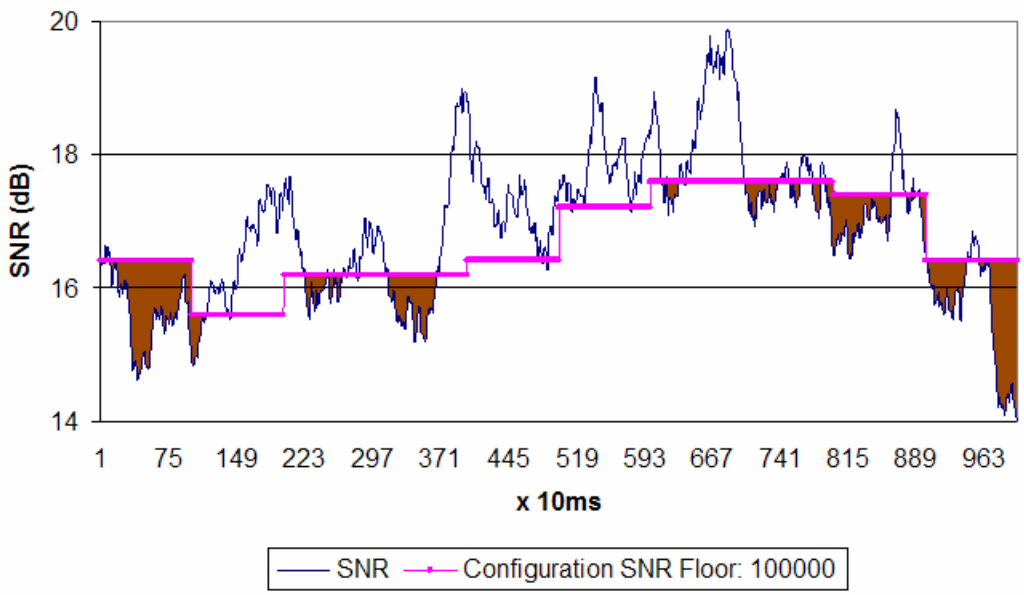


Figure 4.2. SNR variance during operation

Figure 4.2 displays how the SNR can vary between reconfigurations. The pink line displays the floor of the SNR range of the currently instantiated decoder. Notice that it varies periodically due to system reconfiguration. The blue line shows the actual SNR, and the red shading shows times when the SNR is below the desired level for the current configuration. These represent areas where we would expect to see an increase in the CER.

During experimentation we determined the optimal reconfiguration rate in terms of energy consumption, while maintaining the desired CER of 10^{-4}. Results of this analysis are presented in Chapter 6.

4.2. Hardware Optimizations

The second major part of this work applies hardware optimizations to Reed-Solomon decoders at the architectural level, with the goal of reducing energy consumption. The metric of success for this part of the work is the amount of energy required to decode each codeword. As described in Chapter 2, the optimizations used for this process are pipelining, memory operation optimization, and functional unit clock gating. This section provides a detailed look at the modifications, which were applied to the original design. Results of these modifications on design energy consumption characteristics are shown in Chapter 5. Figure 4.3. illustrates a basic system diagram.

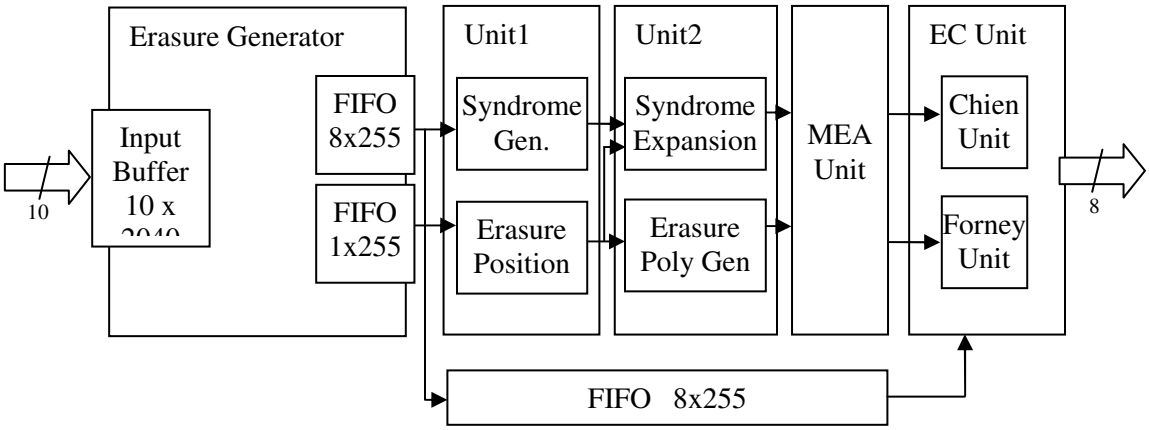


Figure 4.3. System Block Diagram. All connections are 8-bit unless otherwise specified.

4.2.1 Recoding

Before energy reduction techniques were applied, the Verilog source code for each portion of the design was re-coded with an eye towards performing each required operation as efficiently as possible, in terms of both power consumption and the number of required clock cycles. During this re-coding, several design goals were targeted. First, the required number of clock cycles to complete the decoding process was minimized. In most cases, this provides a reduction in required energy to decode each codeword, as energy consumption depends not only on power dissipation, but also on the length of time that power is dissipated. Second, the number of control signals was minimized. Since much of the control circuitry drives clock gating signals for individual modules, some unnecessary signals could be removed. Each design module was re-designed to function only when it receives a clock signal. Thus, the clock distribution to these modules acts as a de-facto control structure. In addition, the decoder structures were written to ensure that the final hardware mapping from Verilog would result in efficient RTL structures. This was done by explicitly defining each individual circuit element and associated connections, as opposed to coding a higher-level description of the functionality, using tasks, for loops, etc. It was observed that this approach leads to a more accurate mapping of functions to the FPGA fabric. Lastly, the designs were coded to include parameterization wherever possible, allowing for easier modifications across all seven decoders.

In addition to these general goals, some specific modifications were performed during the re-coding process, as detailed below.

4.2.1.1. Syndrome Unit

Originally, the syndrome vector, generated by the syndrome unit (see Figure 2.4 for the system diagram), was sent serially to the syndrome expansion unit. Each coefficient was then loaded serially into a register before syndrome expansion began. This inefficiently increased the number of required clock cycles, since the entire syndrome can be transferred in parallel to the expansion unit, allowing for the expansion unit to begin work immediately.

4.2.1.2. Modified Euclidean Algorithm

Several structural changes were implemented for the MEA unit, or Key Polynomial Generation Unit (Figure 4.4a) compared to the structure described in the previous work [7] (Figure 4.4b). First, the dual-ported RAM units, which were used in the previous design, were replaced with shift registers. The use of RAM units to store MEA results after each iteration causes increased delays due to memory accesses. Given the size of the required memories, which range from 128 to 304 bits, using energy-consuming embedded memory blocks is inefficient. In addition, the slightly modified structure detailed in [30] was adopted, as it allows for easy pipelining of the unit. The main difference between the new structure and the previous one is that instead of evaluating the degree of the L polynomial (the number of coefficients) via evaluation before each iteration to see if a stopping point is reached, which causes additional delays

and increased logic usage, we run the unit for a specified number of iterations (n-k) for each decoder that guarantees the processing will be finished. This is controlled by the stop_logic block.

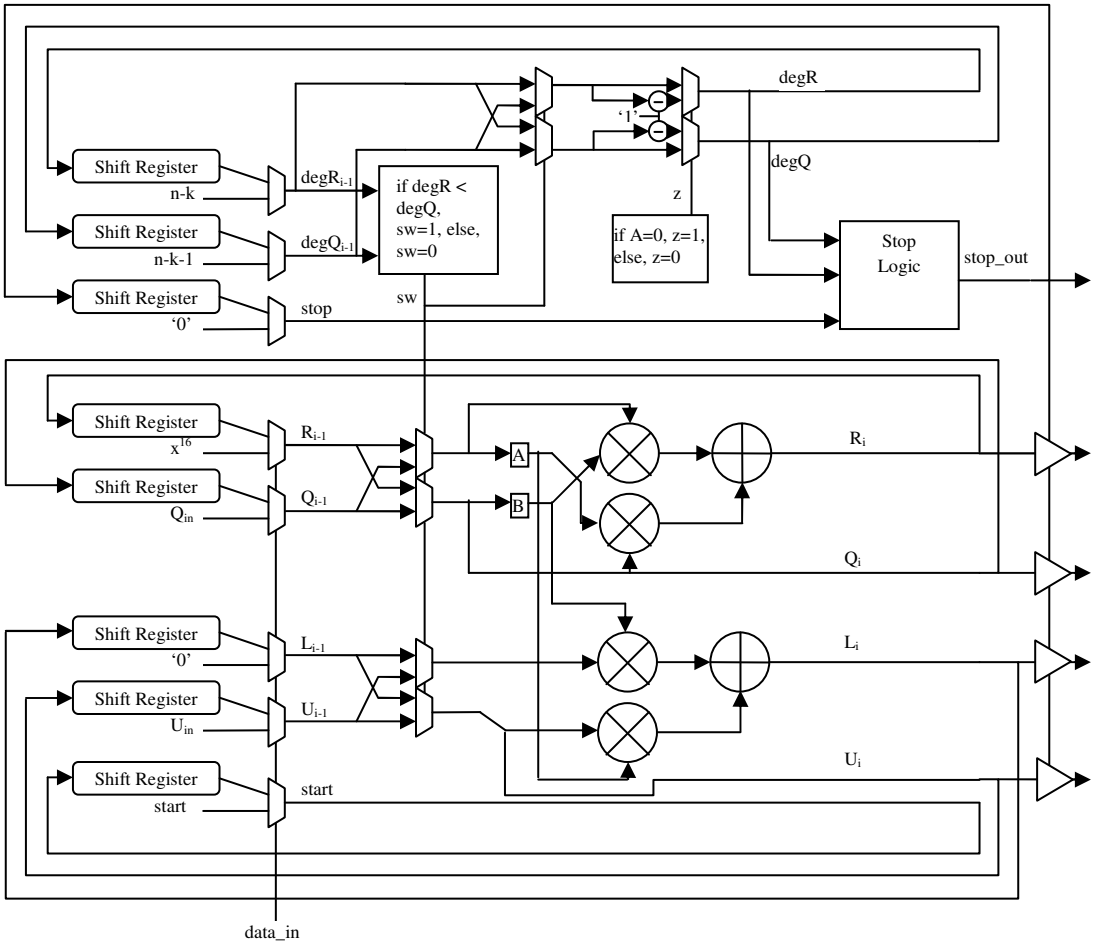


Figure 4.4a. New MEA Structure

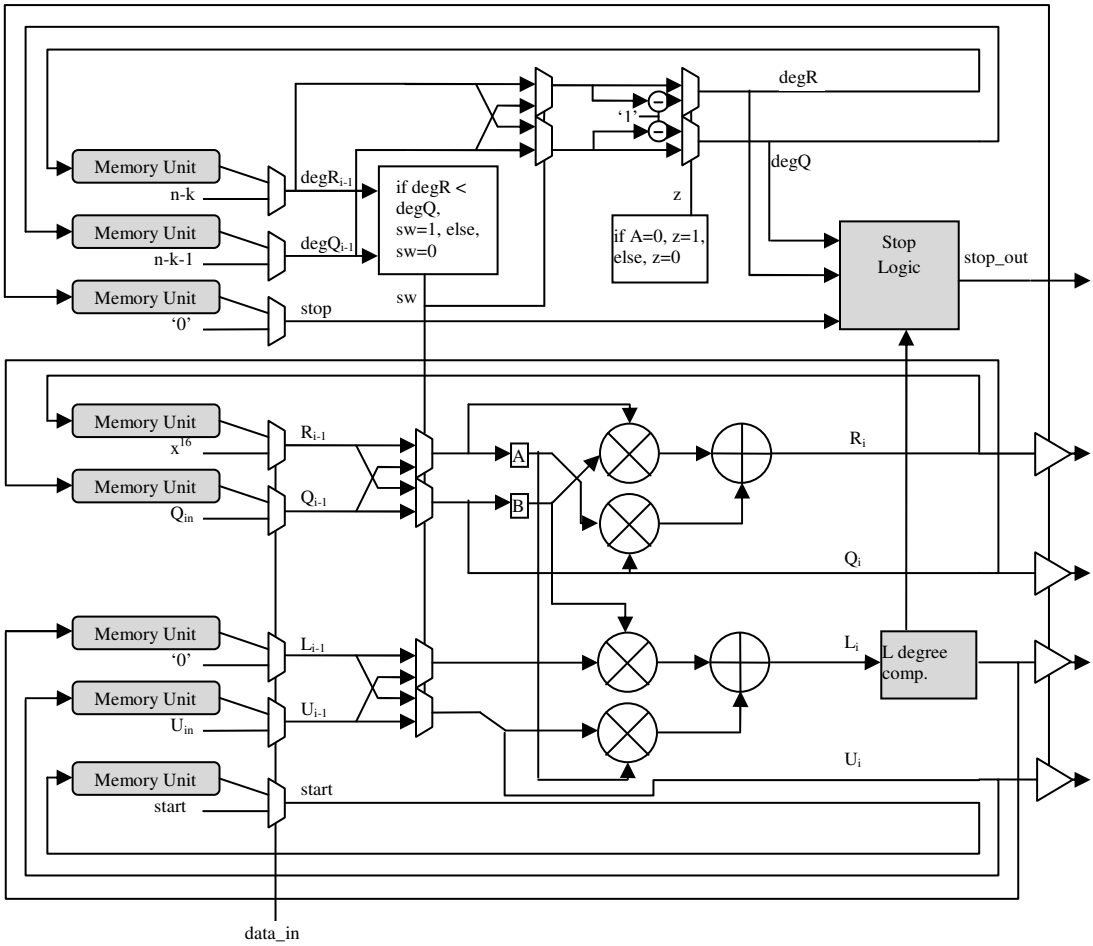


Figure 4.4b. Previous MEA structure [7]. Note the degree of the polynomial L must be computed each iteration.

4.2.1.3. Inverse ROM

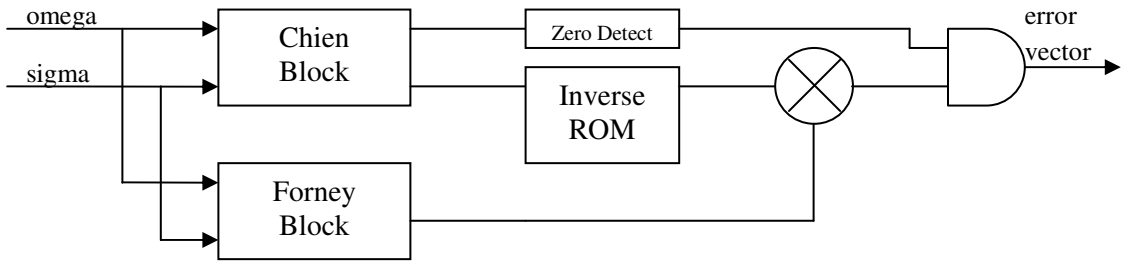


Figure 4.5. Error Correction Unit

For the inverse ROM used in the error correction unit (Figure 4.5), an FPGA embedded memory block was pre-loaded with inverse GF elements. The job of the inverse ROM is to invert values within the Galois Field. The previous approach [7] generated the inverses algorithmically on chip after device reset. The pre-computation approach saves both energy and area.

4.2.2 Pipelining

Pipelining was divided into two separate steps, small-scale pipelining, and global pipelining. Small-scale pipelining is used within a particular functional unit, while global pipelining is performed at functional unit boundaries. Specific applications of pipelining for energy savings in the RS decoder designs are detailed below.

4.2.2.1. Small-Scale Pipelining

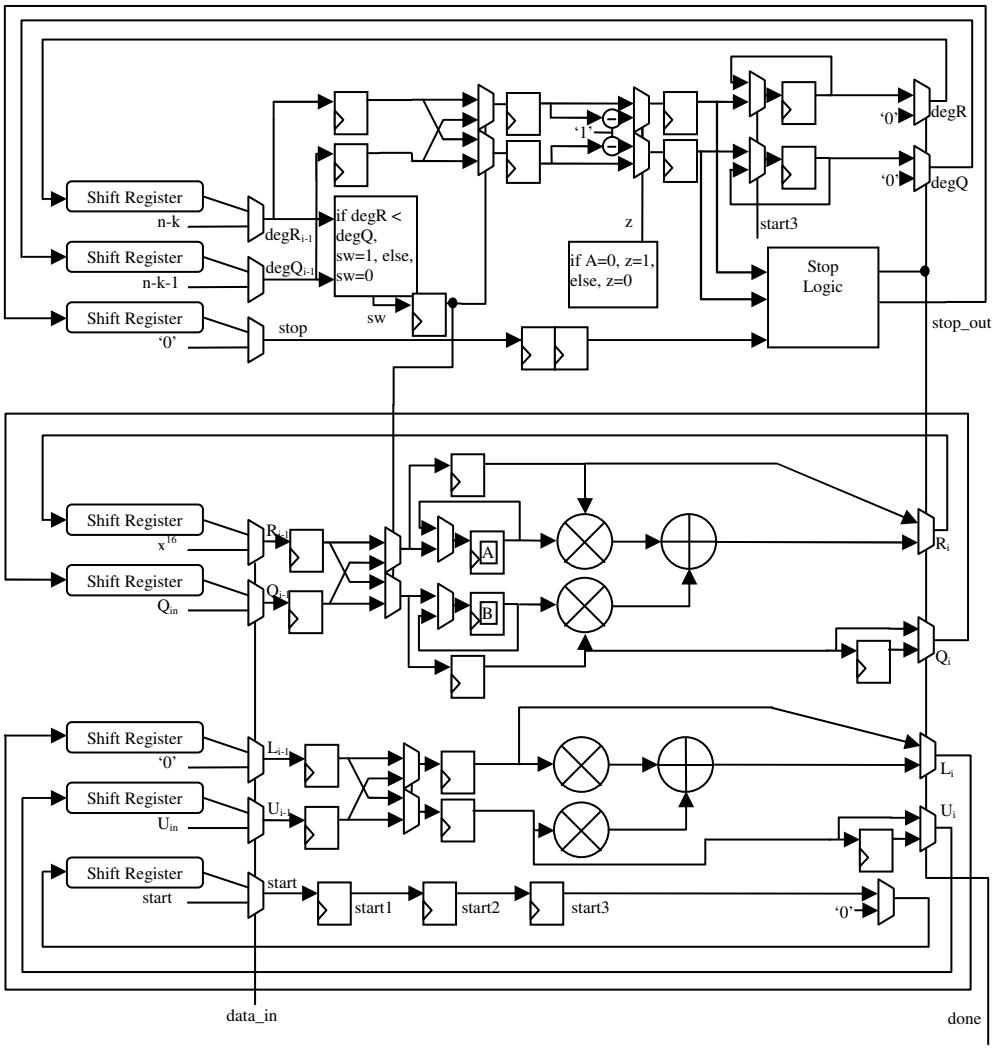
In this step, each functional unit was examined to determine if pipelining could be used to improve energy consumption characteristics. Initially, the most promising location for pipelining appeared to be in the 8-bit GF multiplier circuits. As the design uses hundreds of these small units, any reduction in energy consumption in this unit was expected to have large effects on the overall energy consumption characteristics of the decoder. Initial work was done to develop pipelined versions of the Mastrovito GF multiplier, which was used in the previous design [7]. Pipelining of between zero (combinational) and four stages was examined.

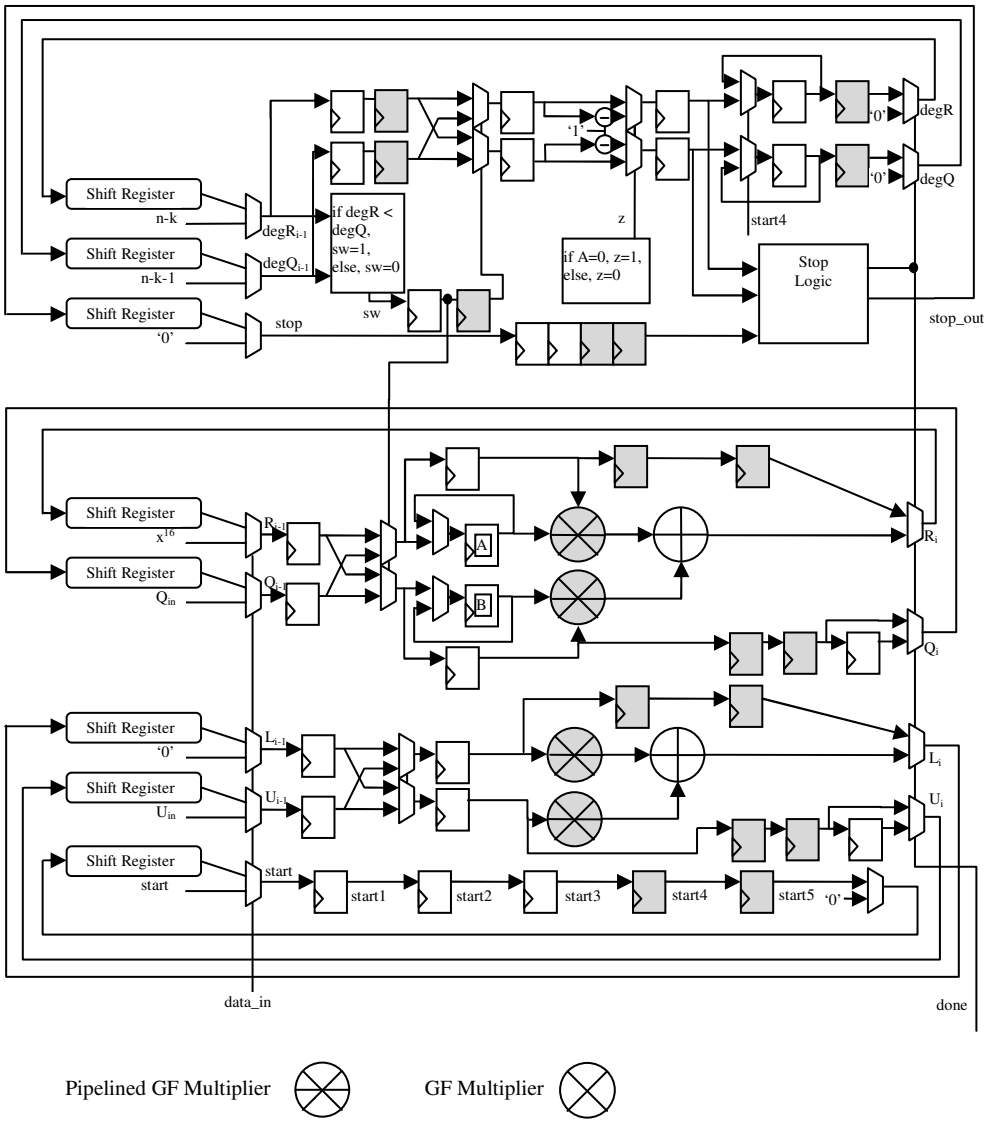
An alternative GF multiplier, described by Paar in [19], was constructed and tested to see if it would be more energy efficient. The Paar multiplier has been shown to have a lower VLSI complexity than the standard Mastrovito multiplier, so the initial thought was that the lower complexity would lead to lower energy consumption. However, despite the fact that the Paar multiplier was implemented using three fewer LUTs than the Mastrovito, it was observed through testing that the Paar multiplier in fact dissipated about 20% more energy than the Mastrovito multiplier because of increased glitching due to mismatched path lengths. The maximum amount of pipelining was found to be four stages due to the critical path length being 4 LUTs. In the end, the 2-stage Mastrovito multiplier was used for our final design as it was found to be the most efficient in terms of energy.

Despite the reduced energy consumption of the pipelined multiplier, it could not be used to replace the majority of the GF multipliers in the decoder design. When the design was examined in detail, it was discovered that the vast majority of the GF

multipliers in the design exist within feedback loops which require a latency of only a single cycle to function properly. Thus, the pipelined multiplier, which requires multiple cycles to perform a multiplication, could not be used in these cases. In the end, only four of the GF multipliers in the design were replaced with pipelined multipliers. These three multipliers are: the two GF multipliers in the MEA unit, and one each in the Forney and Chien units. While these multipliers were replaced with the pipelined units (see Figure 4.5), the overall effect on energy consumption was limited, resulting in a decrease of only about 2.5% on the system level. Full results can be seen in Chapter 5.

The other unit which showed the potential for savings using pipelining was the MEA unit. The paper describing the recursive MEA structure [30] suggests using a 5 stage pipeline within the recursive unit for performance reasons, and it was experimentally determined using 5 stages was in fact optimal for energy characteristics. The results of this analysis will be detailed in Chapter 5.4.2. Figure 4.6 below illustrates the MEA unit with 3 and 5 levels of pipelining.


Shift Register
n-k
Shift Register
n-k-1
Shift Register
'0'
degR$_{i-1}$
degQ$_{i-1}$
stop
if degR < degQ, sw=1, else, sw=0
sw
'1'
z
if A=0, z=1, else, z=0
start3
'0'
degR
'0'
degQ
Stop Logic
stop_out
Shift Register
x^{16}
Shift Register
Q$_{in}$
R$_{i-1}$
Q$_{i-1}$
A
B
R$_i$
Q$_i$
Shift Register
'0'
Shift Register
U$_{in}$
Shift Register
start
L$_{i-1}$
U$_{i-1}$
start
start1
start2
start3
'0'
L$_i$
U$_i$
data_in
done



Pipelined GF Multiplier ⊗ GF Multiplier ⊗

Figure 4.6. MEA unit with 3-stage (above), and 5-stage (below) pipelining. In the 5-stage figure, note that the multipliers are now pipelined 2 stages

4.2.2.2. Global Pipelining

Global pipelining refers to the practice of pipelining various functional units to decrease the overall design throughput. This allows for the more efficient utilization of

functional units, which reduces the energy-per-operation characteristics of the overall design.

The previous decoder was not pipelined at all, so that a codeword was sent into the erasure generator (the first functional unit) only after the previous codeword had been completely processed (see Figure 2.4). As a result, only a single functional unit is active at any given time. Ideally, all functional units should be active at the same time, as idle units receiving clock signals still dissipate energy. Operational restrictions can limit this opportunity, as certain units may require more operating time than others. Placing registers between functional units helps improve operation overlapping.

When examining the design at hand, it helps to break the operation into discrete steps, and to examine the time required and dependencies for each individual functional unit. The diagram below shows a clock cycle description of the activity of each functional unit in the decoder for the decoding of a codeword with K=239.

Erasure Generator	Unit1	Unit2	MEA	Error Correction
2049 Cycles	260 Cycles	8 Cycles	262 Cycles	280 Cycles

Figure 4.7. Time breakdown of the decoding process for an example of K=239 decoding

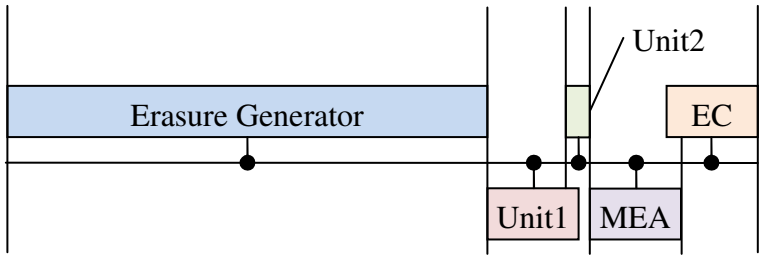


Figure 4.8. Timeline illustrating decoding of a codeword

The above breakdown makes it clear that the bottleneck in the system, in terms of number of required clock cycles, is the erasure generator. The erasure generator requires approximately 3076 cycles regardless of the K value of the design, while the entire operation of the decoder takes only between 764 (K=239) and 1064 (K=217) cycles. For the re-coded design, a similar trend was seen, with the erasure generator requiring approximately 2049 cycles, and the decoding requiring between 903 and 2081 cycles. From this, it was observed that by splitting the design into 2 pipelined stages, all decoders, except for the K=217 decoder, could function with only 2 pipeline stages; the erasure generator, and the decoder. By separating each functional unit in the decoder, a rate of 2049 cycles per codeword could be maintained for all versions of the design. In the case of the K=217 decoder, one codeword is output through the error correction unit while the next codeword is evaluated by the syndrome unit. In all other cases, the decoder only operates on a single codeword at a time, while the erasure generator processes the next codeword. Despite the added pipelining, since there is a memory between the erasure generator and the decoder, no additional pipelining registers were necessary.

This pipelining has several important effects on the design. First, by reducing the time-per-codeword of the design, a significant reduction in energy consumption-per-codeword is achieved (this can be seen in Chapter 5 in section 5.4.3). In addition, all decoders (regardless of K) can operate at the same clock rate, and achieve identical throughput in terms of codewords decoded per second. Lastly, by separating each of the units in a distinct pipeline, each unit can be individually clock gated and turned on and off as needed. The energy reduction results of this global pipelining scheme can be seen

in the following chapter, while the diagram below shows the global pipelining scheme which was adopted for this project.

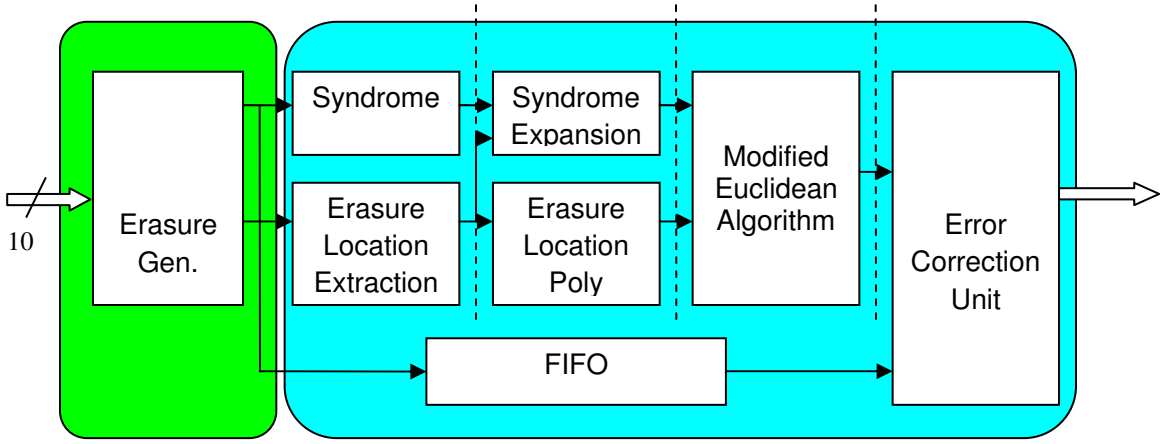


Figure 4.9. Pipelining of Decoder Circuitry
(unless specified otherwise, signals are 8-bit)

4.2.3. Memory Optimizations

As described in Section 2.2.3, the energy required for memory operations using FPGA embedded memories can be reduced by using small buffers before and after the memories which act to collect data coming in and out of the memory units, with the goal of reducing the overall number of memory accesses. Embedded memories in Stratix devices have a physical I/O port size of 32 bits. If several read or write operations can be combined to include most or all of the 32 available bits for each memory access, not only can the number of necessary read and write operations be reduced, but the memory can

be utilized more efficiently. If a value of less than 32 bits is accessed, the RAM will still consume power for the entire 32 bits.

Figure 4.10 illustrates the structure of the buffers and memories when making use of this method. This is just an example, but is representative of the memories between the erasure generator and the decoder, and also the FIFO which holds the received codeword while the decoder is processing. The buffers consist of a number of registers, in this case, 4 8-bit registers because the data is 8-bit and we are packing 4 of the values together, in addition to a small amount of control logic to change the addressing.

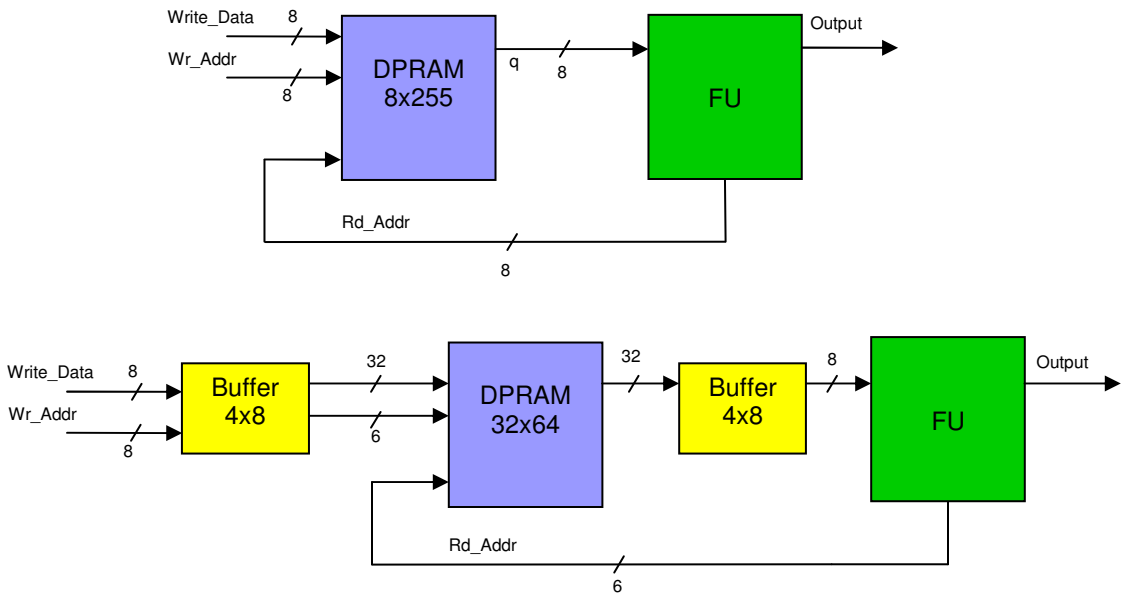


Figure 4.10. Example of Memory Buffering Logic

The Adaptive Reed-Solomon design includes several large memory units, each of which was buffered for fewer accesses as described below:

1. There is a memory unit which receives 10-bit values representing each bit of the codeword from the A/D converter (see figure 4.3). This unit needs to be able to hold 2040 10-bit values, representing one codeword. The values are stored in this memory and withdrawn by the erasure generator.

2. A memory unit serves as a bridge between the erasure generator and the decoder. It consists of a memory which holds the 255 8-bit symbols of the codeword (see figure 4.3).

3. A memory unit serves as a FIFO which holds the received codeword while the decoder determines the error vector needed to correct the codeword. This unit must also hold 255 8-bit values (see figure 4.3).

A detailed discussion of the buffering of each unit is presented in section 5.5., along with the energy benefits of this optimization.

4.2.4. Clock Gating

The RS decoder design includes a linear series of functional units (see figure 4.3). As mentioned in Section 4.2.2.2, the erasure generator serves as the performance bottleneck. Although the erasure generator is always active, we would like to reduce the energy consumption characteristics of the remaining units by shutting off their clock signal when they are not needed.

To reduce energy, the functional units in each decoder stage were clock gated. A small control unit handles the distribution of the clock to each unit, and attempts to minimize the number of clock transitions required by each unit by enabling the functional unit only when it needs to process the codeword. The control unit generates four individual enable signals: one for the syndrome and erasure location extraction, one for the syndrome expansion and the erasure polynomial computation, one for the key polynomials generation or MEA unit, and one for the error correction unit which consists of the Chien search block and the Forney algorithm block. Each unit receives a clock signal immediately before it is presented with data, and once it has output its calculations, the clock signal is discontinued until it is needed again to process the next codeword. Figure 4.11 illustrates how this clock gating was implemented, which is the method suggested by Altera in the Quartus II documentation [31]. The enable signal for a gated clock is clocked into a register on the falling edge of the global clock, and this result is ANDed with the original clock to produce a gated clock signal for the functional unit.

Each enable signal is set high by one trigger, and set low by another. Table 4.1 lists the conditions for activating and deactivating each enable, along with the number of cycles each unit is receiving a clock signal before and after the clock gating was performed. It should be noted that the number of cycles that each unit is active is data dependant and also dependant on K, thus the ranges in the table. The triggering signals were all internal signals which already existed, and did not add any logic to the design. This is why the MEA unit begins when Unit2 has begun, because the activity of Unit2 can be as short as 4 clock cycles, and in order to not create any extraneous control logic,

it is best to start the clock to the MEA unit at this point. Figure 4.12 shows the final system block diagram showing each individual clock domain.

Enable Signal	**Enable Condition**	**Disable Condition**	**Clock cycles to FU (per CW)**	**Clock cycles seen after (per CW)**
Unit1	New_Codeword strobe from erasure generator	syndrome and erasure locations finished being presented to unit2	2049	256 to 294 (dependant on number of erasures)
Unit2	syndrome unit has received 255 symbols	MEA unit has begun processing	2049	4 to 42 (dependant on number of erasures)
MEA	unit 2 has begun processing	error correction has begun	2049	262 to 1450 (dependant on K value)
Error Correction	last iteration of MEA has begun	corrected codeword has been output	2049	280

Table 4.1. Clock Gating Parameters

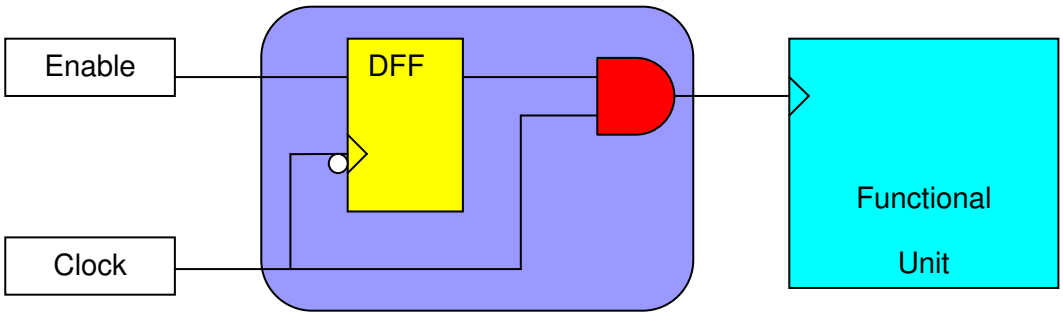


Figure 4.11. Clock Gating Logic

The end result of this optimization is a large reduction (See Table 4.1) in the number of clock cycles seen by each unit in the decoder resulting in a large reduction (~40%) in energy consumption for each gated unit. The full energy numbers are presented in section 5.6.

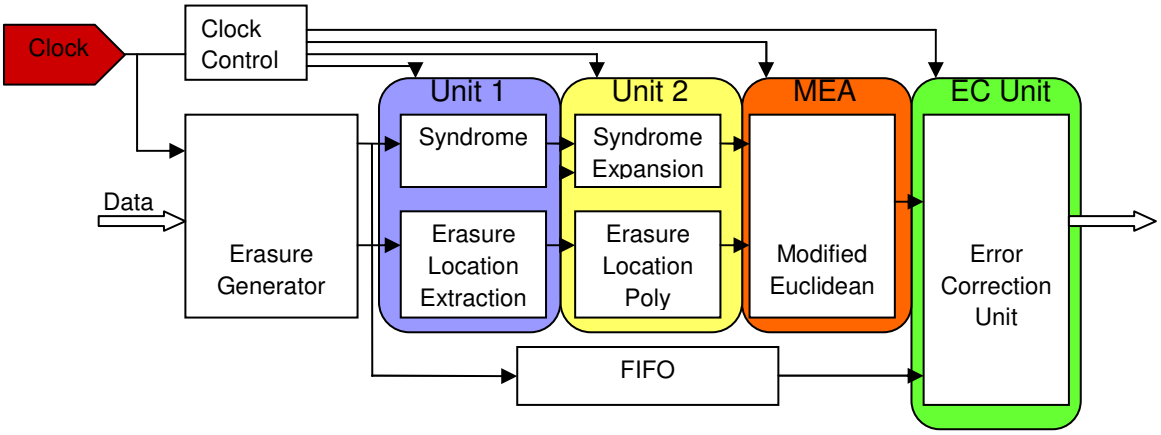


Figure 4.12. Global Clocking Scheme

CHAPTER 5

CIRCUIT LEVEL OPTIMIZATION RESULTS AND ANALYSIS

The next two chapters will provide numerical results generated during the course of this work, along with a detailed analysis of the results. This chapter provides the results of the circuit optimization techniques described in section 4.2. Chapter 6 provides the detailed results of the reconfiguration analysis, described in section 4.1.

5.1. Introduction

This section evaluates the results of applying the energy reduction techniques to the Reed-Solomon decoders developed in [7], both in terms of area and energy consumption. All of the results detailed below were generated by compiling and simulating the RS decoder designs in Quartus II version 7.1, with power numbers from the built in PowerPlay analysis tool used to determine energy consumption values. All designs were simulated at 50 MHz, using waveforms from the previous work. All designs were mapped to the Altera Stratix EP1S10F484C5 FPGA.

5.1.1. Previous Work

To begin evaluation, some modifications to the previous designs were necessary. Previous results were generated using an older version of the Quartus II software, in which the power analysis features had not been fully developed, leading to indeterminate accuracy when measuring power consumption. The old designs were thus recompiled and re-simulated (1 full codeword each), using Quartus II, version 7.1. Energy-per-codeword

and energy-per-Mb of data were determined from these new results. Table 5.1 below illustrates these results.

K Value	LUTs	Regs	Memory Bits	Fmax (MHz)	Pwr @ 50 MHz (mW)	Period (us)	E/Mb (J)
239	5694	1661	35536	65.26	65.14	76.80	2.74E-03
237	6278	1758	35552	62.68	66.27	77.28	2.83E-03
233	6988	1950	35584	65.12	66.34	78.20	2.92E-03
229	7567	2142	35616	64.44	68.29	78.73	3.08E-03
225	8517	2371	35648	64.60	70.27	82.09	3.36E-03
221	9697	2564	35680	63.50	73.27	82.90	3.60E-03
217	10427	2758	35712	64.88	74.13	86.80	3.89E-03

Table 5.1. Results Generated from Designs Developed in [7] using Quartus II, v7.1

5.1.2. Quartus Synthesis Power Optimization

To provide a comparison with an alternate method of reducing power and energy consumption, the original designs were compiled and simulated using the new built-in power-reduction synthesis options available in Quartus II, v7.1. These features have been added to the Quartus II software since the work described in [7] was completed. The tool now provides an automated method of reducing power and energy consumption for FPGA based designs based on low-level logic restructuring. The results of compiling the previous designs with this new option are presented in Table 5.2. The automated power-reducing synthesis algorithms in Quartus result in an energy reduction of 3.48% on average across the seven designs.

K Value	LUTs	Regs	Memory Bits	Fmax (MHz)	Pwr @ 50MHz (mW)	Period (us)	E/Mb (J)	Change
239	6382	1663	45776	63.10	63.07	76.80	2.66E-03	3.18%
237	6774	1760	45792	65.71	63.88	77.28	2.73E-03	3.61%
233	7635	1952	45824	64.89	64.03	78.20	2.82E-03	3.48%
229	8227	2144	45856	63.00	66.05	78.73	2.98E-03	3.28%
225	8945	2373	45888	63.34	68.40	82.09	3.27E-03	2.66%
221	9760	2566	45920	65.21	70.27	82.90	3.45E-03	4.09%
217	10513	2760	45952	64.05	71.14	86.80	3.73E-03	4.03%
							Average	3.48%

Table 5.2. Previous Work with Quartus Automated Power Optimization Results

	LUTs		Regs		Memory Bits		Power (mW)	
Functional Unit	Original	Power Opt.	Original	Power Opt.	Original	Power Opt.	Original	Power Opt.
Erasure Generator	502	498	216	218	24912	24912	33.21	27.78
Unit1	537	441	385	385	128	128	1.49	1.48
Unit2	1827	1824	327	327	0	0	0.22	0.24
MEA	826	683	238	238	2048	2048	8.45	8.29
Error Correction	1841	2692	420	420	6400	16640	20.08	23.7
Fifo	0	0	0	0	2048	2048	1.58	1.58
Top level control	161	244	75	75	0	0	0.11	
Total	5694	6382	1661	1663	35536	45776	65.14	63.07

Table 5.3. K239 Unit-by-unit Power Results

Table 5.3 illustrates the power optimizations on a unit by unit basis. As the table illustrates, the unit which improves the most is the erasure generator. The optimizations reduce the power consumption of the input buffer from 18.33 mW to 13.20 mW. However, for an unknown reason, simultaneously increases the power consumption of the GF inverse lookup table (contained in the EC unit) from 1.38 mW to 4.43 mW

because instead of using logic cells, the table is instantiated as 3 M512 and 2 M4K RAM blocks. This also explains the increase in total memory bits. Why this change is selected by Quartus is unknown.

5.2. Re-Coding

As mentioned in section 4.2.1. the first step in performing the set of optimizations on the decoders was to re-code the basic un-optimized decoders in a more structurally explicit manner, so as to ensure the correct structure when the design is mapped to the FPGA, and also to perform several minor modifications (discussed in section 4.2.1.), and to prepare the designs for the following optimization steps, pipelining (section 5.4), memory optimizations (section 5.5), and clock gating (section 5.6). The results of this recoding process are detailed in Table 5.4, with a unit by unit breakdown for the K239 decoder illustrated in Figure 5.1.

K Value	LUTs	Regs	Memory Bits	Fmax (MHz)	Pwr @ 50MHz (mW)	Period (us)	E/Mb (J)	Change
239	4854	2289	30538	108.25	59.96	57.00	1.87E-03	31.59%
237	5282	2483	30626	107.20	60.79	58.40	1.96E-03	30.62%
233	6201	2866	30802	107.01	62.24	61.68	2.16E-03	26.04%
229	7054	3250	30978	106.37	63.66	65.60	2.39E-03	22.39%
225	7930	3634	31154	105.09	65.49	70.16	2.68E-03	20.34%
221	8867	4025	31390	105.33	67.42	75.36	3.01E-03	16.30%
217	9711	4410	31574	108.34	69.07	81.20	3.39E-03	12.91%
							Average	22.89%

Table 5.4. Recoded Design Results, provides a new baseline for the following optimizations

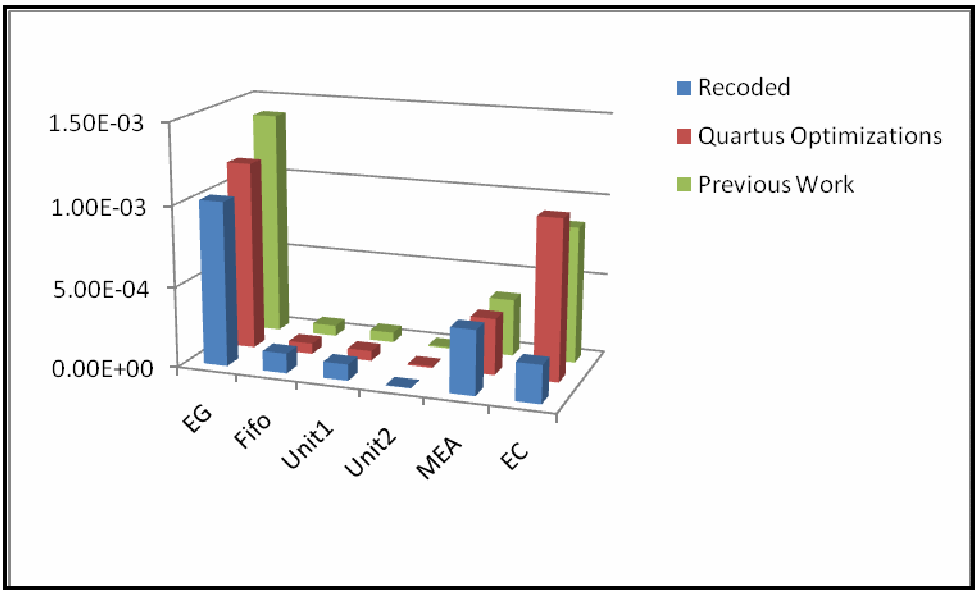


Figure 5.1. K239 Unit by Unit Energy Consumption Breakdowns

	Power (mW)	Energy per CW (J)	Power @ 50 MHz (mW)	Energy per CW (J)	Difference
EG	33.21	1.40E-03	32.61	1.02E-03	-27.12%
Fifo	1.58	6.65E-05	3.93	1.23E-04	84.61%
Unit1	1.49	6.28E-05	3.22	1.01E-04	60.39%
Unit2	0.22	9.27E-06	0.20	6.25E-06	-32.53%
MEA	8.45	3.56E-04	12.77	3.99E-04	12.16%
EC	20.08	8.46E-04	7.51	2.35E-04	-72.24%

Table 5.5. Functional Unit Energy Breakdown for Previous Work and Recoded

The major impact of this recoding was a reduction in the overall number of clock cycles the decoder takes to complete the decoding of a codeword. By eliminating unnecessary handshaking and other communication delays, the decoder can complete the same amount of work in a shorter amount of time (see table 5.6), reducing to overall energy consumption. Although the overall energy results are better with the recoded

version, certain units show higher energy consumption rates than seen in the previous work (as seen in Figure 5.1, Table 5.5). This is because most of the units were designed to use a minimum amount of control logic, since it was assumed that further optimizations (such as clock gating) would be applied later. For instance, the syndrome unit and erasure locator units, collectively referred to as Unit1 (figure 4.3), are designed to run all the time, even though its output is not always necessary..

The benefits are greater for the smaller decoders due to the reduction in MEA run time. As mentioned in Section 4.2.1, the implementation of the MEA unit for this work runs for a distinct number of iterations to avoid expensive control logic, and as a result as the K values decrease, the MEA unit runs for a longer period of time. This is why the benefits of the recoding decrease as the K value increases, when compared to the previous work. All further results will be compared to these new baseline values.

K Value	Previous	Recoded
239	3840	2850
237	3864	2920
233	3910	3084
229	3937	3280
225	4105	3508
221	4145	3768
217	4340	4060

Table 5.6. Cycle Counts for Decoding a Codeword

5.3. Pipelining

This section will detail the numerical results of pipelining, both small-scale and global. In addition, the development and analysis of both the Mastrovito and Paar multipliers with varying degrees of pipelining will be detailed here, despite the fact that they were used sparsely in the final designs.

5.3.1. Galois Field Multipliers

As was described in section 4.2.2.1 several efforts were attempted to reduce the energy consumption characteristics of the GF multipliers, which are used in large numbers throughout the design of the RS decoder.

Two separate implementations of the multiplier circuitry were developed, one using the original Mastrovito[18] structuring, and one using the structure suggested by Paar in [19]. Each of these was examined both in combinational form, and with pipelining between one and four stages. The results are shown below in Table 5.7. The most efficient, the 2 stage Mastrovito multiplier, was adopted for use in the MEA unit, along with the Chien and Forney units.

	LUTs		Regs		Power Consumption	
Pipeline Stages	Mastrovito	Paar	Mastrovito	Paar	Mastrovito	Paar
0	58	53	0	0	2.75	3.54
1	58	53	8	8	2.09	2.25
2	58	53	47	26	**1.63**	1.95
3	92	53	92	40	2.23	2.47
4	97	72	97	72	2.58	2.68

Table 5.7. Pipelined Galois Field Multiplier Results

5.3.2. Small-Scale Pipelining

When examining the design for opportunities to pipeline within functional units, the only one which stood out as providing the opportunity for energy savings was the MEA unit. Based on the amount of pipelining in the GR multipliers in the unit, we have the opportunity to pipeline the unit with between 3 and 5 stages (see section 4.2.2.1 and Figure 4.6). The analysis of the MEA unit versions are shown in Table 5.8 below.

Pipelining Stages	LUTs	Regs	Power @ 50Mhz (mW)
3	783	326	12.60
4	796	482	12.17
5	810	646	9.69

Table 5.8 MEA Unit Comparison

As the results in Table 5.8 indicate, the optimal version uses five pipeline stages, and the overall results of making this change are detailed in Table 5.9. The general trend illustrated by these results is of a larger reduction in energy consumption for the larger decoders. This result makes sense as the larger decoders spend a larger percentage of their decoding time using the MEA unit. As a note, there is an increase in the power dissipation of the EC unit when changing from 3 to 5 pipeline stages. As the EC unit directly follows the MEA unit, it is assumed that the synthesizer is moving logic around to optimize. The net effect is shown in Table 5.9, and in general is a reduction of about 0.5 mW.

K Value	LUTs	Regs	Memory Bits	Fmax (MHz)	Pwr @ 50 MHz (mW)	Period (us)	E/Mb (J)	Change
239	4889	2513	30447	108.34	59.45	57.00	1.86E-03	0.85%
237	5316	2705	30535	107.97	60.16	58.40	1.94E-03	1.04%
233	6231	3089	30711	107.22	61.69	61.68	2.14E-03	0.88%
229	7087	3474	30887	108.73	63.09	65.60	2.37E-03	0.90%
225	7964	3858	31063	103.85	64.45	70.16	2.63E-03	1.59%
221	8904	4253	31239	106.87	66.41	75.36	2.97E-03	1.50%
217	9739	4637	31415	109.49	67.65	81.20	3.32E-03	2.06%
							Average	1.26%

Table 5.9. Small-Scale Pipelining Results. Change values are with regard to Table 5.4.

5.3.3. Global Pipelining

The original decoders [7] processed a single codeword at a time. As described in Section 4.2.2.2, a second codeword is not fed into the erasure generator until the decoder has completely finished processing the previous codeword. Even though the erasure generator finishes processing after 41 us, another codeword is not started until the decoder is finished processing, 15 to 40 us later. As a result, the design was modified to start a new codeword as soon as the erasure generator finishes processing the previous codeword (Section 4.2.2.2). This more efficient use of the available processing resources allows for a throughput across all of the decoders of 40.98 us per codeword, the latency of the erasure generator. The resulting energy reduction results versus the results in Table 5.9 are detailed in Table 5.10. Table 5.11 illustrates the clock cycles per codeword for each decoder before and after.

K Value	LUTs	Regs	Memory Bits	Fmax (MHz)	Pwr (mW)	Period (us)	E/Mb (J)	Change
239	4889	2513	30447	108.34	59.67	40.98	1.34E-03	27.84%
237	5314	2715	30535	107.97	60.38	40.98	1.37E-03	29.57%
233	6231	3089	30711	107.22	61.94	40.98	1.43E-03	33.29%
229	7089	3474	30887	108.73	63.38	40.98	1.49E-03	37.24%
225	7963	3858	31063	103.95	64.77	40.98	1.55E-03	41.30%
221	8904	4253	31239	106.87	66.76	40.98	1.62E-03	45.33%
217	9739	4637	31415	109.49	68.02	40.98	1.68E-03	49.26%
							Average	37.69%

Table 5.10. Global-Pipelining Results, compared to Table 5.9.

	Clock Cycles per Codeword	
K Value	Original	Global Pipelined
239	2850	2049
237	2920	2049
233	3084	2049
229	3280	2049
225	3508	2049
221	3768	2049
217	4060	2049

Table 5.11. Clock Cycles per codeword before and after global pipelining

5.4. Memory Optimizations

There are three major memory units in the adaptive RS decoding unit (see figure 4.3). There is one memory which holds a full codeword's worth of 10-bit data values from the A/D converter unit outside the FPGA, a total of 20,480 bits. There are also two memories that each holds a full codeword of data, one that stores the output of the erasure generator, and one stores the uncorrected codeword while the decoder processes it to determine the correction vectors. Both of these memories are of size 2040 bits.

As the erasure generator works on eight 10-bit values at a time, representing on 8-bit symbol which was received from the channel, the optimal implementation is to pack each set of 80 bits into one read and write. This would reduce the total required number of reads by a factor of 8. The results of reading at different rates are illustrated in Table 5.12 below.

Reading Scheme	Power (mW)
8 x 10	22.19
4 x 20	11.23
2 x 40	5.62
1 x 80	2.89

Table 5.12. Power Consumption Results of Memory Buffering
Of 20,400 bit Memory Units Using M4K Blocks

The other two large memories in the design use 8-bit data values, so the natural choice to make use of all of the physical circuitry available, is to make each read and write 32 bits exactly. The benefits are shown in Table 5.13 below.

Reading Scheme	Power (mW)
4 x 8	3.93
2 x 16	3.08
1 x 32	2.06

Table 5.13. Power Consumption Results of Memory Buffering
2040 bit Memory Units

The overall system effects of performing these optimizations are shown in Table 5.14. The benefits are greater for the smaller decoders as the memory units consume a larger percentage of the overall power in the smaller decoders. The size and activity of the memory units do not vary between decoders.

K Value	LUTs	Regs	Memory Bits	Fmax (MHz)	Pwr @ 50 Mhz (mW)	Period (us)	E/Mb (J)	Change
239	5082	2591	30447	120.55	37.65	40.98	8.46E-04	36.90%
237	5502	2771	30535	117.66	38.73	40.98	8.78E-04	35.86%
233	6428	3155	30711	118.39	40.39	40.98	9.31E-04	34.79%
229	7299	3552	30887	116.04	41.76	40.98	9.80E-04	34.11%
225	8174	3924	31063	120.48	43.61	40.98	1.04E-03	32.67%
221	9121	4319	31295	122.19	46.10	40.98	1.12E-03	30.95%
217	9970	4703	31479	125.57	48.37	40.98	1.20E-03	28.89%
							Average	33.45%

Table 5.14. Results of Memory Optimizations, compared to Table 5.10.

5.5. Clock Gating

Figure 5.2 illustrates the various clock domains used to clock gate design functional units. Each of the units, Unit1, Unit2, MEA, and the Error Correction unit, receives its own gated clock signal.

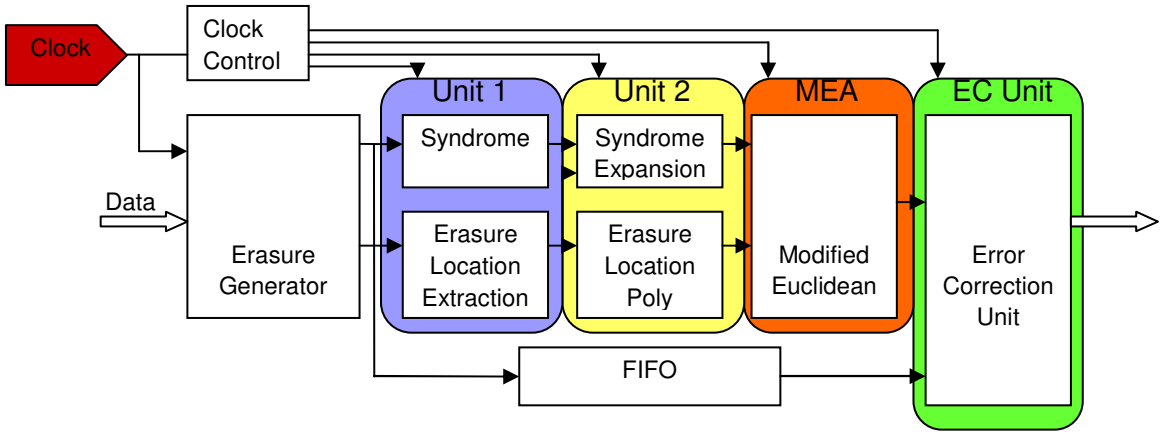


Figure 5.2. Global Clocking Scheme

The colored blocks illustrate the boundaries of different clock gating domains. As the erasure generator is the bottleneck in the system, it is always active and thus no gating is necessary. The other units are Unit 1, Unit 2, MEA unit, and the Error Correction unit, consisting of the Chien and Forney blocks. Each of these units was given its own clock enable signal, and this signal was used to enable the clock just before the unit is needed for processing. The clock is shut off after processing is finished. Table 4.1 illustrates the number of clock cycles that each unit is active before and after clock gating was applied.

The overall system benefits of this clock gating are shown in Table 5.15 below, while Figure 5.3 illustrates the incremental benefits of each of the techniques that were used in this work, while Table 5.16 illustrates the full results in numerical form.

K Value	LUTs	Regs	Memory Bits	Fmax (MHz)	Pwr (mW)	Period (us)	E/Mb (J)	Change
239	5120	2644	30408	124.52	19.48	40.98	4.38E-04	48.26%
237	5552	2836	30496	111.26	19.77	40.98	4.48E-04	48.95%
233	6466	3208	30672	124.18	20.96	40.98	4.83E-04	48.11%
229	7325	3593	30848	121.71	21.97	40.98	5.15E-04	47.39%
225	8225	3989	31024	118.85	22.74	40.98	5.43E-04	47.86%
221	9159	4372	31256	114.93	24.17	40.98	5.87E-04	47.57%
217	10007	4756	31440	117.23	24.57	40.98	6.08E-04	49.20%
							Average	48.19%

Table 5.15. Final Results after Clock Gating, compared to table 5.14.

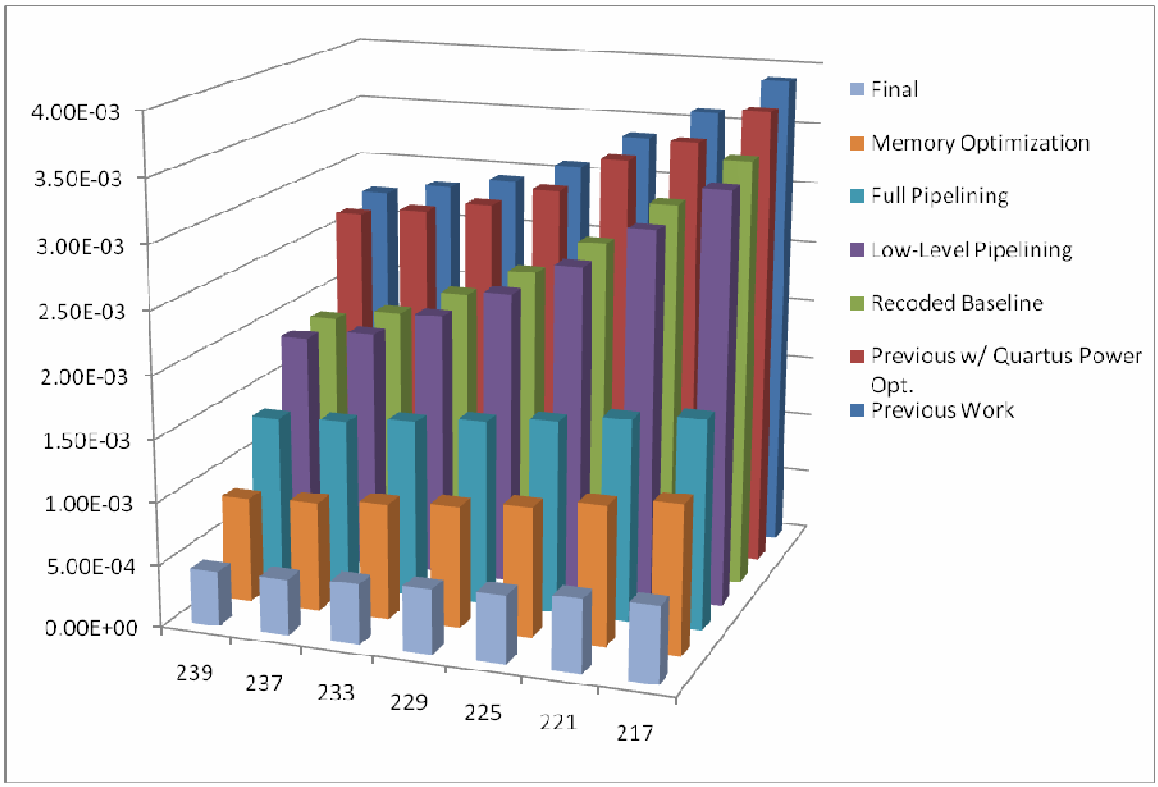


Figure 5.3. Full Incremental Energy per Operation Results Breakdown. Values are in J/Mb

K Value	Recoded (Baseline)	Low-Level Pipelining	% Reduct.	Global Pipelining	% Reduct.	Memory Opt	% Reduct.	Glock Gating (Final)	% Reduct.	Cumulative Reduction
239	1.89E-03	1.86E-03	1.47%	1.34E-03	27.43%	8.46E-04	26.24%	4.38E-04	21.65%	76.79%
237	1.99E-03	1.94E-03	2.12%	1.37E-03	28.95%	8.78E-04	24.72%	4.48E-04	21.65%	77.43%
233	2.19E-03	2.14E-03	2.20%	1.43E-03	32.56%	9.31E-04	22.70%	4.83E-04	20.46%	77.92%
229	2.42E-03	2.37E-03	2.09%	1.49E-03	36.46%	9.80E-04	20.96%	5.15E-04	19.18%	78.70%
225	2.70E-03	2.63E-03	2.41%	1.55E-03	40.31%	1.04E-03	18.72%	5.43E-04	18.46%	79.89%
221	3.05E-03	2.97E-03	2.74%	1.62E-03	44.09%	1.12E-03	16.45%	5.87E-04	17.47%	80.75%
217	3.44E-03	3.32E-03	3.41%	1.68E-03	47.58%	1.20E-03	14.16%	6.08E-04	17.15%	82.30%
Avg.			**2.35%**		**36.77%**		**20.56%**		**19.43%**	**79.11%**

Table 5.16. Final Results, in Energy (J) per Mb of Message Data Reduction in relation to Recoded baseline values (table 5.4.)

5.6. Summary

This section detailed the results of performing energy optimization techniques on the set of Reed-Solomon errors-and-erasures decoders. A new baseline was generated by recoding the designs to be efficient in terms of clock cycles. Using this recoded version of the designs as a new baseline, low-level pipelining was found to provide on average a 2.35% reduction in energy consumption. Global pipelining was found to provide a benefit of 36.77%, while memory optimizations yielded a reduction in energy consumption of 20.56%. Lastly, clock gating of the major functional units provided a reduction of another 19.43%. On the whole, the energy per megabit of data values were reduced by 76.8% to 82.3% over all of the designs, with the average reduction being 79.11%.

CHAPTER 6

RECONFIGURATION RATE ANALYSIS AND RESULTS

This chapter provides numerical results related to the reconfiguration scheduling, along with full system results for the new adaptive Reed-Solomon decoding system. There were several questions which needed to be answered though this analysis, namely, given a more realistic channel model, how does the rate of reconfiguration affect the energy consumption, and how does the rate of reconfiguration affect the codeword error rate. In addition, we will illustrate the benefits of reconfiguration on the overall decoding rate.

6.1. New Configuration Table

Using the final energy consumption values from Tables 5.12 and 5.13, we can construct a table of different decoder configurations based on SNR (Table 6.1). The last column illustrates the energy efficiency benefit of each decoder compared to having a static (non-reconfigurable) K=217 decoder.

K Value	SNR Range (dB)	Mbps	LUTs	Regs	E/Mb Data (J)	Benefit over Static K=217
239	19.6 +	44.50	5120	2644	4.38E-04	28.01%
237	19.0-19.6	44.12	5552	2836	4.48E-04	26.33%
233	17.6-19.0	43.38	6466	3208	4.83E-04	20.55%
229	16.4-17.6	42.63	7325	3593	5.15E-04	15.27%
225	15.6-16.4	41.89	8225	3989	5.43E-04	10.74%
221	14.8-15.6	41.14	9159	4372	5.87E-04	3.41%
217	14.0-14.8	40.40	10007	4756	6.08E-04	----------

Table 6.1. Configuration Table

The SNR ranges for each decoder match the values in [7], which were verified via simulation. In order to determine these values, simulations were run for 10 million codewords with a static SNR (no shadowing), with SNR values from 13 to 21 dB for each decoder. The value where the CER becomes 10^{-4} gives the bottom of each decoder's applicable range. The top of each decoder's range is assigned to the bottom of the next decoder's range.

6.2. CER Analysis

As mentioned in Section 4.1, a simulator was built to determine the performance characteristics that could be expected from the adaptive decoder system. The first parameter which needed to be explored was to see how the reconfiguration rate affects the CER. Previous work [7] assumed that there was no change in the average SNR (the SNR due to shadowing) during the time that a particular decoder was operating. This assumption leads to a somewhat unrealistic representation of real world performance.

As one of our parameters is to keep a static CER rate of 10^{-4}, we must ensure that the variance in the channel between reconfigurations does not reduce the CER below this threshold. In order to determine this, reconfiguration rates from every 5,000 codewords to every 100,000 codewords were tested. Initially the goal was to test reconfiguration rates up to and exceeding the previous work's 125,000 codewords, but as the results illustrate, testing reconfiguration rates that high proved unnecessary.

For these simulations, a total of 10 million codewords were run through the system for each simulation. After the designated amount of codewords had been simulated (5,000 – 100,000), the simulator makes a decision on whether to reconfigure

based on the average SNR during the previous run. If the average SNR is outside of the current decoder's specified range, then the system is reconfigured to insert a decoder which is designed to operate in the current measured range.

Several separate runs were made and the results averaged to eliminate some inherent variance in the system. Figure 6.1 shows a graph of the results and the trend line generated from these results.

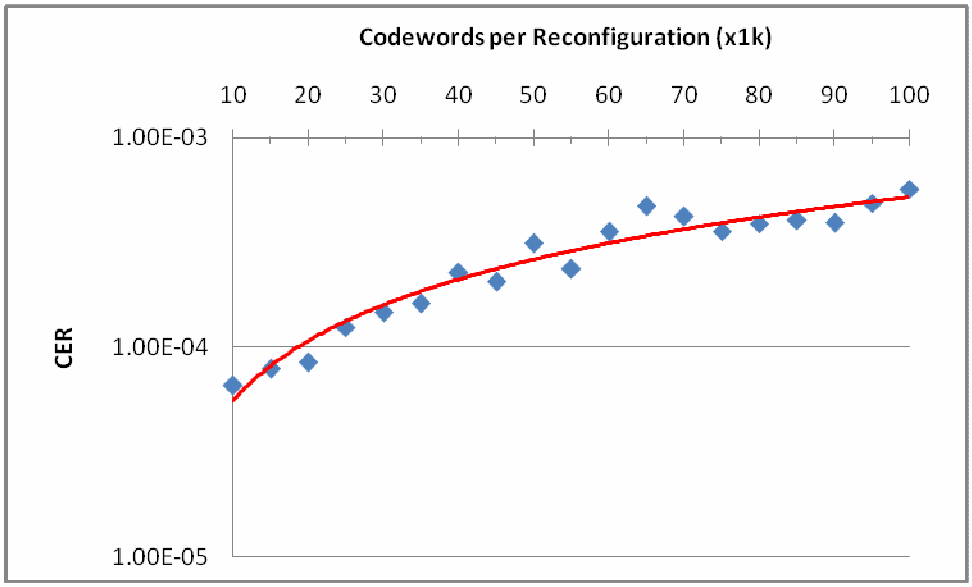


Figure 6.1. Graph of CER vs. Codewords per Reconfiguration

By examining the trend line, we can see that the point at which we reach the desired 10^{-4} threshold is at approximately 18,000 codewords between reconfigurations.

6.3. Energy Efficiency Results

If the total number of reconfigurations, the frequency of use for each decoder and the energy consumption rates of each decoder are known, it is possible to determine the benefit of reconfiguration versus the continuous use of a static K217 decoder. To benefit from reconfiguration, the system must realize an energy savings versus the use of a static decoder. Thus, the energy cost of reconfiguration must be lower than the savings due to using a series of energy efficient decoders.

For these simulations, it was assumed that the system has access to only one clock, running at 50 MHz. The time required to reconfigure was determined from the Stratix Data Sheet [33], assuming configuration in FPP mode, where the configuration data is loaded a byte per clock cycle. In addition to some initialization overhead, it was determined that it takes 8.92ms to load the 3,534,640 configuration bits. During the reconfiguration process, it was determined that a 4Mx32 Micron RAM unit, which holds the configuration data, would dissipate approximately 189 mW. This is based on the maximum power consumption listed in the Micron 4Mx32 data sheet [34], scaled down from 166 MHz to our required 50 MHz. The FPGA's power dissipation was modeled as an 8-bit shift chain, of length 441,830. Previous work in this area [35] had determined the power required for a single shift by modeling a 0.13u shift register in SPICE. This result was modified to assume an 8-bit shift chain as opposed to a single bit, and the result was that the FPGA is expected to dissipate 215 mW of power during the reconfiguration process. This gives us reconfiguration parameters of 8.92 ms and a total power dissipation of 404 mW.

Figure 6.2 illustrates the results of this analysis.

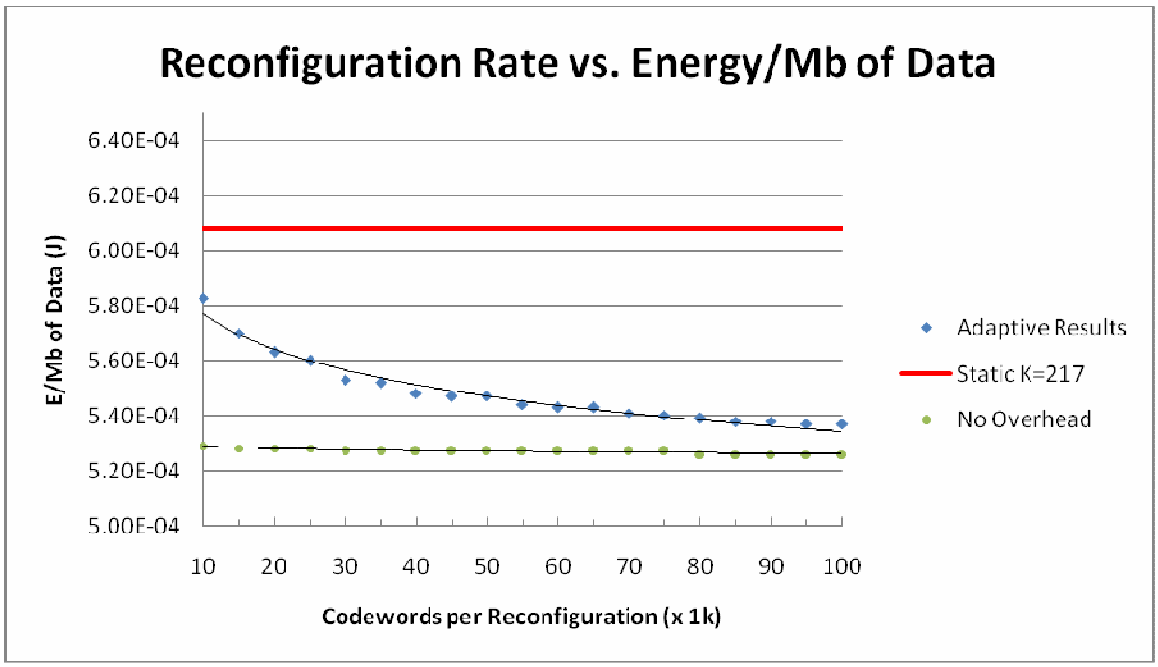


Figure 6.2. Energy per Megabit vs. Codewords per Reconfiguration

As this figure illustrates, reconfiguration allows for us to save from between 4% and 12% in terms of energy per megabit of data, when compared to the static K=217 decoder. As we detailed above, in order to maintain the required CER of 10^{-4}, we need to reconfigure approximately every 18,000 codewords. Based on the above figure, reconfiguring every 18,000 codewords provides a benefit in energy per megabit performance of 6.93%.

The third line in the graph illustrates the benefits of reconfiguring if we do not account for the energy cost of reconfiguration. This serves to illustrate that if the cost of reconfiguration could be reduced further, benefits of up to 13% could be achieved. This is an area of possible future work.

6.4. Decoding Rate Results

An additional benefit of reconfiguration is that we get an increase in effective decoding rate. By making use of a higher K decoder, as opposed to using the K=217 decoder, we reduce the amount of redundancy in each codeword. Because the codewords are all of a fixed length, this effectively increases the amount of data we are processing with each codeword, increasing the overall decoding rate. The effects of reconfiguration rate on decoding rate are detailed in Figure 6.3 below.

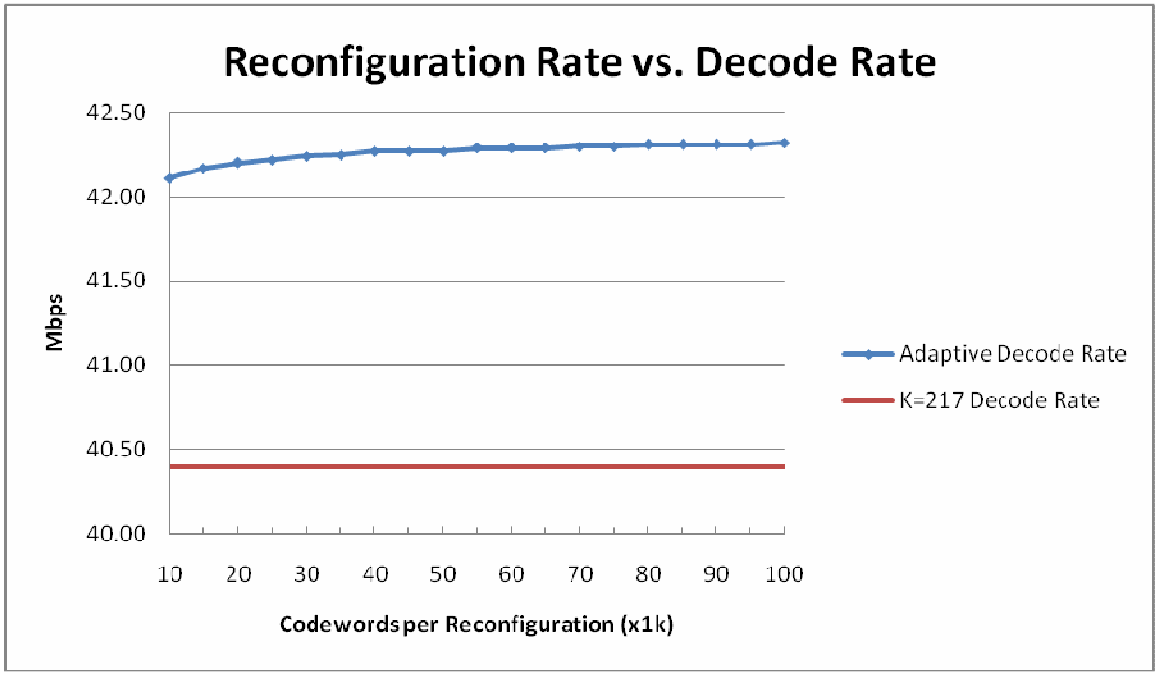


Figure 6.3. Reconfiguration Rate vs. Decode Rate

The increase in decoding rate ranges from 4.23% to 4.75% depending on K value. At our operating rate of reconfiguring every 18,000 codewords, the increase in decoding rate is approximately 4.43%, increasing from 40.40 Mbps to 42.19 Mbps.

6.5. Summary

In this chapter we detailed the results of performing an analysis of the effects of varying rates of reconfiguration on three important parameters of the adaptive Reed-Solomon decoding system. We showed that in order to maintain the minimum codeword error rate of 10^{-4}, we need to reconfigure at a rate no greater than 18,000 codewords. When examining energy consumption, it was shown that using current assumptions of the energy required to reconfigure the FPGA, when reconfiguring every 18,000 codewords, we see a reduction in the energy required to decode a megabit of data from 6.09×10^{-4} per megabit when using the static K=217 decoder to 5.66×10^{-4} J per megabit when using the adaptive system, a reduction of 6.93%. If the cost of reconfiguration could be further reduced, benefits of up to 13% could be achieved. In concert with this energy benefit, reconfiguring also increases the effective decoding rate by reducing the amount of redundancy in each codeword. When reconfiguring at 18,000 codewords, we see an increase over the static K=217 system from 40.40 Mbps to 42.19 Mbps, an increase of 4.43%.

REFERENCES

[1] Lin, Shu, and Costello, Jr. *Error Control Coding: Fundamentals and Applications.* Prentice Hall, 1983

[2] Reed, I. S., and Solomon, G. Polynomial Codes Over Certain Finite Fields. *SIAM J. Appl. Math., Vol 8, No 2, pp. 300-304* (June 1960)

[3] Blahut, R. E. *Theory and Practice of Error Control Codes.* Addison Wesley, 1984

[4] Hoeve, H., Timmermans, J., Vries, L. B. Error correction and concealment in the Compact Disc system. *Philips Tech. Rev. Vol. 40, no. 6, pp. 166-172.* (1982)

[5] Forney, G. D. The Viterbi Algorithm*, Proceedings of the IEEE, Vol. 61, Issue 3, pp. 268-278* (March 1973)

[6] Berrou, C., Glavieux, A., Thitimajshima, P. Near Shannon limit error-correcting coding and decoding: Turbo-codes*, ICC, Volume 2, pp. 1064-1070* (May 1993)

[7] Atieno, L., Allen, J., Goeckel, D., and Tessier, R., An Adaptive Reed-Solomon Errors-and-Erasures Decoder, *in the Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, CA (February 2006)

[8] Wilton, S., Ang, S., Luk, W. The Impact of Pipelining on Energy Per Operation in Field Programmable Gate Arrays, *Lecture Notes in Computer Science, Vol. 3203/2004, pp. 719-728* (2004)

[9] Emnett, F., Beigel, M., Power reduction through RTL clock gating, *Synopsys Users Group, San Jose, CA,* (2000)

[10] Choi, S., Scrofano, R., Prasanna, V. K., Jang, J. Energy-efficient signal processing using FPGAs, *Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays, pp. 225 -234,* (2003)

[11] Shannon, C. E. A Mathematical Theory of Communication. *Bell Syst. Tech. Journal, vol27, pp. 379-423(Part I), 623-656(Pat II).* (July 1948)

[12] Haase, A., Boden, M., Langer, M. Design of a Reed Solomon Decoder Using Partial Dynamic Reconfiguration of XILINX VIRTEX FPGAs – A Case Study. *Design, Automation and Test in Europe, Paris (France),* (March 2002)

[13] Song, M. K., Kim, E. B., Won, H. S., Kong, M. H., Architecture for Decoding Adaptive Reed-Solomon Codes with Variable Block Length. In *IEEE Transactions on Consumer Electronics, vol. 48, No. 3,* (August 2002)

[14] Lee, D., Lee, S., Kim, J. A Reed-Solomon Decoder with Efficient Recursive Cell Architecture for DVD Application. *IEEE International Conference on Consumer Electronics, pp. 184-185,* (2001)

[15] *Run-Time Dynamically Reconfigurable Reed-Solomon Decoder System*, A Masters Thesis by Lilian Atieno, Umiversity of Massachusetts Amherst, November 2004

[16] Hauck, S. The Roles of FPGAs in Reprogrammable Systems. *In Proceedings ofIEEE, pp. 615 – 638,* (April 1998)

[17] Kavian, Y. S., Falahati, A., Khayatzadeh, A., Naderi, M., High Speed Reed-Solomon Decoder with Pipeline Architecture, *Iran University of Science & Technology (IUST), Narmak, Tehran, Iran* (2005)

[18] Mastrovito, E. D., VLSI Design for Multiplication over Finite Fields GF(2^m). *Proc. of Sixth International Applied Algebra, Algebraic Algorithms, and Error Correcting Codes, pp. 297-309* (July 1988)

[19] Paar, C. A New Architecture for a Parallel Finite Field Multiplier with Low Complexity based on Composite Fields. *IEEE Transactions on Computers, 45(7):856-861* (July 1996)

[20] www.opencores.org

[21] Chang, H., Lin, C., Lee, C., A low-power Reed-Solomon decoder for STM-16 optical communications. *IEEE Asia-Pacific conference on ASIC, pp. 351-354* (2002)

[23] CD-ROM Technical Summary, *pauillac.inria.fr/~lang/hotlist/cdrom/Documents/tech-summary.html*

[24] Space Communications Protocol Standards, *http://www.scps.org/*

[25] Schoner, B., Villasenor, J., Molloy, S., Jain, R., Techniques for FPGA Implementation of Video Compression Systems. *ACM FPGA, pp154-159* (1995)

[26] Cummings, M., Haruyama, S., FPGA in the software radio. *IEEE Communications Magazine, Volume 37, Issue 2, pp. 108-112, Feb 1999*

[27] Apple Iphone Technical Specifications, *http://www.apple.com/iphone/specs.html*

[28] Reed, I.S., Shih, M.T., VLSI design of inverse-free Berlekamp-Massey algorithm. *IEE Proceedings on Computers and Digital Techniques, Vol. 138, Issue 5, pp. 295-298, Sep. 1991*

[29] Lee, H., Azam, A., Pipelined recursive modified Euclidean algorithm block for low-complexity, high-speed Reed-Solomon decoder, *Electronics Letters, Vol. 39, Issue 19, 18 Sept 2003*

[30] Bellas, N., Hajj, I.N., Polychronopoulos, C.D., Stamoulis, G., Architectural and compiler techniques for energy reduction in high-performance microprocessors, *IEEE Transactions on VLSI Systems, Volume 8, Issue 3, pp. 317-326, Jun 2000*

[31] Quartus II Development Software Library, *http://www.altera.com/literature/lit-qts.jsp*

[32] Jean, J., Tomko, K., Yavagal, V., Shah, J., Cook, R., Dynamic reconfiguration to support concurrent applications, *IEEE Transactions on Computers, Vol. 48, Issue 6, pp.591-602, Jun 1999*

[33] Altera Stratix Device Handbook, *http://www.altera.com/literature/lit-stx.jsp*

[34] Micron MT48LC4M32B2B5 SDRAM data sheet, *http://www.micron.com/products/partdetail?part=MT48LC4M32B2B5-6*

[35] J. Liang, R. Tessier, and D. Goeckel, A Dynamically-Reconfigurable, Power-Efficient Turbo Decoder, *in the Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa, California, April 2004