## INFORMATION TO USERS

This dissertation copy was prepared from a negative microfilm created and inspected by the school granting the degree. We are using this film without further inspection or change. If there are any questions about the content, please write directly to the school. The quality of this reproduction is heavily dependent upon the quality of the original material.

The following explanation of techniques is provided to help clarify notations which may appear on this reproduction.

1. Manuscripts may not always be complete. When it is not possible to obtain missing pages, a note appears to indicate this.

2. When copyrighted materials are removed from the manuscript, a note appears to indicate this.

3. Oversize materials (maps, drawings and charts are photographed by sectioning the original, beginning at the upper left hand corner and continuing from left to right in equal sections with small overlaps.

UMI®

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

# IMPLEMENTATION OF A MULTIPLIERLESS BINARY ARITHMETIC ENCODER

GILBERTO ISAAC SADA

Electrical and Computer Engineering Department

APPROVED:

Dr. Bryan Usevitch, Chair

Dr. Sergio D. Cabrera

Dr. Vladik Kreinovitch

Associate Vice President for
Graduate Studies

A mis padres, mi hermano y mis amigos.

# IMPLEMENTATION OF A MULTIPLIERLESS BINARY ARITHMETIC ENCODER

by

GILBERTO ISAAC SADA, B.S.E.E.

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

In Partial Fulfillment

Of the Requirements

For the Degree of

MASTER OF SCIENCE

Electrical and Computer Engineering Department

THE UNIVERSITY OF TEXAS AT EL PASO

December, 1999

# Acknowledgements

I would like to express my gratitude to Dr. Bryan Usevitch, Dr. Sergio Cabrera, and Dr. Michael Austin for their guidance throughout my career. I would also like to thank Dr. Vladik Kreinovitch for being part of my Supervising Committee.

I'm indebted to Carlos Betancourt for helping me learn to configure and test the Constellation board. I would like to thank CONEXANT, Inc. and my manager, Javier Avila, for allowing me to further my education and for their support.

I thank my family, the Signal Processing and Communications Affinity group, and some very special friends for their continuous moral support.

This work was supported partially by the NASA Faculty Awards for Research (FAR) contract number 961119. This thesis was submitted to my Supervising Committee on November 1, 1999.

# Abstract

Implementations of a multiplierless binary arithmetic encoder and decoder are presented in this thesis. Two rapid prototyping approaches are adopted. In the first approach, the Signal Processing WorkSystem (SPW) is used to design and develop the encoder and decoder systems while in the second approach Verilog Hardware Description Language (HDL) code is used to describe both systems. Due to propagation delay problems not taken into consideration by the SPW software, only the Verilog HDL system descriptions were successfully ported and tested in an Altera Field Programmable Gate Array (FPGA) chip. The implementations are based on the Rissanen and Mohiuddin arithmetic encoder and decoder [1]. The encoder and decoder operations are based on a 4-bit augend "A" register for simplicity. However, a more standard 16-bit implementation would yield considerable better compression results. The main contribution of this thesis is the development of a practical *bit stuffing* technique that solves the *carry over* problem inherent in most practical arithmetic encoders. The encoder and decoder were tested on a 512 X 512 black and white Lena image. The encoder successfully compressed the file to less than half its original length.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Lossless and Lossy Data Compression

The purpose of a *communication system* is to disseminate and store *information* through the use of *messages*. Words, images, or a stream of numbers are some of the many ways through which messages can be communicated. Transmission and storage of messages cost money. Such costs are directly proportional to the size of the message. The costs of information handling can be lowered if message size is reduced by dispensing some of its *redundancy*. Reduction of message redundancy is known as *data compression*. The field of data compression is therefore dedicated to the representation of information sources through messages with the least possible redundancy.

Data compression can be classified in two broad subgroups: *lossless* data compression and *lossy* data compression. In lossless data compression a message is reversibly transformed into a less redundant compact representation. Since the

transformation is reversible, it is possible to obtain the exact original message from its compact representation. Lossy data compression is dedicated to the search of methods that in general achieve superior data compression, as compared to lossless compression schemes, at the expense of information. Since information is lost when using a lossy compression scheme, the exact original message can no longer be recovered. The difference between the original message and the reconstructed message is known as *distortion.* Lossy compression methods strive to obtain as much compression as possible while keeping distortion to a minimum.

The approach selected to represent a message with the least redundancy will depend on the application. Using lossy data compression methods to represent a computer file or data stream may be disastrous, as the application requires the method to be completely reversible. However, an image can endure certain amount of distortion while still being intelligible. When using a lossy compression method to compress an image, compression results are usually far superior to the ones achieved by a lossless compression scheme, while distortion may be imperceptible.

As a motivation for the study of data compression consider its application in the World Wide Web. Suppose an image file is to be downloaded into a computer for display. Furthermore, suppose a modem is available for image download. The computer connects to a remote server through the modem at a 33.6Kbps rate. Suppose such a rate is maintained through out the download without interruptions. If an image is 1M in size and is not compressed, the 33.6Kbps modem would download the image in about 4 minutes. If the image is losslessly compressed, it can be downsized to 500K, and the download

time would be 2 minutes. If however the image is lossy compressed, it can be reduced to 83K, and it will take 20 seconds to download. Using the same example, it requires less hard drive space to store a 1M file than an 83K file. Hard drive space costs money; obviously a 2G hard drive is cheaper than a 10G hard drive. Internet time costs money as well. This and other applications have been the driving motivation for better data compression methods in recent years.

As a Pre-Processing stage, typical lossy compression methods use the Discrete Cosine Transform or the Wavelet Transform. The result from the Pre-Processing stage is lossy compressed by a *Quantizer*. A lossless compression stage may be used as the final stage of a lossy compression algorithm as in the case of JPEG. Among the most important lossless data compression approaches are the dictionary techniques and entropy coding. ZIP files are generated through the use of dictionary techniques. JPEG, and FAX use lossless entropy coding in conjunction with lossy compression to remove redundancy. Entropy coding includes the Shannon-Fano encoder, the Huffman encoder, and the family of arithmetic encoders. Arithmetic encoders perform better in general than all other entropy coders. In fact, many current state-of-the-art approaches for lossless data compression are realizations of arithmetic encoding.



## 1.2 Research Objectives

The main objective of this thesis is the implementation of a multiplierless binary

arithmetic encoder and decoder through the use of rapid prototyping. Computer Aided Design (CAD) tools provide the necessary elements to quickly develop and implement algorithms in the current competitive market.

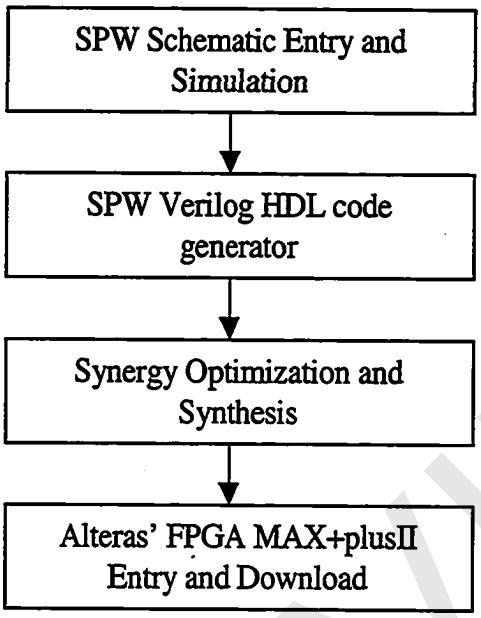


Figure 1.1: Rapid Prototyping first approach.

Two rapid prototyping approaches are presented. The first approach uses the Signal Processing WorkSystem (SPW) CAD graphical environment. SPW provides the necessary tools to simulate and benchmark typical digital designs. In addition, it contains libraries and ready to use optimized hardware blocks to simplify the design workload. SPW supports Verilog Hardware Description Language (HDL) code generation. The HDL code generator is used to quickly port a design from SPW to a standard format. Synergy software is then used to optimize Verilog HDL code and synthesize it to EDIF

format. Finally, Alteras' MAX+plusII software is used to download the optimized design into a 10K Altera Field Programmable Gate Array (FPGA) chip. The process is illustrated in Fig. 1.1.

The second approach uses the fact that Verilog HDL is itself a rapid prototyping tool. The process starts by developing a model of the device using C source code. Once the source code is tested and debugged, direct manual entry of Verilog HDL code, based on the C source code, is done. The Verilog code can be further simulated and debugged if desired. Although Verilog coding rules and C coding rules are very similar, some restrictions apply. Altera does not support some Verilog commands. For example, Altera does not support loops and alternative approaches to looping must be developed. Once the Verilog code is completed, Alteras' MAX+plusII software can generate a symbol for the design. Alteras' software can then compile, synthesize, and simulate the symbol. Finally, the design can be downloaded into a 10K Altera FPGA as described by Fig. 1.2.

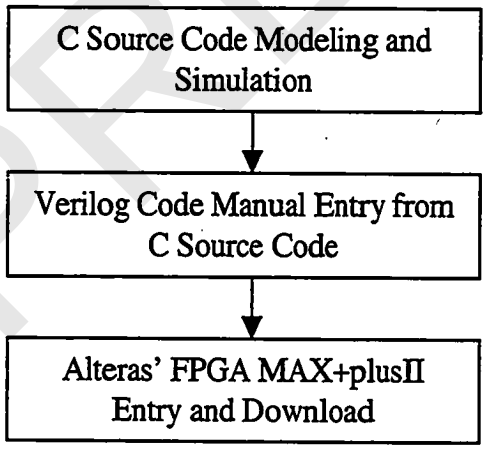


Figure 1.2: Rapid Prototyping second approach.

Rapid prototyping using an FPGA chip provides enough flexibility to quickly test and develop digital system applications. In addition, rapid prototyping designs allow quick and inexpensive upgrades. Development methods using classical Application-Specific Integrated Circuit (ASIC) designs are in general superior as far as power consumption, chip density, and performance speed. Rapid prototyping however is gaining popularity over ASIC designs due to its flexibility and quicker development. Current market interests aim for cheap, high performance FPGAs and programmable DSPs.

To understand the implementation of a binary arithmetic encoder and decoder a discussion on general arithmetic encoding is provided in Chapter 2. Following this, the background obtained in Chapter 2 is applied in Chapter 3 to understand binary multiplierless arithmetic encoders and decoders. In particular, the Rissanen and Mohiuddin arithmetic encoder is discussed at the end of Chapter 3. Chapter 4 introduces a particular bit stuffing implementation for the Rissanen and Mohiuddin binary arithmetic encoder. In particular a problem with classical bit stuffing approaches is explained and a solution is contributed in Section 4.2. Chapter 5 presents an implementation of a binary arithmetic encoder and decoder using SPW. This implementation was not successfully downloaded into an Altera chip due to propagation delay problems not considered by SPW software. However, the implementation was successfully simulated and evaluated with SPW simulation tools. Chapter 6 discusses the implementation of the binary arithmetic encoder and decoder using manual entry of Verilog code. Such implementation was successfully downloaded and tested in an Altera 10K FPGA. In Chapter 7 some conclusions are drawn from the designs and future work is suggested.

# Chapter 2

# Arithmetic Encoding

## 2.1 Background

All source codes can be classified as either block codes or non-block codes [7]. Particularly, block codes concatenate individual symbol codewords to represent a message. Among block codes, prefix codes have the property that no symbol codeword is a prefix of another one. Examples of prefix codes include the well-known Huffman and Shannon-Fano compact codes. Compact codewords are constructed based on symbol statistics. In general, the more likely a symbol is the shorter its codeword will be. When compared to a message comprised of distinct symbol codewords of fixed length, compact codes yield a shorter message representation.

A coding scheme represents a message with minimal redundancy if its average symbol codeword length closely approaches the messages' *average entropy* defined as

$$H(X) = -\Sigma\, p(x_j) \log_2 p(x_j) \qquad \text{(bits/symbol)}, \qquad (2.1)$$

where X represents the message, $x_j$ represents a message symbol event, and $p(x_j)$ represents $x_j$'s probability estimation [12]. Equation (2.1) assumes that the units of the codeword are bits. A more general definition for average entropy can be found in Ziemer and Tranter [12].

Compact codes are optimal in condensing messages information if the symbol probabilities are powers of ½. However, if the symbol probabilities are not powers of ½, compact codes can yield a message average symbol codeword length larger than the message's average entropy. Such a difference between the average codeword length and the message average entropy can be significant if the symbol alphabet is small and the symbol probabilities are *skewed*. Consider a message X whose symbol alphabet is {a, b}. Furthermore, suppose symbol probabilities {p(a), p(b)} = {0.1, 0.9}. Equation (2.1) yields H(X) = .47 bits/symbol. In this example the optimal average symbol length is .47 bits/symbol. However, compact codes can at best represent {a, b} with {0, 1} or {1, 0} which yields an average symbol length of 1 bit/symbol. The message average entropy in this case is less than half the average symbol length.

Typically, alphabet extensions are used to minimize the difference between the average message entropy and the compact code average symbol length. However, alphabet extensions do not yield an optimal solution to compact code average length problems [11]. Non-block codes like Elias and enumerative codes yield average

codeword lengths close to message's average entropy without the need of alphabet extensions. Elias and enumerative codes are among the first known arithmetic codes [7]. However, such codes are not physically realizable since they require the existence of arbitrarily long registers.

The promising results of Elias and enumerative codes gave motivation for the development of physically realizable arithmetic codes [1]-[3],[5]-[8], and [11]. With the development of new realizable arithmetic codes, Rissanen and Langdon extended the definition of arithmetic coding to include some classical block codes and some "near-optimum" data compression non-block codes. In general, a code is called arithmetic if codewords are generated recursively by means of a probability model and certain arithmetic operations [7]. This definition is obviously fairly broad. Under this definition it is possible to consider Huffman coding as a subset of the arithmetic coding family as will be shown in Chapter 3.

## 2.2 Symbol Probability Density and Cumulative Distribution Functions

Arithmetic encoders consist of a *probability model unit* and a *coding unit*. The model unit assigns an event to each symbol in the message alphabet and estimates both the *probability density function* (PDF) and the *cumulative distribution function* (CDF) values for every symbol event. The coding unit then recursively estimates the message's

joint CDF value and range from the individual symbol event PDF and CDF values. The resulting message arithmetic codeword is given by a real number in the messages' joint CDF range as will be shown in next section. The coding rules are such that likely symbols do not significantly increase the size of the codeword while the opposite is true for unlikely symbols. Note there is a clear distinction between the model unit and the coding unit.

The probability model may be *fixed* in which case a look up table can be used to provide symbol PDF and CDF values. Fixed models require knowledge of message statistics prior to encoding. In addition, fixed models may not properly predict symbol statistics for messages that exhibit *non-stationary random* behavior. *Adaptive* models dispense the need for message statistics prior to encoding by calculating them during the coding process. Furthermore, such models can properly adjust to non-stationary random messages. Adaptive models range from simple symbol frequency counts to context dependent finite state machines that closely model message statistics. Moffat [4] discusses some adaptive probability models applied to arithmetic coding.

Suppose a complete message symbol alphabet $\{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \ldots, \alpha_n\}$ is defined and provided to the model unit. The model unit defines an order for the symbols and assigns the events $\{x_1, x_2, x_3, x_4, \ldots, x_n\}$ to the symbol alphabet. Suppose the model unit determines the PDF and CDF values as $\{p(x_1), p(x_2), p(x_3), p(x_4), \ldots, p(x_n)\}$ and $\{P(x_1), P(x_2), P(x_3), P(x_4), \ldots, P(x_n)\}$ respectively where

$$P(x_1) < P(x_2) < P(x_3) < P(x_4) < \ldots < P(x_n) = 1. \qquad (2.2)$$

Once PDF and CDF values are available for each alphabet symbol event $x_i$, the modeling unit assigns a real number $C(x_i)$, defined as the symbol codeword, to each symbol in the alphabet so that

$$P(x_{i-1}) \leq C(x_i) < P(x_i). \tag{2.3}$$

Note that there are no previous symbols to $x_1$. Therefore, $x_1$ is assigned a codeword $C(x_1)$ lower bounded with closure by 0 so that

$$0 \leq C(x_1) < P(x_1),$$

as shown in Fig. 2.1.

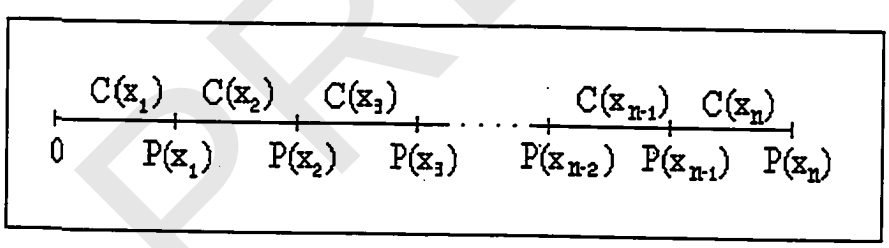


Figure 2.1: Symbol codeword ranges.

Consecutive symbol CDF ranges do not overlap by Equation (2.3). Therefore,