EFFICIENT DATA ENCODER FOR ENDOSCOPIC IMAGING APPLICATIONS

A Thesis Submitted to the

College of Graduate Studies and Research

in Partial Fulfillment of the Requirements

for the degree of Master of Science

in the Department of Electrical and Computer Engineering

University of Saskatchewan

Saskatoon

By

Ramin Tajallipour

PERMISSION TO USE

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Electrical and Computer Engineering

University of Saskatchewan

57 Campus Drive,

Saskatoon SK S7N 5A9

Canada

ABSTRACT

The invention of medical imaging technology revolved the process of diagnosing diseases and opened a new world for better studying inside of the human body. In order to capture images from different human organs, different devices have been developed. Gastro-Endoscopy is an example of a medical imaging device which captures images from human gastrointestinal. With the advancement of technology, the issues regarding such devices started to get rectified. For example, with the invention of swallow-able pill photographer which is called Wireless Capsule Endoscopy (WCE); pain, time, and bleeding risk for patients are radically decreased. The development of such technologies and devices has been increased and the demands for instruments providing better performance are grown along the time. In case of WCE, the special feature requirements such as a small size (as small as an ordinary pill) and wireless transmission of the captured images dictate restrictions in power consumption and area usage.

In this research, the reduction of image encoder hardware cost for endoscopic imaging application has been focused. Several encoding algorithms have been studied and the comparative results are discussed. An efficient data encoder based on Lempel-Ziv-Welch (LZW) algorithm is presented. The encoder is a library-based one where the size of library can be modified by the user, and hence, the output data rate can be controlled according to the bandwidth requirement. The simulation is carried out with several endoscopic images and the results show that a minimum compression ratio of 92.5 % can be achieved with a minimum reconstruction quality of 30 dB. The hardware architecture and implementation

result in Field-Programmable Gate Array (FPGA) for the proposed window-based LZW are also presented. A new lossy LZW algorithm is proposed and implemented in FPGA which provides promising results for such an application.

ACKNOWLEDGEMENTS

This research project would not have been possible without the support of many people. The author wishes to express his gratitude to his supervisor, Dr. Khan Wahid who was abundantly helpful and offered invaluable assistance, support and guidance. Deepest gratitude are also due to the members of the supervisory committee, Dr. Seok-Bum Ko and Dr. Li Chen without whose knowledge and assistance this study would not have been successful.

The author would also like to convey thanks to the NSERC and Electrical and Computer Engineering department for providing the financial means and laboratory facilities.

The author wishes to express his love and gratitude to his beloved family; for their understanding and endless love, through the duration of his studies.

This thesis is dedicated to my father, who taught me that the best kind of knowledge to have is that which is learned for its own sake. It is also dedicated to my mother, who taught me that even the largest task can be accomplished if it is done one step at a time.

CONTENTS

Permission to Use i

Abstract ii

Acknowledgements iv

Contents vi

List of Tables viii

List of Figures ix

List of Abbreviations x

1 Introduction 1
- 1.1 Introduction . . . 1
- 1.2 Motivation . . . 4
- 1.3 Thesis Objectives . . . 5
- 1.4 Thesis Organization . . . 6

2 Data Compression 7
- 2.1 Introduction . . . 7
- 2.2 Compression Methods . . . 8
- 2.3 Run-Length Encoding . . . 10
- 2.4 Huffman Encoding . . . 11
- 2.5 Adaptive Huffman Coding . . . 14
- 2.6 Arithmetic Coding . . . 15
- 2.7 LZ or LZ77 Coding . . . 18
- 2.8 LZ78 Coding . . . 19
- 2.9 LZW Coding . . . 20
- 2.10 LZW–Flush Coding . . . 22
- 2.11 Lossless JPEG Encoding . . . 23
- 2.12 Summary . . . 23

3 Simulation Result 25
- 3.1 Introduction . . . 25
- 3.2 DCT-Based Compression . . . 25
- 3.3 ZigZag Scan . . . 27
- 3.4 Comparative Study . . . 29
  - 3.4.1 Determining the Library Size . . . 33
- 3.5 Summary . . . 35

4 Hardware Implementation **38**
4.1 Introduction . . . 38
4.2 Hardware Architecture . . . 38
4.2.1 Dictionary . . . 39
4.3 Controller . . . 42
4.4 Counter . . . 44
4.5 Summary . . . 46

5 Performance Evaluation **47**
5.1 Introduction . . . 47
5.2 Hardware Comparison . . . 47
5.3 Reconstructed Images . . . 50

6 Lossy LZW–Flush **55**
6.1 Introduction . . . 55
6.2 Lossy LZW–Flush . . . 55
6.2.1 Weber–Fechner Law . . . 55
6.2.2 Algorithm . . . 56
6.2.3 Proposed Lossy LZW–Flush . . . 58
6.3 Performance Evaluation . . . 61
6.4 Summary . . . 63

7 Conclusion and Future Work **69**
7.1 Summary Of Accomplishments . . . 69
7.2 Future Works . . . 71

References **72**

LIST OF TABLES

2.1 Probability table for each symbol 16
2.2 Probability table for ”a” interval 17
2.3 Probability table for ”ab” interval 17

3.1 The quantization table developed in [1] 27
3.2 Performance analysis of the encoder 35

5.1 The average CR of 20 medical images for different encoding algorithms . 48
5.2 Hardware comparison of different encoder 49

6.1 Compression performance of the proposed lossy encoder 62
6.2 Hardware cost of lossy LZW-Flush algorithm 62

LIST OF FIGURES

1.1 Block diagram of a typical endoscopic system [1] 3

2.1 Lossless and lossy compression and decompression method 9
2.2 An example for RLE algorithm 11
2.3 RLE hardware 11
2.4 Example of Huffman compression algorithm 13
2.5 Example of LZ77 algorithm [2] 19
2.6 Example of LZ78 algorithm [3] 20
2.7 Example of LZW algorithm taken from [4] 21
2.8 Flow chart of LZW–Flush algorithm 24

3.1 Overview of DCT operation 26
3.2 Image compression system used in this work 27
3.3 Functional diagram of ZigZag Scan 28
3.4 20 endoscopic colour-scale images 30
3.5 Effects of *n* on compression ratio (CR) 31
3.6 Effects of *n* on PSNR 32
3.7 Compression ratio (CR) of different data encoding methods 34
3.8 Effect of different library size on CR(%) 36
3.9 Effect of library size on mean CR for 20 endoscopic colour-scale images . 37

4.1 Hardware block diagram of the LZW–Flush encoder 40
4.2 Hardware architecture and RTL view of a CAM cell 41
4.3 The architecture of entire proposed dictionary 43
4.4 The architecture of the proposed controller 45
4.5 The architecture of the counter 46

5.1 The image reconstruction process block diagram 50
5.2 Image 8 PSNR = 30.914 dB 51
5.3 Image 9 PSNR = 30.409 dB 52
5.4 Image 13 PSNR = 32.249 dB 53
5.5 Image 16 PSNR = 30.271 dB 54

6.1 The lossy LZW–Flush algorithm flow chart 59
6.2 Block diagram of proposed lossy LZW–Flush 60
6.3 Effects of *Tr* on PSNR 60
6.4 Effects of *Tr* on CR of five arbitrary capsule endoscopy images 61
6.5 Gray scale of image 6 (Tr = 1 to 10) 64
6.6 Gray scale of image 13 (Tr = 1 to 10) 65
6.7 Gray scale of image 16 (Tr = 1 to 10) 66
6.8 Proposed filter hardware architecture and RTL view 67
6.9 Reconstructed colour image Lena 68

List of Abbreviations

FPGA	Field-Programmable Gate Array
RLE	Run-Lenght Encoding
chr	Character
CR	Compression Ratio
PSNR	Peak Signal Noise Ratio
WCE	Wireless Capsule Endoscopy
CAM	Content-Addressable Memory
Tr	Threshold
JPEG	Joint Photographic Experts Group
LZW	LempelZivWelch
SQ	Scaling and Quantization
DFT	Discrete Fourier Transform
VLSI	Very Large Scale Integration
DPCM	Differential Pulse Code Modulation

CHAPTER 1

INTRODUCTION

1.1 Introduction

The enhancement of technology has radically affected everything and the medical science is not excluded. One of the major effects is the invention of medical imagining technology. Prior to the end of the nineteenth century, doctors cut open the patient to diagnose most of the internal medical problems. With the invention of X-rays by William Roentgen [5], the first medical image emerged to medical science. As the invisible gets visible through X-ray, the cut open was removed from the process of diagnosing many internal medical problems. The first revolution in medical science occurred in the earliest twentieth century. During this time, scholars started to enhance the Roentgen's discovery which led to the invention of computed tomography scanning (CT scan), magnetic resonance imaging (MRI), position emission tomography (PET) scanning, and ultrasonography [6].

Medical imaging step by step has shaped its own concept. In medicine context, the technique or process which leads to create an image from human body for the medical proposes is called medical imaging [7, 8]. This concept opened new disciplines in medical imaging such as biological imaging, radiology, nuclear medicine, endoscopy, etc. As

other types of science, these technology enhancements have brought some disadvantages particularly in medical profession, such as bringing pain or bleeding risk to patient during the imaging process. To overcome these problems, another enhancement is required which dictates some restrictions for engineers which are discussed in the following.

In endoscopy discipline, in order to study inside of the human gastrointestinal (GI) and digestive track and diagnose the GI diseases, the first gastro–endoscopy prototype was developed in 1952 by a Japanese team of a doctor and optical engineers [9]. It was integrated from a flexible tube attached to a tiny camera and light bulb on top of it. To capture images, this tube should be immersed to the intestine. Meanwhile, the patient suffers from a lot of pain caused by this device and there might be some scratch while the device gets trough. To alleviate the pain and scratch risk and study the rest one third of bowl, scholars brought up new solution and made a wireless swallow–able device so–called capsule endoscopy (CE) invented by Dr. Gavriel Iddan [10] in 1998.

The patient swallows the capsule (size: 11×30 mm) that captures images of the GI tract and sends them wirelessly through the RF signal [10]. The device is comprised of four main modules: battery, image sensor, image compressor, and wireless transmitter as shown in figure 1.1. The image sensor converts the physical image to electrical signals. The captured digital signal needs many bytes to represent the image and takes a long time for transmission [11]. To make it efficient for sending, an image compression is a must. Therefore, images are compressed in image compressor and then transmitted to the outside world by the RF transmitter block where they are reconstructed back to image in a

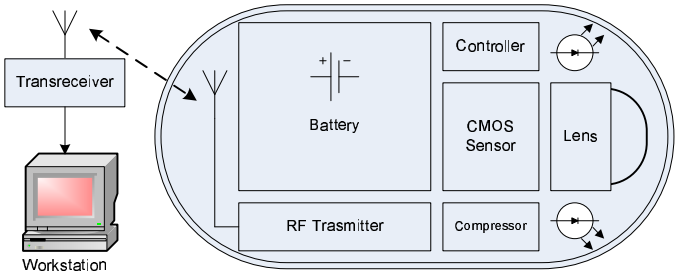


Figure 1.1: Block diagram of a typical endoscopic system [1]

workstation and used for diagnosis. The signals are received by the receiver tied to the patient's belt. The received images are uncompressed and enhanced inside the computer and are provided for doctors to study. The capsule is energized by the built–in battery. It is important to mention that for such an application, the number of transmitted bits has linear relationship with the amount of power consumption.

In order to save the battery life, it is important to have modules of ultra–low power consumption. At the same time, the reconstructed images should be of high resolution for the physician to examine correctly [12].

The image compression can be accomplished in two ways: Lossless and Lossy [13]. The lossless algorithm has no effect on the quality of the reconstructed image, while the lossy degrades the quality; however, the compression rate of the lossy algorithm is more than the lossless. Therefore, there is a trade-off between compression rate and the driven quality.

For the low–power medical imaging system, an efficient data encoder is proposed based on Lempel–Ziv–Welch (LZW) algorithm. The LZW algorithm searches for patterns inside of the input stream and replaces each with a unique index. The larger the patterns in the data, the better compression rate is achieved. The encoder is library–based and the size of the library can be controlled by the user which results in variable data rate. The simulation is carried out on endoscopic colored images using several encoder algorithms. The hardware architecture of the proposed encoder is also presented.

1.2 Motivation

How much image quality is expected in medical imaging is an important issue. The visual image quality can be determined by Peak–Signal–to–Noise–Ratio (PSNR) and their appearance [12]. Due to the importance of taken images on the study of human body, the PSNR of the images should be higher than 30dB [12]. In [1], efficient image compressor architecture is presented which achieved a very good compression rate along with an average PSNR above 30dB. The image processor is constructed with three main blocks; 2D forward transformer, Scaling and Quantization (SQ), and encoding.

In order to assess the suitability for an area– and power–sensitive medical imaging system such as capsule endoscopy, the previous works in literature have focused mostly on the two first components of the image compressor which include the 2D cosine transform and the scaling and quantization [14]. A very few research has been devoted on investigating the procedures and methods that can be applied to the data encoding component to

achieve more efficient images in terms of quality with regard to power consumption and size.

Our concern in this work is to find a simple encoder with high compression ratio and PSNR for this configuration. This encoder converts the 2D data given from SQ block to stream of symbols suitable for the Transmitter stage. The encoder can be departed to two parts; zigzag scan and data compression, which will be discussed in details later in this thesis.

1.3 Thesis Objectives

In [1], Wahid et al presented the low–power design of an image compressor for the wireless capsule endoscopic system. As mentioned, our focus here in this research is to find an efficient data encoding algorithm which is the part of the image compressor. The objective in finding such an algorithm can be summarized into:

1. Selecting an encoding algorithm which produces high quality reconstructed images. This feature increases the chance of diagnosing diseases.

2. Selecting an encoding algorithm which produces highly compressed data. The motivation for this purpose is that as the data is compressed the compression results need less data to be transmitted through the wireless transmitter which results in decreasing the amount of power consumption. Besides, it ends up to have a decrease in the size of encoded data which requires less space for storing the data.

3. Selecting a simple encoding algorithm which would be a high potential candidate for hardware implementation. This feature would yield to less power and area consumption. The motivation behind this reason relies on the limitations of the size of the capsulate and the power it exploits. This work leads to the development of the architecture, and its implementation on FPGA and Very-Large-Scale Integration (VLSI) platform.

1.4 Thesis Organization

This thesis is organized into seven chapters. In Chapter 2, information on different data compression methods and algorithms are provided. In Chapter 3, a comparative study is accomplished between different encoding methods and the simulation results are presented. Chapter 4 provides the hardware FPGA implementation of the most efficient encoding method investigated and selected in Chapter 3. Later, in Chapter 5, the performance of LZW–Flush encoder is evaluated in terms of algorithm simplicity and FPGA hardware cost. Chapter 6 presents a developed lossy encoder which is similar to JPEG-LS [15] and investigates its compression rate and hardware design. Chapter 7 concludes the thesis by summarizing the accomplishment of the research work and presenting future extensions that can be applied to this research.

CHAPTER 2

DATA COMPRESSION

2.1 Introduction

Data transmission is the physical transfer of data over channels. The main issues regarding data transmission can be classified into the limitation of bandwidth as well as capacity problems. In order to tackle these issues, solutions exist such as building large storage capacity or increasing the bandwidth. These solutions are not desirable as they require expensive resources such as hard disk space or large transmission bandwidth [3]. In order to reduce resource consumptions, data compression techniques were developed to provide better solution for these challenges. Compression techniques aim to find an efficient algorithm to remove various redundancy from a certain type of data.

In this chapter, we provide an introduction to compression techniques. First, the two major categories of compression techniques, lossless and lossy compression, are described. Later, different algorithms of lossless compression techniques are presented. These algorithms include Run-Length Coding, Huffman Coding, Arithmetic, LZW, LZ77, LZ78, and Joint Photographic Experts Group (JPEG).

2.2 Compression Methods

Data compression has become a requirement for most applications in different areas such as computer science, information technology, communications, medicine, etc. In computer science, *Data Compression* is defined as the science or the art of representing information in a compact form [3]. In communications, data compression enables devices to transmit or store the same amount of data in fewer bits. In this area, it is viewed as a means for efficient representation of a digital source of data such as text, image, sound, or any combination of these. Examples of some application areas that require data compression include:

- personal communication systems such as facsimile, voice mail and telephony
- multimedia, signal processing, imaging
- image archival
- memory structures or disk optimization
- better usage of connection bandwidth

Compression techniques can be categorized into two major types: lossless and lossy compression. If it is possible to exactly rebuild the original data from the compressed version, the compression approach is referred to as lossless compression. It can be understood that this method is called lossless as there is no loss of any information during the compression process. This approach is also called reversible compression since it is

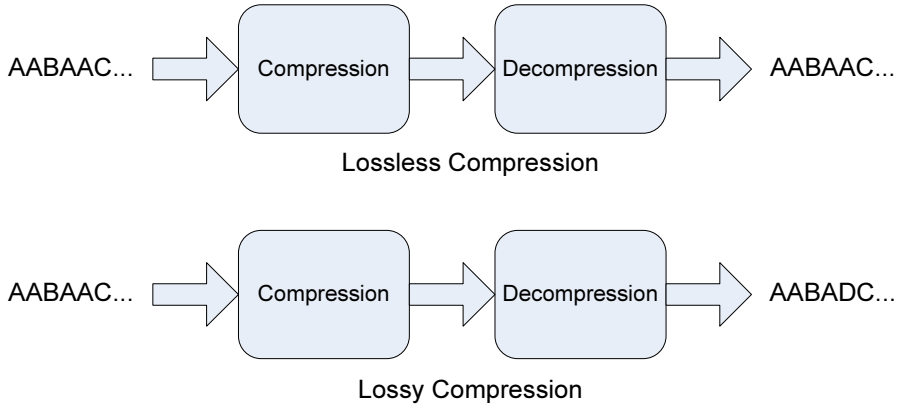


Figure 2.1: Lossless and lossy compression and decompression method

possible to recover the original data perfectly by decomposition.

In 1948 Claude E. Shannon formulated the theory of data compression in [16]. Shannon established that there is a fundamental limit to lossless data compression. This limit, called the entropy rate, is denoted by H. The exact value of H depends on the information source, more specifically, the statistical nature of the source. It is possible to compress the source, in a lossless manner, with compression rate close to H. It is important to mention that mathematically it is impossible to do better than H.

Shannon [16] also developed the theory of lossy data compression. Lossy Data Compression methods are not able to reconstruct the original data exactly from the compressed version. Some insignificant details might get lost during the process of compression. This type of compression is also referred to as irreversible since it is not possible to recover the original data exactly by decomposition. Multimedia, images, video and audio are more easily compressed by lossy compression techniques.

Figure 2.1 shows an example of data compression and decompression using lossless and lossy algorithms. In the rest of this chapter, different types of lossless compression algorithms are provided.

2.3 Run-Length Encoding

Run-Length Encoding (RLE) is a simple, easy, famous, lossless compression method used in different applications such as facsimile communication or image formats such as Graphics Interchange format (GIF) and windows Bitmap (BMP) [17]. In some applications, a combination of this method along with other techniques would be used to archive with higher compression ratio. In this method, the length of each repeating character $(calledrun)$ in a stream of data, is calculated and the data is encoded into two bytes. The first byte represents the number of characters in the run and the second byte is the value of the character in the run.

In terms of hardware, this method has very basic structure. It does not need data processing and can be easily implemented. Thus, it is a suitable candidate for those applications which have restriction on hardware or energy consumption.

The pseudocode for RLE algorithm is presented below:

1. read $chr(i)$
2. if it's not the same as $chr(i+1)$, output $\{chr(i)\}$ otherwise, count run's of $chr(i)$ and output $\{run, chr(i)\}$

3. if there is more chr go to step 1

In order to provide a better understanding of the algorithm an example is given in figure 2.2.

stream $...AAAAAAAAAAAAAAAABBBBBBBBBCCCAEE...$

compressed $...17A8B3CA2E...$

Figure 2.2: An example for RLE algorithm

As it can be observed, the stream takes 31 bytes while the compressed stream requires just 9 bytes.An example of hardware implementation is provided in figure 2.3

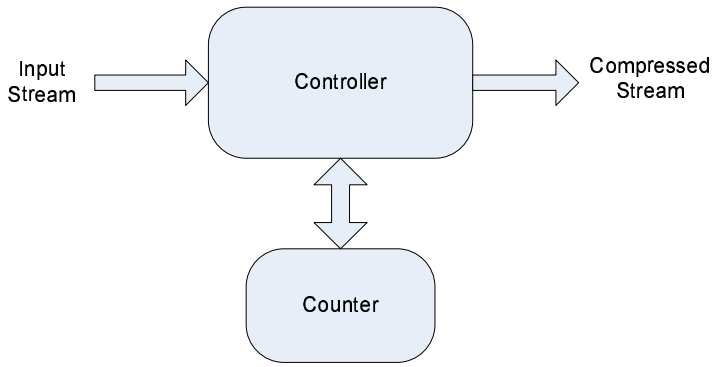


Figure 2.3: RLE hardware

2.4 Huffman Encoding

The Huffman encoding (static Huffman) is a popular lossless data compression algorithm that produces variable length code to represent symbols [18]. The driven length depends

on the probability of the occurrence of each symbol in the input string. Therefore, for more frequent symbols, this method uses shorter codewords and for less common ones, it uses longer codewords. The Huffman optimum algorithm is developed by David A. Huffman in 1952, while he was a Ph.D. student at MIT, and has been applied to many software and hardware applications. For example, Huffman compression method is included in JPEG and MPEG standards, as well as ZIP standards, for compressing data files [19].

In order to specify the frequency of each symbol and replace the symbols with relative code-words, two passes of the entire string to the encoder are required. In the first pass, the frequency of each character is calculated and the Huffman binary tree is produced. Later, Huffman's algorithm minimizes the weighted path length $\sum w_j f_j$ among all binary trees, where w_j is the weight of the jth leaf, and f_j is its depth in the tree [18]. The codewords related to each symbol are determined and inserted in a look up table. In second pass, each character is replaced by it's codeword which was calculated in the first pass. The pseudo code of this algorithm is presented below [20].

1. *C* is a set of n characters
2. n is the total number of chars in *C*
3. insert all the chars to in ascending frequency order in queue *Q*
4. n-1 times
5. create a new node *Z*

6. left child of z is the least frequent char popped from *Q*

7. now pop another char from *Q* to create the right child

8. frequency of *Z* is the sum of frequencies of it's children

9. insert the newly created object into the min-priority queue

10. loop ends

11. return the root of the tree

An example for the Huffman algorithm is shown in figure 2.4. It is important to mention that Huffman coding would be the best variable-length entropy coding when the probability of the occurrence of each symbol is negative powers of 2 [18]. For example, if the frequency of the symbol is 0.4, the ideally assign code should be $\log_2 0.3 \approx 1.73 bits$ which Huffman normally can assign 1 or 2 bits for it.

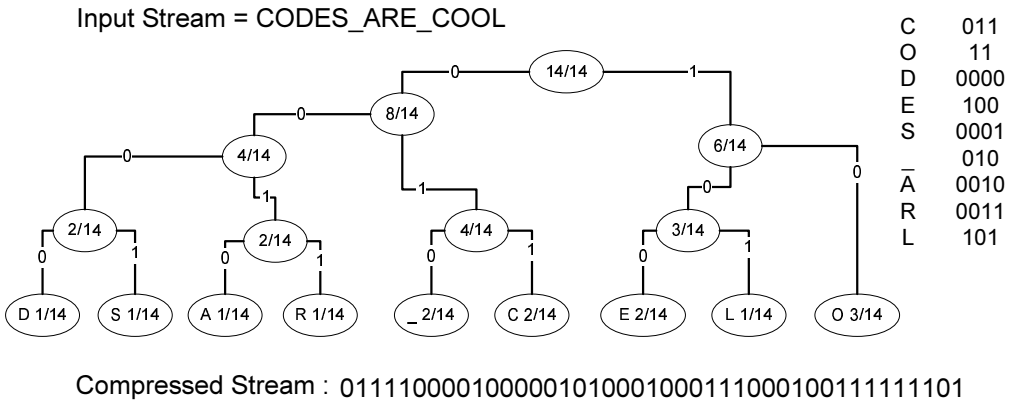


Figure 2.4: Example of Huffman compression algorithm

In [19], [21], the authors have proposed different hardware implementations of the Huffman algorithm. As Huffman method requires two passes, it is not suitable for real-

time applications. Thus, in order to eliminate the pre-scan step for these types of applications, known code word tables [22] are provided. These tables are tuned for different applications. For a large file or big stream of data, it is somehow very difficult to find the frequency of each chr. Therefore, it is needed to store the whole stream which requires a huge space. If the frequency of each chr varies, the related codeword is varied. In order to find the codeword, we should re-compute the Huffman code by running through the entire file.

One problem is that the statistical model of the data source must be known before the design of the Huffman code. The better the model and data source agree, the better the Huffman code performs. A bad model can lead to a low compression efficiency, sometimes even worse than no compression at all. This is a very real problem because in many applications, the prior knowledge of the statistics of a data source is impossible, or the statistics are changeable with time. [23]

2.5 Adaptive Huffman Coding

In order to tackle the two-pass issue of the Huffman method and make it suitable for real-time applications, Faller and Gallager [24] presented a solution which is referred to as Adaptive Huffman algorithm (also called Dynamic Huffman). In the adaptive coding, a fixed model is no longer used. Instead, a counter is set up for each symbol of the source alphabet, and the count is initially set to 1 or to an expected occurrence frequency. Each time a symbol occurs, its count increases by 1, and the code tree is then updated to fit the

accumulative counts which approximately represent the local statistics of the data source. Every time a certain amount of data has been encoded, equivalent to a time constant, each count is multiplied by some fixed factor. Since both sender and receiver update their own code tree based on the same previous data sequence, the codes used by both sides always agree [23].

However, the adaptive coding provides low compression efficiency when it is applied to segmented data. In fact, there is still a low compression efficiency at the beginning of the data even with the adaptive coding. The reason resides on the fact that not enough statistical information has been gained in the beginning to establish a good model. However, the problem becomes more serious when segmenting data into segments. Every segment cannot use the previous segment's model as the segments could have been corrupted by channel noise. Thus, initialization is basically necessary for every segment which results in a large percent of data encoded with lower compression efficiency. [23].

2.6 Arithmetic Coding

Almost for 25 years, arithmetic coding was the most successful substitute for Huffman coding algorithm. This method has superior performance compared to Huffman specially for the situations where the input stream is small. This algorithm was an extension of encoding work of Shanon, Fano and Elias [3]. This algorithm solves the problem of assigning integer to each symbol for those do not have occurrence frequency as a power of 2. In this algorithm, instead of looking for each symbol, the method that most entropy encod-

ing algorithms apply, the entire file stream is checked and encoded into a single number, n which is 0 $\leq$ n $<$ 1. In Arithmetic coding , there exists a probability line, 0 – 1. A range is assigned to each symbol based on its probability. As the ranges are assigned to each symbol, the encoding process starts.

In order to clarify how the algorithm behaves, an example is provided in the following. Assume that we want to encode abd. Table 2.1 shows the probability value assigned to each symbol.

Table 2.1: Probability table for each symbol

Symbol	Probability	Interval
a	0.2	$[0.0, 0.2)$
b	0.3	$[0.2, 0.5)$
c	0.1	$[0.5, 0.6)$
d	0.4	$[0.6, 1.0)$

The first symbol to be encoded is a. For this purpose, we *zoom* to the line which represents the probability of a and re-calculate the probability of each symbol of the table 2.1 for the new interval. The new values retrieved are presented in table 2.2

The next symbol in this stream is b which results in the table 2.3

The encoded stream is any number between $[0.1608, 0.2)$, which for this example

Table 2.2: Probability table for ”a” interval

Symbol	New ”a” Interval
a	$[0.0, 0.04)$
b	$[0.04, 0.1)$
c	$[0.1, 0.102)$
d	$[0.102, 0.2)$

Table 2.3: Probability table for ”ab” interval

Symbol	New ”ab” Interval
a	$[0.102, 0.1216)$
b	$[0.1216, 0.151)$
c	$[0.151, 0.1608)$
d	$[0.1608, 0.2)$

0.1608 was chosen. In the binary, the symbol is represented as 11001001000. As it can be seen from the example, this algorithm needs a lot of calculation and the output of the encoded stream depends on the floating point unit precision. On the other hand, floating point arithmetic is not suitable in terms of hardware implementation as it requires a lot of hardware. Apart from that, this method has binary rounding which brings error. Thus, this algorithm is not a suitable choice for large files and is fairly slow due to big numbers of calculations.

2.7 LZ or LZ77 Coding

In variable-length compression algorithms, a prior knowledge about the source characteristic is important which helps in achieving higher compression ratio. In many cases, accessing prior information is impossible or unreliable. Therefore, there was a need for a mechanism which learns the characteristics of the input stream and applies it for the purpose of gaining higher compression ratio. Abraham Lempel and Jacob Ziv, in 1977, introduced a dictionary-based compression algorithm which is called LZ or LZ77 [25].

This algorithm looks for patterns while encoding the input stream, and replace those patterns with a fixed size codeword. This algorithm is very simple and does not need huge resources to be implemented. LZ77 exploits a windows which slides through the input stream. The window can be divided into two parts, the left one is called *History buffer* or search buffer which includes a portion of recently seen input stream, and the right one is called *Look-ahead buffer* which contains the next portion of the input required to be compressed. The window size is fixed and the size of the history buffer is much bigger than look-ahead buffer. The pseudo code for LZ77 algorithm is shown below:

1. Read from input stream till look-ahead buffer (L) gets full.
2. search through history buffer (h) to find a prefix match with L
3. if the size of match is bigger than one , output $<$ location of first match symbol in window = f , match length = l, mismatch symbol following the match = $c >$.
4. if there is no match output $< 0, 0, ASCII(c) >$

5. if there is more symbole to encode, slides window one symbol to right and run from step 2.

An example of this algorithm is shown in figure 2.5.

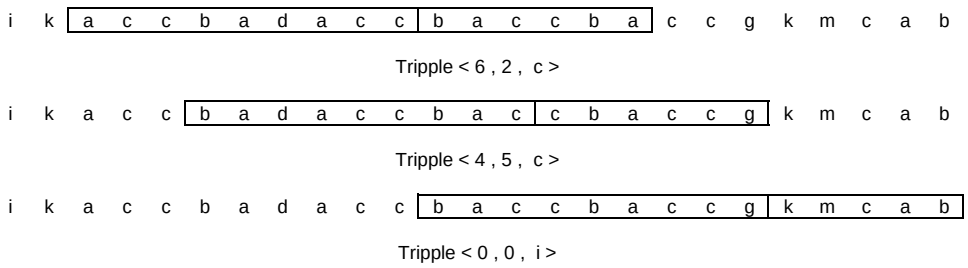


Figure 2.5: Example of LZ77 algorithm [2]

2.8 LZ78 Coding

In LZ77, the pattern would easily be lost if it is shifted out from the history window; therefore, no match would be found for the next symbol in the input stream. This deficiency causes this algorithm to be incapable of having highly compression ratio. A year after the development of LZ77, Lemple and Ziv introduced a new algorithm, referred to as LZ78, which overcomes the discussed problem and provides the capability of holding all patterns [26]. Another improvement of this algorithm compared to LZ77 is that the new method encodes the input into a two component codeword $< index(word), c >$ instead of a triple. In LZ78, the history buffer is replaced with a dictionary which collects all previously observed patterns. The size of look-ahead window is restricted to one symbol length. The pseudo code for this algorithm is shown below [3], [25]

1. input: input stream, and empty dictionary
2. empty *word*
3. send next symbol to *c*
4. search for $word+c$ in the dictionary, if there is a match, replace $word$ with $word+c$ and take next symbol to c and redo step 3
5. if there is not a match, output token $<$ (index of *word* in the dictionary $), c >$ and add $word + c$ into the dictionary
6. if still there is a symbol to compressed go to step 3

In order to provide a better understanding of how this algorithm works, an example is presented in figure 2.6. The example takes *a _date* as input stream.

i	*w*	*c*	*w+c*	*output*	0	1	2	3	4	5
1		a	a	0 , a		a				
2		_	_	0 , _		a	_			
3		d	d	0 , d		a	_	d		
4		a	a							
	a	t	at	1 , t		a	_	d	at	
5		e	e	0 , e		a	_	d	at	e

Figure 2.6: Example of LZ78 algorithm [3]

2.9 LZW Coding

In 1984, four years after the introduction of LZ78, Terry Welch published a paper about a new algorithm called LZW. Welch in this algorithm provides small changes to LZ78 algorithm which yields to higher compression ratio. In LZW, the first 256 rows of dictionary

i	*w*	*c*	*w+c*	*output*	257	258	259	260	261
1		a	a						
	a	c	ac	a	ac				
2	c	b	cb	c		cb			
3	b	b	bb	b			bb		
4	b	a	ba	b				ba	
5	a	a	aa	a					aa
6	a	c	ac						
	ac			257					

Figure 2.7: Example of LZW algorithm taken from [4]

are initially pre-filled with 0 to 255. Also, the number of output components is decreased to one [27]. The LZW idea is to identify the longest pattern from the input stream and replace it with dictionary indices. The pseudo code for this algorithm is shown below [3]. The implementation speed of this algorithm, as can be seen from pseudo code, is very fast.

1. empty *word*

2. send next symbol to *c*

3. search for $word+c$ in the dictionary, if there is a match, replace $word$ with $word+c$ and take next symbol to c and redo step 2

4. if there is not a match, output the dictionary index for $word$ and add $word+c$ to the dictionary, and replace $word$ with x.

5. if still there is a symbol to compressed go to step 2

6. output the dictionary index for the $word$

An example of this method is provided in figure 2.7 to clarify the pseudo code. The example takes $acbbaac$ as input stream, and outputs *a c b b a 257*.

2.10 LZW–Flush Coding

The implementation of LZW has an important restriction; size of dictionary. In practise, we cannot have unlimited dictionary size. Besides, the size of the dictionary is increased by feeding the encoder with larger file size. Therefore, to tackle this issue, a modified LZW algorithm, called LZW-Flush [28] was developed in which the dictionary gets flushed if it is full and gets refilled again through the process of compression for the rest of symbols. The process of flushing and refilling continues till the whole input stream is compressed. The pseudo code of this algorithm is written below.

1. empty *word*
2. send next symbol to *c*
3. search for $word+c$ in the dictionary, if there is a match, replace $word$ with $word+c$ and take next symbol to c and redo step 2
4. if there is not a match, output the dictionary index for $word$.
5. if dictionary is not full, add $word + c$ to the dictionary.
6. if dictionary is full, clear the dictionary and add $word + c$ to the dictionary.
7. replace $word$ with x
8. if still there is a symbol to compressed go to step 2
9. output the dictionary index for the $word$

Figure 2.8 shows the flow chart of LZW–Flush coding algorithm.

2.11 Lossless JPEG Encoding

JPEG is designed to exploit known limitations of the human eye. It is based on the fact that the small color changes are perceived less accurately than small changes in brightness. The encoding method uses a predictive scheme based on the three nearest causal neighbours which include the upper, left, and upper-left neighbours. To assess the prediction error entropy coding is used [29]. The lossless coding process employs a simple predictive coding model called Differential Pulse Code Modulation (DPCM). In this model, the predictions of the sample values are estimated from the neighbouring samples that are already coded in the image. In most predictors, the average of the samples are taken immediately of the above and to the left of the target sample. Once all the samples are predicted, Huffman coding or Arithmetic coding is applied to differentiate between the samples in a lossless fashion.

2.12 Summary

This Chapter provided a review of different encoding methods. The description of each method was broken down into three parts of algorithm explanation, psudo code, and examples. The performance of all explained method is evaluated in the next chapter.

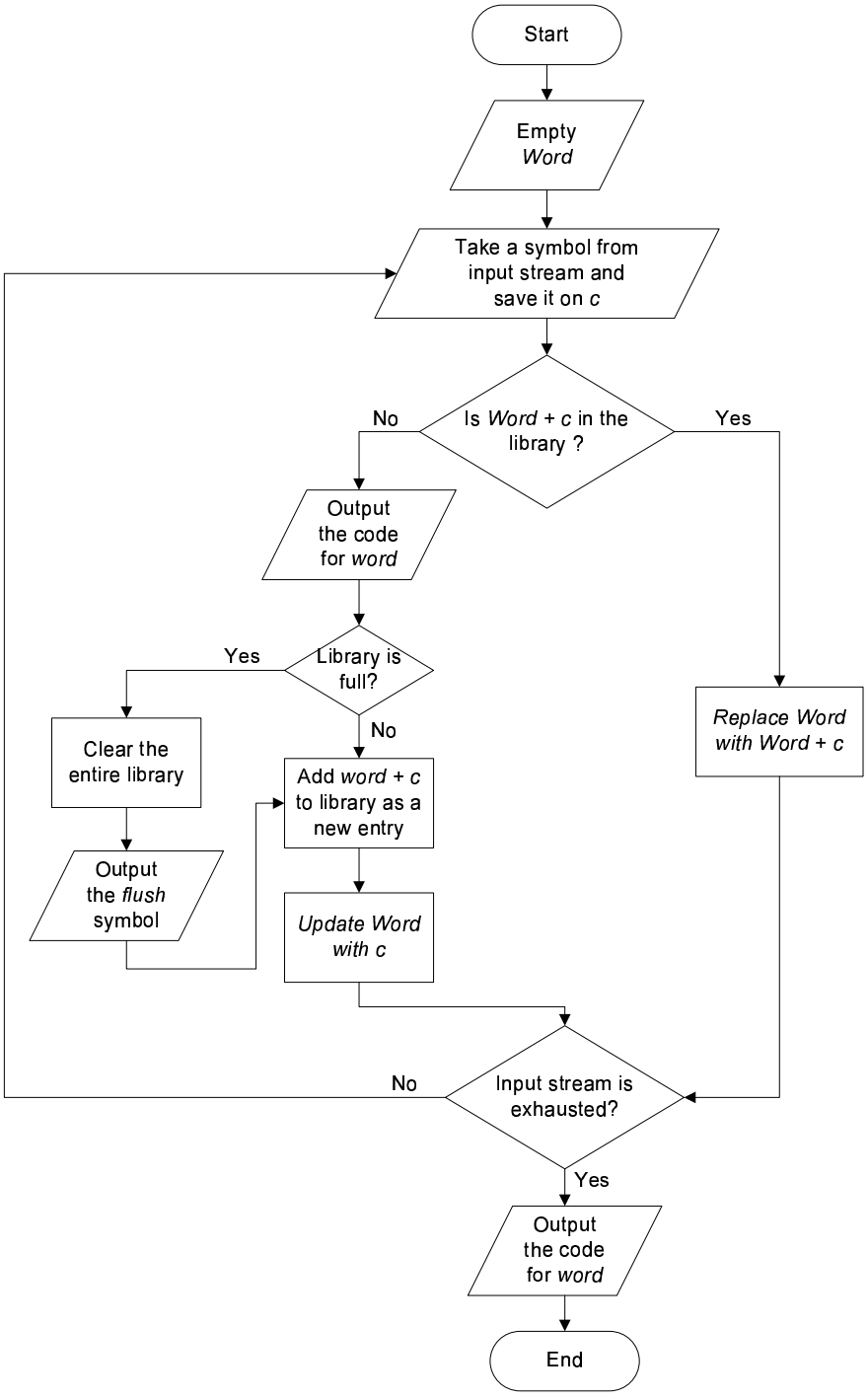


Figure 2.8: Flow chart of LZW–Flush algorithm

CHAPTER 3

SIMULATION RESULT

3.1 Introduction

Due to the different requirements of different applications, different methods are applied. In order to select an efficient algorithm for this particular application, first of all the performance of different algorithms should be assessed. This chapter presents the analysis of different algorithms through a specific image compression method [1]; DCT-based compression. At the beginning, the DCT-based image compression method is explained. Later, a modified zigzag scan is presented and at the last section of this chapter an efficient algorithm is selected based on the implementation difficulties of different algorithms and their compression ratio.

3.2 DCT-Based Compression

The rapid growth of digital imaging applications, including desktop publishing, multimedia, teleconferencing, and high-definition television (HDTV) has increased the need for effective and standardized image compression techniques. These techniques employ a basic technique known as the discrete cosine transform (DCT), developed by [30], which is a close relative of the Discrete Fourier Transform (DFT). DCT helps separate the image

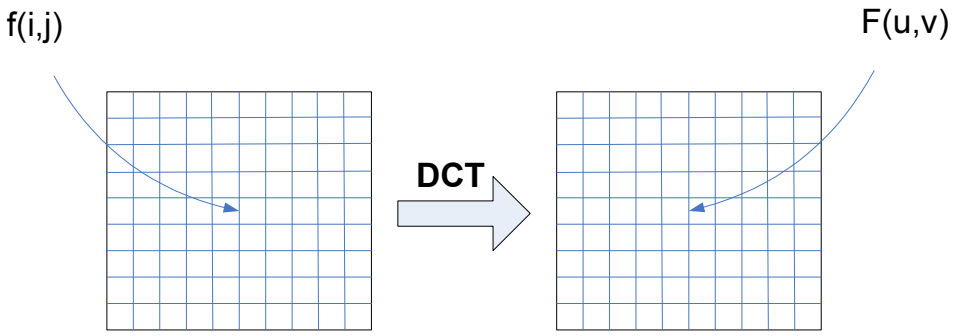


Figure 3.1: Overview of DCT operation

into parts (or spectral sub-bands) of differing importance with respect to the image's visual quality.

The DCT is similar to the discrete Fourier transform: it transforms a signal or image from the spatial domain to the frequency domain. Figure 3.1 provides a high level overview of this transformation.

The general equation for a 2-D DCT for an image sized N $\times$ M is defined by the following equation:

$$F(u,v) = (\frac{2}{N})^{\frac{1}{2}}(\frac{2}{M})^{\frac{1}{2}} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} \Lambda(i).\Lambda(j). \cos(\frac{\pi u}{2N}(2i+1)) \cos(\frac{\pi v}{2M}(2j+1)).f(i,j) \tag{3.1}$$

and the corresponding inverse 2D DCT transform is simple $F^{-1}(u,v)$, i.e. where

$$\Lambda(\zeta) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } \zeta = 0 \\ 1 & \text{otherwise} \end{cases}$$

The image compressor component of a typical endoscopic system is shown in figure

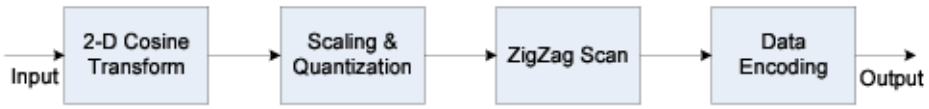


Figure 3.2: Image compression system used in this work

Table 3.1: The quantization table developed in [1]

64	128	128	512
128	256	256	1024
128	256	512	1024
512	1024	1024	2048

3.2. It starts in the format of Bayer patterns [31] and separately processes R-G1-G2-B signals of the raw image. Later, it computes the 2-D DCT, quantizes it, and outputs the encoded bit-stream for transmission. The 2-D DCT method which we exploit in our model is based on the technique proposed in [1] which has the advantage of reducing the hardware space as well as processing time. The proposed architecture in [1] eliminates the need of transpose operation by direct mapping of the 2D Discrete Cosine Transform. Wahid et al used an error-free algebraic integer encoding instead of the conventional binary finite precision approach which guarantees lossless computation. Table 3.1 shows the corresponding quantization table developed in [1] to comply with the JPEG standard.

3.3 ZigZag Scan

In order to send the scaled/quantized coefficients, data reordering (from 2-D to 1-D) is a must. This procedure is accomplished by scanning (as shown in figure 3.3). Different

scanning methods exist. The reforming from 2-D to 1-D can be done by sampling row by row, or column by column or an arbitrary pattern. The best method brings more redundancy of data which yields to better compression rate from the encoder stage. Since the most significant information of the image is located in the top-left region of the matrix, the zigzag scan would be a good candidate. This method is depicted on figure 3.3.

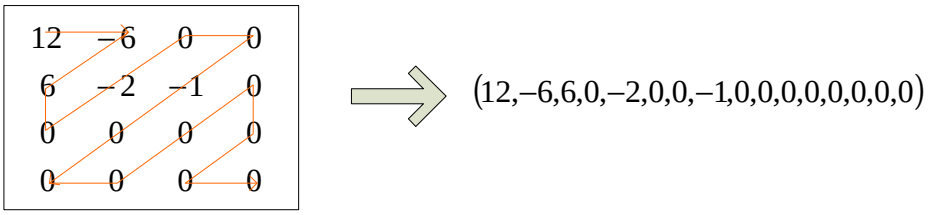


Figure 3.3: Functional diagram of ZigZag Scan

We can pass only such nonzero significant information (i.e. low frequency components) to the next stage while discarding the rest (zero coefficients) through the zigzag scan. To make this process more efficient, we propose a method which is dicussed bellow. The effect of the method on the compression ratio and reconstructed quality (in peak-signal-noise-ratio, PSNR) is anaylsed later in this section.

The devised algorithm performs the zigzag scan and stops scanning when it encounters a number of consecutive zeros, n, where the parameter n is variable and defined by the user; a symbol is added to represent the end of scanning process. We have applied this method on 20 endoscopic coloured images, shown in figure 3.4 (a resolution of 256×256), with different values of n to investigate its effect on the PSNR and the CR (as defined in Eq.

3.2 and 3.3).

$$CR = (\frac{\text{no. of discarded coefficients}}{256 \times 256}) \times 100\% \tag{3.2}$$

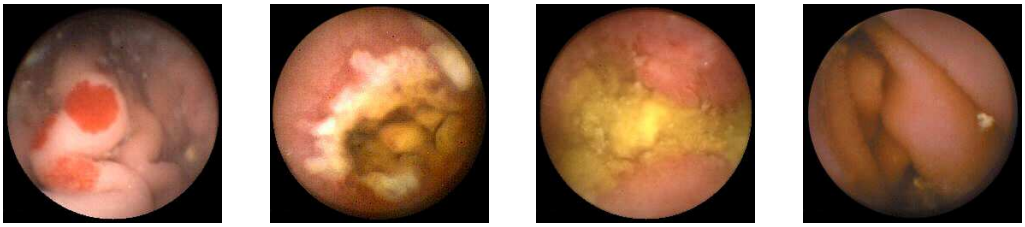
$$PSNR = 10 \times \log_{10}(\frac{(256 \times 256) \times (255 \times 255)}{\sum_{i=0}^{255} \sum_{j=0}^{255} (x_{i,j}^{reconstruct} - x_{i,j}^{original})^2}) \tag{3.3}$$

For the reconstruction, the output of the scan is inverse-scanned, followed by the inverse SQ and 2-D transformation. The results are shown in figures 3.5 and 3.6. As it can be seen from these two figures, for n less than 3 the PSNR is above 30 dB and with bigger values of n, a better CR is achieved. Hence, we have chosen n = 3 for the proposed encoder.

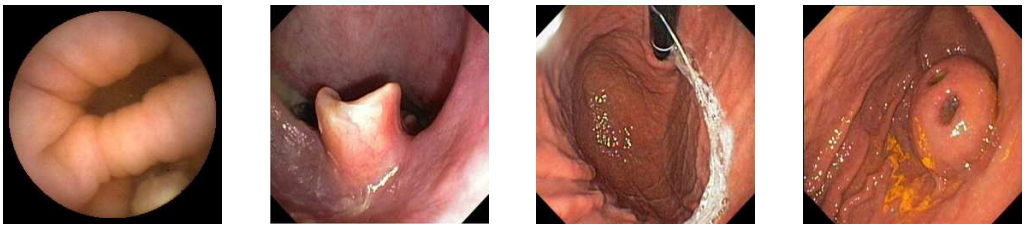
In summary, the algorithm will discard the remaining coefficients, if three consecutive zeros are detected in the scanning chain. Hence, it speeds up the scanning process over the traditional approach.

3.4 Comparative Study

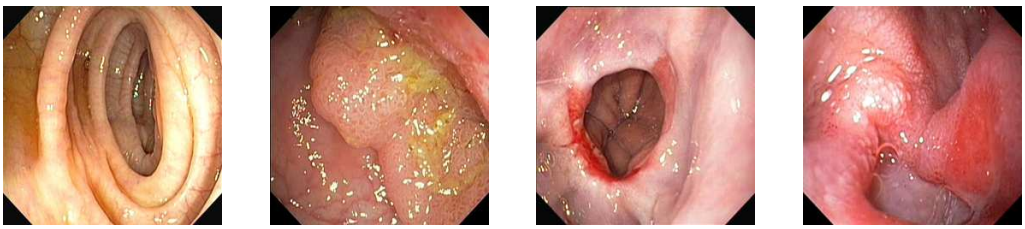
In order to choose an efficient encoding algorithm for the image compressor, we have implemented the all mentioned encoding algorithms discussed in the previous chapter in MATLAB software [32] to investigate the performance of each method. The simulation was carried out on the same endoscopic images. The output from the SQ unit (figure 3.2) has undergone the zigzag scanning (as defined in Section 3.3) before becoming the input



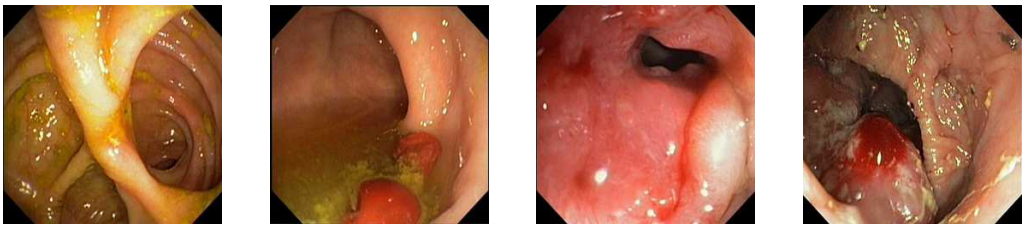
(a) Image 1 **(b)** Image 2 **(c)** Image 3 **(d)** Image 4



(e) Image 5 **(f)** Image 6 **(g)** Image 7 **(h)** Image 8



(i) Image 9 **(j)** Image 10 **(k)** Image 11 **(l)** Image 12



(m) Image 13 **(n)** Image 14 **(o)** Image 15 **(p)** Image 16

(q) Image 17 **(r)** Image 18 **(s)** Image 19 **(t)** Image 20

Figure 3.4: 20 endoscopic colour-scale images

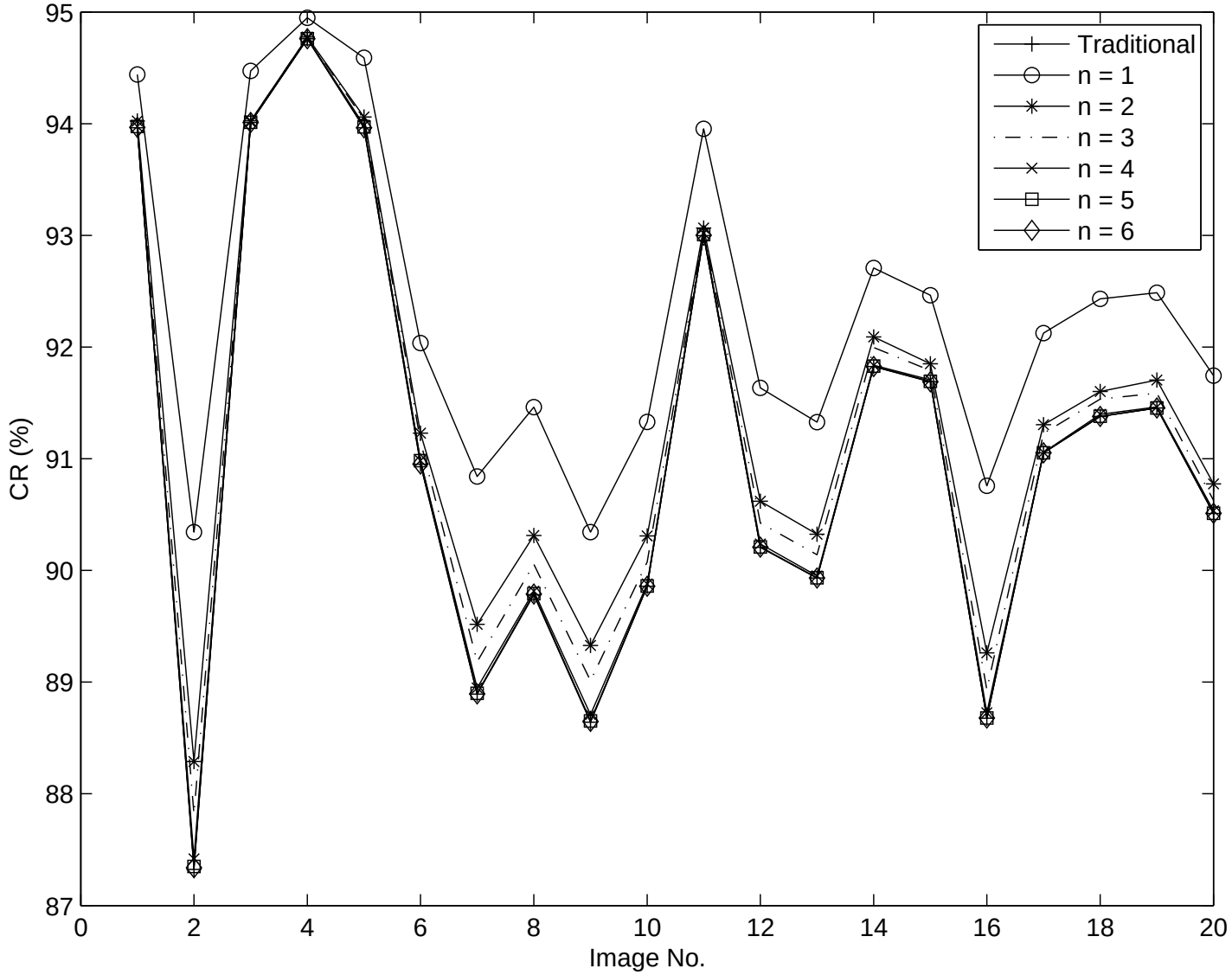


Figure 3.5: Effects of *n* on compression ratio (CR)

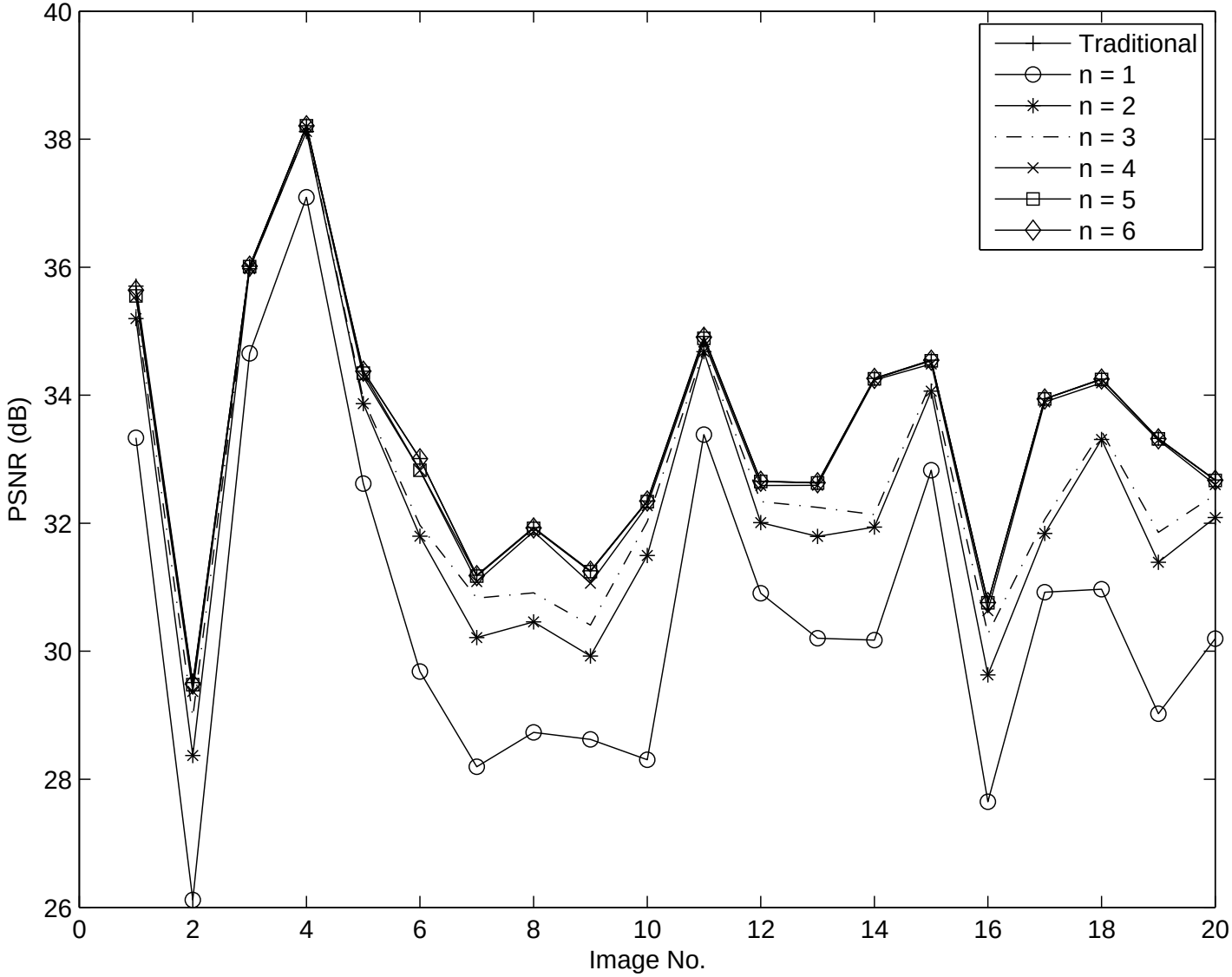


Figure 3.6: Effects of *n* on PSNR

to the encoder. The results are shown in figure 3.7. This result is just for the encoder stage. In order to represent the study better, we applied plot chart instead of bar graph. It can be seen from the figure 3.7 that both Huffman and LZW produce higher compression ratio compared to LZ78 and LZW-Flush (with a library size of 240 bytes) where the later twos performance is very similar. The PSNR for this comparative study for different method is the same as shown for n = 3 on figure 3.6.

As discussed earlier, the implementation of static Huffman encoder requires more memory, while LZW and LZ78 require larger dictionary and library size. In this case, the LZW-Flush would be a good option as the library size can be set by the user. Therefore, keeping the application in mind, where hardware resources (or design space) and power consumption are critical, we propose to use LZW–Flush encoding scheme.

3.4.1 Determining the Library Size

In order to find the best library size for the proposed WCE application, we have examined those endoscopic images with different library sizes of the LZW–Flush encoder. These images have been passed through the encoder and then reconstructed back to calculate the CR. The results are summarized in figure 3.8. It can be seen from figure 3.8 that the results for all 20 images are consistent in nature. As expected, by increasing the library size, higher compression rate can be achieved. Figure 3.9 shows the average CR of all images for different library size. However, it can be seen that the gain in CR slows down after a certain threshold (in library size) is passed. Considering the degree of compression we have achieved in the previous stages and the complexity and hardware cost of the encoder,

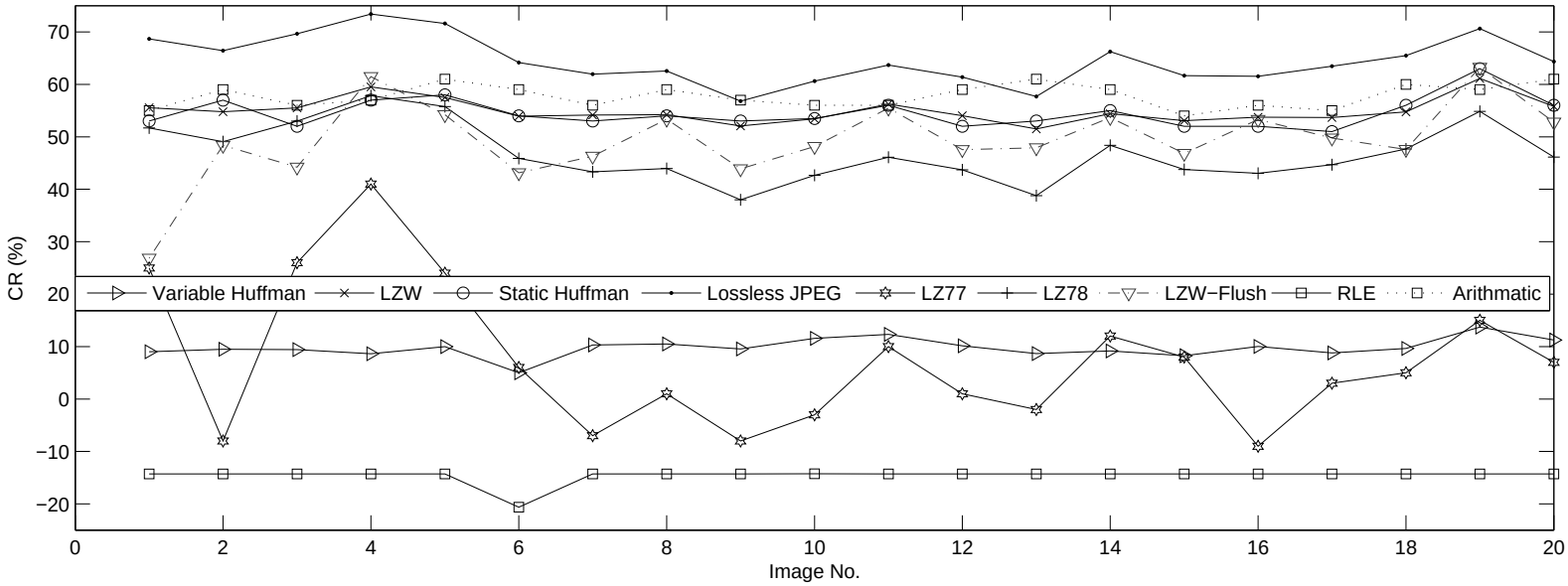


Figure 3.7: Compression ratio (CR) of different data encoding methods

Table 3.2: Performance analysis of the encoder

CR(%) (Zigag)	CR(%) LZW–Flush	CR(%) (Overall)	Energy Consumption
90 %	49.40 %	94.9 %	75 μJ / frame

we have chosen a library size of 240 bytes that will give us an average CR of 30% (figure 3.9).

In table 3.2, the performance analysis of the encoder is presented. Throwing out coefficients in the zigzag stage makes this encoding scheme lossy. As it can be observed, by applying the proposed window-based (library size of 240 bytes) LZW–Flush algorithm with a defined (n=3) zigzag pattern, a minimum compression ratio of 94.9% with acceptable image reconstruction (minimum PSNR of 30dB) can be achieved. As mentioned earlier, we have used 20 colour scale endoscopic images of 256$\times$256 resolutions for our simulation. According to [33], It takes 1.91nJ of energy for a BFSK transmitter to transmit one bit (assuming 2.07mA current consumption). By considering the image size (256x256, 8bpp, colour) and the CR achieved in this work, the total energy consumption is estimated to be 75 μJ/frame. As a result, a 1-min endoscopic video (captured at a rate of 4 frames per second) will consume only 18mJ of energy.

3.5 Summary

In this chapter the performance of different encoding methods in a DCT-based image compressor are evaluated. Also, a modified zigzag scan is presented and their effects is discussed. Hence, an efficient method (LZW–Flush), is selected. Simulation regarding the

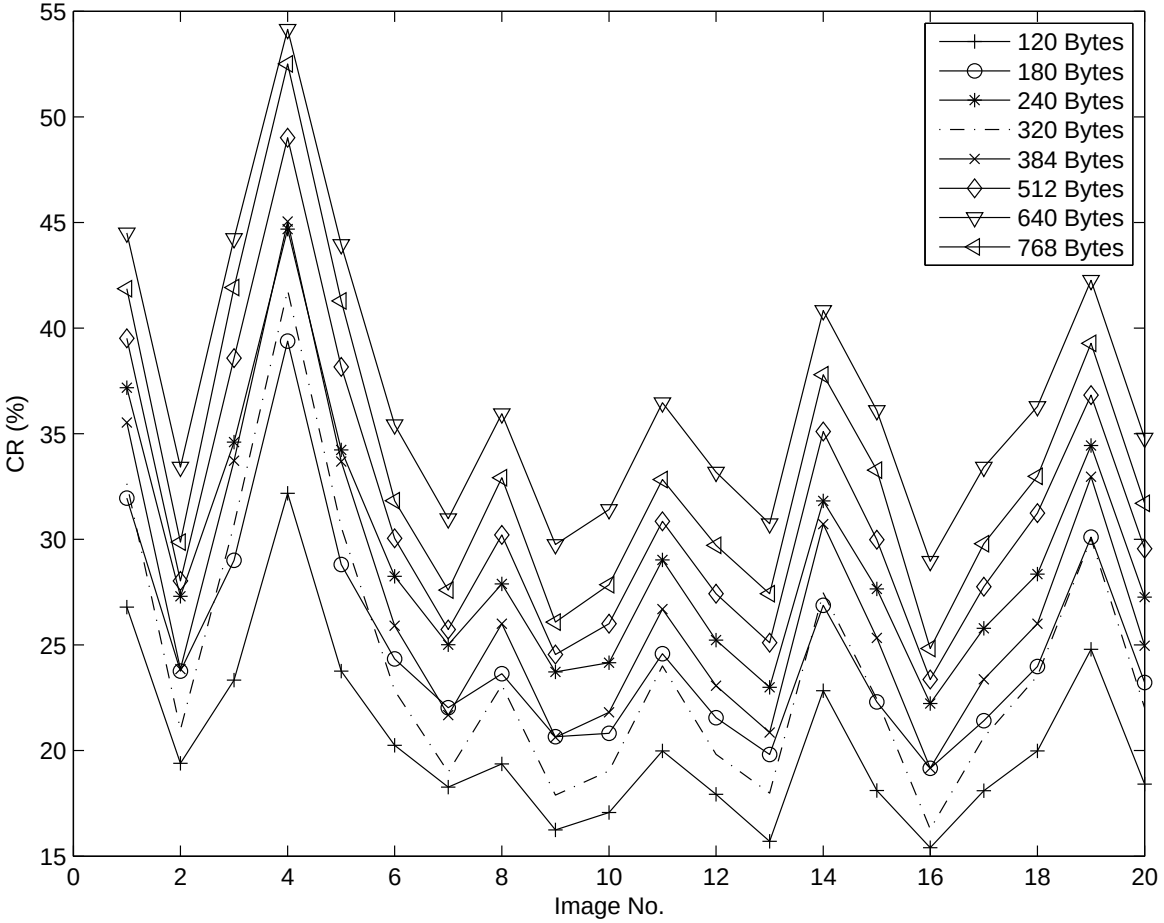


Figure 3.8: Effect of different library size on CR(%)

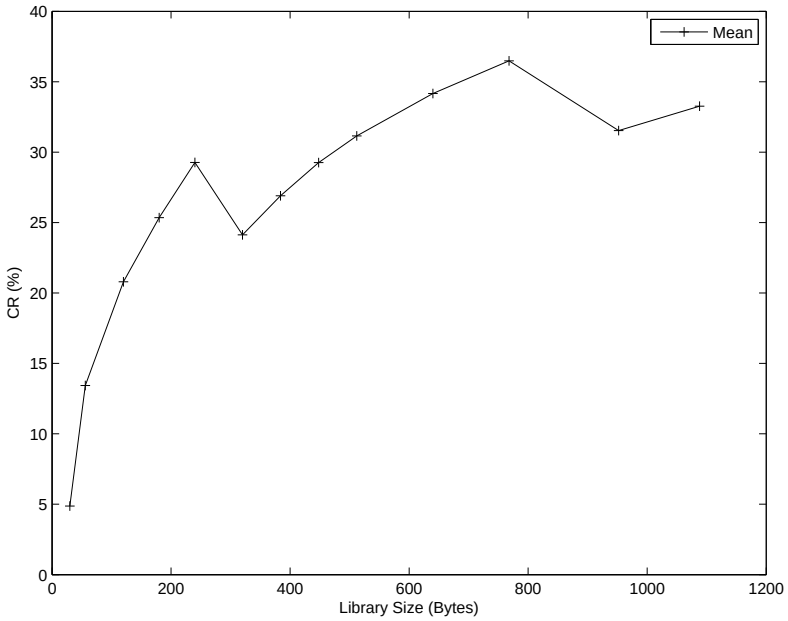


Figure 3.9: Effect of library size on mean CR for 20 endoscopic colour-scale images

effects of different library sizes for LZW–Flush were accomplished and the best library size was chosen.

CHAPTER 4

HARDWARE IMPLEMENTATION

4.1 Introduction

In the previous chapter, the simulation results of different encoding algorithms were presented and it was observed that the LZW–Flush provides better performance compared to other methods. In this chapter, the hardware implementation of the LZW–Flush algorithm is presented. The LZW–Flush implementation includes the Content-Addressable Memory (CAM) and a state machine to compress the input stream. The FPGA code is written in Verilog. The overall block diagram of the LZW–Flush encoder is also presented.

4.2 Hardware Architecture

The block diagram of LZW–Flush is shown in figure 4.1. The input stream is connected to *input* register. This register is updated in each clock when its *En* pin is set to high. The *Word* register contains the previous match index. It is concatenated to the *input* and is fed to the input of the *Dictionary*. Dictionary is a CAM which records the observed patterns and is updated whenever a new pattern is arrived. Whenever the input stream matches a row of the dictionary, the *matched index* is updated with the content index. In addition to this update, the *controller* updates the *output* register with the previously matched index,

word.

The *matched flag* determines the occurrence of a match. The *Dictionary* can be cleared, read or written by *Flush Dic* and *W/R* pins respectively. Whenever the *Dic. Full flag* is set, it is not able to record any more patterns. In case the dictionary gets full, the *controller* sets the *flush dic* signal to clear the dictionary in order to record the coming patterns. The *controller* controls the entire compression process and throughputs the compressed symbols. In the following sections, the details of the Dictionary and controller components are described.

4.2.1 Dictionary

In order to record and retrieve patterns, memory is needed. Two types of RAM can be used for this purpose which include SRAM and CAM. SRAM can not be considered a good candidate since it requires large numbers of clock cycles in order to search through the RAM and discover the matching pattern. The search time also depends on the library size. If we have a modified hash based SRAM, it needs few clock cycles [34]. CAM is a special memory architecture. Rather than providing an address to retrieve the data, it gets the data, and returns the address back. This operation only consumes one clock cycle. In CAM, rather than having few clock cycles, the index can be accessed by just one clock cycle. Obviously it needs more power and occupies more area. But in such an application, its a need to use this type of memory due to the time constraint that exists for data compression.

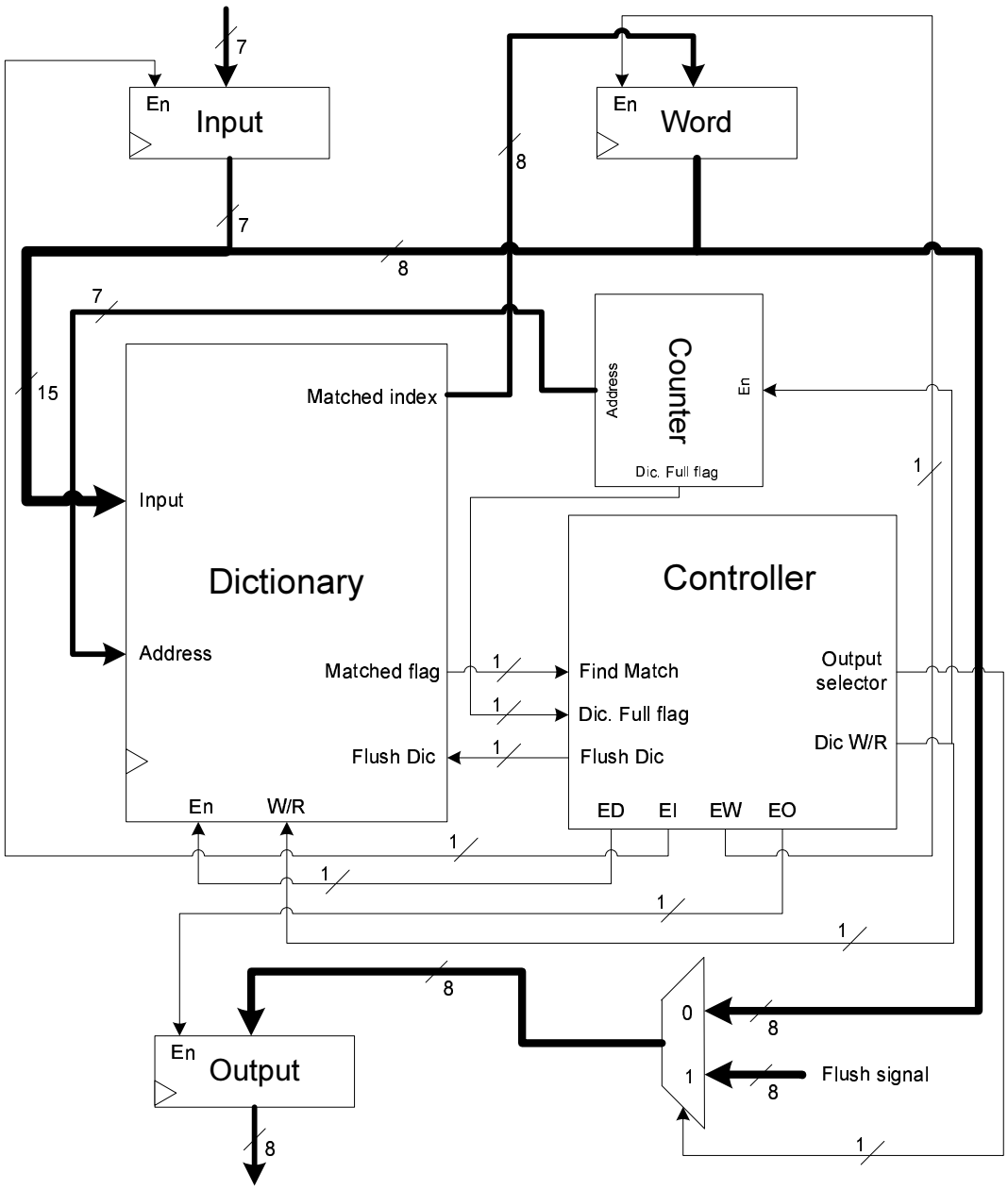
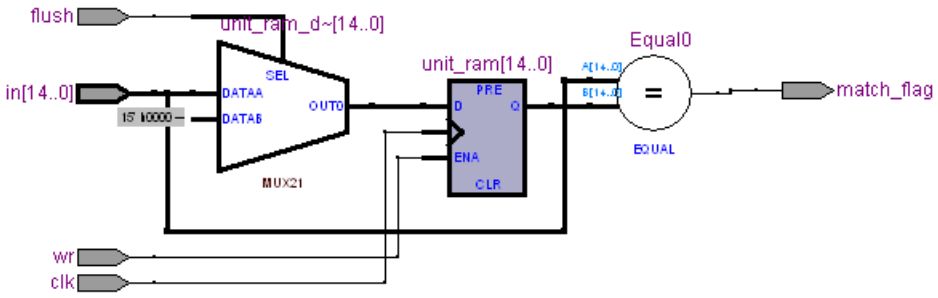
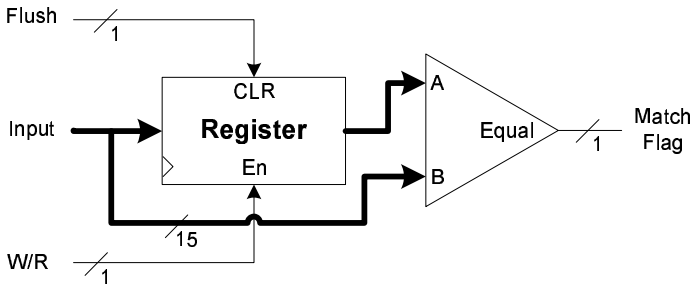


Figure 4.1: Hardware block diagram of the LZW–Flush encoder



(a) RTL view



(b) Hardware architecture

Figure 4.2: Hardware architecture and RTL view of a CAM cell

The architecture of the implemented cell of a typical CAM is shown in figure 4.2. It records 15 bits which is divided into 7 bits for input stream and 8 bits for the index. If the flush bit is set to high, the register content gets cleared. The W/R pin is connected to activate the D-latch register pin. If this pin is low, it prohibits the write operation for D-Latch, and the content of the register is compared with input. In case both are the same, the match flag is set to high. In the other cases, it is always assigned to low.

To extend the size of the dictionary to 128 cells, the architecture depicted in figure 4.3 is proposed. The decoder takes the input address and decodes it to W/R pin of related cell. The encoder takes all match flag pins as input and produces an eight bit length output

which contains the address of the proper matched index cell.

The match-index is shifted to above 128. The reason behind this shift is that the first 128 (0 to 127) are used for the purpose of predefining characters. (refer to LZW algorithm description). This operation can be easily done by a left shift of match index.

4.3 Controller

The controller is composed of two step state machine and some logics, controls the dictionary and the peripheral components. The architecture of this controller is depicted in figure 4.4. It has eight output pins and two inputs. If ED pin is activated, the Dictionary is enabled, otherwise if the EI pin is activated, the input register gets enabled. The EW and EO enable the word and the output register respectively. Dic W/R selects the operation of the dictionary. If it is low, it forces the dictionary to operate in read mode and when it is set to high, the dictionary accepts writing operation. The output selector selects between the flush signal and Compressed symbol. Flush dictionary clears the dictionary contents. Dic Full Flag, is an input signal which comes from counter and shows that the dictionary is full. Find match tells the controller the combination of input register and word which is found inside of the dictionary. Word-match selector selects the output. The next clock turns back the state level to state number one.

Controller is based upon the match flag pin and changes its state. In case it is set to high, a match has been found in the dictionary. Thus, it enables the input register to update

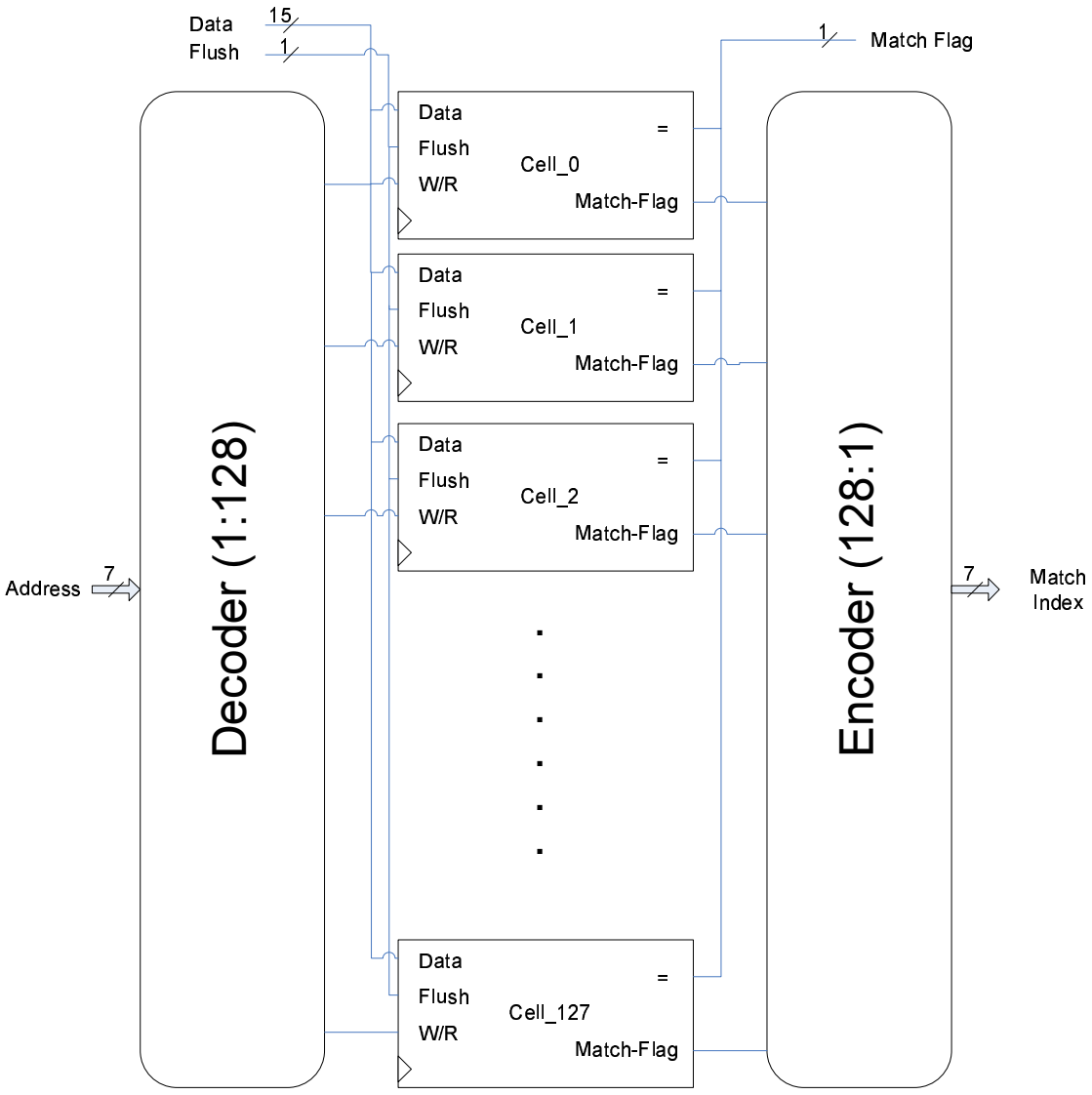


Figure 4.3: The architecture of entire proposed dictionary

with new symbol of input stream in next clock. It also enables word register to update with the match index. In the occurrence of a match condition, the write enable pin of the output register is cleared. If there is no match, the match flag is set to low, and the controller enables the input register to update with the new symbol of the input stream in the next clock. Hence, the word-matched selector selects input and feeds the word input by the content of the input register. W/R is set to high which activates the writing operation of the dictionary. It also enables the output register. If the counter does not pull the flush dic to high, the output selector selects the word register. In the next clock, the state machine steps to the second state and all registers are updated.

If in the first step the Dic full flag is set to high, the *flush dic* of Controller is enabled, the dictionary is flushed, and the flush signal is selected by the output selector. Hence, input register and word are not enabled, and while the state machine is entered to second state, their values are not updated. Therefore, the next clock sends back the state machine to state number one, and the process starts from the beginning as described earlier.

4.4 Counter

The architecture of the counter is shown in figure 4.5. It is a simple 7-bit length counter. It counts from 0 up to 127. It is enabled when the W/R pin of the controller is one. The overflow pin is set if there is an overflow, and in the next clock, it is cleared by the controller. The counter output is connected to the address pins of the dictionary. The reset

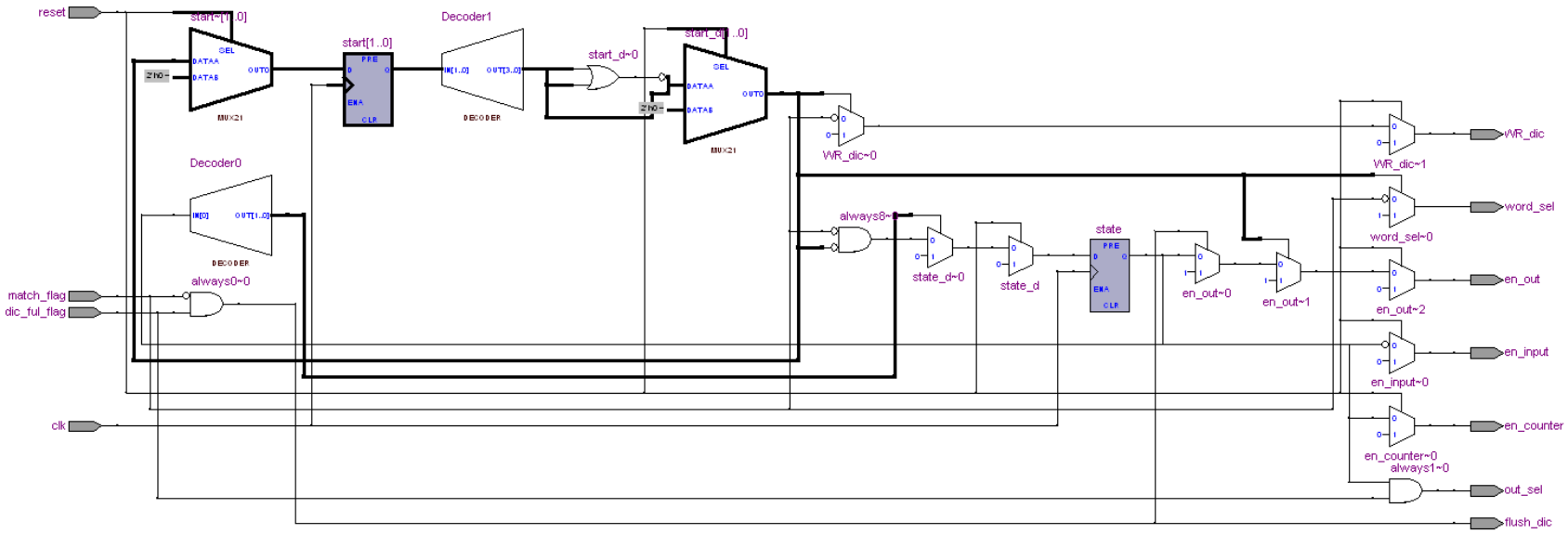


Figure 4.4: The architecture of the proposed controller

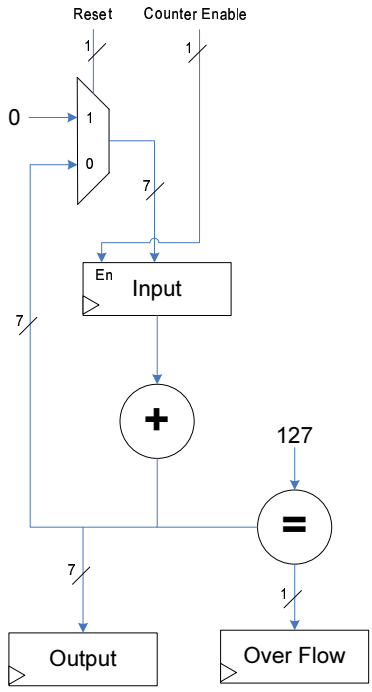


Figure 4.5: The architecture of the counter

pin, resets the counter.

4.5 Summary

This Chapter discussed a hardware implementation of LZW–Flush. An overall block description was provided as well as an in depth discussion of the implementation. The implementation was broken down into three parts: Dictionary, Controller, and Counter. Each part was discussed in detail. In the next chapter, the performance of the encoder methods discussed in chapter 2 is evaluated, and the proper encoder for the application is proposed.

CHAPTER 5

PERFORMANCE EVALUATION

5.1 Introduction

In Chapter 4, the simulation results of different encoding algorithms were presented and it was observed that LZW–Flush provides better compression ratio compared to the other methods. In this Chapter, we are going to evaluate the performance of the algorithms which provided us with highest compression results. The evaluations are accomplished with regard to the library size, memory consumption, and the simplicity of hardware architecture. In the last part of this chapter, we provide some of the test images, fed into the proposed compression system along with their corresponding decoded images.

5.2 Hardware Comparison

As it was discussed earlier, the LZW–Flush encoding algorithm provides better compression ratio compared to the other methods provided in previous chapter. Table 5.1 presents the average compression ratio of the 20 simulated images for different encoding algorithms. As it can be observed from the table, the encoding methods which guarantee the best compression ratios for the tested images respectively includes: LZW, Lossless JPEG, Arithmatic, Static Huffman, LZW–Flush, LZ78, and Adaptive Huffman. Therefore, we

Table 5.1: The average CR of 20 medical images for different encoding algorithms

	Arith.	Stat. Huff.	Var. Huff.	LZW	LZ78	LZ77	LZW Flush	RLE	L.L JPEG
CR (%)	54.95	54.52	9.75	64.60	46.91	7.35	49.40	-14.59	57.75

established the performance evaluation on these algorithms to detect the encoding method that best suits the compression system and provides better overall performance in terms of simplicity of hardware architecture. It can be noticed from the results shown in the table that the compression results of the Arithmetic, Huffman, LZW, LZ78, and LZW–Flush are better than the rest. Therefore, the evaluations were done to investigate the performance of these algorithms in terms of hardware.

Before we go through further investments, it is important to mention that the LZW algorithm is not a perfect choice for the compression system. The reason behind this argument is that LZW needs a large library to accomplish the encoding [35, 36, 37, 38]. The mean size of the library used for the experiment performed sums up to 8.1 KByte. Thus, the RAM power consumption as well as the area are large for such an application. The static Huffman method has some basic issues as well. First, this algorithm needs two passes to do the encoding which reduces the speed and thus decrease the performance. In addition, this method requires a large RAM to store the image. For an image of size 256×256, the Huffman algorithm needs more than 65 KB of memory which is a big number. Arithmetic encoding method would not be a good choice due to the algorithm hardness. It requires several multiplication to encode the input stream. Lossless JPEG

exploits Arithmetic or Static Huffman as the part of the entropy encoder, which makes it difficult to be implemented in hardware. The RLE algorithm expands the input stream due to not having much runs in the input stream.

Table 5.2: Hardware comparison of different encoder

	Scheme	LUT	Registers	RAM
Proposed 1	LZW–Flush	2624	1947	0
Proposed 2	LZW–Flush	966	49	1920
Rigler et al. [39]	Huff1	2066	573	40234
Rigler et al. [39]	Huff2	1980	533	4410
Rigler et al. [39]	Huff3	1842	502	2969
Jamro et al. [40]	Var. Huff.	3007	1042	4112
Rigler et al. [39]	LZ	2077	734	8192
Cui [41]	LZW	6436	–	272 K
Abdelghany et al. [42]	LZ	419	408	1040

Therefore, the focus was put on the rest of the algorithms which would shorten the list to LZ78, LZW-Flush, and variable Huffman. While LZ78 requires a larger library compared to LZW–Flush, but the difference in terms of algorithm simplicity is not much. The hardware implementation of both methods is almost the same while LZW–Flush provides better results in terms of the compression. The only algorithms remained to be compared, are variable Huffman encoding and LZW–Flush. The hardware details of the variable Huffman encoding is presented in [39, 40]. These details along with the hardware

implementation details of the LZW–Flush which was presented in Chapter 4, are presented in table 5.2. The presented data for LZW-Flush proposed 1 is simulated on FPGA Board Cyclon II EP2C35F484C7. Proposed 2 is simulated on FPGA Board APEX20KE EP20K30ETC144. In proposed 1, the dictionary is made of the CAM architecture described earlier. In proposed 2, the dictionary takes advantage of built-in CAM memory of the FPGA board. As it can be observed from the table, variable Huffman encoding requires more hardware compared to LZW–Flush. Thus, from the arguments and evaluations, we can justify that LZW–Flush guarantees better performance compared to the other approaches as it supports higher compression ratio, and smaller hardware cost.

5.3 Reconstructed Images

In the previous section, it was discussed that LZW–Flush provides the best functionality. In this section, four out of the twenty images are randomly chosen and shown along with their reconstructed images. Figure 5.1 shows the process of reconstruction from the original image. Figures 5.2,5.3, 5.4, and 5.5 present these images along with their reconstructed images achieved through the LZW–Flush based compression system.

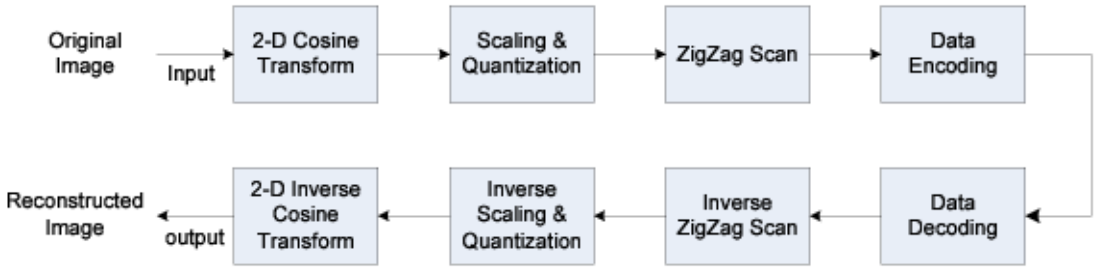
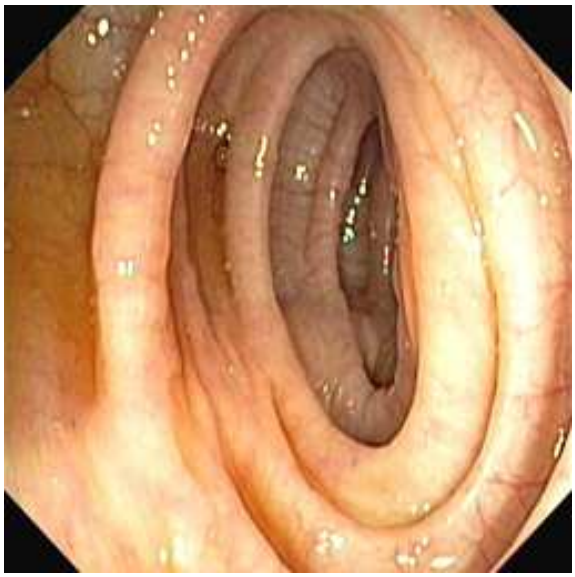


Figure 5.1: The image reconstruction process block diagram

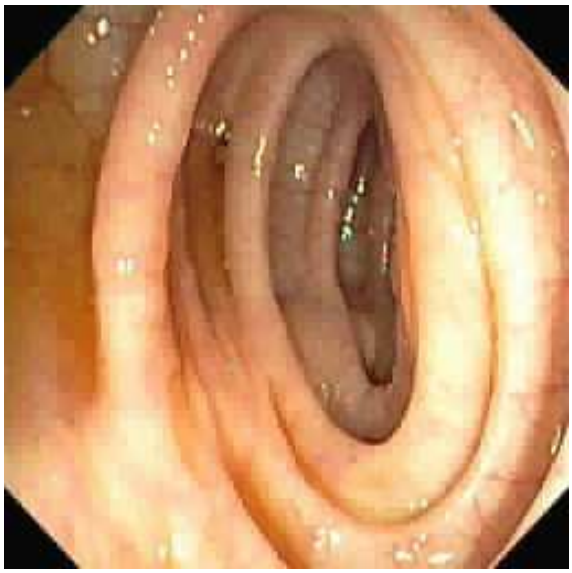
(a) Original

(b) Reconstructed

Figure 5.2: Image 8 PSNR = 30.914 dB

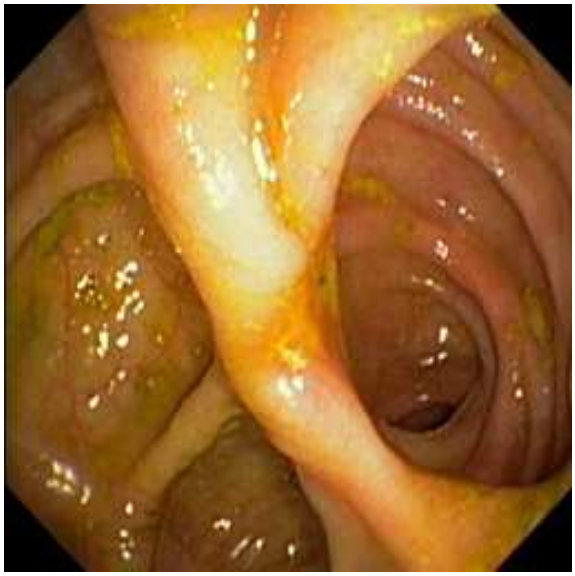


(a) Original

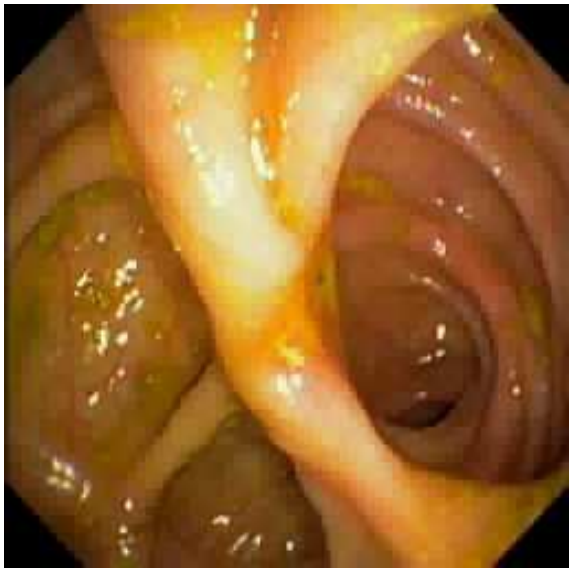


(b) Reconstructed

Figure 5.3: Image 9 PSNR = 30.409 dB

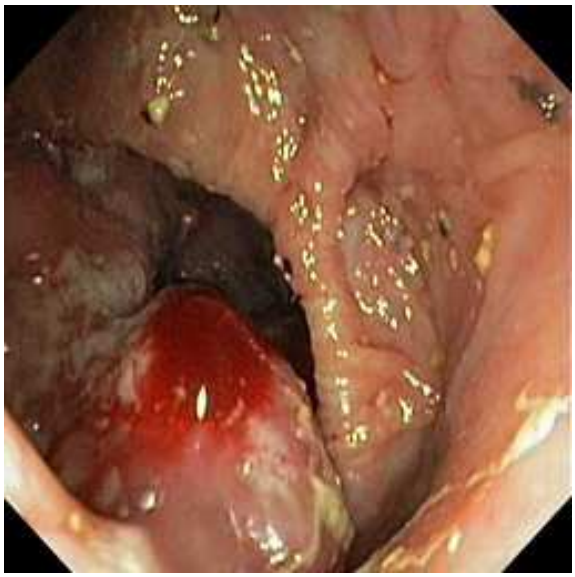


(a) Original



(b) Reconstructed

Figure 5.4: Image 13 PSNR = 32.249 dB



(a) Original

(b) Reconstructed

Figure 5.5: Image 16 PSNR = 30.271 dB

CHAPTER 6

LOSSY LZW–FLUSH

6.1 Introduction

In endoscopy applications, the quality of captured images plays an important role in diagnosing a disease. Higher quality images require higher level representative data. As mentioned in the first chapter, there are two concerns; power consumption, and area usage. Therefore, streaming of high quality images is somehow impossible. In such applications, the streaming of high quality images is not considered an issue, rather, the quality of the image taken from the interested zone is important. If by a simple change in the data encoder (tolerable hardware cost), the quality of images becomes under control, the target would be achieved. In this chapter, a filter is proposed to investigate this problem. Later, the algorithm and the proposed hardware are presented.

6.2 Lossy LZW–Flush

6.2.1 Weber–Fechner Law

The actual magnitude of a physical quantity measured by a proper instrument is not the same as the perceived one. For example, assume a person is asked to hold a 1 Kg package

of sand for a few seconds. In case a few seconds later, the sand pack is replaced with another one weights 1.01 Kg, that person can not discriminate the weight difference of the two packs. Weber–Fechner attempts to define the relationship of physical magnitude and their perceived one [43]. In case of vision, it is discovered that the eye senses brightness approximately logarithmically [44]. This means that the discrimination boundary of the human vision, i.e. the threshold, varies logarithmically with respect to the intensity range.

In [45], the intensity range versus threshold curve is depicted. By taking advantage of the Weber–Fechner feature, a set of pixels having a non-differentiable range of intensity (Weber's law) can be converted to a set of pixels with the same intensity. For example, assume that we have three different intensities of $\{30, 31, 28\}$. If these three intensities are converted to $\{30, 30, 30\}$, a human eye can not differentiate between the two sets from each other. Hence, the redundancy of some intensities are increased in the image. In the following section, taking advantage of this feature, a new filter is designed to exploit the Weber–Fechner law to increase the redundancy of image data.

6.2.2 Algorithm

In the following, the proposed filter is presented. The pixel intensity of the first symbol of input stream is taken as reference and is stored in r. Then, the absolute difference of the next symbol and the reference is calculated and stored in *d*. If *d* is less than or equal to the defined threshold (calculated from weber's law), the symbol value in the input stream is replaced with *r*, in other cases *r* takes the symbol value. This process is carried on for the rest of symbols in the input stream.

As mentioned in the previous section, there are different thresholds for different intensity ranges. These thresholds are logarithmically related to the intensity ranges. Therefore, to find the proper threshold, there is a need to implement the intensity range–threshold curve on a lookup table. The lookup table size for the full range is 256 bytes. To simplify the process and get rid of the adaptive threshold, a fixed threshold is considered for all ranges. The flowchart of this algorithm is depicted in figure 6.1. The pseudo code for the algorithm is shown in the following:

1. take the first symbol from input stream, and store it as reference, r
2. take the next symbol, x
3. if $|r - x| = d \leq threshold$ then replace x with r
4. if $|r - x| = d \geq threshold$ then replace r with x
5. if the input stream is not exhausted get back to step 2

In the following, an example is given which presents how the filter algorithm works. In this example, the input stream is $\{15, 29, 28, 20, 14\}$ and Threshold is tr = 3.

- $r \leftarrow 15$ and $x \leftarrow 29$
- $|r - x| = |15 - 29| = 14 > 3 \Rightarrow r \leftarrow 29, x \leftarrow 28$
- $|r - x| = |29 - 28| = 1 < 3 \Rightarrow r \leftarrow 29, x \leftarrow 20$

- $|r - x| = |29 - 20| = 9 > 3 \Rightarrow r \leftarrow 20, x \leftarrow 14$
- $|r - x| = |20 - 14| = 6 > 3 \Rightarrow r \leftarrow 14, (nomoreinputsymbol)$
- filtered output = $\{15, 29, 29, 20, 14\}$

6.2.3 Proposed Lossy LZW–Flush

The architecture of the proposed lossy LZW–Flush is shown in figure 6.2. Lossy LZW–Flush is composed of the proposed filter and the LZW–Flush encoder. This algorithm is called lossy as the filter pixels can not be retrieved back in the reconstruction process. The proposed compressor, has two parameters which affect the compression results; the filter threshold value, and the LZW–Flush library size. To study the effect of each parameter, 20 medical images are encoded by the proposed data compressor.

The threshold value affects the PSNR and CR of the reconstructed image. The simulation results are depicted in figures 6.3, and 6.4. In figure 6.4, five arbitrary images are chosen to show the effect of the threshold on CR. In this simulation, the library size is fixed to 4096 bytes. Figures 6.5, 6.6, and 6.7 show three different medical reconstructed sample images. In each figure, the effect of the threshold on the reconstructed images can be observed.

As it can be seen from the figures, when the threshold exceeds 5, the art effect starts to appear significantly. In other aspect, threshold 5 results in acheiving the mean PSNR above 41.04dB which provides satisfying results which are also acceptable by medical

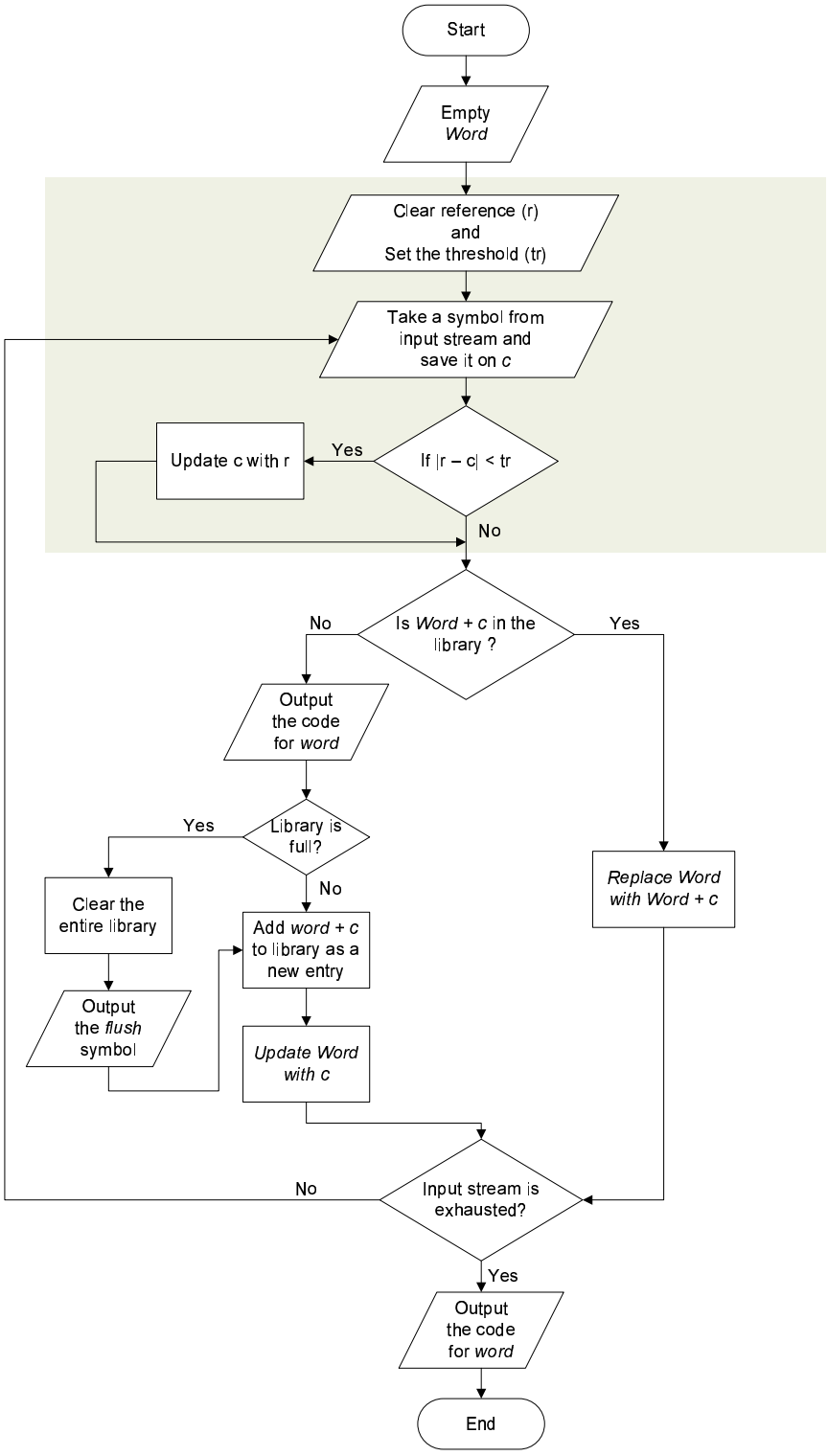


Figure 6.1: The lossy LZW–Flush algorithm flow chart

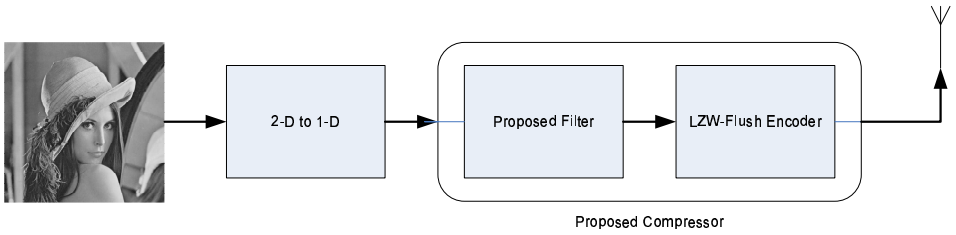


Figure 6.2: Block diagram of proposed lossy LZW–Flush

phicisions. By increasing the threshold, the compression ratio is increased respectively (figure 6.4).

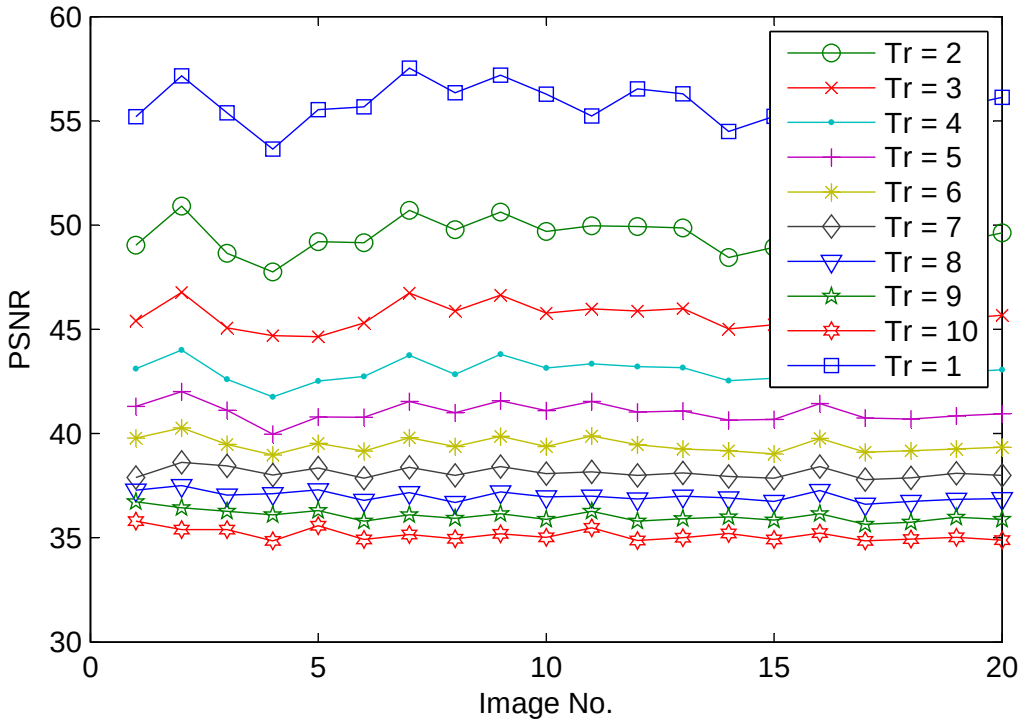


Figure 6.3: Effects of *Tr* on PSNR

The hardware architecture of the proposed filter is shown in figure 6.8. It consists of a register, two comparators, two adders, and three multiplexers 2:1. The register has two functionalities which include working as the reference register as well as holding

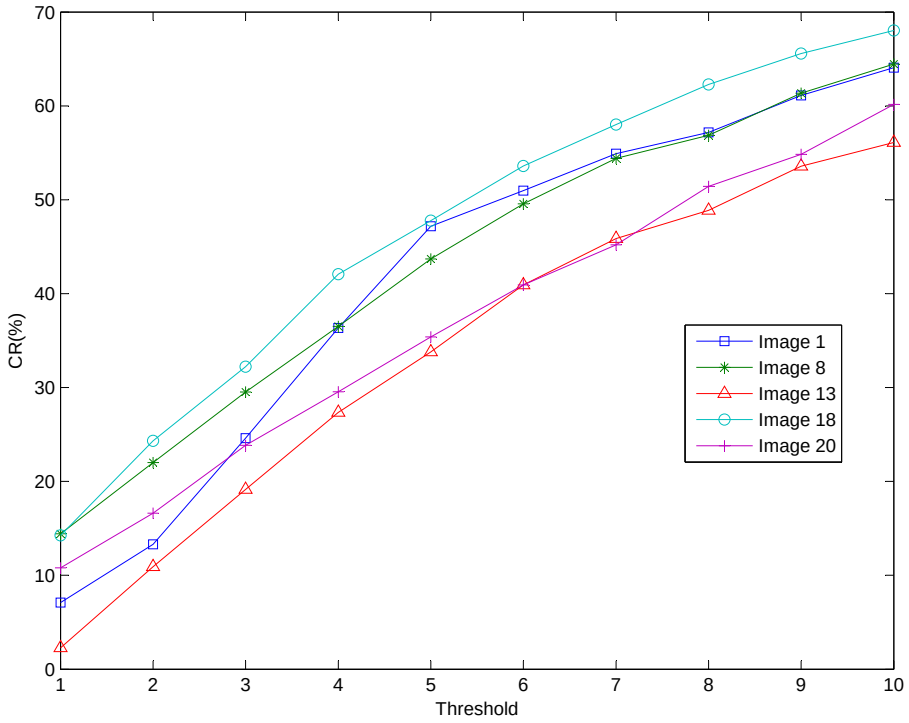


Figure 6.4: Effects of *Tr* on CR of five arbitrary capsule endoscopy images

the filtered input symbol. The reset pin clears the content of the output register. The threshold value is added to and subtracted from the output register value and respectively is compared with the input symbol by two comparators. If the input symbol is not in the defined boundary, the output register is updated with the input symbol.

6.3 Performance Evaluation

In [45], Chiang,el proposed a new lossy LZW. In their work, they exploit Weber's law and apply the threshold to the index of the LZW library. In order to implement that, an exhaustive search through the dictionary is required. From the hardware implementation aspect, it obviously needs more significant hardware resources compared to the proposed lossy LZW–Flush. In table 6.1, the CR and the PSNR of two standard images are shown.

Table 6.1: Compression performance of the proposed lossy encoder

Image Name	Chiang et al. [45]			Proposed	
	Lossless LZW	Lossy LZW		Lossy LZW–Flush	
	CR(%)	CR(%)	PSNR(dB)	CR(%)	PSNR(dB)
Lena	7.06	43.68	33.58	45.84	38.10
Air-plane	15.22	66.47	34.90	67.83	32.16
Baboon	3.83	27.18	29.31	26.44	36.41

Table 6.2: Hardware cost of lossy LZW-Flush algorithm

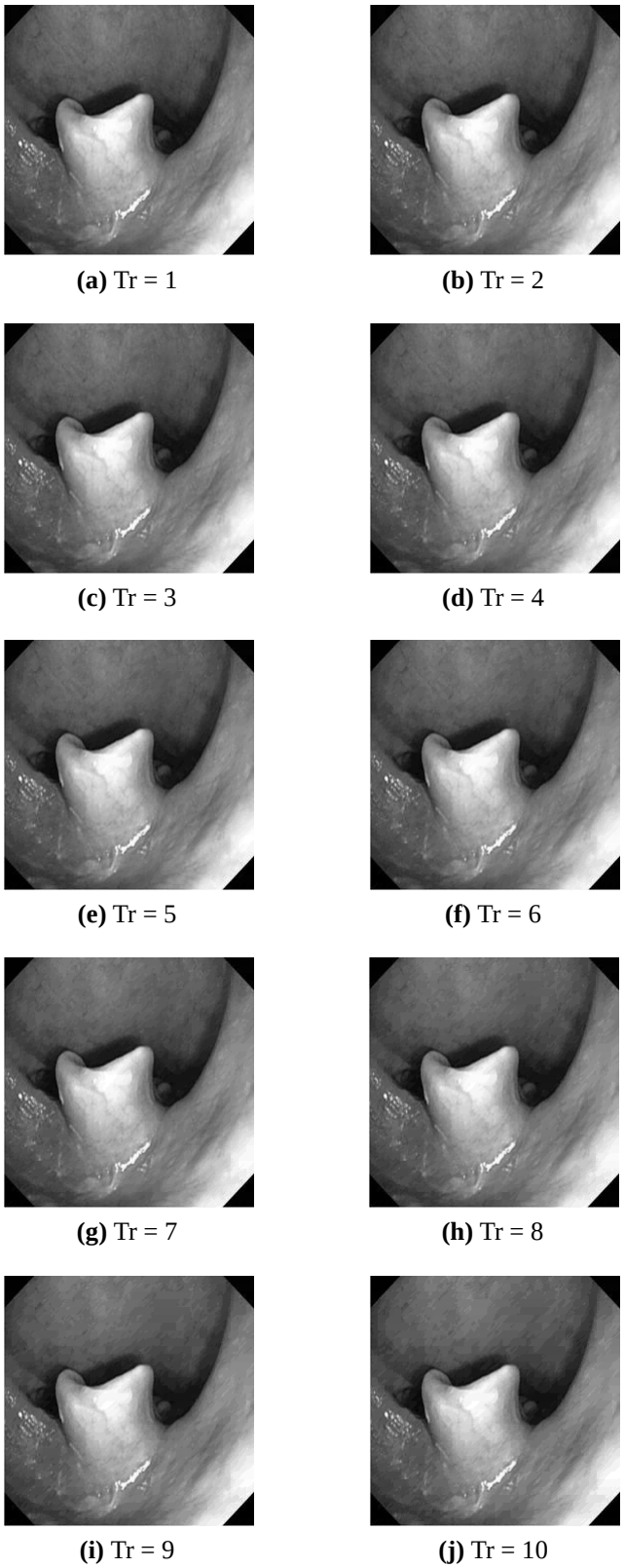
Technology	Logic cells	Registers	Freq.	Power
Cyclone II	2662	1954	69.05 MHz	35.6 mW

As it can be observed, the Chiang's method provides better results in terms of CR, while both algorithms' PSNR are about the same.

In this regard, due to having a simple algorithm, a simple hardware architecture is expected. Therefore, the proposed method would be a good candidate in application but it has limitations of power consumption and area usage. Table 6.2 shows the hardware cost of the implementation of this algorithm on to the Cyclone II FPGA. Figure 6.9 shows the reconstructed images which were compressed by the proposed algorithm.

6.4 Summary

This chapter proposed a new filter which increases the redundancy of the input stream. The composition of this filter with the proposed LZW-Flush results in a novel lossy compression algorithm. The effect of this algorithm was evaluated and simulated with over 20 medical images. The hardware architecture and the cost of this algorithm were also presented. The results guarantee that this algorithm would be a good solution for those applications which require relatively high quality but also have the limitations of resource consumption, area and power usage.



(a) Tr = 1 **(b)** Tr = 2

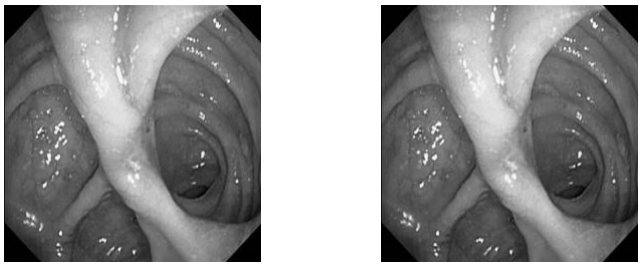
(c) Tr = 3 **(d)** Tr = 4

(e) Tr = 5 **(f)** Tr = 6

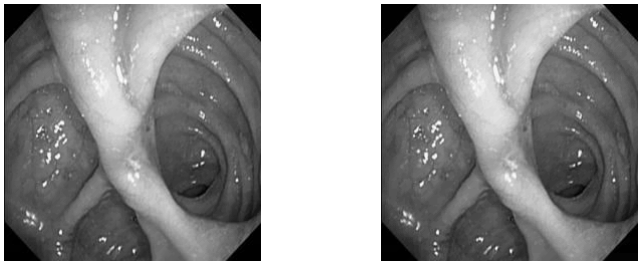
(g) Tr = 7 **(h)** Tr = 8

(i) Tr = 9 **(j)** Tr = 10

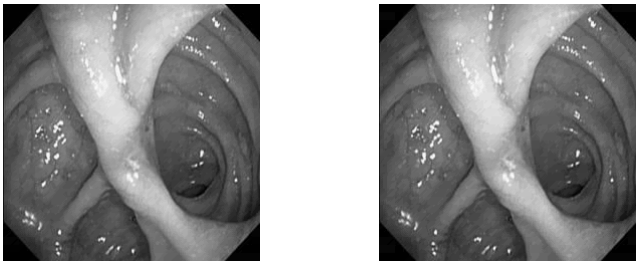
Figure 6.5: Gray scale of image 6 (Tr = 1 to 10)



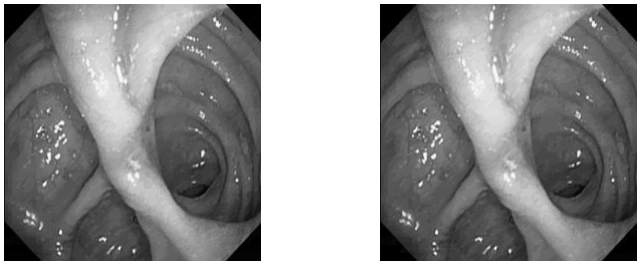
(a) Tr = 1 **(b)** Tr = 2



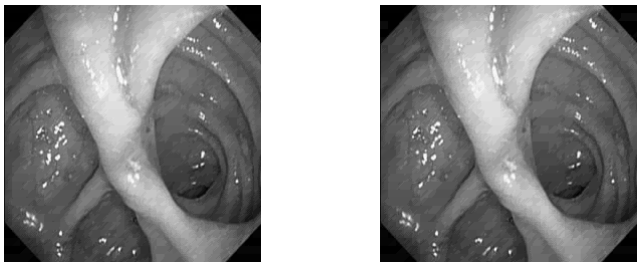
(c) Tr = 3 **(d)** Tr = 4



(e) Tr = 5 **(f)** Tr = 6

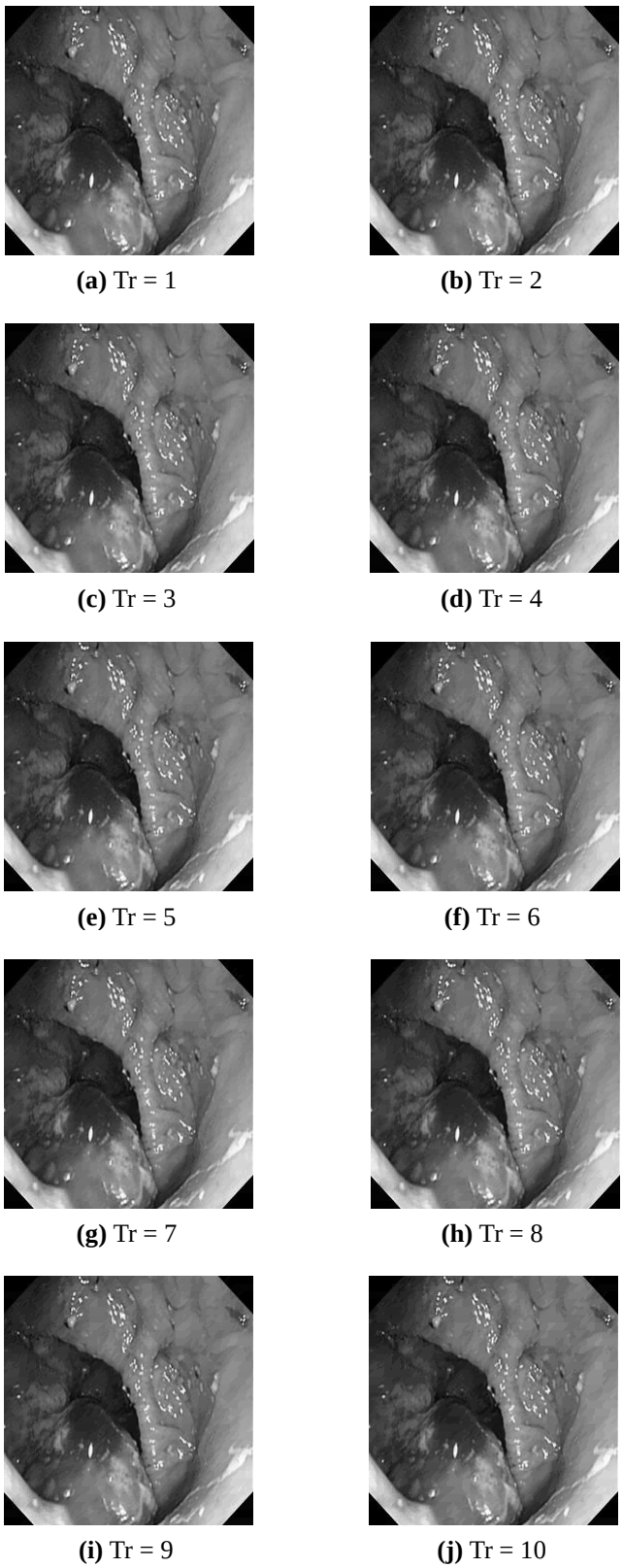


(g) Tr = 7 **(h)** Tr = 8



(i) Tr = 9 **(j)** Tr = 10

Figure 6.6: Gray scale of image 13 (Tr = 1 to 10)



(a) Tr = 1 **(b)** Tr = 2

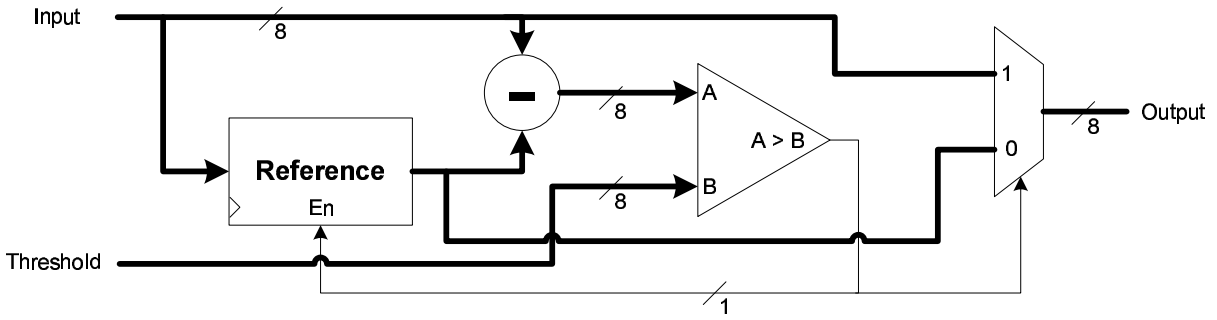
(c) Tr = 3 **(d)** Tr = 4

(e) Tr = 5 **(f)** Tr = 6

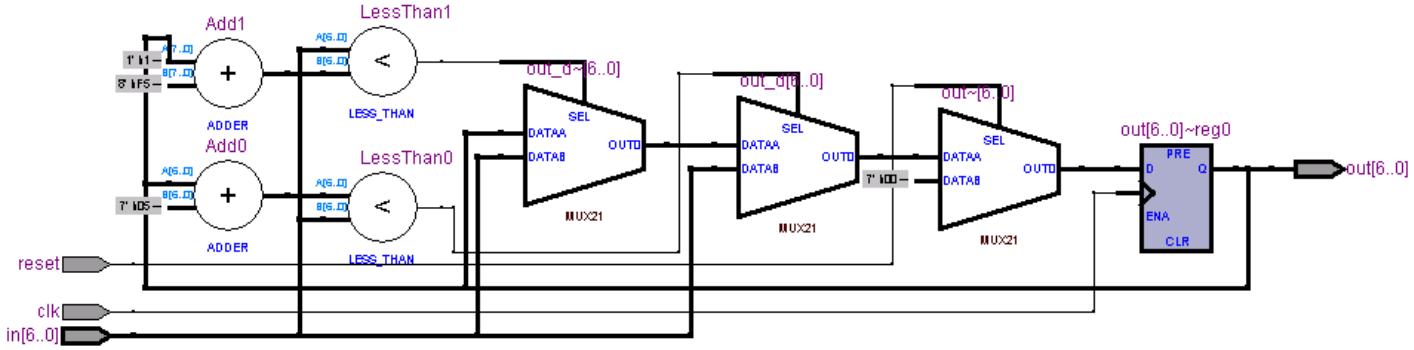
(g) Tr = 7 **(h)** Tr = 8

(i) Tr = 9 **(j)** Tr = 10

Figure 6.7: Gray scale of image 16 (Tr = 1 to 10)



(a) Hardware architecture



(b) RTL view

Figure 6.8: Proposed filter hardware architecture and RTL view

Figure 6.9: Reconstructed colour image Lena

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Summary Of Accomplishments

Capsule endoscopy examines some parts of the tracts that can not be discovered using other types of endoscopies. The architecture of such a capsule includes an image compression system which is responsible for doing transformations as well as making compressions using encoding algorithms. In this thesis, we put forward a comparative study of different encoding algorithms and their differences in terms of compression ratio, reconstructed image quality, and the algorithm simplicity which result in a potential encoder exploiting low power and area consumption.

Different encoding algorithms, including Huffman encoding, LZW, LZW–Flush, Arithmetic, LZ77, LZ78, and JPEG were introduced and a study of each approach was presented. Later, a new zigzag scan method was presented and applied to the image compression component. The results of the proposed scan method along with different encoding algorithms were simulated using different images. LZW–Flush presented better compression ratio compared to other approaches. Thus, LZW–Flush was chosen as the encoder providing higher compression rates and its hardware was implemented to investigate the

hardware cost of this method.

The compression ratio results of other approaches; LZW, Huffman encoding, LZ78, and LZW, were not significantly different from LZW–Flush. Thus, evaluations were done to compare the hardware cost, power consumption, and area usage of these methods. The evaluation results showed that in overall, still LZW–Flush provides better performance in terms of all these parameters. The claims behind this argument can be summarized as follows:

- The reconstructed images provide high quality in terms of PSNR. Thus, the first objective of selecting a high qulaity encoding algorithm was achieved.
- LZW–Flush has the highest compression rate compared to other methods. Therefore, as less data is transmitted, the amount of power consumption is less.
- It requires an affordable size of RAM to accomplish the encoding.
- Due to the simplicity of the algorithm and its low hardware cost, LZW–Flush would be a potential candidate having low power and area consumption.

As the encoding approaches used in the simulation phase were all lossless compression techniques, we later studied the effects of applying lossy LZW–Flush approach to the compression system. The experiments showed that with the addition of a small tunable hardware to the lossless LZW–Flush, an encoder is achieved which has a compression ratio better than lossless. The results provide high quality reconstructed images having satisfiable compression ratio.

7.2 Future Works

As the implementation has been done in FPGA, our future trends are toward VLSI implementation of the encoder to find the actual power consumption and area usage. Different search techniques are to be investigated in order to discover the optimal method which best fits these types of applications in terms of VLSI design, hardware implementation, and latency. In order to extend the research accomplished on lossy encoding, different schemes of the applied filter will be investigated with the purpose of reordering data toward increasing pattern redundancy. This we believe would help in increasing the compression ratio of the lossy approach.

REFERENCES

[1] K. Wahid, S.-B. Ko, and D. Teng, "Efficient hardware implementation of an image compressor for wireless capsule endoscopy applications.," in *IJCNN*, pp. 2761–2765, 2008.

[2] Zver, "http://projects.hudecof.net/diplomovka/online/ucebnica/html/alg_lz77.html/."

[3] I. M. Pu, *Fundamental Data Compression*. Butterworth-Heinemann, 2006.

[4] W. Kinsner and R. Greenfield, "The Lempel-Ziv-Welch (LZW) data compression algorithm for packet radio," in *IEEE Western Canada Conference on Computer, Power and Communications Systems in a Rural Environment*, pp. 225–229, 1991.

[5] "http://nobelprize.org/nobelprizes/physics/laureates/1901/rontgen-bio.html."

[6] A. Meyer-Bse, *Pattern recognition for medical imaging*. Academic Press, 2003.

[7] Wikipedia, "http://en.wikipedia.org/wiki/medicalimaging."

[8] J. T. Bushberg, J. A. Seibert, E. M. L. Jr., and J. M. Boone, *The Essential Physics of Medical Imaging*. Lippincott Williams and Wilkins, 2001.

[9] Wikipedia, "http://en.wikipedia.org/wiki/endoscopy."

[10] G. Iddan, G. Meron, A. Glukhovsky, and P. Swain, "Wireless capsule endoscopy," *Nature*, vol. 405, no. 6785, p. 417, 2000.

[11] H. Kotze and G. Kuhn, "An evaluation of the Lempel-Ziv-Welch data compression algorithm," in *Proc. of the Southern African Conf. on Comm. and Signal Proc.*, pp. 65–69, 1989.

[12] P. Cosman, R. Gray, and R. Olshen, "Evaluating quality of comppressed medical images: SNR, subjective rating, and diagnostic accuracy," *Proceedings of the IEEE*, vol. 82, no. 6, pp. 919–932, 1994.

[13] D. Salomon, *Data Compression*. Springer, 3rd ed., 2004.

[14] F. Gong, P. Swain, and T. Mills, "Wireless endoscopy," *Gastrointestinal endoscopy*, vol. 51, no. 6, pp. 725–729, 2000.

[15] M. J. Weinberger, G. Seroussi, and G. Sapiro, "From logo-i to the jpeg-ls standard," in *Proc. Int. Conf. Image Processing ICIP 99*, vol. 4, pp. 68–72, 1999.

[16] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 22, pp. 379–423, 1948.

[17] J. Trein, A. T. Schwarzbacher, B. Hoppe, and K. H. Noff, "A hardware implementation of a run length encoding compression algorithm with parallel inputs," in *Signals and Systems Conference*, pp. 337–342, 2008.

[18] D. Huffman, "A method for the construction of minimum redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098 –1101, 1952.

[19] T. Kumaki, Y. Kuroda, T. Koide, H. Mattausch, a. K. D. H. Noda, K. Arimoto, and K. Saito, "CAM-Based VLSI architecture for huffman coding with real-time optimization of the code word table," in *IEEE International symposium on Circuits and Systems*, vol. 5, pp. 5202–5205, 2005.

[20] "http://anupom.wordpress.com/2006/10/09/huffman-coding/."

[21] M. L. Lu and C. F. Chen, "An encoding procedure and a decoding procedure for a new modified huffman code," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 38, no. 1, pp. 128–136, 1990.

[22] "ISO \ IEC 10918 Information Technology," 1994.

[23] W. W. Lu and M. Gough, "A Fast-Adaptive Huffman coding algorithm," *IEEE Transactions on Communications*, vol. 41, no. 4, pp. 535–538, 1993.

[24] J. S. Vitter, "Design and analysis of dynamic huffman coding," *26th Annual Symposium on Foundations of Computer Science*, vol. 34, no. 4, pp. 825–845, 1985.

[25] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337–343, 1977.

[26] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Transaction on Information Theory*, vol. 24, no. 5, pp. 530–536, 1978.

[27] T. Welch, "A technique for high-performance data compression," *Computer*, vol. 17, no. 6, pp. 8–19, 1984.

[28] S. Bunton and G. Borriello, "Practical dictionary management for hardware data compression," *Communication of the ACM*, vol. 35, no. 1, pp. 95–104, 1992.

[29] M. Yang and N. Bourbakis, "An overview of lossless digital image compression techniques," in *Proc. 48th Midwest Symp. Circuits and Systems*, pp. 1099–1102, 2005.

[30] N. Ahmed, T. Natarajan, and K. R. Rao., "Discrete cosine transform," *IEEE Transactions on Computers*, vol. 23, no. 1, pp. 90–93, 1974.

[31] R. C. Gonzalez and R. E. Woods, *Digital Image Processing (2nd Edition)*. Prentice Hall, 2002.

[32] "http://www.mathworks.com/."

[33] M. Shen, C. Lee, and J. Bor, "A 4.0-mW 2-Mbps programmable BFSK transmitter for capsule endoscope applications," in *Asian Solid-State Circuits Conference*, pp. 245–248, 2005.

[34] R. Samanta and R. N. Mahapatra, "An enhanced CAM architecture to accelerate LZW compression algorithm," in *20th International Conference on VLSI Design*, pp. 824–829, 2007.

[35] L. Ming-Bo, "A hardware architecture for the LZW compression and decompression algorithms based on parallel dictionaries," *Journal of VLSI Signal Processing Systems*, vol. 26, no. 3, pp. 369–381, 2000.

[36] N. Ranganathan, "Efficient VLSI for Lempel-Ziv compression in wireless data communication networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, no. 3, pp. 475–483, 1998.

[37] J. Venbrux, P. Yeh, and M. Liu, "A VLSI chip set for high-speed lossless data compression," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 2, no. 4, pp. 381–391, 1992.

[38] C. Su, C.-F. Yen, and J. Yo, "Hardware efficient updating technique for lzw codec design," in *IEEE International Symposium on Circuits and Systems*, pp. 2797–2800, 1997.

[39] S. Rigler, "FPGA-Based lossless data compression using GNU Zip," Master's thesis, University of Waterloo, 2007.

[40] E. Jamro, M. Wielgosz, and K. Wiatr, "FPGA implementation of the dynamic Huffman encoder," in *Proceedings of IFAC workshop on programmable devices and embedded systems*, 2006.

[41] W. Cui, "New LZW data compression algorithm and its FPGA implementation," in *Picture coding symposium*, 2007.

[42] M. Abdelghany, A. Salama, and A. Khalil, "Design and implementation of FPGA-based systolic array for LZ data compression," in *IEEE international symposium on circuits and systems*, pp. 3691–3695, 2007.

[43] M. R. Longo and S. F. Lourenco, "Spatial attention and the mental number line: Evidence for characteristic biases and compression," *Neuropsychologia*, vol. 45, no. 7, pp. 1400–1407, 2007.

[44] J. Shen and Y.-M. Jung, "Weberized Mumford-Shah model with Bose-Einstein photon noise," *Applied Mathematics and Optimization*, vol. 53, no. 3, pp. 1432–0606, 2006.

[45] S. Chiang and L. Po, "Adaptive lossy LZW algorithm for palettised image compression," *Electronics Letters*, vol. 33, no. 10, pp. 852–854, 1997.