IMAGE COMPRESSION SYSTEM FOR A 3U CUBESAT

by

Gutembert Nganpet Nzeugaing

Thesis submitted in partial fulfilment of the requirements for the degree of

Master of Technology: Electrical Engineering

in the Faculty of Engineering
at the Cape Peninsula University of Technology

Supervisor: Prof. E Biermann

Bellville

Date: 05 August 2013

DECLARATION

I, Gutembert Nganpet Nzeugaing, declare that the contents of this thesis represent my own unaided work, and that the thesis has not previously been submitted for academic examination towards any qualification. Furthermore, it represents my own opinions and not necessarily those of the Cape Peninsula University of Technology.

Signed **Date**

ABSTRACT

Earth observation satellites utilise sensors or cameras to capture data or images that are relayed to the ground station(s). The ZACUBE-02 CubeSat currently in development at the French South African Institute of Technology (F'SATI) contains a high resolution 5 megapixel on-board camera. The purpose of the camera is to capture images of Earth and relay them to the ground station once communication is established. The captured images, which can amount to a large volume of data, have to be stored on-board as the CubeSat awaits the next cycle of transmission to the ground station. This mode of operation introduces a number of problems, as the CubeSat has limited storage and memory capacity and is not able to store large amounts of data. This, together with the limitation of the downlink capacity, has set the need for the design and development of an image compression system suitable for the CubeSat environment.

Image compression focuses on reducing the size of images to be stored as well as reducing the size of the images to be transmitted to the ground station. The purpose of the study is to propose a compression system to be implemented on ZACUBE-02. An intensive study of current, proposed and implemented compression methods, algorithms and techniques as well as the CubeSat specification, served as input for defining the requirements for such a system.

The proposed design is a combination of image segmentation, image linearization and image entropy coding (run-length coding). This combination technique is implemented in order to achieve lossless image compression. For the proposed design, a compression ratio of 10:1 was obtained without negatively affecting image quality.The on-board storage memory constraints, the power constraints and the bandwidth constraints are met with the implementation of the proposed design, resulting in the downlink transmission time being minimised.

Within the study a number of objectives were met in order to design, implement and test the compression system. These included a detailed study of image compression techniques; a look into techniques for improving the compression ratio; and a study of industrial hardware components suitable for the space environment.

Keywords: CubeSat, hardware, compression, satellite image compression, Gumstix Overo Water, ZACUBE-02.

To God for His everlasting grace and glory.

To my mother, Pauline Nketcha, for her love and care.

ACKNOWLEDGEMENTS

“What has been will be again, what has been done will be done again; there is nothing new under the sun.” Ecclesiastes 1:9

Throughout my research studies in these two years, I have encountered the state-of-the-art techniques and theories within the scope of image processing and satellite engineering which have, and continue to, inspire me to do research in image compression system for a 3U CubeSat. It is my pleasure to acknowledge and thank everyone who has influenced me through my research studies.

First of all, I express my gratitude to my research supervisor Prof. Elmarie Biermann for her support and encouragement.

Many thanks to the French South African Institute of Technology (F’SATI) for my research funding. To all my friends and colleagues at F’SATI, I thank you.

I would like to thank Dr. John R. Samson, Jr., Principal Engineering Fellow, Honeywell International, Dr. Matthew Clark, Principal Systems Engineer, Honeywell International, and Dr. Eric Grobelny, Senior Systems Engineer, Honeywell International for assisting me on the hardware platform for CubeSats.

I am grateful to my parents for all the encouragement and support which they have provided me in all my years spent on the pursuits of my educational Excellency. I would like to thank my brothers and sisters in Christ for their prayers.

To you my brother Jamot Bami, this journey of my Masters study would not have started without your tremendous input; I am thankful for your support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF LISTINGS

LIST OF ABBREVIATIONS

1D	One-Dimensional
1U	One Unit
2D	Two-Dimensional
3U	Three Units
AC	Alternating Current
ADCS	Attitude Determination and Control System
b	bit
B	Byte
BMP	Bitmap
BR	Bit Rate
CALIC	Context Based Adaptive Lossless Image Coder
Cal Poly	California Polytechnic State University
CFP	Call for Proposal
CMY	Cyan, Magenta, and Yellow
COM	Computer On Module
CPU	Central Processing / Processor Unit
CPUT	Cape Peninsula University of Technology
CR	Compression Ratio
dB	decibel
DC	Direct Current
DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
DNS	Domain Name System
DPCM	Differential Pulse Code Modulation
DSP	Digital Signal Processing / Processor
DWT	Discrete Wavelet Transform
EBCOT	Embedded Block Coding with Optimized Truncation
FDCT	Forward Discrete Cosine Transform
FIFO	First In First Out
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
F'SATI	French South African Institute of Technology
GDI	Graphics Device Interface

GEVS	General Environmental Verification Standard
GIF	Graphics Interchange Format
GPP	General Purpose Processor
GSFC	Goddard Space Flight Center
I2C	Inter Integrated Circuit
IDCT	Inverse Discrete Cosine Transform
I/O	Input / Output
JPEG	Joint Photographic Experts Group
JPEG2000	Joint Photographic Experts Group 2000
JPL	Jet Propulsion Laboratory
LEO	Low Earth Orbit
LOCO-I	Low Complexity, Context-Based Lossless Image Compression Algorithm
LSB	Least Significant Bit
LTS	Long Term Support
LVQ	Lattice Vector Quantisation
LZ77	Lempel-Ziv 1977
LZW	Lempel-Ziv-Welch
M	Mega
MP	Megapixel
MSB	Most Significant Bit
NASA	National Aeronautics and Space Administration
NFS	Network File System
OBC	On-Board Computer
OMAP3530	Open Media Applications Platform 3530
OS	Operating System
P	Pixels
PC	Personal Computer
PNG	Portal Network Graphics
PSNR	Peak Signal-to-Noise Ratio
RGB	Red, Green, and Blue
RLC	Run-Length Coding
RMSE	Root-Mean-Squared-Error
SCP	Secure Copy
SD	Secure Digital
SNR	Signal-to-Noise Ratio

SQ	Scalar Quantisation
SSDL	Space Systems Development Laboratory
TIFF	Tagged Image File Format
TT&C	Tracking Telemetry and Command
TUT	Tshwane University of Technology
USB	Universal Serial Bus
VQ	Vector Quantisation

CHAPTER 1

PROPOSAL

1.1 Introduction

Low Earth Orbit (LEO) refers to any altitude less than 2 000km above Earth (Odenwald, Green & Taylor, 2006:280-297). Satellites located at this altitude are able to capture images of Earth using various imaging sensors. The development of large satellites such as, for example, the Planck satellite (ESA, 2013) and Landsat-D (Schefter, 1984:92-95) is resource intensive and very expensive. To minimise development time and especially costs, Prof. Jordi Puig-Suari at the California Polytechnic State University (Cal Poly), San Luis Obispo and Prof. Bob Twiggs at Stanford University's Space Systems Development Laboratory (SSDL) proposed the CubeSat as a collaborative effort in 1999 (Heidt, Puig-Suari, Moore, Nakasuka & Twiggs, 2001). CubeSats are small satellites that have advantages such as low development cost, low complexity, good imaging sensors as well as short development time. The CubeSat specification also provides universities and research institutes with opportunities to develop and enhance space programmes and train students in the field of space engineering.

The CubeSat initiative is regarded as the ideal platform for the development and transfer of satellite engineering skills especially in developing countries. Africa joined this opportunity to invest in space exploration through the development of CubeSats at a number of universities. The South African Government, in cooperation with the French Government, established the French South African Institute of Technology (F'SATI) together with the Cape Peninsula University of Technology (CPUT) and the Tshwane University of Technology (TUT), with the aim of developing human capital for space science.

F'SATI, operating within the Faculty of Engineering at CPUT, introduced a CubeSat programme in 2007. The main focus was to develop human capital for space science by providing an opportunity for graduate students to be involved in the design and implementation of satellites. As a result of this programme, the first 1U CubeSat (ZACUBE-01) is set for launch in 2013, making it Africa's first CubeSat in space. The design and development of the 3U CubeSat (ZACUBE-02) also commenced in 2011 and is set to be completed in 2014.

The primary mission of ZACUBE-02 is to capture and relay still images of Earth to serve as input into several applications such as, for example, the monitoring of bush fires and weather predictions. Due to the limited memory and storage on-board the 3U CubeSat and the limited

bandwidth, still images captured have to be compressed and stored temporarily before being relayed.

ZACUBE-02 includes a 5 megapixels (high resolution) camera. According to Chartier, Mackay, Ravalico, Russell and Wallis (2010:14), image sensors with large resolutions (1x1 m pixel size) record file sizes that are typically larger than what can be transmitted by the majority of CubeSat transceivers in one overhead pass. Therefore, image compression is identified as a crucial on-board facility on the CubeSat (Yuhaniz, Vladimirova & Sweeting, 2005:90-103).

In this chapter background information regarding CubeSats and especially ZACUBE-01 and ZACUBE-02 are provided. Image processing is also introduced, leading to an elaboration of the problem statement, research objectives, solution approach, significance and contribution of research, ending with a synopsis.

1.2 CubeSat

A CubeSat is classified as a pico-satellite with a volume of 10 cm^3 and a mass of up to 1.33 kg for one unit (1U), which can be extended up to three units (3U). CubeSats are mainly used for space exploration and consist of the following subsystems (Larson & Wertz, 1999:44):

- Payload: the combination of hardware and software on the satellite that interacts with the target to accomplish the mission objectives.
- Attitude Determination and Control System (ADCS): used to point the satellite accurately and to stabilise it during image capturing.
- Thermal: ensuring the satellite is protected against solar radiation in space. It also protects the satellite's electronic components against very high and very low temperatures.
- Tracking Telemetry and Command (TT&C): provides communication between the satellite and the ground station by transferring payload mission data and handling spacecraft housekeeping. The TT&C is used as the platform to pass operation commands to the satellite.
- On-Board Computer (OBC): the computing centre of the CubeSat. It gathers telemetry from subsystems, performs general housekeeping, controls other subsystems, communicates with the ground station and handles the execution of the flight plan.
- Power system (solar panel and batteries): satellite power system utilises a solar energy system due to solar energy being the most favourable source of energy in space; the electric power produced by the solar panel is then dispatched to each subsystem.

A CubeSat has to pass vigorous tests before being launched to ensure its survival in orbit. These testing requirements are provided by the launch vehicle entity, or alternatively standard requirements can be obtained from the General Environmental Verification Standard (GEVS) for Goddard Space Flight Center (GSFC) Flight Programs and Projects GSFC-STD-7000 (NASA, 2005).

ZACUBE-01 as illustrated in Figure 1.1 (adapted from F'SATI, 2011) contains a camera with resolution 640x480 pixels (0.3 megapixels). Captured images of Earth can be transmitted to the ground station without the need to be compressed due to the low resolution.

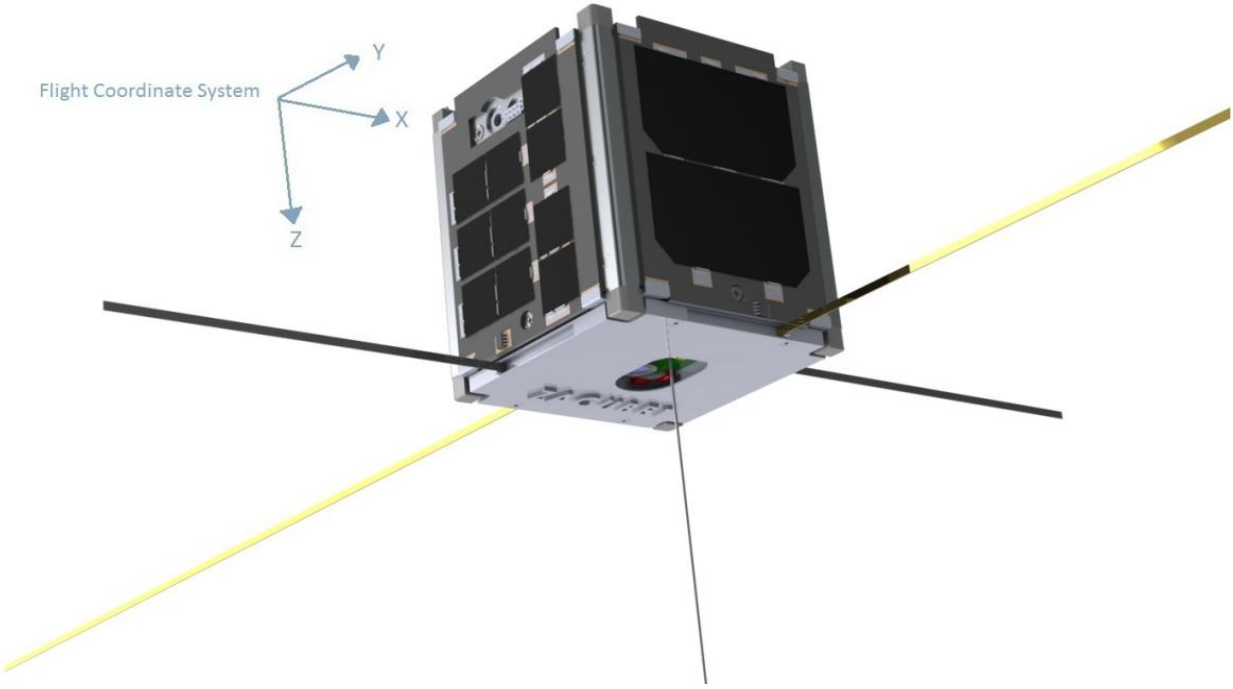


Figure 1.1: CPUT 1U CubeSat (ZACUBE-01)
(Adapted from F'SATI, 2011)

Figure 1.2 illustrates the camera block diagram used for ZACUBE-01 which operates in the following way:

The UHF/VHF antenna alerts the OBC through the microcontroller (PIC24) during communication between the CubeSat and the ground station. The OBC instructs the ADCS through the ATMEL MEGA chip to point the satellite accurately to Earth and stabilise it during image capturing. Once communication between the CubeSat and the ground station is established, the OBC sends instructions to the PIC24 and the ATMEL MEGA chip through the

I2C bus for data transmission. The PIC24 then alerts the HF BEACON transmitter for image downlink to occur. With the 0.3 megapixels camera on-board ZACUBE-01, images taken will be easily transmitted to the ground station without the need for compression.

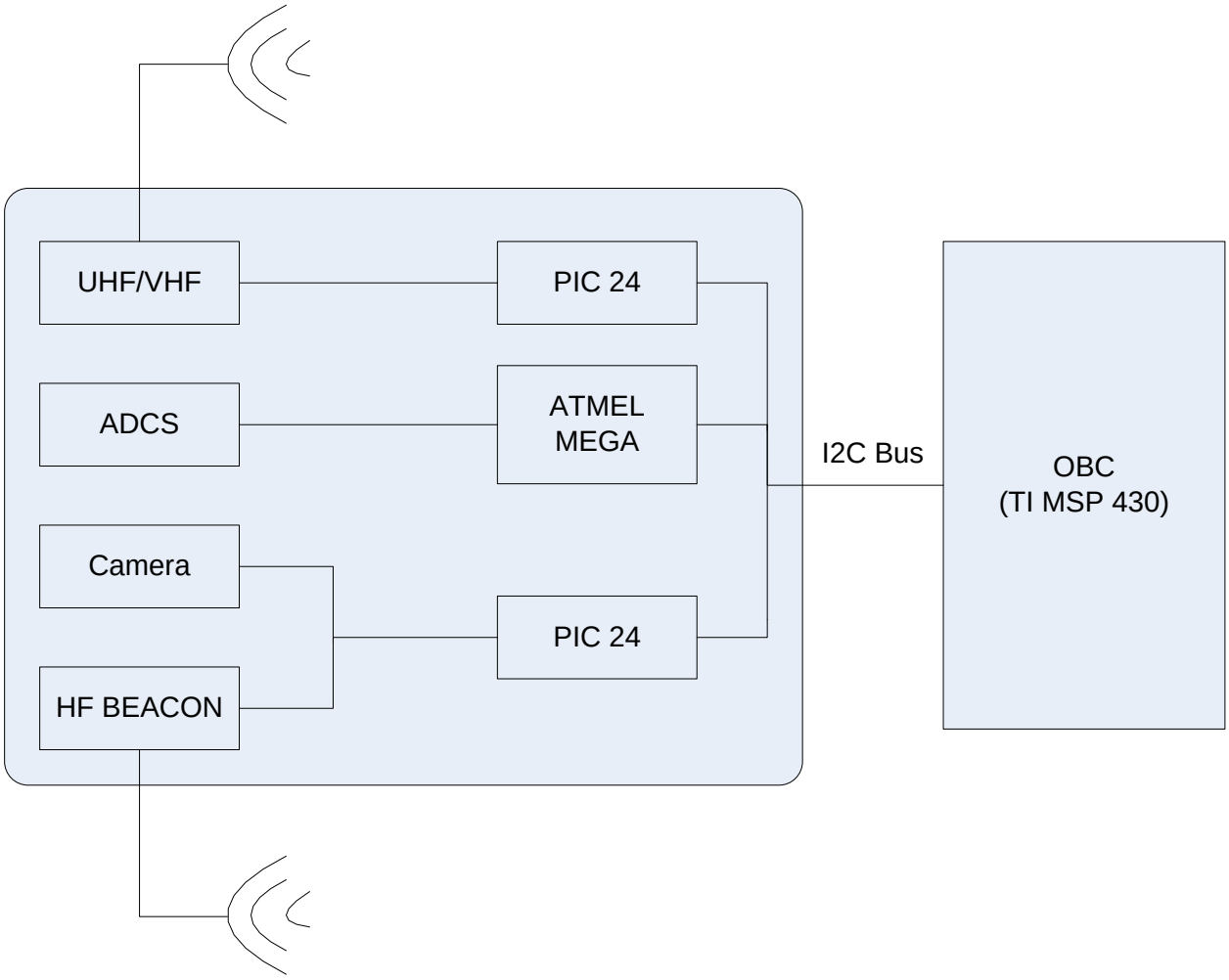


Figure 1.2: CPUT 1U CubeSat (ZACUBE-01) camera block diagram

The proposed model of ZACUBE-02 is depicted in Figure 1.3 (adapted from F'SATI, 2011). It contains a 5 MP, high resolution on-board camera. Due to the high resolution and resulting size of the images, captured images cannot be transmitted to the ground station in one pass of the satellite over the ground station.

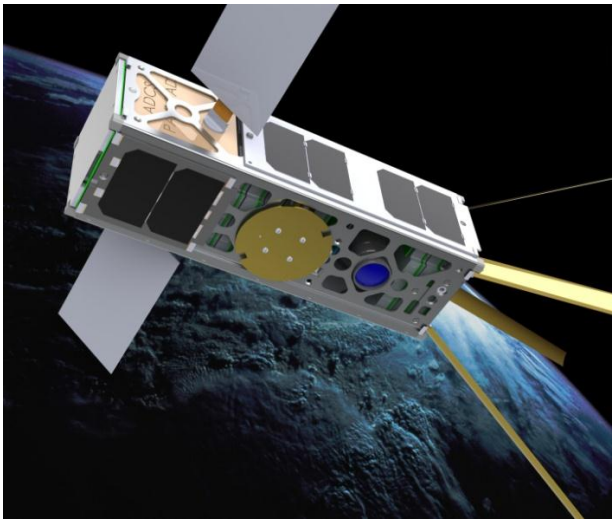


Figure 1.3: ZACUBE-02
(Adapted from F'SATI, 2011)

Therefore, captured images need to be buffered, compressed and stored on-board ZACUBE-02 while awaiting communication between the ground station and the CubeSat. Once communication occurs, the OBC will instruct the FPGA via the I2C cable for image downlink. The FPGA will alert the S Band transmitter for image downlink as illustrated in Figure 1.4.

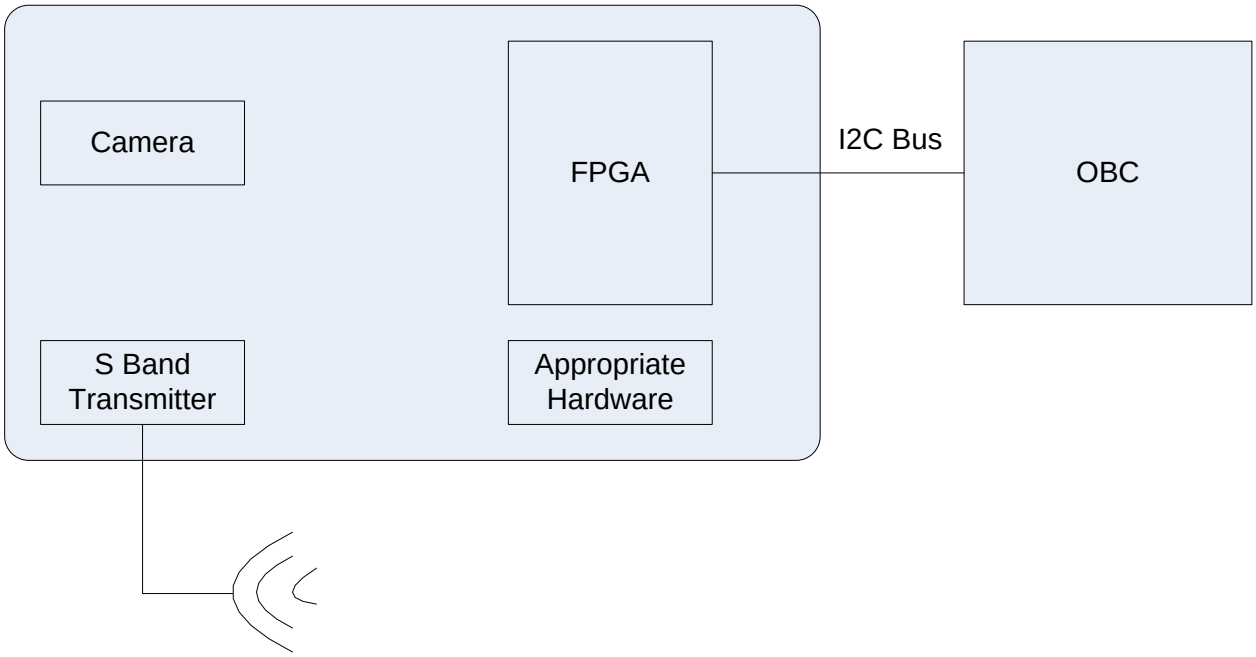


Figure 1.4: ZACUBE-02: Image processing block diagram

CPUT has its own ground station used to track satellites utilising amateur radio frequencies. When ZACUBE-02 passes the horizon, the compressed images on-board will be transmitted to this ground station. The CPUT ground station antenna is illustrated in Figure 1.5.

Figure 1.5: CPUT ground station antenna
(Adapted from F'SATI, 2011)

1.3 Image processing

An image is an array, or matrix, of square pixels (picture elements) arranged in columns and rows. In science-related applications, the two main image colour spaces are RGB (Red, Green, and Blue) and CMY (Cyan, Magenta, and Yellow). RGB are the basic colours used in television or any other medium that projects colour with light. They are the basic colours used in computers and for web graphics.

The CMY colours are formed by mixing two of the primary colours (Red, Green or Blue) while excluding the third colour. The combination of Red and Green gives Yellow, combining Green and Blue gives Cyan, and combining Blue and Red gives Magenta. A full combination of Red, Green, and Blue makes white (Meher, 2010).

The reduction of image size is often the first critical step in image processing, especially in applications such as medical image compression and satellite imagery. Image processing (image compression) minimises the storage space, the required bandwidth, the downloading time, and the transmission time.

In this research project, still images captured by the camera on-board ZACUBE-02 will be used as an input signal to the selected hardware. The input image will be processed (compressed) on

the selected hardware resulting in an output signal. The output image will then be stored temporary while waiting for communication between the CubeSat and the ground station. This process is referred to as image processing (Dougherty, 2009).

Due to downlink restrictions, large image files can't be sent to ground station. One possible way in which large image files can be sent to a ground station is to process them directly on-board ZACUBE-02 and compress the image files before transmission to the ground station. The size, mass, power restrictions and space environment of a CubeSat present challenges for on-board computing. Signal processing on-board a CubeSat presents four key challenges namely the capture, process, storage and transfer of images (Karagiannakis, Weiss & Bowman, 2012). Image compression will alleviate some of these challenges.

Image compression is vital for image processing, especially when the storage memory and the transmission bandwidth are limited. Image compression techniques can either be lossy or lossless. Lossy image compression technique consists of reducing the original image file's size while losing some image information, whereas lossless image compression technique consists of reducing the original image file's size without loss of image information. Digital images are compressed and stored in various standards and formats, such as GIF, PNG, JPEG and JPEG2000.

1.4 Problem Statement and Purpose

Most CubeSats employ a high resolution camera on-board and it is inevitable that the images captured by the camera are larger than what can be transmitted to the ground station in a single pass over the ground station. The purpose of ZACUBE-02 is to capture images of Earth and relay these images to the ground station with the limited resources at hand. ZACUBE-02 will host a 5 MP camera: captured images of this size leads to strains in terms of transmission size, storage as well as power. Image compression is thus required to ensure that images are relayed for use in specialised applications. The purpose of this study is to propose a suitable design and implementation to compress images on-board ZACUBE-02.

1.5 Research objectives

The main objective of this study is:

To design, implement and test an image compression system for the 3U CubeSat (ZACUBE-02) that will meet the requirements of the harsh space environment and provide quality images for advanced applications.

In order to reach this objective, a number of sub-objectives are identified:

- Conduct a detailed study and comparison of image compression techniques.
- Study and compare techniques for improving the compression ratio while retaining image quality.
- Conduct a detailed study of imagery and compression systems especially tailored for the CubeSat environment.
- Put forward detailed design criteria and requirements.
- Study and choose industrial hardware components to meet environmental requirements, design requirements and application requirements.
- Evaluation of the proposed design.

1.6 Solution approach

The image compression system on-board ZACUBE-02 requires industrial hardware that meets the stringent environmental requirements. These requirements include for example passing vibration tests, being able to operate in extreme LEO temperatures and a high computing processor for real time image processing. The CubeSat's constraints such as low power consumption, small dimensions (less than 30x10x10 cm^3), and low cost (Boghosian & Valerdi, 2012; Clyde Space, 2012; Space Micro, 2012), also provide additional limitations on the design.

The approach to be followed in order to propose a solution is highlighted in Figure 1.6. The study commences with extensive research into possible compression techniques and compression ratios. Existing imagery and compression systems provide needed input in terms of requirements and possible design limitations. Once the requirements are set and the design proposed, the study focuses on the implementation and testing phase. The evaluation phase concludes the study.

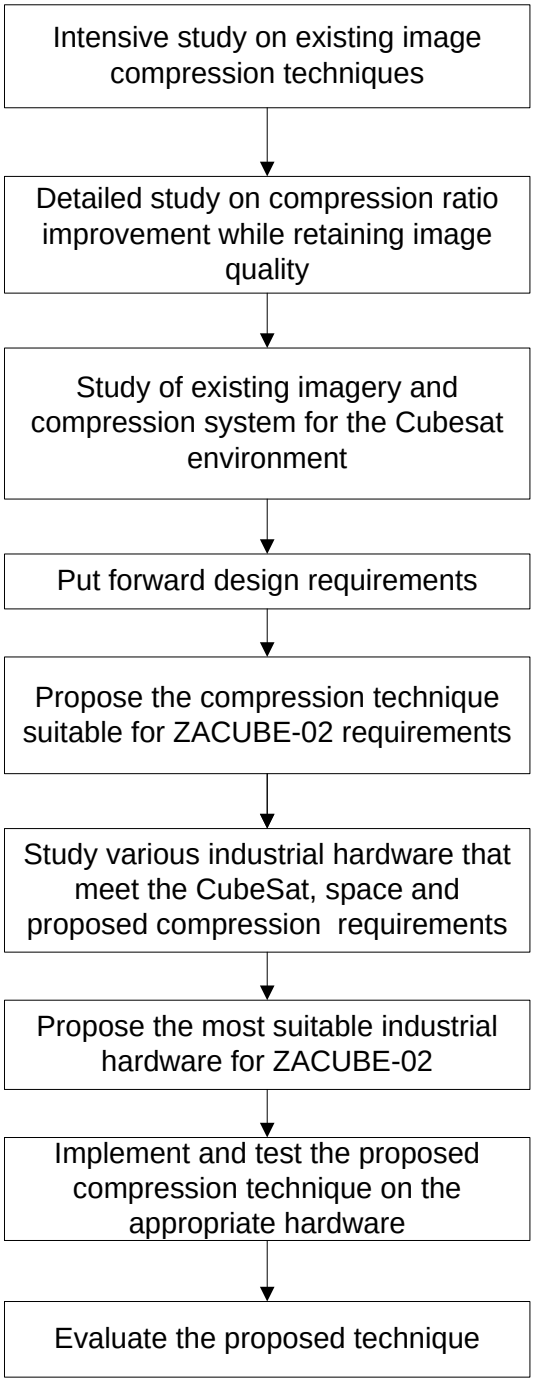


Figure 1.6: Solution approach schematic

1.6.1 Delineation

This research project only focuses on the CubeSat on-board image compression. Captured images will be buffered then compressed before being stored in temporary memory. Receiving systems and the ground station will not be covered within this project.

1.7 Significance and contribution of research

Advancements within the field of imagery hardware and software lead to larger image sizes. Compression of such images is thus becoming a vital part of satellite research and development. Suitable and optimised compression techniques are required within advanced satellite applications such as, for example, weather systems.

Outputs include a conference paper (CPUT Postgraduate Research Conference 2012), symposium paper (South African Institute for Computer Scientists and Information Technologists 2012 Masters and Doctoral Symposium), journal paper (*International Journal of Computer Applications*, 36(10):32-36, December 2011. Published by Foundation of Computer Science, New York, USA) as well as a thesis, to be used by scientists, engineers and researchers.

1.8 Synopsis

The research and development process is presented within seven chapters. Chapter 2 focuses on lossless image compression, while Chapter 3 details lossy image compression. These chapters provide a comparative study in order to select a suitable compression technique.

Chapter 4 describes the imagery and compression systems on CubeSats. The requirements and proposed design are highlighted and discussed in Chapter 5. This chapter also contains information and data relating to the implementation and testing. Chapter 6 evaluates the proposed solution with the conclusion summarised in Chapter 7.

CHAPTER 2

LOSSLESS IMAGE COMPRESSION

2.1 Introduction

Image compression reduces the data (image) size, minimises the storage, and reduces the required bandwidth, the data transmission time, and the power consumption.

Image compression is vital in the digital world since digital images are continuously expanding in size. It is mostly used in commercial photography, industrial imaging, satellite imaging, and academic applications in order to reduce the amount of data to be stored or transmitted (Masud & Canny, 1998:2581-2584). Image compression is the process used to minimise the physical size of the image. Often in satellite imaging, the interest is to reduce the size of the image so that more images can be stored on-board the satellite, and to relay images to the ground station in one pass.

Image compression is classified as either lossy or lossless; during lossy compression, some of the original information contained in the image is not preserved, thus resulting in better compression ratios but with some image quality degradation. During lossless compression the original information contained in the image is preserved (Murray & Van Ryper, 1996).

Both lossy and lossless image compression can be achieved using coding methods, spatial domain compression methods, transform domain compression methods or the combination of these methods (Bassiouni, Tzannes, Tzannes & Tzannes, 1991:2817-2820).

This chapter discusses and analyses lossless image compression techniques.

2.2 Lossless Image Compression Techniques

Lossless image compression refers to reversible image compression, meaning the exact original image can be recovered (Starosolski, 2007:65-91). Lossless image compression presents the advantage of recovering the exact original image, thus preserving the original image quality. Compression efficiency and complexity are two parameters used to measure the lossless compression algorithm performance. The compression ratio (CR) or bit rate (BR) is the component used to measure compression efficiency. The compression ratio is defined as the ratio of the number of bits representing the original image to the number of bits representing the compressed image. A variation of the compression ratio is known as bits per sample or bit rate, which is the ratio of the number of bits of a single uncompressed image to the compression ratio. For example, an $512x512\ pixels$, $12\ bits\ per\ pixel$ image requires

$512 * 512\ pixels * 12\ bits/pixel = 512 * 512 * 12\ bits = 512 * 512 * \frac{12}{8}\ bytes = 393216\ bytes$ as an uncompressed image size. If the compressed image requires $78643\ bytes$, the compression ratio will be $\frac{393216}{78643} = 5$. Taking into consideration the uncompressed image of $512x512\ pixels, 12\ bits/pixel = 262144\ pixels, 12\ bits/pixel$, the compressed image requires $78643\ bytes * \frac{8}{262144\ pixels} = 2\ bits\ per\ pixel$ or $\frac{12\ bits}{CR} = \frac{12\ bits}{5} = 2.4 \cong 2\ bits\ per\ pixel$ which means the bit rate is 2. The bit rate of the compressed image can be compared to the entropy of the source image for compression efficiency. The source entropy represents the amount of information contained in the source image. For example, an image with grey values range from 0 to $M - 1$ and its probability p_i (frequency divided by the total number of pixels) has an entropy of $H(I) = -\sum_{i=0}^{M-1} p_i \log p_i$ (Shannon, 1948:379-423).

Compression efficiency of a lossless compression method is measured by determining how closely the bit rate approximates the source entropy; since the source entropy is a lower bound on the bit rate any lossless compression method can achieve better compression efficiency (Kou, 1995). If the source entropy ($H(I)$) of an image is 2 bits/pixel and the lossless compression system has a bit rate of 2 bits/pixel, the lossless compression system has achieved the best compression efficiency possible.

The compression complexity of an image is measured by its run time (i.e. the time that the compression system takes to accomplish the total process); this is very important for applications where speed is crucial.

Lossless image compression is achieved via a number of methods, namely entropy coding, spatial domain compression, transform domain compression or a combination of these methods.

2.2.1 Coding

Image coding is a systematic way in which to condense extensive image sets into smaller analysable units through the creation of categories and concepts derived from the image (Lewis-Beck, Bryman & Liao, 2004:137-138). Coding methods are direct applications to the raw images, treating them as a sequence of discrete numbers. Different coding methods have been proposed and used such as Arithmetic coding (Abramson, 1963:61-62; Witten, Neal & Cleary, 1987:520-540), Range coding (Cho & Pearlman, 2007:2005-2015), Huffman coding (Huffman, 1952:1098-1101), Adaptive Huffman coding (Pigeon & Begio,1998), Lempel-Ziv Welch (LZW) coding (Welch, 1984:8-19), LZ77 coding (Ziv & Lempel, 1977:337-343), LZ78 coding (Ziv & Lempel,

1978:530-536), Lempel-Ziv-Oberhumer algorithm (Oberhumer, 2011), Lempel-Ziv-Markov algorithm (Pavlov, 2013), Run-Length coding (RLC) (Akhtar, Qureshi & Qamar-ul-Islam, 2011:81-85), Golomb coding (Golomb, 1966:399-401), Rice coding (Rice, 1979; 1991), Golomb-Rice coding (Salomon, Motta & Bryant, 2010), Unary coding (Gallager & Van Voorhis, 1975:228-230), Truncated binary coding (Bayat, Johansen & Jalali, 2011), Elias coding (Elias, 1975:194-203), Fibonacci coding (Kautz, 1965:284-292), and Shannon-Fano coding (Xiaoyu & Katti, 2006:1-7).

Coding methods such as these can be used to compress images (ITU-T & ISO/IEC, 2011) as a set of two-dimensional (2D) arrays of integer while first ensuring that the two-dimensional array is converted into a one-dimensional (1D) array image sequence by means of linearization. Histogram packing technique is also used by these coding methods in order to improve the image compression ratio.

2.2.1.1 Linearization

Satellite images are generally in 2D; as an example, an image M of $4x4$ pixels (width x height), linearization of $M_{2,1}$ can be achieved as depicted in Figure 2.1 and computed as $row\ x\ width + column = 2\ x\ 4 + 1 = 9$ (Kirk & Hwu, 2012).

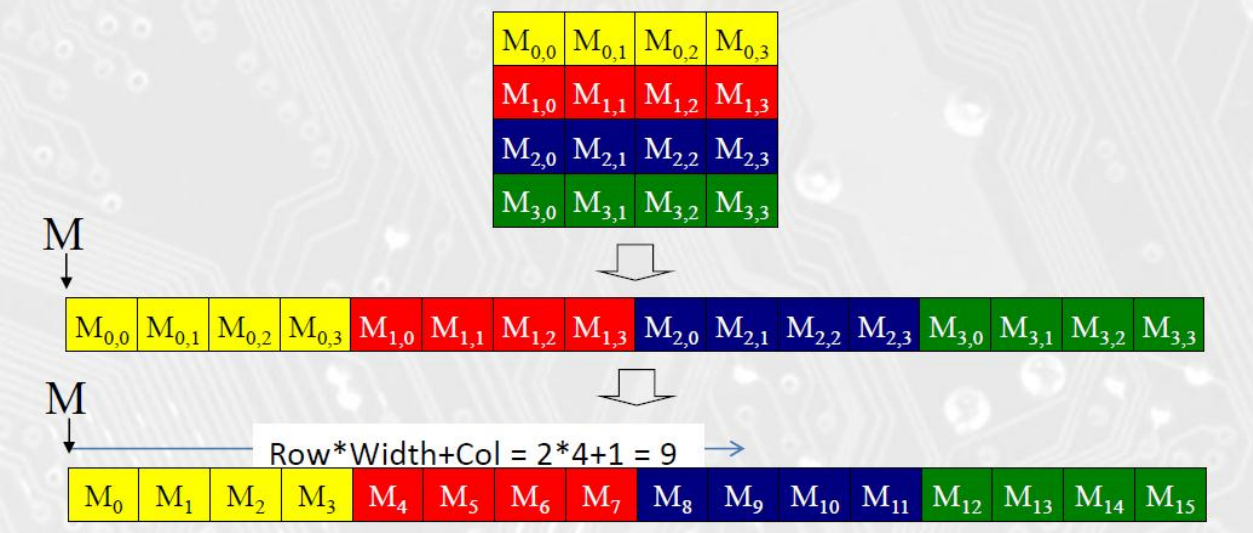
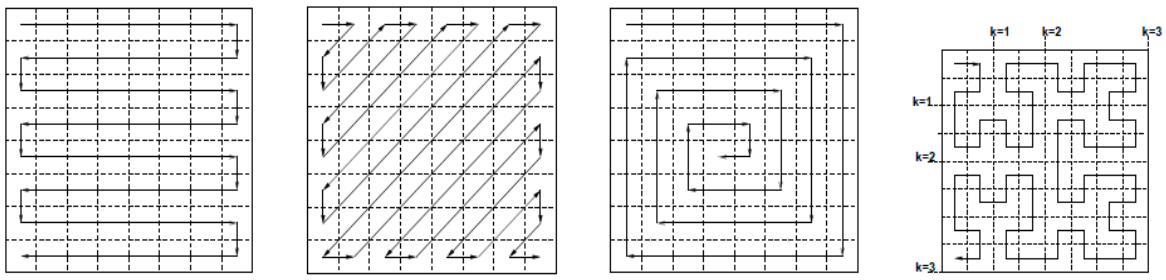


Figure 2.1: Image linearization
(Adapted from Kirk & Hwu, 2012)

During linearization, each linearization scans the image pixels in some order to produce the one-dimensional sequence. According to Portoni, Combi, Pozzi, Pinciroli, Fritsch and Brennecke (1997:185-204), common linearization schemes used are:

- **Row-Major Scan**: scans image in row by row from top to bottom and from left to right within each row.
- **Column-Major Scan**: scans image in column by column from left to right and from top to bottom within each column.
- **Diagonal Scan**: scans image along the anti-diagonals (lines with constant row plus column value) beginning with the top-most anti-diagonal. Each anti-diagonal is scanned from the left bottom corner to the right top corner.
- **Snake-like Row-Major Scan**: is derived from a row-major scan method. It scans image row by row from top to bottom, and alternately scans rows from left to right and from right to left. The top-most row is scanned from left to right as illustrated in Figure 2.2 (a). The same approach is applicable to the "snake-like" linearization scheme derived from the "column-major" and "diagonal scans" linearization schemes as illustrated in Figure 2.2 (b).
- **Spiral Scan**: scans image from the outside to the inside, tracing out a spiral curve starting from the top left corner of the image and proceeding clockwise as illustrated in Figure 2.2 (c).
- **Peano-Hilbert Scan**: scans image from the left-most pixel of the first row and ends at the left-most pixel of the bottom row. This linearization scheme was introduced by Peano and Hilbert and is best described recursively as depicted in Figure 2.2 (d). It requires an image of size $2^n x 2^n$ pixels where n is an odd number. If n is an even number, the image is scanned from the left-most pixel of the first row and ends at the right-most pixel of this row. The *Peano-Hilbert scan* scans an image quadrant by quadrant. The *Peano-Hilbert scan* for an image of size $2^n x 2^n$ pixels where n is 1, 2, and 3, is depicted in Figure 2.2 (d).



(a) (b) (c) (d)

Figure 2.2: (a)Snake-like row major scan, (b)Snake-like diagonal scan, (c)Spiral scan, (d)Peano-Hilbert scan (Adapted from Portoni *et al*., 1997:185-204)

Different linearization schemes result in different compression ratios. Although no single linearization method provides the highest compression ratio for all images, the *Peano-Hilbert* performs better (Vemuri, Sahni, Chen, Kapoor, Leonard & Fitzsimmons, 2007).

Images have local and global redundancy. Local redundancy in images is defined as coherence or correlation in image pixels, resulting in image smoothness (Carpentieri, Weinberger & Seroussi, 2000:1797-809), while global redundancy in images is defined as a repetition of patterns within an image (similarity).

- **Smoothness**

  Smoothness of the data is often referred to as coherence or correlation property exhibited from a given neighbourhood in an image. Neighbourhood in an image represents different pixels located in a specific or given block of the image. The variation within a neighbourhood is often due to the illumination or shadowing effects. The smoothness is then measured by the difference between the minimum and maximum grey levels within a neighbourhood. For example, if the difference is zero, it means that every pixel in the block is of the same grey level. Therefore, the block can be represented by two components namely: a pixel and the block's size.

  The pixel and block's size representation is identical to the run-length coding and requires less memory storage (Russ, 2011). Generally, if the difference is not zero, it is usually small, resulting in the block being represented by the offset value between the minimum grey level value and the neighbouring pixels in the block. To encode each offset, N_bit code-words length can be used as derived from equation (2.1).

$$N = \log_2(V_{th} + 1) \tag{2.1}$$

  From Equation (2.1), V_{th} is the offset value between the minimum grey level value and the neighbouring pixels in the block. It is very important to carefully choose a proper threshold value since if the difference value is smaller than the fixed threshold value, the compression is only implemented using the minimum grey level value and the corresponding offsets (Deshpande & Sane, 2007).

- **Similarity**

  According to Bertelli (2009), similarities in images are:

  1) Similar pixel intensities in some areas;
  2) Similar pixel probabilities;
  3) Comparable histograms in some areas;

4) Similar edge distributions within an image;
5) Analogous distributions of features within an image.

Most often, images with a given pattern might repeat itsel. For example, if a pattern repeats itself for m times in an image, then $(m-1)$ patterns can be replaced by pointers to the first occurrence of the pattern. From the compression point of view, to determine how good the similarity characteristic is, the image has to be partitioned into sub-regions where each sub-region is considered as a pattern, which help to estimate how many times each pattern repeats itself in the entire image. Once the frequency of occurrence is known, the estimated resulting compression that can be achieved on the image is known. To find the basic pattern, the similarity of the whole image must be studied. The manipulation of one basic pattern block of a certain size can be done by geometric modifications such as magnification, reduction, rotation or brightness scaling in order to represent the original image (Se-Kee, Jong-Shill, Dong-Fan & Je-Goon, 2006:50-56).

Taking advantage of redundancy consists of saving information on pixels' similarity/correlation. Coding schemes are more effective if they can preserve the local/global redundancy of the image and are expected to result in a better compression ratio since the compression efficiency is measured by the compression ratio or by the bit rate. In order to improve the compression ratio without affecting the image quality, the compression system has to be coupled with a coding scheme that can take advantage of local and global redundancy (Ranganathan, Romaniuk & Namuduri, 1995:1396-1406). The histogram packing technique (Pinho, 2001:442-445; Ferreira & Pinho, 2002:259-261) is a method to improve a lossless image compression ratio.

2.2.1.2 Histogram packing technique

The histogram packing technique improves the compression ratio by introducing a variation-reducing reversible mapping (images with smaller total variation are easier to compress). By reducing the image variation, the histogram packing technique reduces the approximation error roughly by a constant amount. It has been shown (Pinho, 2002a:5-7) that the performance of lossless image coding methods, such as JPEG-Lossless, lossless JPEG-2000 and CALIC, can be improved by taking advantage of pre-processing techniques that can be applied to images with sparse histograms. The histogram packing technique as illustrated in Figure 2.3 consists of mapping the image in order to pack its histogram. This technique is very useful when the histograms are only locally occurring at widely spaced intervals (Pinho, 2002b).

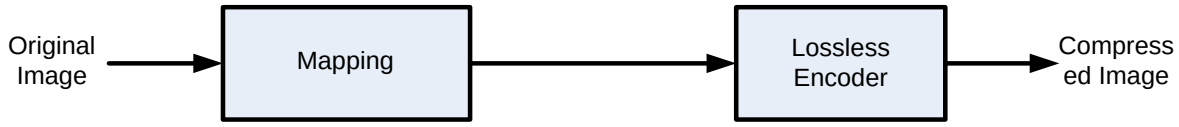


Figure 2.3: Pre-processing (histogram packing) for lossless image compression improvement (Adapted from Pinho, 2002a:5-7)

The histogram packing technique can either be online or offline. Online histogram packing (Carpentieri *et al*., 2000:1797-809) operates independently from the encoders, resulting in no modification of the encoding algorithms. Offline histogram packing is performed by mapping the intensity values of each image into a contiguous set, while maintaining the original order. Offline packing requires *a priori* knowledge of the sparseness of the histogram or two-pass operation.

2.2.1.3 Coding Methods

Common coding methods include: Arithmetic, Huffman, Lempel-Ziv Welch (LZW), and Run-Length.

- **Arithmetic Coding**

  Greyscale digital image is an image in which each pixel's value (grey value) carries only intensity information. The arithmetic coding scheme depends only on the relative order of grey values. Intensity information often varies from the darkest (0) to the brightest (1) value; which can be grouped as interval [0, 1). Image coding in arithmetic coding is based on values of the interval [0, 1), where [a, b), represents a half open interval with *a* denoting an included value and *b* an excluded value (Witten *et al*., 1987:520-540). Arithmetic coding has two fundamental concepts namely the probability of a grey value, and the encoding interval range for a grey value. The compression efficiency and the interval ranges of source grey values for the encoding process are determined by the occurrence probabilities of the grey values. The main idea is to assign short code-words to probabilities that occur frequently and longer code-words to probabilities of fewer occurrences. The compression output is determined by the interval ranges defined within the interval from zero included to one excluded ([0, 1)). For arithmetic coding, both high-precision floating point and integers are used during computation.

  To achieve better compression, the data structure has to be taken in consideration; for example in images, numerical intensity values of nearby pixels are used to predict the

intensity of each new pixel and to use a suitable probability distribution for the residual error to allow for noise and variation between regions within the image.

Pure arithmetic codes supplied with accurate probabilities provide optimal compression. Arithmetic coding runs slow and are fairly complicated to implement, but it finds common applications in frequently occurring sequence of pixels (Howard & Vitter, 1994:857-865).

- **Huffman Coding**

  This coding scheme focuses only on the frequency of occurrence of different grey values (pixels' value) where each grey value of the uncompressed image is replaced by a code (Huffman, 1952:1098-1101). Grey value codes are of variable length and grey values with higher frequency of occurrence in the uncompressed image have codes of smaller size than grey values with lower frequency of occurrence. By encoding these grey values, an overall reduction in image size is accomplished. The grey value codes satisfy a prefix property (no code is a proper prefix of another code). In images, an individual pixel represents an individual set of bits. For a 12 bits/pixel image, the total bits are $2^{12} = 4096$. If the image to be compressed has $4x4$ pixels (16 pixels), then it can be represented by its pixels as:

  $\{P_{0,0}, P_{0,1}, P_{0,1}, P_{0,1}, P_{1,0}, P_{1,1}, P_{1,2}, P_{1,3}, P_{2,0}, P_{2,1}, P_{2,2}, P_{2,3}, P_{3,0}, P_{3,1}, P_{3,2}, P_{3,3}\}$

  with its respective probabilities as follows:

  $\{p_{0,0}, p_{0,1}, p_{0,1}, p_{0,1}, p_{1,0}, p_{1,1}, p_{1,2}, p_{1,3}, p_{2,0}, p_{2,1}, p_{2,2}, p_{2,3}, p_{3,0}, p_{3,1}, p_{3,2}, p_{3,3}\}$. The Huffman coding for m pixels can be computed in $O(m \log_2 m)$ time using a greedy algorithm (Horowitz, Sahni & Mehta, 2006). In Huffman coding, a fixed preset coding table is often used which is based on an estimated probability of the pixels to be coded. In adaptive Huffman coding, the coding table is generated individually (adaptively) from the pixel probabilities for the data to be coded.

  Huffman coding although relatively easy to implement and mostly used in JPEG, introduces a number of disadvantages (Leondes, 2002) such as:

  - Compression of images that contain long runs of identical pixels by Huffman coding is not as efficient when compared to Run-Length coding (discussed later in this section).
  - Efficiency depends on the accuracy of the statistical model used, and the type of image.
  - Huffman algorithm design depends on the image formats.
  - It is required to have Huffman table at the beginning of the compressed image.
  - All codes of the coded images are of different sizes (not of fixed length).

- **Lempel-Ziv Welch Coding (LZW)**

  This coding scheme depends only on the relative order of grey scale values. An adaptive coding method has been proposed by Ziv and Lempel (1978:530-536), and Welch (Welch, 1984:8-19) which does not require all the images needed to be available at the beginning of the compression process. The LZW technique examines the source file (image) for compression from beginning to end in order to generate codes (compressed image). In Huffman coding, a variable length code is constructed for each pixel in the source image but in LZW coding, fixed-length codes are constructed. LZW compression is the perfect candidate for reducing the size of files containing more repetitive data (Ziv & Lempel, 1977:337-343), however improving the compression ratio using LZW is not easy especially when the image has variant colour and large size (Taleb, Musafa, Khtoom & Gharaybih, 2010:502-509).

  LZW is mostly used in TIFF and GIF files; it presents an advantage to compress images of fixed-length size.

- **Run-Length Coding (RLC)**

  This coding scheme depends only on the relative order of grey scale values. In RLC, the source file (image) is decomposed into segments of identical grey values (Kou, 1995). Each segment is replaced by a pair of the form (grey value, frequency of occurrence). If the maximum difference within a segment is zero (all pixels in the segment have the same grey value), the RLC is then used to optimally encode the segment regarded as one pixel with its frequency of occurrence. The code-word generated by the RLC contains only the pixel's value and not the segment's size since the segment's size is automatically given by the category of the segment. The category of the segment is defined as a square block MxM where $M = 2^n$ and n is an integer (e.g. $MxM = 2^n x 2^n$ =1x1, 2x2, 4x4, 8x8, 16x16, etc). The length of the code-word depends on the size of the image ($\log_2(NxN)$) with $NxN = 2^l$ (l = length of the code-word, and NxN the size of the image). The length of the code-word also depends on the probability of occurrence of pixels ($l = -\log_2(Prob(pix))$ with $Prob(pix)$ = the probability of occurrence of pixels).

  RLC is a perfect candidate to compress images that contain identical pixels; therefore it is simple to implement, fast to execute, and efficient (Nunez, 2003).
  RLC is mostly used in TIFF, BMP, PCX, and JPEG.

Table 2.1 presents the advantages, disadvantages and application of common coding methods.

Table 2.1: Comparison of common coding methods

	Arithmetic Coding	Huffman Coding	LZW	Run Length Coding
Type of Compression	Lossless compression.	Lossless compression.	Lossless compression.	Lossless compression.
Advantages	Reduces image size dramatically (optimal compression). Represents efficiently frequent sequences of pixel values with fewer bits.	Easy algorithm mplementation.	Fixed-length size of all codes of coded images. Fast compression. Dictionary based technique.	Simple to implement. Fast to execute. Efficient.
Disadvantages	Relatively slow. Complicated to implement.	Not efficient when compared to Run-Length coding. Depends upon statistical model of image. Algorithm varies with different formats. Variable length sizes. Huffman table required at beginning of compression.	Difficulty of compression ratio improvement of large size or variant colour of images.	Performs poorly when compressing images that do not contain long runs of identical pixels
Application	Mostly used for frequent sequence of pixels.	Used in JPEG.	Used in TIFF and GIF files.	Mostly used for TIFF, BMP, PCX files and JPEG.

2.2.2 Spatial Domain Algorithms

Spatial domain compression methods combine spatial domain algorithms and coding methods. Besides operating directly on the grey values in an image, spatial domain compression methods also set to eliminate the spatial redundancy (Starosolski, 2007:65-91). Spatial redundancy has been defined by Shannon (1948:379-423) as a property of codes.

Lossless image compression techniques via spatial domain algorithms operate directly on the raw image in order to reduce the number of bits required to represent the information contained in the image. This method has two stages. The first stage is pre-processing that involves techniques such as image segmentation, image sub-sampling and image interpolation. This is followed by the second stage, an efficient encoder, responsible for encoding the results of the

pre-processing stage. During pre-processing, image segmentation schemes partition the image into regular shapes (rectangles). This has the advantage of taking less storage to code the shape (Ranganathan *et al*., 1995:1396-1406) which has smaller sizes. The smaller shape sizes require fewer bits in order to code the shape. However, a large number of bits might be required to code the entire image. Lossless image compression algorithms that use spatial domain algorithms in pre-processed method (image segmentation) include variable block size compression, "lossy + residual" approaches, and context-based compression algorithms.

2.2.2.1 Variable Block Size Compression

Keissarian (2008:285-292) developed an image compression algorithm for hiding secret data which exploits the smooth area of the host image. The algorithm segments the image into variable size blocks at different rates according to the level of activity inside the block and encodes them by employing quadtree segmentation. Quadtree is a well-known data structure that describes the spatial information of an image. Quadtree segmentation decomposes an image into variable block sizes, and then encodes them at different rates according to the level of activity inside the block. In conclusion, this technique consists of segmenting an image into variable size blocks/segments and then encodes the blocks based on the pixels property.

2.2.2.2 Lossy + Residual Approaches

This technique includes a number of methods of which the basic concept is to first send a lossy image, followed by the residual with respect to the original image. The combination of lossy and residual approach results in an exact reconstruction of the original image. The residual is defined as the difference between the original and the compressed images. This residual image first has to pass through a sequential linearization before being effectively compressed via coding schemes such as Arithmetic coding and Huffman coding. Since lossy compression techniques display high compression ratios, the "lossy + residual" approach has the main advantage of exploiting the high compression ratios obtained from the lossy image compression discussed in Chapter 3.

The general technique of lossy and residual approach uses a sub-sampling and an interpolation method; it exploits an interpolation from a sub-sampled set of points derived from the original image in order to estimate the image.

2.2.2.3 Context-based Compression Algorithms

In image compression algorithms, context-based modelling has become a popular technique; according to Langdon and Haidinyak (1995:21-27), the context is defined in terms of the prediction errors incurred in previous encodings.

The context-based compression algorithms originated from text compression wherein the context of a symbol is defined as the N symbols preceding it. Text compression includes a modelling stage where the model assigns a probability to a symbol based not only on the frequency of occurrence of this symbol but also on the frequency counts of the symbol's context. In the coding stage, the symbols are coded based on their probabilities defined on their contexts.

According to Rissanen and Langdon (1981:12-23), context modelling in data compression consists of two parts: the *structure*, which represents the set of occurrences of the symbols and their contexts, and the *parameters*, which represent the probabilities assigned to each occurrence of the symbols. This can be interpreted as a *tuple* (*parameters, structure*).
The context-based compression algorithms can be classified as CALIC (Wu & Menon, 1996:1890-1893) and LOCO-I (Weinberger, Seroussi & Sapiro, 1996:140-149).

- **CALIC: A Context Based Adaptive Lossless Image Coder**
  Developed by Wu and Menon (1996:1890-1893), CALIC is basically a "lossy + residual" approach. CALIC includes a mechanism to trigger the binary mode automatically in order to code either uniform or binary sub-images or both. CALIC algorithm has two steps namely:

  1. In the first step, the algorithm exploits a gradient-based non-linear prediction to obtain the lossy + residual image. The gradient-based non-linear prediction is located at the current pixel x as depicted in Figure 2.4.

     Figure 2.4 estimates the horizontal and vertical gradients as:
     $$d_h = |w - ww| + |n - nw| + |ne - n| \quad (2.2)$$
     $$d_v = |w - nw| + |n - nn| + |ne - nne| \quad (2.3)$$
     This prediction is still not precise enough for a complete removal of the spatial redundancy in the image. Therefore, the CALIC algorithm presents some complexity during the computation process.

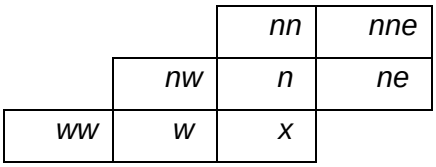


Figure 2.4: CALIC template
(Adapted from Wu & Menon, 1996:1890-1893)

2. In the second step, the algorithm uses the arithmetic coding method to encode the residual probabilities of the symbols in different contexts. However, during this process many unnecessary symbols are introduced, thus wasting memory space.

- **LOCO-I: A Low Complexity, Context-Based Lossless Image Compression Algorithm**
  Developed by Weinberger *et al*., (1996:140-149), LOCO-I uses modelling and prediction units based on the causal template illustrated in Figure 2.5.

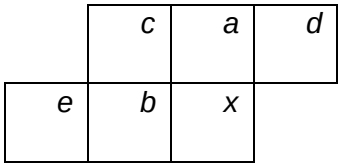


Figure 2.5: LOCO-I causal template
(Adapted from Weinberger *et al.*, 1996:140-149)

In Figure 2.5, x represents the current pixel, while a, b, c, d, and e represent the neighbouring pixels in corresponding locations. The following prediction:

$$\hat{x}_{i+1} = \begin{cases} \min(a,b)\ if\ c\ \geq \max(a,b) \\ \max(a,b)\ if\ c\ \leq \min(a,b) \\ a+b-c\ otherwise \end{cases} \tag{2.4}$$

is used for LOCO-I, which presents a low complexity. This is then followed by the encoding of the prediction residuals using the Golomb-Rice codes (Salomon *et al*., 2010). Golomb-Rice codes are a combination of Golomb codes and Rice codes. Golomb codes (Golomb, 1966:399-401) are codes used for data compression that are distributed geometrically. Rice codes (Rice, 1979; 1991) are a special case of the Golomb codes (when the number to be coded is a power of 2). Rice codes depend on the chosen base (n) and are computed as follows:

1. Separate the sign bit from the rest of the number (this becomes the Most Significant Bit "MSB" of the Rice codes).
2. Separate the n Least Significant Bits "**LSB**s" (they become the **LSB**s of the Rice codes).
3. Code the remaining bits in unary and make this the middle part of the Rice code (if the remaining bits are, say 11, then the unary code is either three zeros followed by a 1 or three 1's followed by a 0).

Golomb-Rice codes are computed with few logical operations, which are faster than Huffman code.

LOCO-I presents low complexity and good compression performance due to its superior classification in terms of compression/complexity trade-off curve, and therefore it has been adapted as part of the new lossless and near-lossless compression of continuous-tone still image JPEG standard known as JPEG-LS (ITU-T ISO/IEC, 2011). However, CALIC can perform a slightly better compression ratio due to the Arithmetic coding but at the expense of additional computation time (run time/complexity).

2.2.3 Transform Domain Algorithms

In transform domain compression methods, an appropriate basis set is used to represent the image with the aim to obtain a spare coefficient matrix. Transform domain compression methods have two examples namely: the Discrete Cosine Transform (DCT) based compression and the wavelet transform based compression.

Image compression using transform domain algorithms is achieved by exploiting the spatial and frequency information contained in the image. According to Heer & Reinfelder (1990:354-365), Rabbani & Jones (1991) and Takamura & Takagi (1994:155-174), for lossless image compression, the most popular multi-resolution transform-based scheme in the field of medical imaging, commercial photography, industrial imaging and satellite imaging, are the S-transform (sequential transform). Said and Pearlman (1996a:1303-1310) have developed an image multi-resolution transform suited for both lossless and lossy compression. This transform requires only integer addition and bit-shift operations.

Calderbank, Daubechies, Sweldens and Yeo (1998:332-369) developed wavelet transforms that map integers to integers for lossless image compression.

2.2.3.1 Wavelet Transform Based Image Compression

The discrete wavelet transform (DWT) is a digital transform technique that can be used for wavelet-based image compression techniques. The wavelet transforms that map integers to integers for lossless image compression were developed by Zandi, Boliek, Schwartz & Gormish (1995) and Dewitte & Cornelis (1996). The modification of the precoder method (Laroia, Tretter & Farvardin, 1993:1460-1463), the combination of the lifting scheme method (Sweldens, 1996:186-200) and the reversible way of rounding to the nearest integer are all methods to achieve integers to integers mapping using wavelet transforms.

The discrete wavelets transform uses a high integer precision that allows high transformation of blocks of pixels. The DWT module takes image resolution in bit per pixel as inputs, to allow a degree of freedom in how the image is decomposed.

According to Tzanetakis, Essl and Cook (2001), the DWT input image is decomposed such that the only unknown variable is the current pixel, which represents a time function with finite energy and fast decay called the mother wavelet.

The wavelet transform based image compression resulting from the new JPEG committee standard on lossy and lossless image compression is the JPEG2000 (Christopoulos, Skodras & Ebrahimi, 2000:1103-1127; ITU-T ISO/IEC, 2004).

Since the wavelet transform coefficients contain information about both spatial and frequency content of an image, discarding a high-frequency coefficient leads to some image degradation in a particular location of the compressed image rather than across the whole image.

2.2.4 Lossless image Compression Standards and Formats

Standard is vital for any image compression system. This section focuses on lossless image compression standards and formats.

2.2.4.1 JPEG lossless compression standard

The JPEG lossless compression standard incorporates coding methods, namely:

1. Lossless method with Huffman coding.
2. Lossless method with Arithmetic coding.

The JPEG lossless compression standard (Bhaskaran & Konstantinides, 1995:47-49) is based on a predictive coding technique. The JPEG lossless compression mode compared to the JPEG lossy compression mode is fully independent of the transform-based coding (JPEG lossy) since

it applies differential coding to form the residuals which are then coded via the Huffman or the arithmetic coding methods. The residual is predicted using previously encoded pixels in the current row and/or previous row of the image. The predicted residual at a pixel location x is defined as $R_x = P_x - x$, with P_x the predicted value from Table 2.2 (Langdon, Gulati & Seiler, 1992:172-180). Figure 2.6 illustrates the pixel values a, b and c that are available at the encoder as well as the decoder prior to processing of location x. The chosen prediction function is encoded into the header of the compressed stream and is therefore easily readable at the decoder end. If the prediction methods (4) through (7) from Table 2.2 are used, values in the first row are predicted using method (1) and those in the first column are predicted using method (2).

Table 2.2: JPEG lossless prediction
(Adapted from Langdon *et al.*, 1992:172-180)

Selection-value	Prediction equation
0	No prediction
1	$Px = a$
2	$Px = b$
3	$Px = c$
4	$Px = a + b - c$
5	$Px = a + (b - c)/2$
6	$Px = b + (a - c)/2$
7	$Px = (a + b)/2$

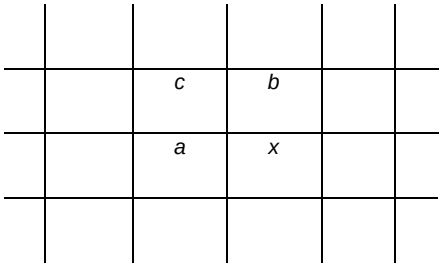


Figure 2.6: Pixel samples used in JPEG lossless prediction

JPEG lossless introduces advantages such as simple computations, easy implementation, and it can easily and efficiently be mapped to the hardware since the algorithm has some simplicity (Vemuri *et al*., 2007).

For JPEG2000 lossless compression standard, the 5/3 wavelet filter is used by default (LeGall & Tabatabai, 1988:761-764). The JPEG2000 (Joint Photographic Experts Group 2000) standard is a wavelet-based image compression standard, which was initiated in March 1997 (JPEG2000 Standards, 1997) in order to overcome the original DCT-based JPEG standard disadvantages.

The 5/3 wavelet filter is used in the JPEG2000 standard by default for lossless compression, but can also be used in order to achieve a lossy compression. The JPEG2000 standard is an improvement of the JPEG standard, in that it offers the possibility for lossy and lossless compression techniques. The primary reason for JPEG2000's superior performance is attributed to the wavelet transform (Lyons, 2007). It offers advantages such as random access localisation within the image, parallelisation, improvement of cropping and rotation functionality, improvement of error resilience, efficient rate control and maximum flexibility in arranging progression orders (Taubman, 2000:1158-1170).

JPEG2000 provides a strong compression technique that can be modified in different ways to better suit any application. Recognising redundancies such as coding redundancy, inter-pixel redundancy, and psycho-visual redundancy in data, is a crucial step in image compression (Gonzalez & Woods, 2004). Data redundancies are defined as unnecessary data used to represent information. By removing unnecessary data, the size of the data decreases without losing any vital information, therefore compression is achieved (Al-Mualla, Canagarajah & Bull, 2002).

According to Skodras, Christopoulos and Ebrahimi (2001:36-58), the JPEG2000 standard has a better Peak Signal-to-Noise Ratio (PSNR) than the JPEG standard; for several images across all compression ratios, the JPEG2000 standard has a margin of approximately two decibels (dB) ahead of the JPEG standard.
The PSNR is commonly defined as the objective image quality degradation, which is the ratio of the maximum possible power of signal and the power of the reconstruction errors known as the root-mean-squared-error (RMSE) (Smith, 2007). It can be expressed in terms of the logarithmic decibel scale defined by equation (2.8).

$$PSNR = 20\log_{10}\left(\frac{Max_1^2}{RMSE}\right) \quad (2.5)$$

with $$Max_1^2 = \sqrt{\frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1} f^2(i,j)} \quad (2.6)$$

$$RMSE = \sqrt{\frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\|f(i,j) - f'(i,j)\|^2} \quad (2.7)$$

therefore, $$PSNR = 10\log_{10}\left(\frac{\sum_{i=0}^{M-1}\sum_{j=0}^{N-1} f^2(i,j)}{\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\|f(i,j) - f'(i,j)\|^2}\right) \quad (2.8)$$

where Max_1^2 is the maximum possible power of signal, $RMSE$ is the power of the reconstruction errors, $f(i,j)$ is the original signal/image, $M\ x\ N$ is the size of the original signal/image, $f'(i,j)$ is the reconstructed signal/image, and i,j refer to the pixel position in the image.

The corresponding PSNR in dB for an 8-bit image is computed as

$$PSNR = 20\log_{10}\left(\frac{255}{RMSE}\right) \quad (2.9)$$

2.2.4.2 Lossless Image Compression Formats

Lossless image compression formats exist in various types and each of them has specific applications in digital image storage and retrieval. The basic image formats are:

- **GIF Image Format**

  The GIF (Graphics Interchange Format) uses lossless compression to achieve compression on images with colours up to 256. GIF compression works best on identical, adjacent pixels or rows of identical, adjacent pixels. The compression acts best on images with colours or images in which one colour is dominant (Fu, Fan & Wang, 2011:2747-2750). GIF utilises the LZW algorithm.

- **PNG Image Format**

  The PNG (Portal Network Graphics) is a bitmap image format created in 1996, to improve and replace the GIF image format with a non-licensed image format.

  PNG uses the combination of the LZ77 algorithm as well as Huffman coding. PNG is designed for image distribution over the internet, but not for professional graphic applications such as 3D images (Huang & Zheng, 2008:1-4).

- **JPEG2000 Image Format**

  JPEG2000 image format utilises EBCOT (Taubman, 2000:1158-1170) and presents numerous advanced features as improvements on JPEG (Lee, 2005:32-41) which are discussed in the following:

  1. *Superior compression performance:* JPEG2000 has a higher percentage of compression compared to JPEG at high bitrates, where artefacts are imperceptible. At lower bitrates, JPEG2000 still has a much more significant advantage over JPEG.

  2. *Multiple resolution representation:* Using segmentation, JPEG2000 can handle large image size in one single code-stream which provides similar compression of image components whereby each component is provided by 16 bits per component sample.

  3. *Progressive transmission by pixel and resolution accuracy:* JPEG2000 code-stream is accurately organised in the progressive manner using the image pixel, image quality Signal-to-Noise Ratio (SNR), image resolution or image size.

  4. *Lossless and lossy compression:* JPEG2000 exploits its reversible (integer) wavelet transforms to achieve both lossless and lossy compression from the same compression architecture.

  5. *Random code-stream access and processing:* JPEG2000 code-streams present many possibilities to enhance access to a specific region or region-of-interest of images.

  6. *Error resilience:* Since JPEG2000 uses image segmentation, images are segmented in small independent blocks that make JPEG2000 robust to bit errors introduced by noisy communication channels such as wireless.

  7. *Sequential buildup capability:* JPEG2000 provides image compression based on top-down approach in a sequential way with no need to buffer the entire image.

  8. *Flexible file format:* JPEG2000 presents different file formats such as the JP2, J2K and JPX.

2.3 Conclusion

In conclusion, lossless image compression techniques preserve the information so that the exact reconstruction of the image is possible from the compressed data. Consequently, lossless image compression is suitable for applications where a better image quality after compression is required such as in the transmission of reference images, especially on medical images since the effect of lossy coding artifacts on diagnostic accuracy is an important consideration.

Since images are reproduced exactly, the main goal of a lossless compression is to maximise the achievable compression, specified in terms of the *compression ratio*.

CHAPTER 3

LOSSY IMAGE COMPRESSION

3.1 Introduction

Lossy image compression is also referred to as irreversible image compression due to the loss of some information during the compression process. Lossy image compression techniques are achieved via coding methods, spatial domain compression methods, frequency domain compression methods or the combination of these methods.

3.2 Lossy Image Compression Techniques

Lossy image compression techniques are different from those of lossless due to the fact that a quantisation step is introduced that defines the amount of compression and distortion associated with the lossy system. The distortion introduced in the system results in the loss of some information during the compression process. The quantisation step is the simplest type of lossy compression and maps a range of values to a smaller range of values. The quantisation step is always implemented before the compression method i.e. coding methods, spatial domain compression methods, or the frequency domain compression method.

3.2.1 Coding Methods

For lossy image compression technique, common coding methods include: Arithmetic, Huffman, Lempel-Ziv Welch (LZW), and Run-Length as discussed in Lossless Image Compression. The quantisation step is always implemented before these coding methods.

In order to improve the compression ratio, the compression system has to be coupled with a coding scheme that can take advantage of local and global redundancy. Fractal coding (Jie, Zhongshan, Huiling, Peihuang & Guoning, 2008:487-490) is a method to improve a lossy image compression ratio.

Fractal coding technique developed in the seventies by Mandelbrot (1977) is based on the theory of fractal geometry defined as a concept of self-similarity in fractals. It uses fractal components that have self-similarity to the rest of the surrounding area in the picture. Basically, a fractal coding represents the overall image structure where some regions have similarity or correlation. Fractal coding technique applies a technological approach that is a structure iterated function system (IFS) representing the self-similarity of the original image (Ebrahimpour-Komleh, Chandran & Sridharan, 2004:664-672). This approach consists of using a sub-block being closer

to another sub-block of the image through self-affine transform. In order to achieve an image compression improvement, only the corresponding IFS must be transmitted or stored. The first step in implementing a fractal coding technique is to segment the image to non-overlapping blocks (e.g. 8x8) or larger (e.g. 16x16) into domain blocks (possibly overlapping). The segmentation defines a set of admissible block transforms consisting of a reduction of the blocks. Each block is supported by a factor of two on each side through average neighbouring pixels. The fractal coding technique is accomplished by finding each range block, a domain block for which the pixel values can be made close to those of the range block while applying an admissible transform.

It is vital to select the transforms with care since their union is a contractive transform on the image as a whole. Fractal coding technique is a very broad framework which can be classified in the following categories (Monro & Dudbridge, 1992a:485-488, 1992b:1053-1055; Monro, 1993:362-363):

1. The segmentation imposed on the image by the range blocks (Segmentation Techniques).
2. The composition of the pool of domain blocks, which is restricted to some extent by the range segmentation (The Pool of Domain Blocks).
3. The class of transforms applied to the domain blocks (Block Transforms).
4. The type used in order to locate suitable domain blocks (Search Strategies).
5. The quantisation of the transform parameters and any subsequent entropy coding (Quantisation).

3.2.1.1 Segmentation Techniques

Image segmentation can result in rectangular, triangular, or polygonal blocks although the majority of coding techniques are based on square or rectangular segmentation.

- **Rectangular Range Segmentation Techniques**

  These techniques are represented by fixed square blocks, quadtree blocks, and horizontal-vertical blocks depicted in Figure 3.1.

  - **Fixed square blocks**

    Fixed square blocks represent the simplest possible range segmentation (Oien, Lepsoy & Ramstad, 1991:2773-2776; Fisher, 1992:903-919; Fisher, Jacobs & Boss, 1992:35-61; Barnsley & Hurd, 1993). Monro and Dudbridge (1992a:485-488, 1992b:1053-1055) & Monro (1993:362-363) developed the most prominent example of fractal coding

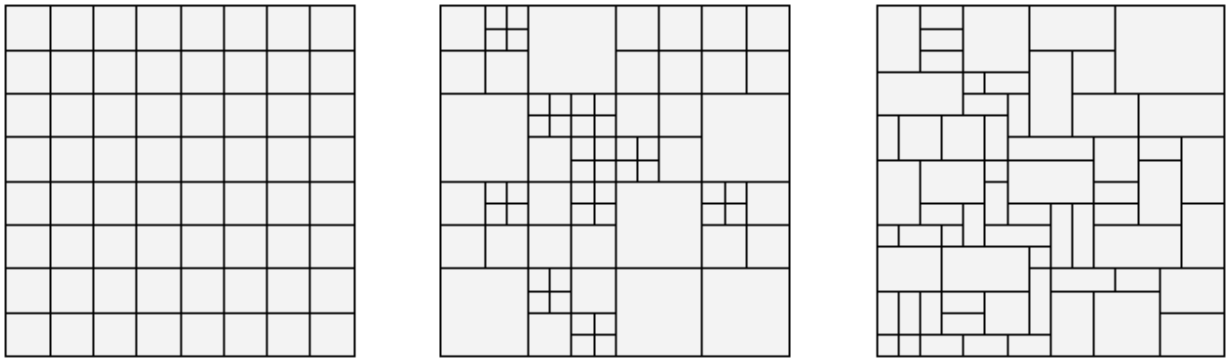
technique based on this segmentation. This type of block segmentation is very successful in transform coding of individual image blocks.

- **Quadtree blocks**

  The quadtree segmentation (Saupe, 1994) is implemented by splitting a selected image into quadrants, while enabling the resulting partition to be represented as a tree structure in which each node has four descendents. The segmentation is made by selecting an initial level in the tree and recursively segmenting any block which has a better match than some preselected threshold which is not found.

- **Horizontal-vertical blocks**

  The Horizontal-vertical (HV) segmentation (Fisher *et al*., 1992:35-61; Fisher, 1992:903-919, 1995) depicted in Figure 3.1 produces a tree-structured segmentation of the image as in the quadtree. In HV each image block is split into two either horizontal or vertical lines. Horizontal or vertical segmentations along prominent edges, while avoiding the creation of narrow rectangles, are created by exploiting the heuristic algorithm (Fisher, 1995).



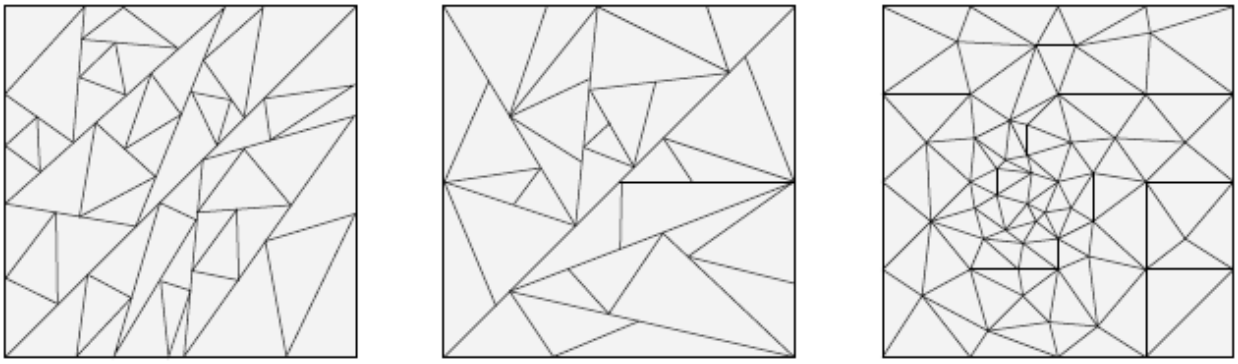
(a) Fixed Square Blocks **(b) Quadtree Blocks** **(c) Horizontal-vertical blocks**

Figure 3.1: Rectangular range segmentation techniques
(Adapted from Fisher, 1995)

- **Triangular Range Segmentation Techniques**

  The triangular range segmentation techniques can be generated in different ways. The triangular segmentation is generated by splitting the image into two triangles, followed by a recursive splitting of triangles into four sub-triangles by inserting lines between split points on each side of the original triangle (Fisher, 1992:903-919). Novak (1993:6-15) also employed a similar recursive scheme by splitting each triangle into two while inserting a line from a vertex

of the triangle to a point on the opposite side. Alternatively, triangle segmentation is based on a Delaunay triangulation (Preparata & Shamos, 1985) of the image depicted in Figure 3.2 , which is constructed by exploiting an initial set of “seed points”, then adapted to the image by adding extra seed points in regions of high image variance (Davoine, Bertin & Chassery, 1993:56-57; Davoine & Chassery, 1994:801-803; Davoine, Antonini, Chassery & Barlaud, 1996:338-346).



(a) Fisher Split **(b) Novak Split** **(c) Delaunay Triangulation**

Figure 3.2: Triangular range segmentation techniques

(Adapted from Preparata & Shamos, 1985; Fisher, 1992:903-919; Novak, 1993:6-15)

- **Polygonal Range Segmentation Techniques**

  The polygonal range segmentation techniques (Reusens, 1994:171-174) consist of constructing a polygonal segmentation (illustrated in Figure 3.3) by recursively subdividing an initial coarse grid. The information required in encoding the segmentation details can be achieved by ensuring that each polygon is subdivided at an arbitrary position in the polygon by inserting a line segment at one of the restricted set of angles.

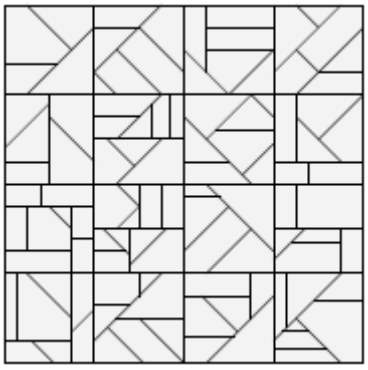


Figure 3.3: Polygonal range segmentation techniques

(Adapted from Reusens, 1994:171-174)

3.2.1.2 The Pool of Domain Blocks

The domain pool used in fractal coding technique is often referred to as a virtual codebook, compared to the Vector Quantisation (VQ) codebook (Jacquin, 1993:1451-1465). From this comparison it is crucial to have a suitable domain pool for efficient representation. Although increased fidelity may be obtained by allowing searching over larger set domains, there is a corresponding increase in the number of bits required to specify the selected domain.

3.2.1.3 Block Transforms

Block transforms are critical elements of the fractal coding techniques since their quantised parameters comprise the majority of information in the compressed representation.

Block transforms are classified as block support, defined as “geometric” transforms in Jacquin’s terminology (Jacquin, 1993:1451-1465). Transforms are applied and restricted by the block segmentation techniques studied previously (rectangular blocks, triangular blocks, and polygonal blocks). The block intensity is defined in terms of “massic” transforms by Jacquin (1993).

3.2.1.4 Search Strategies

Search strategies result in lengthy coding times for fractal encoding techniques and require significant computation. Existing design of efficient domain search techniques in fractal coding have been proposed by Saupe and Hamzaoui (1994:211-229).

3.2.1.5 Quantisation

Quantisation is usually uniform (Fisher, 1995, Øien *et al*., 1991:2773-2776) with the possibility of compensation for inefficiency by subsequent entropy coding. Monro and Woolley (1994a:557-560, 1994b:168-171) have discussed the quantisation optimisation for polynomial fixed block transforms, and the VQ of the transform coefficients for the frequency domain transform is explained by Barthel and Voyé (1994:33-38).

3.2.2 Spatial Domain Algorithms

Lossy image compression techniques via spatial/time domain algorithms are mostly based on delta modulation. In this coding the step size is changed in accordance with the time-varying slope characteristics of the input signal. Types include: Differential Pulse Code Modulation (DPCM), Scalar Quantisation (SQ) and Vector Quantisation (VQ).

3.2.2.1 Differential Pulse Code Modulation (DPCM)

The DPCM (Goyal & O'Neal, 1975:660-666) method eliminates the inter-pixel redundancy that exists between adjacent pixels. The redundancy is removed by encoding the difference between the predicted value of the pixel and its actual value. This difference is known as the predicted error that represents the new information in the pixel. The DPCM model for lossy predictive coding (Kobayashi & Bahl, 1974:164-171) is obtained by introducing the quantisation step and feedback, as depicted in Figure 3.4.

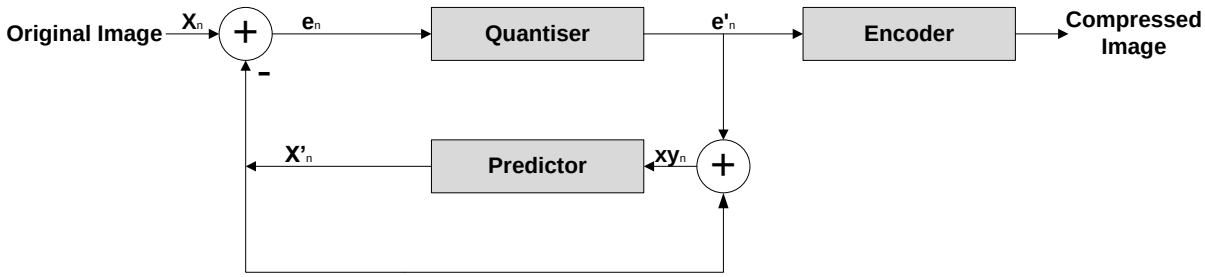


Figure 3.4: Differential Pulse Code Modulation model
(Adapted from Kobayashi & Bahl, 1974:164-171)

Lossy compression techniques include a quantisation step which maps the prediction error (e_n) to a limited number of outputs (e'_n). In the lossy system, the predictor and the quantiser define the amount of compression and distortion associated to the image.

In Figure 3.4, X_n is the input pixel and X'_n is the prediction of X_n determined by the predictor. The prediction error e is the difference between the actual pixel value X_n and its predicted value X'_n ($e_n = X_n - X'_n$). The quantisation step quantises the prediction error e_n to e'_n, then the quantised value e'_n is added to the predicted value X'_n to form the estimated value of X_n (xy_n). The estimated value of X_n (xy_n) is an input to the predictor, resulting in the next predicted value X'_{n+1} (the prediction value of X_{n+1}). The encoder encodes each quantised value of e'_n in order to accomplish the compressed image.

3.2.2.2 Scalar Quantisation (SQ)

During the quantisation process in lossy compression techniques of continuous random variable, the distortion is introduced and consequently, insufficient bits are available which represent the full range of the original image. The lossy system may be represented as a single function $y = Q(x)$ where the original image (source value) is x and the quantiser output value is y as illustrated in Figure 3.5.

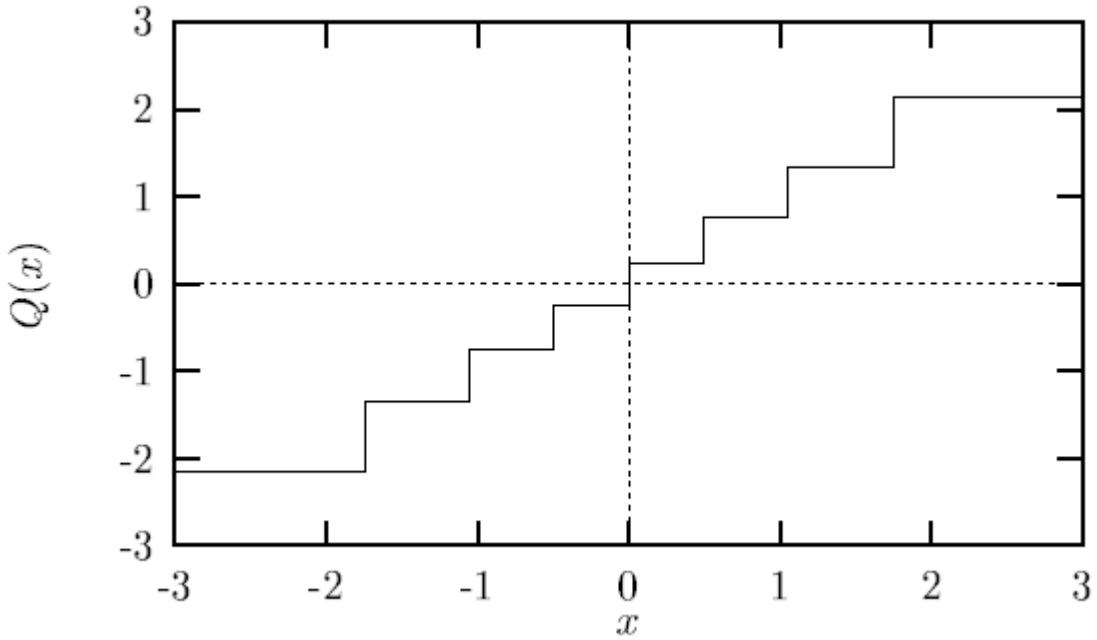


Figure 3.5: Scalar quantisation function Q(x)

The scalar quantization system consists of encoding each output value individually. Let (x, d) be a quantization problem with source (input) alphabet X and output alphabet Y. To approximate the source value x and the quantiser output value y, the encoder represents a source value $x \epsilon X$ and the quantiser output value $y \epsilon Y$. A scalar quantiser is therefore a mapping $q : X \rightarrow Y$ such that $|q(X)|$, the number of possible quantised values, is either finite or countably infinite.

3.2.2.3 Vector Quantisation (VQ)

Vector quantisation is a pure representation of scalar quantisation to joint quantization of a vector of scalar values. The VQ is more advantageous for lossy compression compared to the scalar quantisation due to the existence of dependence between the scalar elements of a vector. In image compression, vector quantisation plays both functions – as primary coding mechanism and as secondary quantisation step – after the primary and secondary function, the following methods such as transform or sub-band coding (Gersho, 1982:157-166; Ramamurthi & Gersho, 1986:1105-1115; Nasrabadi & King, 1988:957-971) are implemented.

The VQ in its primary function consists of tiling the image by sub-blocks (e.g. 8x8, 16x16), each being considered as separate vector. A codebook is constructed based on a large number of these vectors generated from a training set of images, and the encoding is used to represent each block with the index of the closest codebook vector. From the rate of distortion theory, the VQ becomes more efficient with the increase of vector size; therefore the individual image sub-

blocks should be as large as possible. Practically, block size is limited due to the rapid increase in both codebooks size and block size. The larger the codebook, the bigger is the computational effort. For better computation, the computational effort can be compensated by using a Tree Structured VQ (TSVQ) which requires an enlarged codebook. This achieves a significant improvement in search time by providing a tree structure on the codebook. The challenges associated with large vectors is dealt with by Lattice VQ (LVQ) (Gersho & Gray, 1992), by designing a codebook based on a lattice of vectors arranged according to some regular structure. According to Fischer (1986:568-583), efficient quantisation of large vectors with a variety of distributions is possible using lattice VQ techniques.

3.2.3 Frequency Domain Algorithms

Lossy image compression techniques via frequency domain algorithms are classified as filter based and transform based algorithm.

3.2.3.1 Filter Based Algorithm

This technique consists of compressing images using filter based algorithms such as sub-band and wavelet.

- **Sub-band**

  The sub-band coding presents the possibility of coding the sub-band separately by adapting the coding procedure to the statistics of each sub-band; different scalar quantisation is applied to each sub-band for the simplest system (Jayant & Noll, 1984) while vector quantisation is applied to each sub-band for more complex coding techniques (Cosman, Gray & Vetterli, 1996:202-225) of vectors belonging to each sub-band (Antonini, Barlaud, Mathieu & Daubechies, 1992:205-220). A filtering based algorithm via sub-band is accomplished by passing a signal (image pixels) through a multichannel filter bank that is represented by two-channel filter bank (Pande & Zambreno, 2008) as depicted in Figure 3.6.

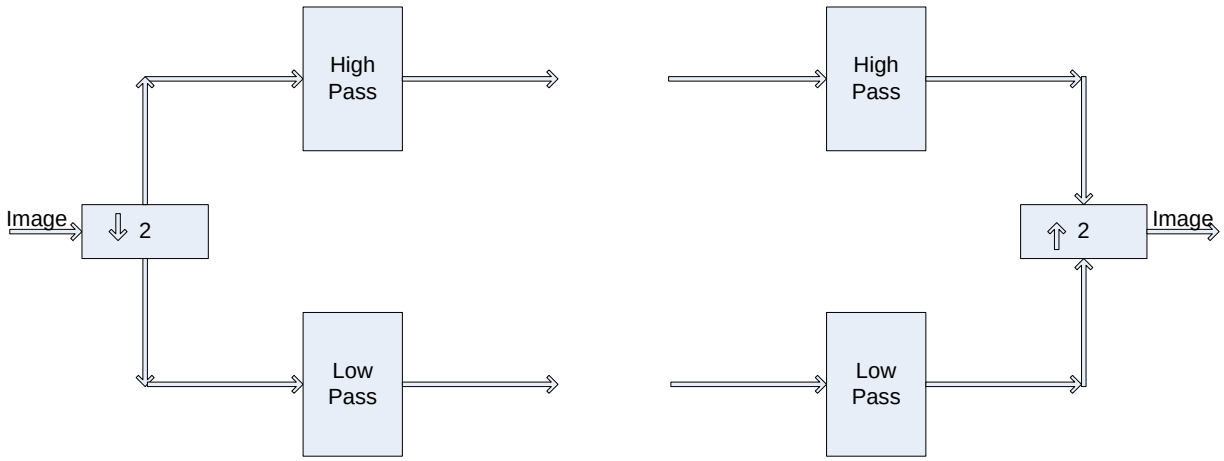


Figure 3.6: A Two-channel Filter Bank (Signal/Image analysis and reconstruction)
(Adapted from Pande & Zambreno, 2008)

In Figure 3.6, the initial filter (left hand side) is denoted as an analysis filter (image compression), and the final filter (right hand side) is denoted as a synthesis filter (image reconstruction).

a) The analysis filter is a process that consists of splitting the input signal/image in Low-Pass and High-Pass filter and sub-sampling the respective filters in order to produce respectively the length of Low-Pass and High-Pass filter used for image decomposition (image compression).
b) The synthesis filter is a process that consists of up-sampling the length of Low-Pass and High-Pass filter in order to gather Low-Pass and High-Pass filter to produce an image reconstruction (image decompression).

Due to the fact that sub-band decomposition is computed using linear operations, the filtering based algorithm via sub-band decomposition results from a linear transformation (Habibi, 1971:948-956; Wohlberg & de Jager, 1999:1739-1742). The analysis part of the filter bank can be decomposed at multichannel levels as illustrated in Figure 3.7.

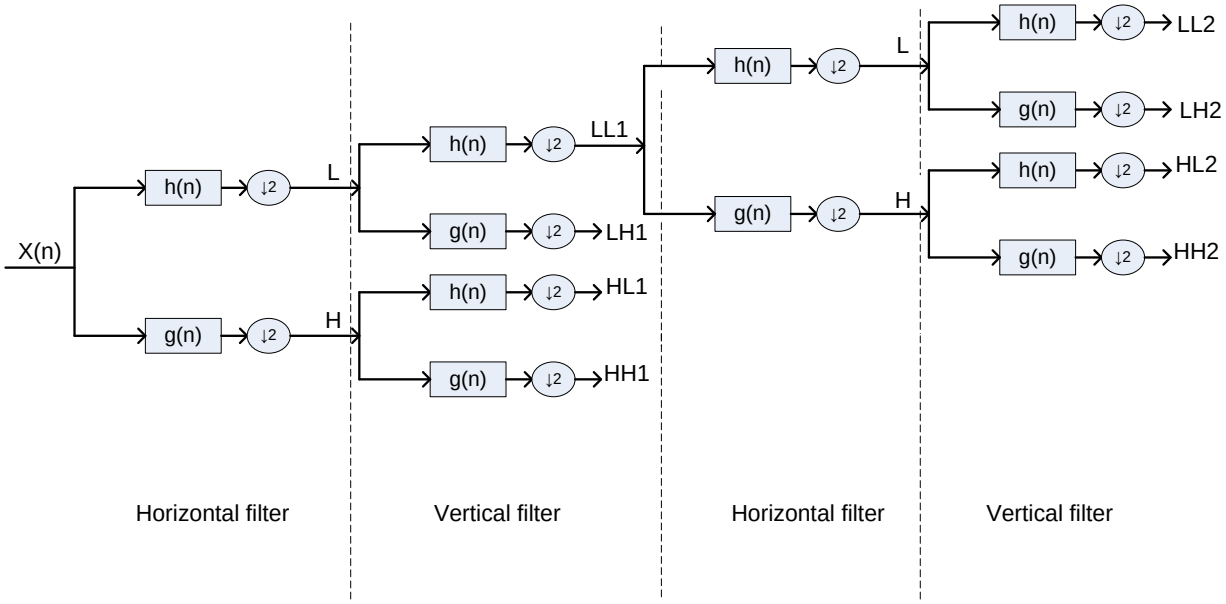


Figure 3.7: Sub-band decomposition

In Figure 3.7 , the input signal $X(n)$ is split through two stages of filters respectively, namely Low-Pass ($h(n)$) and High-Pass ($g(n)$). $X(n)$ is processed horizontally, and then sub-sampled by two. The outputs $L,\ H$ respectively from the Low-Pass ($h(n)$) and High-Pass ($g(n)$) filter are processed vertically. The outputs $LL_1,\ LH_1,\ HL_1,\ and\ HH_1$ are represented as one-level decomposition; this decomposition process can be expanded to multi-level decomposition i.e. $LL_2,\ LH_2,\ HL_2,\ and\ HH_2$.

The number of decompositions' levels can be up to 32 but practically, five-level decomposition provides superior image quality (Taubman & Zakhor, 1994:572-588).

- **Wavelet**

The wavelet is described as small waves used in mathematics to decompose functions (Sifuzzaman, Islam & Ali, 2009). The decomposed function has two fundamental components: the wavelet coefficients and the wavelet bases. If function f is defined to be decomposed using wavelet coefficients $a_{x,y}$ and wavelet bases $\Psi_{x,y}(t)$:

$$f = \sum_t a_{x,y}\ \Psi_{x,y}(t) \qquad (3.1)$$

To have a real representation of a function f, it is very important to choose a suitable family of functions $\Psi_{x,y}$. In this research project, the image is the data; therefore the functions $\Psi_{x,y}$ should match its features.

To be able to locate a *resolution*'s variable (time-frequency), Figure 3.8 is used to define a wavelet function $\Psi(t)$ called *prototype* or *mother wavelet*.

Figure 3.8: Mother wavelet function

The DWT has a resolution level i at time j. To achieve a 2D wavelet image filtering based algorithm, the following expression is used:

$$\Psi_{i,j}(t) = 2^{\frac{-i}{2}} \Psi(2^{-i}t - j) \tag{3.2}$$

Figure 3.9 illustrates the resolution functions at different resolutions. At different resolutions, the wavelet must form a basis biorthogonal, meaning they must be mutually biorthogonal.

High frequencies **mid frequencies** **low frequencies**

Figure 3.9: Representation of wavelets at different resolutions

The image (signal) can be interpreted as the summation of wavelet coefficients at infinite number of resolutions as defined in equation (3.3).

$$f(t) = \sum_{i,j} a_{i,j} \Psi_{i,j}(t) \;\Rightarrow\; a_{i,j} = \int_{-\infty}^{+\infty} f(t)\, \Psi_{i,j}(t) dt \tag{3.3}$$

According to Usevitch (2001:22-35), equation (3.3) is called the *classic wavelet transform* which is similar to the Fourier Transform Theorem.

In order to maintain a high image quality, an image filtering based algorithm via wavelet domain can only result in the LeGall (5/3) wavelet filter for lossless compression and the Daubechies (9/7) wavelet filter for lossy compression (Acharya & Tsai, 2005:149-151).

a) 5/3 Wavelet Filter

5/3 wavelet filter is a filter with a length of 5; 5 and 3 represent its Low-Pass and High-Pass filter's coefficients respectively (Gholipour, 2011:161-172). The paired Low-Pass and High-Pass filters are biorthogonal. The 5/3 wavelet filter is implemented using integer arithmetic that generates reversible transform, facilitating lossless compression (Du & Swamy, 2010:723).

The biorthogonal 5/3 filters are symmetrical and since they are almost orthogonal (Sweldens, 1996:186-200) they find more applications in image compression.

b) 9/7 Wavelet Filter

9/7 wavelet filter is a filter with a length of 9; 9 and 7 represent its Low-Pass and High-Pass filter's coefficients respectively. 9/7 implies that the analysis filter is formed of a 9-tap Low-Pass Finite Impulse Response (FIR) filter and a 7-tap High-Pass FIR filter; they are both symmetric. Due to truncation involved in the irrational filter coefficients, this filter will lead to lossy compression but with the advantage of a better compression ratio than the 5/3 wavelet filter (Pande & Zambreno, 2008).

A filtering based algorithm via wavelet takes an input image and splits it into odd and even coefficients, does some predictions, then updates the coefficients before the encoding (compression) process.

By splitting the input signal, the wavelet filter is divided in Low-Pass and High-Pass filter respectively. Figure 3.10 illustrates the wavelet filter block diagram.

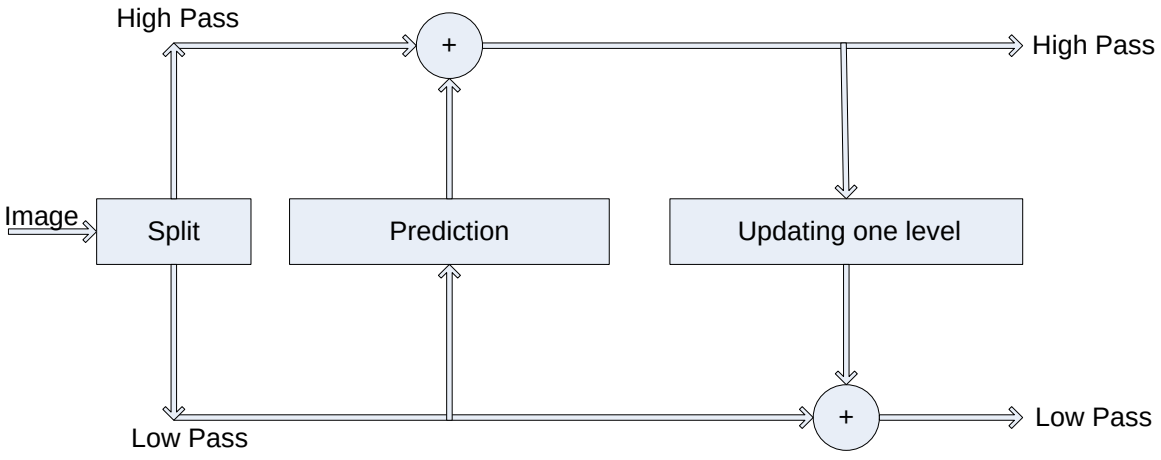


Figure 3.10: Wavelet filter block diagram

(Adapted from Dia, Zeghid, Saidani, Atri, Bouallegue, Machhout & Tourki, 2009:1-5)

❖ **Split**

The input signal (image) is split respectively in a signal with coefficients at odd positions (High-Pass) and a signal with coefficients at even positions (Low-Pass). After the prediction stage, the signal is updated and the split process is carried out by dropping one of the signals (Low-Pass or High-Pass) in order to generate the wavelet coefficients.

❖ **Prediction**

Prediction is a step whereby the predicted next stage input signal $y(2n+1)$, is defined using the previous signal with coefficients at odd position $x(2n+1)$, the previous signal with coefficients at even position $x(2n)$, and the sampling of the two. The predicted next stage input signal is then defined as:

$$y(2n+1) = x(2n+1) + \left\{-\frac{1}{2}[x(2n) + x(2n+2)]\right\} \quad (3.4)$$

❖ **Up-dating**

The predicted stage input signal $y(2n+1)$ is updated resulting in a new signal with coefficients at even position $z(2n)$ defined as:

$$z(2n) = x(2n) + \frac{1}{4}[y(2n-1) + y(2n+1) + 2] \quad (3.5)$$

Equations (3.4) and (3.5) are depicted in Figure 3.11.

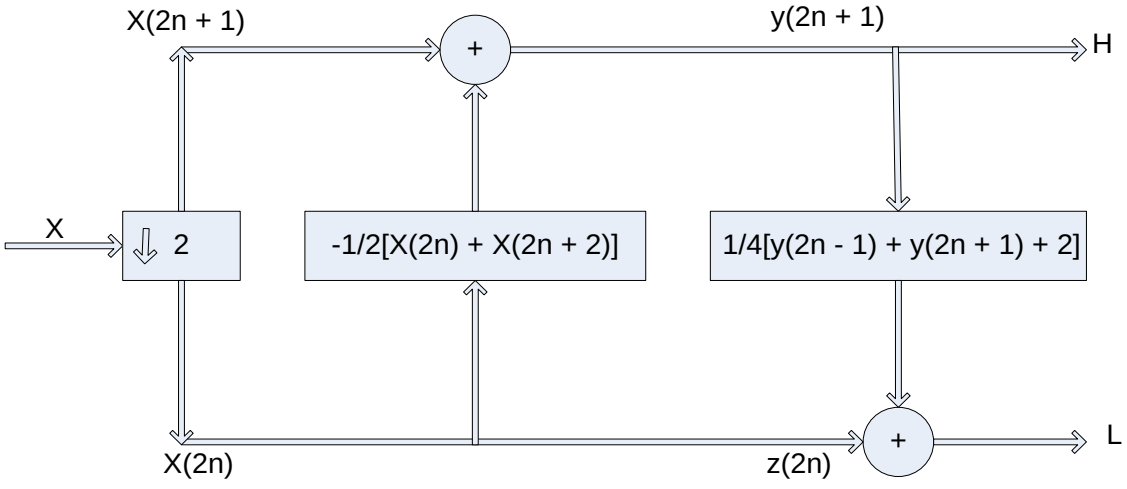


Figure 3.11: Decomposition of the wavelet filter

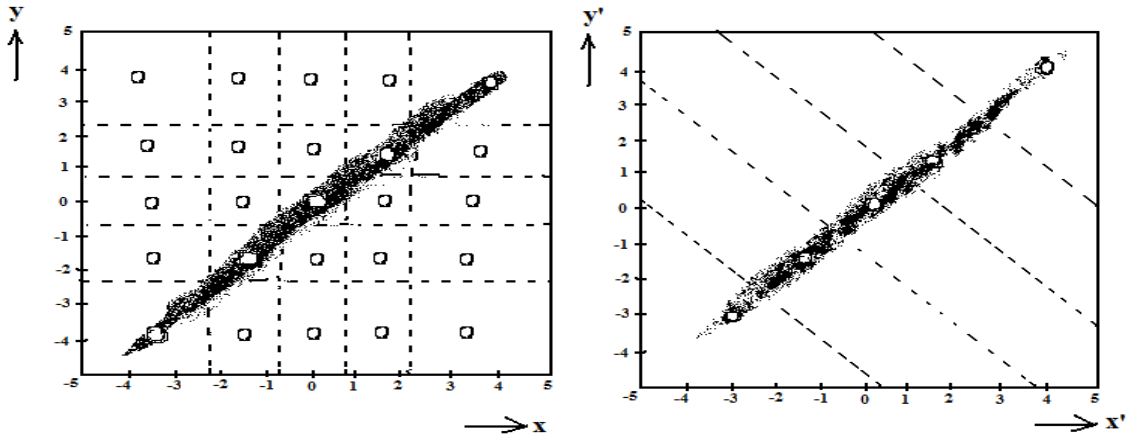
(Adapted from Dia *et al*., 2009:1-5)

Image filtering via wavelet filter has the advantages of multi-scale image representation and non-linear filtering (Wu, Schulze & Castleman, 1998:325-328) mostly used in medical imaging, commercial photography, industrial imaging, and satellite imaging.

3.2.3.2 Transform Based Algorithm

This technique consists of reducing or removing inter-pixel correlations in an image by applying linear transformation to the image, or to individual sub-blocks of the image. Compression is achieved through the subsequent quantisation of the transform coefficients.

Since an image is a continuous data sequence of pixel values, the pre-processing step partitions these pixel values into rectangular and non-overlapping tiles of equal sizes. Transform coding implements the tiles independently in order to de-correlate the pixels so that redundancy and irrelevant information can be removed. The transform encoding method is basically the discrete cosine transform (DCT) or the discrete wavelet transform (DWT). Transform coding converts the image (pixels) into a series of cosines/wavelets (frequency domain) for better storage. By converting pixels to frequencies, the pictures can easily be de-correlated so that statistical redundancy and irrelevant information are removed as depicted in Figure 3.12.



(a) Correlated image with redundancies (white dots) **(b) De-correlated image by rotating the co-ordinate axe**

Figure 3.12: Transform Coding

- **Fourier**

  Image compression via transform based algorithm consists of incorporating the Fourier Transform (Pratt, Kane & Andrews, 1969:58-68) and its digital equivalent known as the Discrete Fourier Transform (DFT). The DFT is calculated using the Fast Fourier Transform (FFT).

- **Discrete Cosine Transform (DCT)**

  The DCT is widely used for the JPEG (Pennebaker & Mitchell, 1993) lossy compression standard, which is based on the coding of 8x8 image blocks in tiling of the full image. In order to ease the coding process, the threshold sampling is applied by dividing each DCT coefficient by the corresponding entry in a quantisation table before rounding to the nearest integer. The normalised values are implemented through the *zigzag* linearization process before being encoded using either Huffman or Arithmetic coding.

  The DCT is related to the Discrete Fourier Transform (DFT) in such a way that it represents the cosine components of the DFT. According to Wallace (1992:xviii-xxxiv), DCT compression can be obtained by using the Forward Discrete Cosine Transform (FDCT) as a harmonic analyser (encoder) and the Inverse Discrete Cosine Transform (IDCT) as a harmonic synthesizer (decoder). For the FDCT process, each 8x8 block of source image samples containing one of the 64 unique two dimensional (2D) *spatial frequencies* which have the input signal's *spectrum* to the FDCT are converted to the image data in a domain where the data representation is more compressible. The amplitude of the converted signal is called DCT coefficients. The FDCT decomposes its input into 64 orthogonal basis signals. The resulting output from the FDCT is a set of 64 DCT coefficients whereas the coefficient with zero frequency is the *DC coefficient*, and the remaining 63 coefficients are the *AC coefficients*. This resulting output is quantised then coded.

During the quantisation process, each DCT coefficient is divided by its corresponding quantisation step-size followed by rounding to the nearest integer value as from equation (3.6). In the JPEG standard, the quantisation step-size is always at the threshold value (max) meaning greater than one, in order to achieve the best compression ratio. At any quantisation step-size greater than one, it will be distortions during image reconstructions consequently, lossy image compression. This is the fundamental reason why the JPEG standard always results as lossy (Said & Pearlman, 1996b:243-250).

$$F^Q(u,v) = IntegerRound\left(\frac{F(u,v)}{Q(u,v)}\right) \qquad (3.6)$$

Where $F^Q(u,v)$ is the quantised FDCT coefficient, $F(u,v)$ is each of the 64 DCT coefficients, and $Q(u,v)$ is the quantisation step-size.

3.3 Lossy image Compression Standards and Formats

Standards and formats are crucial for any image compression system. This section focuses on lossy image compression standards and formats.

3.3.1 JPEG lossy compression standard

The JPEG compression standard is a combination of The DCT and DWT. The JPEG standard was created in 1992 which is the DCT method of 8x8 blocks of source image samples specified as lossy compression. The JPEG standard represents different simple lossy compression modes of operation known as the Baseline methods and expanded as:

- The DCT based-mode, the DCT sequential-mode, the DCT progressive-mode, and the DCT hierarchical-mode.
- The DCT based compression method is an essential compression of a stream of 8x8 blocks. The DCT sequential-mode compression technique that includes the Baseline sequential compression provides the means by which a single compression element works in a fairly complete way.
- The DCT progressive-mode compression technique consists of an image buffer that exists just before the entropy encoding step to allow images to be stored for multiple scans in order to successively improve the image quality.
- The DCT hierarchical-mode of compression technique is mostly used within a larger framework as building blocks (Salomon *et al*., 2010).
- JPEG can also be achieved via a DWT method by inserting a quantisation step between image transformation (DWT), and the entropy encoding.
- For lossy compression, the 9/7 wavelet filter is used in the JPEG2000 standard by default (see Section 3.2.3.1).

3.3.2 Lossy Image Compression Formats

Lossy image compression format has specific applications within digital image storage and retrieval. The basic image format is:

JPEG Image Format

The JPEG image format is a DWT or DCT-based image compression. The DWT was initiated in March 1997 (Standards, 1997) in order to overcome the original DCT-based JPEG *artefacts* which resulted in the JPEG2000.

In summary, lossy image compression is used for applications that are not too restricted for better image quality but that satisfy the storage and bandwidth constraints. Although lossy image compression does not provide a better image quality, it provides a better compression ratio, therefore there is a strong relationship between the compression rate and the quality of the resulting image. This relationship depends upon the image being compressed, the compression method, and the choice of some parameters (region of interest, quantisation step size) used by the compression method.

Figure 3.13 summarises a digital image compression by showing both lossy and lossless compression techniques.

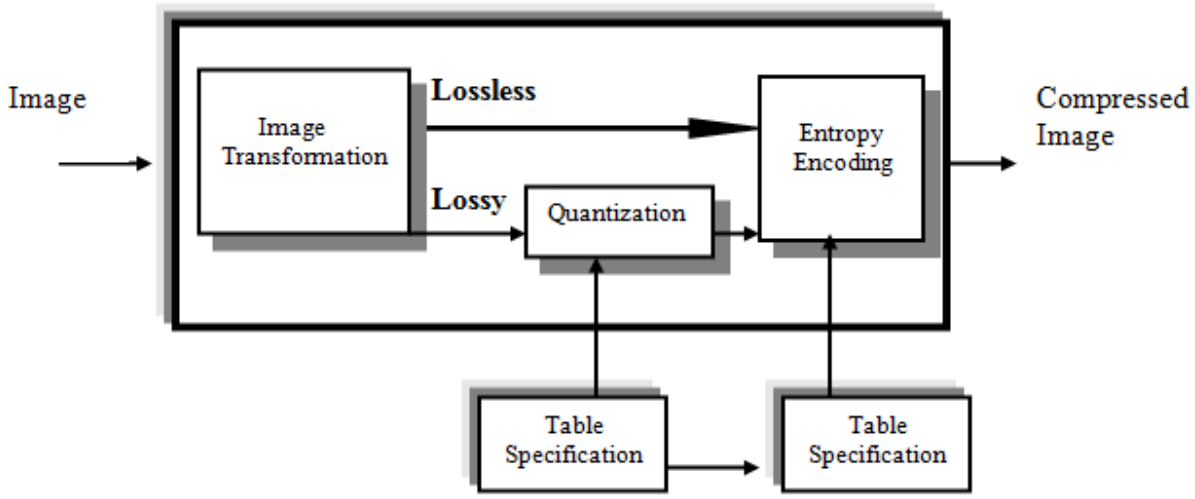


Figure 3.13: Digital Image Compression
(Adapted from Huang, 2004)

3.4 Conclusion

Image compression theory has this far been introduced in general view with emphasize on specific application (image compression system for a 3U CubeSats). Images are subjected to both lossless and lossy compression techniques, depending on the application.

In conclusion, there are two types of image compression techniques:

1. “Lossless” or reversible compression technique

   In reversible compression technique, there is no loss of information during the compression process. Lossless compression technique does not use a quantisation process, but employs image transformation and encoding in order to achieve a compressed image.

2. “Lossy” or irreversible compression technique

This compression technique is implemented using at least three steps namely: image transformation, image quantisation, and image encoding. According to Erickson (2002), image transformation is a lossless step whereby the image is transformed from grey scale values in the spatial domain to coefficients in another domain. During the transformation step process, no loss of information occurs. The quantisation step is the step in which data integrity is lost. The quantisation process may be as simple as converting floating point values into integer values. Finally, the quantized coefficients are encoded for efficient image storage or transmission.

CHAPTER 4

IMAGERY AND COMPRESSION SYSTEMS ON CUBESATS

4.1 Introduction

In this Chapter, various imagery and compression systems utilised in CubeSats are introduced. Imagery system on-board CubeSats is vital to applications if the CubeSats' system/s or subsystem/s include/s a camera as a payload system or subsystem. Current cameras on-board CubeSats are mostly high resolution which require high on-board storage capacity. High storage capacity on-board the CubeSats is the main challenge for small nano-satellites. The transmission of images to Earth requires some important aspects to take into consideration, of which the primary aspect is the size of the image data to be transmitted. The available bandwidth that links the CubeSats to Earth is limited and consequently images with a very large data size require a considerable amount of time for downlink which can not be transmitted without being compressed. The considerable amount of time taken during transmission will also require considerable power consumption.

4.2 Imagery on CubeSats

The main mission of ZACUBE-02 is to capture still images of Earth and relay them to the ground station(s). According to Rasib, Hashim, Omar and Tan (2011), satellite imagery is useful in fields such as: cadastral surveying for property management, land cover studies, detecting shoreline changes, as well as forest and wetland mapping. Using CubeSats as sensor platforms eased access to high resolution multispectral imagery data of the Earth's surface on a global basis (Wolf, 1983:495). Image sensors on CubeSats may have different resolutions depending on the CubeSat mission. For high (full) image quality, high resolution and colour depth or bit depth (bits/pixel) must be considered. The bit depth commonly used in satellite imaging is 8 bits/pixel or 12 bits/pixel (8 $bit = 1\ byte$ and 12 $bit = 1.5\ byte$).

4.2.1 CubeSats on-board Camera

Cameras are often sensors used on-board CubeSats in order to record video images or to capture still images as part of the CubeSat mission. These unprocessed images (uncompressed) are referred to as raw images (in the raw format). The camera used on-board ZACUBE-02 will be 5 mega pixels (5MP). Converting 5MP to pixels (P):

$5MP = 5x1024x1024 = 5242880P$,

To represent the entire picture as two-dimensional hxw (height multiplied by width):

$5242880 = \sqrt{5242880}x\sqrt{5242880} = 2290x2290 \leftrightarrow 5MP = 2290x2290 = hxw$.

From the total pixels, the picture file size can be determined:

$$(Number\ of\ pixels)x(Number\ of\ total\ bits/pixel). \quad (4.1)$$

This image file size unit is measured in *bit (b)*. Conversion to bytes (B) is realised by:

$$\frac{(Number\ of\ pixels)x(Number\ of\ total\ bits/pixel)}{8} \quad (4.2)$$

The still images taken by the CubeSat on-board camera are converted in picture file size as raw image files (raw formats) before they are buffered and compressed. After the buffering and compression process, the compressed images are relayed to the ground station(s) or temporarily stored while waiting for communication between the CubeSat and the ground station(s).

For example, using the 5MP camera on-board ZACUBE-02, if the bit depth is 8 bits/pixel, any picture or still image captured will portray an image file size of:

$$\frac{(5MP)x(8\ bits/pixel)}{8} = 5MB \quad (4.3)$$

And if the bit depth is 12 bits/pixel, any picture or still image captured will have an image file size of:

$$\frac{(5MP)x(12\ bits/pixel)}{8} = 7.5MB \quad (4.4)$$

Since ZACUBE-02 will be deployed in LEO at approximately 600km from Earth, it will pass over the ground station only once a day for a period of approximately 11 minutes; thus allowing a maximum of 1MB of image file size to be relayed to the ground station per day at a download rate of approximately $1589\frac{B}{s}$ ($\text{Download rate} = \frac{Image\ file\ size}{Time}$ and

$$image\ file\ size = \text{Download rate x } Time$$

$$= 1589\ B\ x\ 11\ x\ 60$$

$$= 1048740\ B$$

$$= \frac{1048740\ B}{1024\ x\ 1024}$$

$$= 1\ MB).$$

$$Image\ file\ size = 1\ MB$$

Images of file size 5MB or 7.5MB captured by ZACUBE-02 can't be transmitted to the ground station(s) in one pass of the satellite. In order to relay these images of file size deduced from equation (4.3) and (4.4), a compression ratio of 5:1 $\left(\frac{5MB}{1MB}\right)$ and 7.5:1 $\left(\frac{7.5MB}{1MB}\right)$ are required respectively. To ensure that the still images captured by ZACUBE-02 are relayed to the ground station regardless of the bit depth, a minimum compression ratio of 7.5:1 is required.

4.3 Compression Systems on CubeSats

Due to the large image file size captured by ZACUBE-02, a compression system is crucial on-board ZACUBE-02 in order to compress images to less than or equal to 1MB. Previous CubeSats missions such as AAUSAT-III (AAU CUBESAT, 2013), Astrid 2 (ISA, 2012), BeeSat-1 (ESA, 2013), CanX-1 (SFL, 2011), GOLIAT (GOLIAT, 2012), The FalconSAT Project (Kauderer, 2013), M-Cubed (M-Cubed, 2008), QuakeSat (Krebs, 2013a), and UniSat-4 (Krebs, 2013b) implemented compression systems on-board as illustrated in Table 4.1 so that images captured could be relayed to ground station(s). By minimising the image file size, various benefits such as fast image downlink, less bandwidth used for downlink, and less power consumption for downlink are obtained.

Table 4.1: Table stating the compression employed

CubeSat	Compression employed
AAUSAT-III	Advanced Image Compression (AIC)
Astrid 2	Standard lossless compression algorithm
BeeSat-1	JPEG compression algorithm
CanX-1	Zip
GOLIAT	JPEG compression algorithm
The FalconSAT Project	JPEG compression algorithm
M-Cubed	JPEG compression algorithm
QuakeSat	Bzip
UniSat-3	JPEG compression algorithm

ZACUBE-02 is considered a small nano-satellite. Given the imposed constraints (low power consumption, mass of 3 kg and volume of 30x10x10 cm^3) and its limited financing, all subsystems are integrated in one (Klofas, Anderson & Leveque, 2008). The integration of all subsystems reduces the mass, saves energy and therefore lowers the cost. One benefit of such integration is the possibility to substitute some on-board hardware with software. For example, the storage hardware could be optimised by programming one Central Processor Unit (CPU)/Digital Signal Processor (DSP).

CubeSats current and past mission subsystems are listed in Table 4.2.

Table 4.2: Current/past mission subsystems

CubeSat	Mission subsystems
AAUSAT-III	Imaging system
Astrid 2	Imaging system
BeeSat-1	Orbit verification system
CanX-1	Evaluation of several novel technologies in space system
GOLIAT	Imaging system
The FalconSAT Project	Plasma bubbles distribution measurement in the upper ionosphere system
M-Cubed	Imaging system
QuakeSat	Earth quake system
UniSat-3	Real time satellite tracking system

The lack of essential observations from space is currently a major limiting factor in space observation research (Cowing, 2012). Resent advances in sensor and satellite technologies make it feasible to obtain key measurements from low-cost, small satellite missions such as ZACUBE-02 mission (capture and relay the images of Earth). ZACUBE-02 mission requirements are:

a) Low power consumption (less or equal to 2W)
b) Mass of 3 kg and volume of 30x10x10 cm^3
c) Limited financing
d) The compressed image must be identical to the original image

These requirements lead us then to carefully study and choose/propose the appropriate compression technique and hardware necessary for image compression system on-board the CubeSat (ZACUBE-02).

Table 4.3: Comparison of lossless and lossy compression

		Compression Ratio	Compression Methods	Image quality after compression	Meet ZACUBE-02 requirements
Lossless	**GIF**	85.9%-91%	LZW coding	Very good if the image does not contain more than 256 colours. Can only store 8 bit of information per pixel	No, because it can only store 8 bits of information per pixel
	PNG	78.6%	Combination of the LZ77 as well as Huffman coding	Very good and can store more than 8 bit per pixel	No, because of poor compression ratio
	JPEG-LS	88%-95%	Modelling Prediction	Good and can store more than 8 bit per pixel	No, because of poor image quality
	JPEG2000	86%-91.4%	Discrete wavelet transform Huffman coding Arithmetic coding Run-length coding	Very good and can store more than 8 bit per pixel	Yes, because of very good image quality and can store more than 8 bits per pixel
Lossy	**JPEG**	91.4%-99.3%	Discrete cosine transform Quantization Entropy coding	Not good if the image is going to be edited and saved numerous times, because of loss in quality every time	No, because of the loss in image quality

A summary and analysis of the extensive study into lossless and lossy compression techniques from Chapter 3 is provided in Table 4.2. The table highlights the possibilities and maps this to the requirements for ZACUBE-02. From this analysis, since the JPEG2000 meets the possibilities and all the requirements for ZACUBE-02, the JPEG2000 is the chosen method to secure as a basis for the design proposed in this chapter.

4.4 Notion of Image Probabilities

Images are made of pixels; all pixels are equal in size but do not always have equal intensity density. Still image redundancy is highly predictable since intensity values of neighbouring pixels are highly correlated. In this proposed design, the focus is more on pixels size equality since this will be exploited as image redundancy. This redundancy (equality) is exploited. Since each pixel is equal in size, the probability P_i of each pixel of an image made of $NxN = N^2$ pixels is calculated such that:

$\sum_{i=1}^{N^2} P_i = 1$ (4.5)

For equation (4.5) to be true, $P_i = \frac{1}{N^2}$, since $\frac{1}{N^2} + \frac{1}{N^2} + \frac{1}{N^2} + \frac{1}{N^2} + \ldots + \frac{1}{N^2} (N^2\ times) = 1.$

Image probabilities are calculated for each pixel and since the whole image has to be compressed, it is crucial to cover all pixels. By covering all pixels, it is guaranteed that the sum of pixels' probabilities is equal to one. The pixel's probabilities are then used in order to determine the image entropy discussed in the next section.

4.5 Notion of Image Entropy

According to Shannon (1948:379-423) in information theory, the concept of entropy is defined as a measure of information. The image entropy measures the amount of information an image contains. The entropy H is defined as (Shannon, 1948:379-423):

$$H = -\sum_{i=1}^{N^2} P_i . \log(P_i) = \sum_{i=1}^{N^2} P_i . \log\left(\frac{1}{P_i}\right) \quad (4.6)$$

Since we are dealing with images (data), the entropy unit should be in bit, therefore equation (4.6) is stated as:

$$H = -\sum_{i=1}^{N^2} P_i . \log_2(P_i) = \sum_{i=1}^{N^2} P_i . \log_2\left(\frac{1}{P_i}\right) \quad (4.7)$$

For $P_i = \frac{1}{N^2}$, we deduce from equation (4.7):

$$H = \sum_{i=1}^{N^2} P_i . \log_2\left(\frac{1}{P_i}\right) = \sum_{i=1}^{N^2} \left(\frac{1}{N^2}\right) . \log_2(N^2)$$

$$\sum_{i=1}^{N^2} \left(\frac{1}{N^2}\right) . \log_2(N^2) = \left[\left(\frac{1}{N^2}\right) . \log_2(N^2) + \left(\frac{1}{N^2}\right) . \log_2(N^2) + \ldots + \left(\frac{1}{N^2}\right) . \log_2(N^2)\right] \ (N^2\ Times,$$

$since\ i\ varies\ from\ 1\ to\ N^2)$

$$= \left[\left(\frac{1}{N^2} + \frac{1}{N^2} + \frac{1}{N^2} + \frac{1}{N^2} + \ldots + \frac{1}{N^2}\ (N^2\ Times)\right) . \log_2(N^2)\right]$$

$$= [(1) . \log_2(N^2)]$$

$$= \log_2(N^2)$$

$$= \log_2(NxN)$$

Therefore, the entropy is:

$H = \log_2(NxN)$ bits. (4.8)

The entropy represents the code-length necessary for image compression. Image entropy is well used in image compression. In order to verify whether the compressed image is identical to the original image, image entropy is calculated and the bit rate of the compressed image can be compared to the entropy of the source image for compression efficiency as discussed in section 2.2 (lossless image compression techniques).

4.6 Notion of Image Bit Depth

The unit of the image file size is in bit; however an image consists of pixels (NxN) and therefore its bit depth (K) is deduced as:

$$Number\ of\ total\ bits/pixel = \frac{(Image\ file\ size)}{(Number\ of\ pixels)} \leftrightarrow K = \frac{(Image\ file\ size)}{(NxN)} \quad (4.9)$$

4.7 Proposed Design

The proposed design is addressed in terms of image information, image segmentation, image linearization, image entropy coding, and finally the proposed design flowchart.

4.7.1 Image Information

The input image is received as a string. The function (code) is written such that it returns structure containing information regarding the image stored as a file in a current or a specific directory. The output structure contains the following elements:

Image file size in bits, image height in pixels (N), image width in pixels (N), the number of total bits/pixel (K), the probability per pixel, and the code-length

The image information pseudo-algorithm is then:

(i) Get image file size (for reusability)

(ii) Get image size in pixels (NxN) (for reusability)

(iii) Get image bit depth (K)

(iv) Get the probability per pixel $\left(\frac{1}{NxN} = \frac{1}{N^2}\right)$

(v) Get the code-length ($\log_2(NxN) = \log_2(N^2)$)

4.7.2 Image Segmentation

The known image size (NxN) requires a large computational effort, is time consuming and might result in poor compression ratio if the NxN pixels size has to be encoded at once. By

segmenting NxN pixels in small sizes (KxK pixels), the computational effort, time consumption and poor compression ratio will be overcome. By segmenting NxN pixels in small segments (KxK pixels), the new image consists of $B = NxN/KxK$ blocks of KxK pixels. The function (code) for image segmentation is written such that it returns a structure containing information regarding the image.

The image segmentation pseudo-algorithm is:

(vi) Get image segment's size (KxK)

(vii) Get the number of blocks $\left(\frac{NxN}{KxK}\right)$

(viii) Get the tuple $(P_i\ , KxK) = \left(\frac{1}{NxN}\ , KxK\right)$

4.7.3 Image Linearization

The new image is structured into smaller sizes (KxK pixels). Since it is still in 2D, it is necessary to convert the 2D to 1D sequence by means of linearization. Linearization techniques have been studied in the literature review and according to Vemuri *et al*., (2007) the Peano-Hilbert scan provides the best compression (see Section 2.2.1 Coding).

The linearization technique Peano-Hilbert scan is used.

(ix) Linearization of each block using Peano-Hilbert scan technique

4.7.4 Image Entropy Coding

After the linearization stage, the 1D sequence is compressed using the corresponding coding technique. The coding techniques have been studied in the literature review and since the image is represented as a tuple ($P_i\ , KxK$) = ($P_i\ , K^2$)

$$= [Probability\ (P_i), Frequency\ of\ occurence\ (K^2)].$$

This notation is referred as dictated by the run-length coding scheme described in Table 2.1. The run-length coding scheme is then used for entropy coding in order to output the code-words. Since the sequential linearization, entropy encoding, and code-words have been done for only one block thus far, the current output is stored, then a loop of B times is implemented such that each block (all the pixels) is covered. At the end of the loop, the compressed image is the output.

(x) Compress the sequence obtained from each block with the corresponding coding technique (run-length coding)

(xi) Output and store the code-words

(xii) Go to step viii until all pixels are covered

(xiii) Output the compressed image in JPEG2000 format

4.7.5 Proposed Design Flowchart

Figure 4.1 illustrates the flowchart of the design described above as pseudo-algorithm.

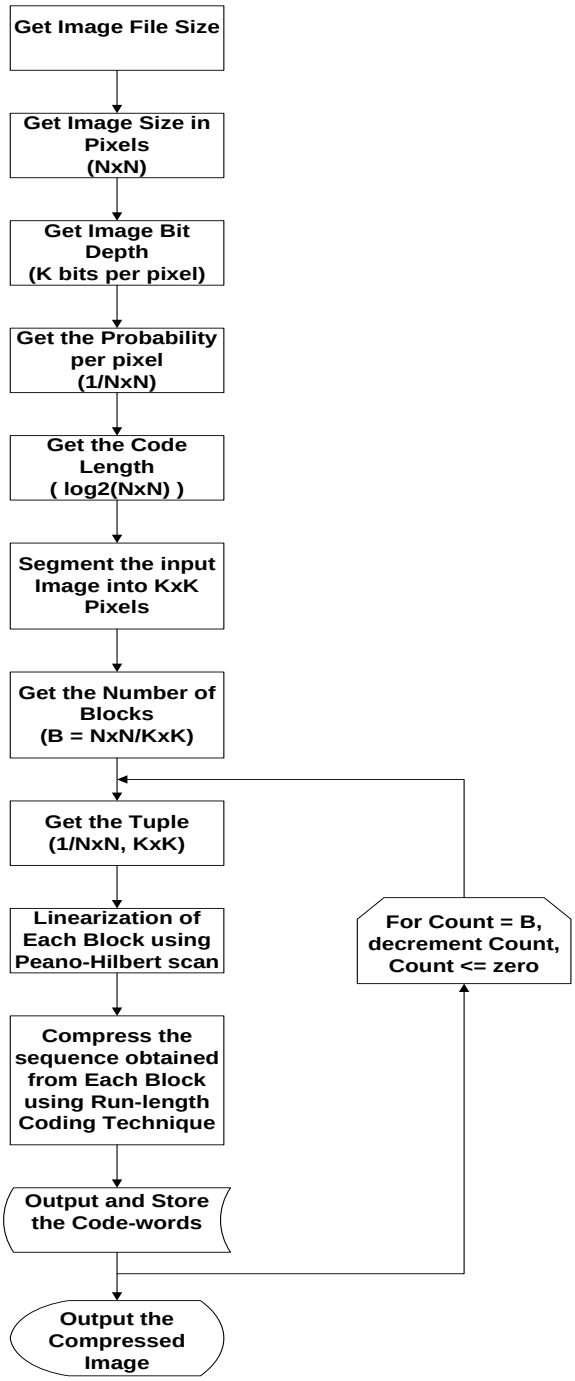


Figure 4.1: Proposed Design Flowchart

4.8 Conclusion

After an extensive study on the lossless image compression technique presented in Chapter 2 and the lossy image compression technique presented in Chapter 3, our specific application required the compressed image to be lossless, thus the design proposed here in Chapter 4 provides a suitable lossless image compression system.

The proposed design exploits a global redundancy (pixels equality) in order to target a better compression ratio. Image segmentation is used to speed up the compression time, Peano-Hilbert scan technique is used for sequence linearization, and the run-length coding for entropy encoding.

The compressed output image will be saved in JPEG2000 format.

CHAPTER 5

IMPLEMENTATION AND TESTING

5.1 Introduction

In this chapter we provide a step by step environment setup for compilation, hardware, and software development.

The appropriate hardware's CPU/DSP is tested for its functionality as well as the implementation and testing of the proposed design.

5.2 Hardware for Compression Systems on CubeSats

The image compression system on-board ZACUBE-02 requires industrial hardware that meets the environmental requirements. These include vibration tests and suitable temperature range for LEO, high computing processor for real time image processing such as the CPU or DSP, as well as the 3U CubeSat's constraints such as low power consumption, low cost (Boghosian & Valerdi, 2012), and small dimensions (less than 30x10x10 cm^3).

Table 5.1: Industrial hardware features

		Industrial Hardware			
		TILERA	**GUMSTIX**	**TRITON**	**CONGATEC**
Features	**Power**	15 – 22W @ 700MHz all cores active	Less than 1W	Less than 1W	5 W @ 5V
	Price	$435 for each Tile	$229.00	$715.00	$1088.00
	Dimensions	482.6x482.6 mm^2	17x58x4.2 mm^3	67.6x36.6x7.3 mm^3	70x70 mm^2
	Temperatures	0-70°C Commercial	Built with components rated -40°C < T < 85°C	-40°C to 85°C	Operating: 0°C to 60°C Storage: -20°C to 80°C
	Processor/s	16 to 100 identical processor cores (tiles)	CPU & DSP @ 720MHz	CPU & DSP @ 400MHz	CPU & DSP from 400MHz to 1.33 GHz

From Table 5.1 and with respect to ZACUBE-02 mission requirements set previously (see Section 4.3), the required hardware is mostly based on its physical size, its power consumption, its price, its operational or storage temperature and its processor speed.

We deduce that GUMSTIX is the best suitable industrial hardware for image compression system on-board ZACUBE-02 in terms of price (it has the lowest price), dimension (it has the smallest dimension), power consumption (it consumes less than 1 Watt which makes it one of

the two lowest power consumption hardware), temperature (temperature ranges between -40°C to 85°C), and processor speed (contains a CPU and DSP running at 720MHz).

The GUMSTIX industrial hardware for image compression will be interfaced with the camera board module.

GUMSTIX industrial hardware has a number of modules such as Gumstix Overo Air COM, Gumstix Overo Earth COM, Gumstix Overo FE COM, Gumstix Overo Fire COM, Gumstix Overo Tide COM and Gumstix Overo Water COM (Gumstix, 2012a). Gumstix Overo Water (Adapted from Gumstix, 2012a), depicted in Figure 5.1, has a CPU/DSP chip to be used for implementing image compression on-board ZACUBE-02.

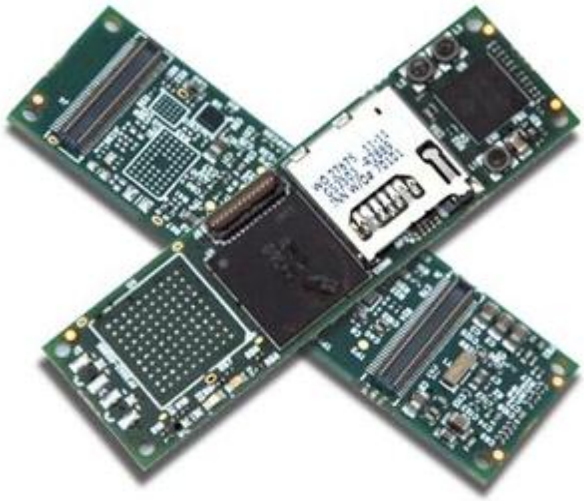


Figure 5.1: Gumstix Overo Water Industrial Hardware

High resolution still images captured by ZACUBE-02 have to be compressed on-board ZACUBE-02 using the Gumstix industrial hardware. Figure 5.2 illustrates the block diagram of ZACUBE-02 image processing system.

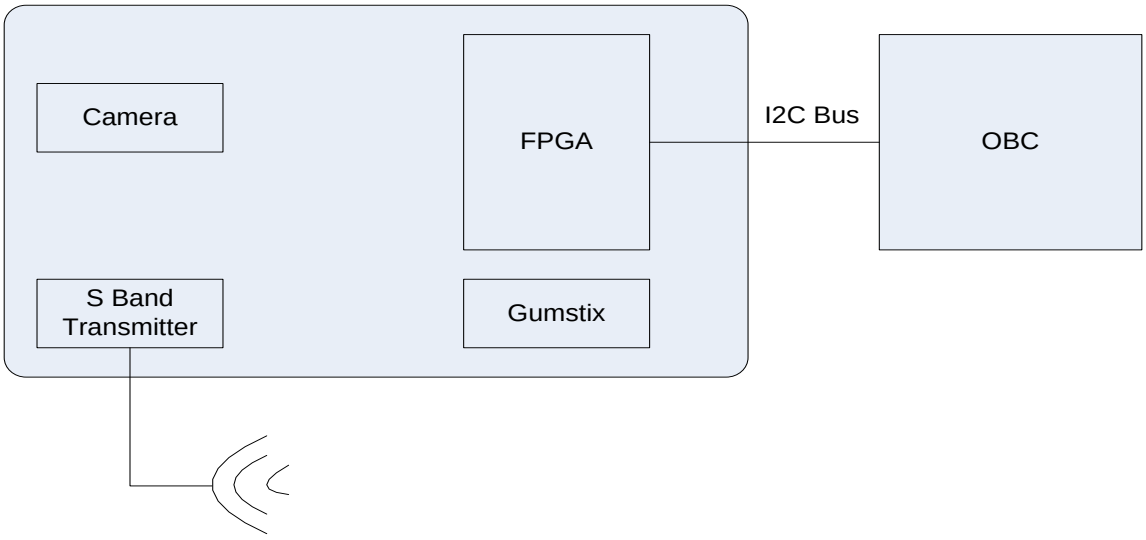


Figure 5.2: ZACUBE-02 Image Processing System Block Diagram

The image of Earth is captured by the 5 megapixels on-board camera; the image is buffered, compressed and temporarily stored by the Gumstix industrial hardware while waiting for communication between the ground station and the CubeSat. When communication occurs, the on-board computer (OBC) will instruct the field programmable gate array (FPGA) via the I2C bus, and then the FPGA will alert the S Band transmitter for image downlink.

5.3 Compilation Environment

In this research study, the proposed design implementation (coding) is done using the Linux Operating System (OS). The proposed design software is written in C programming language which exploits the Open Embedded framework for cross compilation. A detailed implementation study led to choosing Ubuntu 10.04 LTS (Long Term Support), and the Linux distribution for the Open Embedded Framework.

Ubuntu 10.04 LTS is installed on a machine with 120GB permanent storage and 1GB RAM. The packages are installed using the command line “sudo apt-get install package.” For a restricted network provider, the git protocol has to be allowed by the network administrator, as the git protocol utilises port 9418. It is necessary to ensure that the proper proxy and port are used.

Details regarding the compilation steps are provided in Appendix A.

5.4 Gumstix Hardware Compilation Environment

Gumstix's hardware is bootable via its 512MB NAND memory, or using a micro SD Card.

In this research project, Gumstix is booted from a 16GB micro SD Card. The Gumstix hardware requires a 5V DC and a connection to a PC via a USB A-to-mini cable in order to operate.
In this research project, Gumstix is running on a Tobi expansion board, and the USB A-to-mini cable is used to attach to the serial console port. The serial device is detected and recorded in "/dev" folder as "ttyUSB0" (indicated by a green LED) – as shown in Figure 5.3. The *Picocom* terminal program is used and configured as: "picocom -b 115200 /dev/ttyUSB0".
In order to avoid an error during the boot process, it is necessary to implement the changes in Listing 5.1 to the u-boot environment:

```
Overo # setenv mem mem=50M@0x80000000 mem=384M@0x88000000

Overo # setenv mmcargs setenv bootargs console=\${console} mpurate=\${mpurate} vram=\${vram} \
${mem}
             omapfb.mode=dvi:\${dvimode} omapdss.def_dsp=\${defaultdisplay} root=\${mmcroot}
             rootfstype=\${mmcrootfstype}

Overo # run mmcargs

Overo # saveenv

Overo # boot
```

Listing 5.1: u-boot environment

This ensures the availability of the modules listed in Listing 5.2.

```
Module                  Size  Used by
sdmak                   4030  0
lpm_omap3530     6895  0
dsplinkk             130309  1
lpm_omap3530
cmemk                 22192  0
rfcomm                55180  0
ipv6                 244760  10
hidp                  16651  0
bluetooth            218428  4
rfcomm,hidp
rfkill                17575  1
bluetooth
ads7846               10516  0
```

Listing 5.2: Modules

Upon start up, the default physical starting point of the kernel 2.6.34 and DSP memory for the "cmemk" module is at 86300000, which is not always compatible with all DSP applications. The most compatible physical starting point of the kernel 2.6.34 and DSP memory for DSP applications is 85000000. The script to unload the default module is shown in Listing 5.3.

```
if [ 'grep cmem /proc/
modules' ]
    then
        rmmod cmemk
fi
```

Listing 5.3: Unload script

and the script to load the module required at the right physical starting point is shown in Listing 5.4

```
insmod /lib/modules/2.6.34/kernel/drivers/dsp/cmemk.ko allowOverlap=1 phys_start=0x85000000 phys_end=0x86000000
pools=20x4096,10x131072,2x1048576
```

Listing 5.4: Load script

The module can be mounted automatically upon reboot by changing the values of the “cmemk” physical starting (phys_start) and physical ending (phys_end) point in Overo’s to 85000000 and 86000000 respectively.

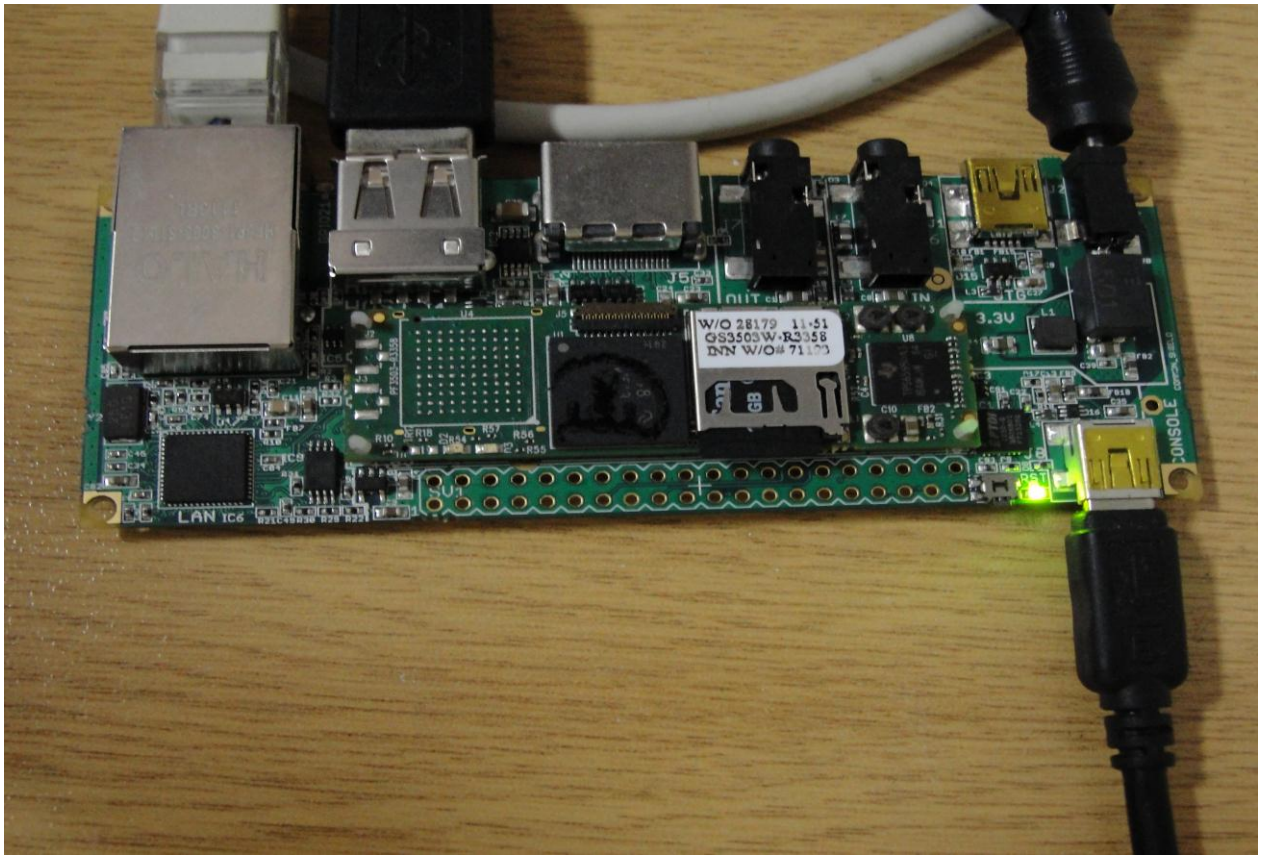


Figure 5.3: Gumstix’s serial device detected

It is necessary to erase the Gumstix's NAND memory in case a new bootable card is inserted. This is achieved through Listing 5.5 in the u-boot environment.

```
Overo# nand erase 240000 20000
Overo# reset
```

Listing 5.5: NAND memory's erase commands

Gumstix by default has Angstrom (Linux distribution) installed; the flash drive or bootable micro SD Card, the Network File System (NFS) and the USB external hard drive can be used for the software development environment. The flash drive or bootable micro SD Card is not meant for heavy Input/Output (I/O), and the NFS has occasionally had some lag/performance issues. Therefore, the USB external hard drive is the most stable for a software development environment.

5.4.1 Creating the Software Development Environment

The USB external hard drive (250GB Powered external hard drive) is used to create a full development environment by connecting the local PC, and formatting it as a single ext3 Linux partition (see Listing 5.6).

```
sudo umount /dev/sdb1
sudo mkfs.ext3 /dev/sdb1
```

Listing 5.6: Partitioning the USB external hard drive

Assuming the USB drive is mounted at "sdb1".
The following steps are necessary for creating the software development environment:

1. Attach the USB drive to a host Linux PC.
2. If it's auto-mounted to the host PC, determine the device name. Note the mount point and skip to step 5.
3. If not auto-mounted, use the "dmesg" command, and scan the last bit of output to determine the device name.
4. Find or create a mount point (e.g., "sudo mkdir /mnt/usbdisk") and mount the device.
5. Use the Linux command "cd" to locate wherever the tar archive of an existing file system for the Overo is.
6. Extract the contents to the mount point of the USB drive, e.g., "sudo tar xaf dsp-console-image. tar.bz2 –C /mnt/disk".

7. “sync” command to ensure everything’s written out to disk. “ls –l /mnt/disk” to verify that all the tarball’s contents are indeed on the disk (“bin”, “etc”, “home”, “usr”, …)
8. Change directory and un-mount the USB disk.
9. Attach the USB drive to the Overo.
10. Repeat steps 2-4 to mount the USB drive to the Overo.
11. Assuming the USB drive is attached to /mnt/disk, edit the file “/mnt/disk/etc/resolv.conf” and add the Domain Name Server/service (DNS) IP address found in the hosts PC’s /etc/resolv.conf file. Ensure the proper proxy and port for “https_proxy”, “http_proxy”, and “ftp_proxy” are set in “/mnt/disk/etc/wgetrc”. Mount the contents of the Overo’s “/proc” to the ones on the disk using the command “mount -o bind /proc /mnt/disk/proc” and “mount -o bind /dev /mnt/disk/dev”. Create an executable file “mounting-Proc-Dev.sh”, copy this executable file into the “/etc/init.d” directory. Add the executable file to startup using the command “update-rc.d mounting-Proc-Dev.sh defaults".

    Since the executable file is accessible only during the booting process, it is necessary to ensure that the disk is attached to the Overo. If attached after the booting process, the Overo has to be rebooted.
12. Switch to the new environment with “chroot /mnt/disk /bin/bash”.
13. Execute “opkg update” command to download the latest packages.
14. Secure copy all “libopencv” binary files, from the host PC to the new environment library (scp libopencv* root@Overo_IP:/mnt/disk/lib); make a new directory (IPK) at the “disk” home directory and secure copy all “.ipk” files, from the host PC into the new “IPK” directory. Use “opkg” to install all development tools available in the “IPK” directory (opkg install *.ipk).

5.4.2 Testing the CPU/DSP with existing example applications

In the Gumstix’s software development environment home directory, the folder/directory “DSPTest” is created in order to test a few DSP applications

- **DSPLink**

  For testing purposes, the files “ringiogpp” and “ringio.out” are chosen in order to test communication between the Gumstix’s General Purpose Processor (GPP) or CPU and its DSP. The files “ringiogpp” and “ringio.out” are tested from the Gumstix’s software development environment directory/folder “DSPTest” using the command line in Listing 5.7.

```
./ringiogpp ringio.out
1024 5
```

Listing 5.7: GPP test using ringio

resulting in the output in Listing 5.8

```
========== Sample Application : RING_IO ==========
Bytes to transfer :128
Data buffer size  :1024
Entered RING_IO_Create ()
Leaving RING_IO_Create ()
Entered RING_IO_ReaderClient ()
Entered RING_IO_WriterClient ()
GPP-->DSP:Sent Data Transfer Start Attribute
GPP-->DSP:Sent Data Transfer Start Notification
GPP<--DSP:Received Data TransferStart Attribute
GPP-->DSP:Total Bytes Transmitted  128
RingIO_setAttribute succeeded to set the  RINGIO_DATA_END. Status = [0x8100]
GPP-->DSP:Sent Data Transfer End Attribute
GPP-->DSP:Sent Data Transfer End Notification
Leaving RING_IO_WriterClient ()
GPP<--DSP:Received Data TransferEnd Attribute
GPP<--DSP:Bytes Received 128
GPP<--DSP:Received Data Transfer End Notification
Leaving RING_IO_ReaderClient ()
Entered RING_IO_Delete ()
Leaving RING_IO_Delete ()
=================================================
```

Listing 5.8: Ringio output

It has to be taken in consideration that after testing a DSPlink application, in order to test a new one the local power manager module (lpm_omap3530) has to power-cycle the DSP while switching between different applications; this is done by conducting a secure copy of the files “lpmOFF.xv5T” and “lpmON.xv5T” from “$HOME/overo-oe/tmp/sysroots/overo-angstrom-linux-gnueabi/usr/share/ti/ti-lpm-utils” to the “usr/bin” directory of the Gumstix’s software development environment. Implement the commands to switch between different applications shown in Listing 5.9.

```
lpmOFF.xv5T

lpmON.xv5T
```

Listing 5.9: Switch commands

❖ **Codec Engine**

Before testing the codec engine application, it is vital to set the kernel 2.6.34 and the DSP memory starting point for the “cmemk” module appropriately by unloading the default module (see Listing 5.10).

```
if [ 'grep cmem /proc/
modules' ]
    then
        rmmod cmemk
fi
```

Listing 5.10: unloading the default module

and loading the module required at the right physical starting point (see Listing 5.11)

```
insmod /lib/modules/2.6.34/kernel/drivers/dsp/cmemk.ko allowOverlap=1 phys_start=0x85000000 phys_end=0x86000000
pools=20x4096,10x131072,2x1048576
```

Listing 5.11: loading the module at the right physical starting point

For testing purposes, the application “image_copy” is chosen in order to test the Gumstix’s DSP. The folder/directory “image_copy” is copied securely to the Gumstix’s software development environment directory/folder “DSPTest”. Execute Listing 5.12 within the “image-copy” directory.

```
CE_DEBUG=2 ./app_remote.xv5T
```

Listing 5.12: DSP test

This result in the output displayed in Listing 5.13.

```
…
…
…
[DSP] @2,820,525tk: [+0 T:0x8783f28c] ti.sdo.ce.image.IMGDEC - IMGDEC_process> Exit
(handle=0x8783ee28, retVal=0x0)
[DSP] @2,820,580tk: [+0 T:0x8783f28c] OM - Memory_cacheWb> Enter(addr=0x8553e000,
sizeInBytes=1024)
[DSP] @2,820,636tk: [+0 T:0x8783f28c] OM - Memory_cacheWb> return
[DSP] @2,820,669tk: [+5 T:0x8783f28c] CN - NODE> returned from call(algHandle=0x8783ee28,
msg=0x87e04880); messageId=0x000209ff
@2,727,416us: [+0 T:0x4001df90] CE - Engine_fwriteTrace> returning count [1935]
@2,727,478us: [+0 T:0x4001df90] CV - VISA_call Completed: messageId=0x000209ff, command=0x0,
return(status=0)
…
…
...
@2,761,688us: [+2 T:0x4097c490] OP - Processor_delete_d> Stopping DSP...
@2,761,779us: [+2 T:0x4097c490] OP - Processor_delete_d> Closing pool...
@2,761,962us: [+2 T:0x4097c490] OP - Processor_delete_d> Detaching from DSP...
@2,770,324us: [+0 T:0x4001df90] ti.sdo.ce.osal.Sem - Leaving Sem_post> sem[0x641c0]
@2,770,446us: [+0 T:0x4001df90] ti.sdo.ce.osal.Sem - Entered Sem_pend> sem[0x641d8]
timeout[0xffffffff]
@2,770,538us: [+2 T:0x4097c490] OP - Processor_delete_d> Destroying DSP... (object, that is)
@2,770,935us: [+0 T:0x4097c490] ti.sdo.ce.ipc.Power - Power_off> Enter (handle=0x648a8)
@2,770,996us: [+2 T:0x4097c490] ti.sdo.ce.ipc.Power - Power_off> Turning off DSP power...
@2,771,179us: [+2 T:0x4097c490] ti.sdo.ce.ipc.Power - Power_off> Closing Local Power Manager
object...
@2,771,240us: [+0 T:0x4097c490] ti.sdo.ce.ipc.Power - Power_off> return (0)
@2,771,301us: [+0 T:0x4097c490] OP - Processor_delete_d> return
@2,771,331us: [+0 T:0x4097c490] ti.sdo.ce.osal.Sem - Entered Sem_post> sem[0x641d8]
@2,771,423us: [+0 T:0x4001df90] ti.sdo.ce.osal.Sem - Leaving Sem_pend> sem[0x641d8] status[0]
@2,771,484us: [+0 T:0x4001df90] OP - doCmd> Exit (result=1)
@2,771,545us: [+1 T:0x4001df90] OP - Processor_delete(0x64880) freeing object ...
@2,771,575us: [+0 T:0x4001df90] OP - Processor_delete> return.
@2,771,636us: [+0 T:0x4001df90] CE - rserverClose(0x62f38) done.
@2,771,667us: [+0 T:0x4001df90] CE - Engine_close exit
@2,772,644us: [+1 T:0x4001df90] ti.sdo.ce.examples.apps.image_copy - app done.
```

Listing 5.13: DSP test output

- **Dmai**

The file “image_encode_io1_omap3530.x470MV” is chosen in order to test the Gumstix’s DSP. This file has to be tested while being in the same directory as the codec server file “cs.x64P”.

Both files (“image_encode_io1_omap3530.x470MV” and “cs.x64P”) are executed by Listing 5.14, with the output provided in Listing 5.15.

```
./image_encode_io1_omap3530.x470MV -c jpegenc -e encode -i jpeg_test_output.yuv -o Output.jpeg -r 720x576 --
iColorSpace 3 --oColorSpace 1
```

Listing 5.14: Execution command

```
…
…
…
[DSP] @0,063,466tk: [+0 T:0x87f0d334] OM - Memory_alloc> Enter(size=0x18)
[DSP] @0,063,521tk: [+0 T:0x87f0d334] OM - Memory_alloc> return (0x87f11e08)
[DSP] @0,063,565tk: [+0 T:0x87f0d334] OM - Memory_alloc> Enter(size=0xa)
[DSP] @0,063,602tk: [+0 T:0x87f0d334] OM - Memory_alloc> return (0x87f11e20)
[DSP] @0,063,649tk: [+0 T:0x87f0d334] OM - Memory_alloc> Enter(size=0x20)
[DSP] @0,063,688tk: [+0 T:0x87f0d334] OM - Memory_alloc> return (0x87f11e30)
[DSP] @0,063,731tk: [+0 T:0x87f0d334] OM - Memory_alloc> Enter(size=0x24)
[DSP] @0,063,769tk: [+0 T:0x87f0d334] OM - Memory_alloc> return (0x87f11e50)
[DSP] @0,063,838tk: [+0 T:0x87f0d334] ti.sdo.ce.image1.IMGENC1 - IMGENC1_create> Enter
…
…
...
@0,906,646us: [+1 T:0x4001f070] OP - Processor_delete(0x60820) freeing object ...
@0,906,707us: [+0 T:0x4001f070] OP - Processor_delete> return.
@0,906,768us: [+0 T:0x4001f070] CE - rserverClose(0x5f058) done.
@0,906,799us: [+0 T:0x4001f070] CE - Engine_close exit
End of application.
```

Listing 5.15: Execution command output

5.5 Proposed Design Implementation

It is important to implement the proposed design software, cross-compile it on the host PC, and secure copy the executable binary file to the Gumstix code development environment for integration and testing.

The Gumstix hardware used in this research project utilises the Open Media applications Platform 3530 (OMAP3530) processor which requires a specific code. The specific code in this case is an executable binary file generated by cross-compilation of the proposed design script written in C. The cross-compiler consists of several components such as the toolchain, the kernel and various packages necessary to build the binary code to be integrated and tested on the Gumstix code development environment. This is then ported to the Gumstix's hardware for image compression on-board the 3U CubeSat (ZACUBE-02).

The various components (directories) necessary for implementing the proposed design in this research project include:

1. Proposed Design Scripts: (ProDesScr)
2. Codec Engine: (CE)
3. CMEM: (CMEM)
4. Framework Components: (FC)
5. Local Power Manager: (LPM)
6. XDAIS Algorithm Standard: (XDAIS)
7. multimedia codecs: (CODEC)
8. DSP Link: (LINK)
9. Linux libraries: (LINUXLIBS)
10. Linux kernel: (LINUXKERNEL)
11. GCC cross compilation tool chain: (CROSS)
12. RTSC tools: (XDC)
13. Accelerator tools: (C6ACCEL)
14. EXEC: the DMAI applications should be installed when 'make install' is executed
    make at /dmai, make at /ProDesScr, and make install at / ProDesScr. The executable file or binary code is available at /EXEC.

The proposed design specifies four steps/elements: the information regarding the image, the segmentation of the image in blocks, the linearization of the image (Peano-Hilbert scan) followed by entropy coding (run-length coding). Figure 5.4 provides a better understanding of these four steps/elements.

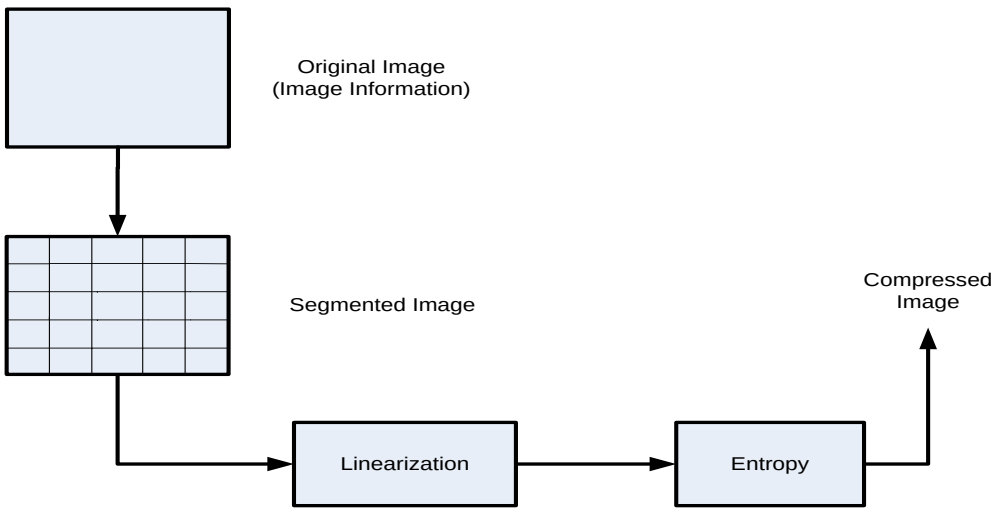


Figure 5.4: Steps involved in the proposed design

Figure 5.8 details the entropy coding process.

1. Original Image (Image Information)

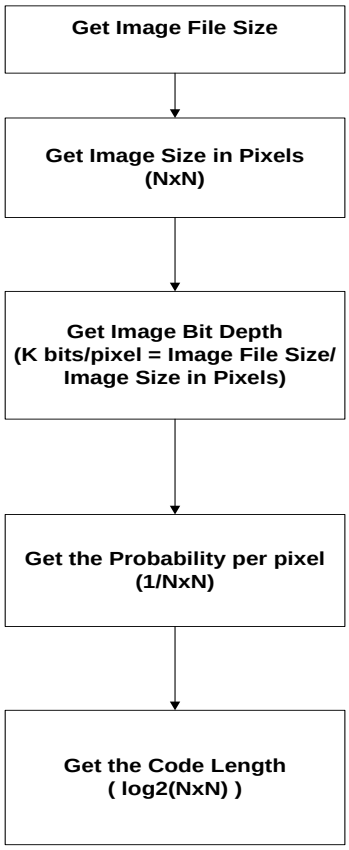


Figure 5.5: Original Image (Image Information) code flowchart

2. Image Segmentation

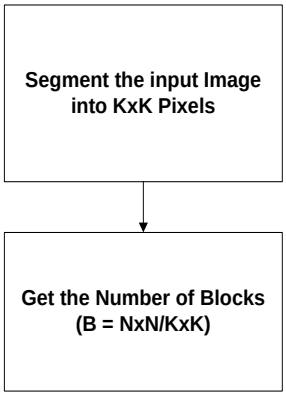


Figure 5.6: Image Segmentation code flowchart

3. Image Linearization (Peano-Hilbert scan)

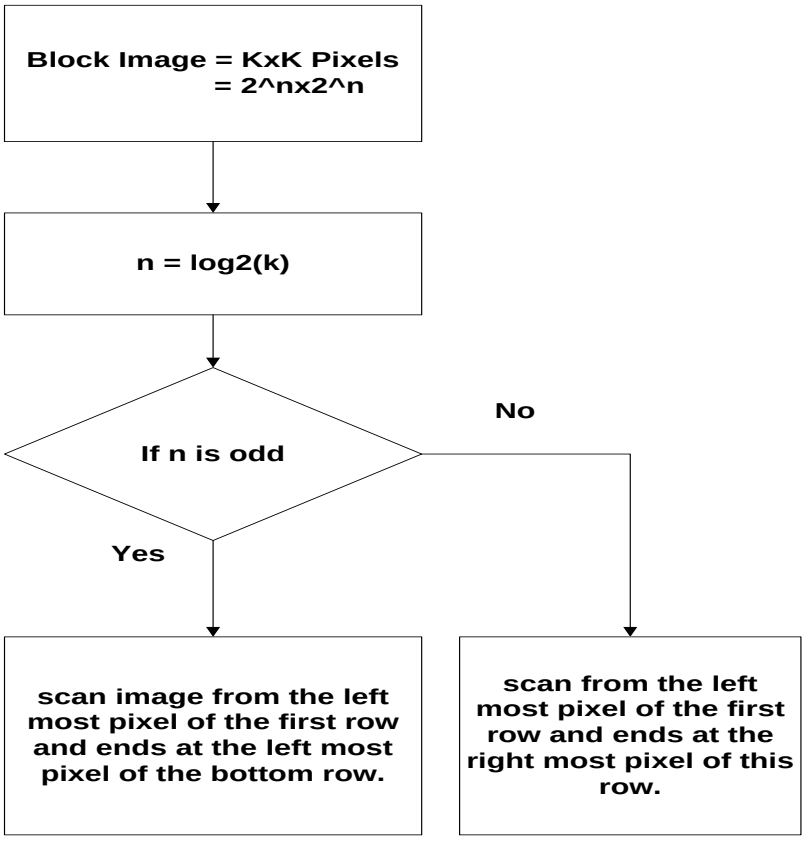


Figure 5.7: Image Linearization (Peano-Hilbert scan) code flowchart

4. Entropy coding (Run-length coding)

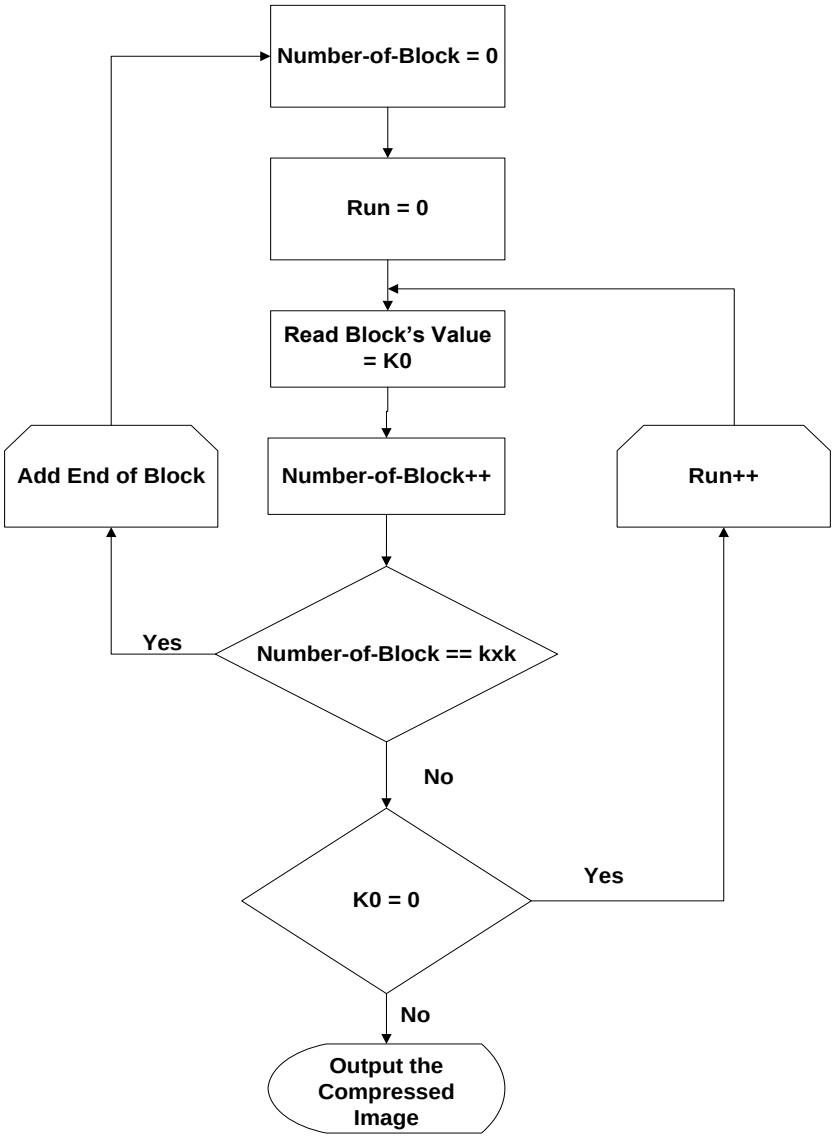


Figure 5.8: Entropy coding (Run-length coding) flowchart

The complete code listing is provided in Appendix B.

The executable file or binary code available at /EXEC, is copied to the Gumstix code development environment using the secure copy command (scp). After successful integration and testing of the binary code, it is ported to the Gumstix's hardware for image compression system on-board the 3U CubeSat (ZACUBE-02).

5.6 Conclusion

The implementation and testing of the proposed design is achieved through a proper study and choice of an appropriate hardware which in this study is the Gumstix hardware known as the world smallest super computer. Also the necessity of the compilation environment as well as creating the software development environment has been taken in consideration.

Before attempting any implementation and testing on hardware, the programming environment and all the hardware's components/elements necessary to achieve the targeted goal have to be in place.

The complete code generated within the implementation of the proposed design for compression of images is provided in Appendix B.

CHAPTER 6

EVALUATION OF THE PROPOSED DESIGN

6.1 Introduction

To ensure the design and subsequent implementation adhere to all set requirements, a thorough evaluation has to be conducted. The complete system is evaluated in terms of compression technicalities, performance and comparison with current systems as well as adherence to requirements. As ZACUBE-02 is only set to be launched in 2014, the system cannot be evaluated within a real satellite environment.

6.2 Image Compression ratio

To evaluate the compression ratio, (i) a Google satellite image of a street map of 720896 pixels (8 bits/pixel), (ii) an image taken by a 10 megapixels (12 bits/pixel) CANON digital camera, (iii) an Earth picture taken by a Masat-1 CubeSat (100x75 pixels, 8 bits/pixel), and (iv) the image of Earth (MIMAS) taken by the National Aeronautics and Space Administration (NASA)/ Jet Propulsion Laboratory (JPL)/Space Science Institute (500x500 pixels, 8 bits/pixel), are selected. The four images were successfully compressed at the compression ratio of 10:1.

Figure 6.1 and Figure 6.2 illustrate the original Google satellite image of a street map and the compressed Google satellite image of a street map respectively.

Figure 6.3 and Figure 6.4 illustrate the original image taken by a digital camera and the compressed image taken by a digital camera respectively.

Figure 6.5 and Figure 6.6 illustrate the original Earth picture taken by a Masat-1 CubeSat and the compressed Earth picture taken by a Masat-1 CubeSat respectively.

Figure 6.7 and Figure 6.8 illustrate the original image of Earth (MIMAS) taken by NASA/JPL/Space Science Institute and the compressed image of Earth (MIMAS) taken by NASA/JPL/Space Science Institute respectively.

Figure 6.1: Original Google satellite image of a street map

Figure 6.2: Compressed Google satellite image of a street map

Figure 6.3: Original image taken by a digital camera

Figure 6.4: Compressed image taken by a digital camera

Figure 6.5: Original Earth picture taken by a Masat-1 CubeSat

Figure 6.6: Compressed Earth picture taken by a Masat-1 CubeSat

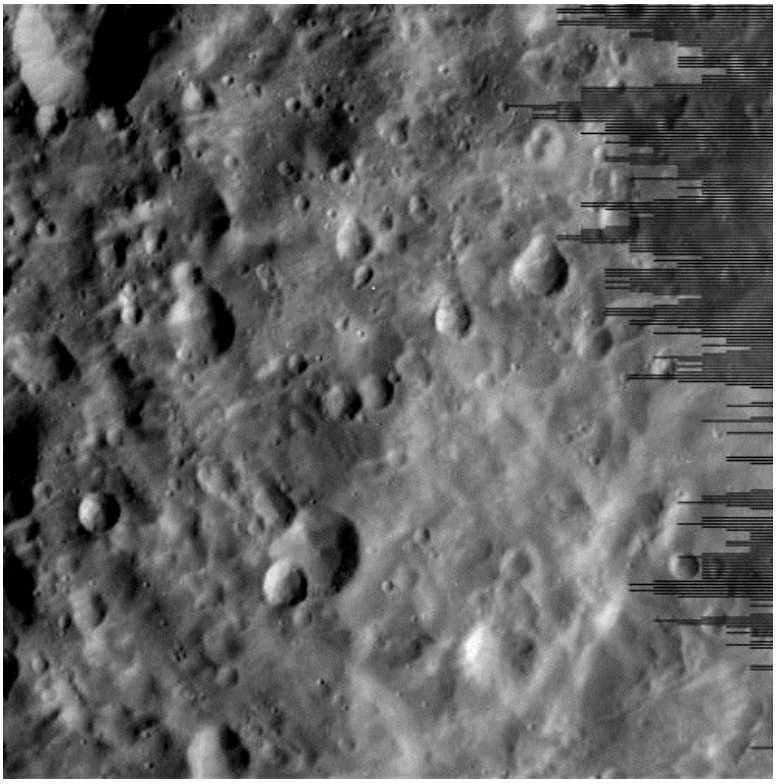


Figure 6.7: Original image of Earth (MIMAS) taken by NASA/JPL/Space Science Institute

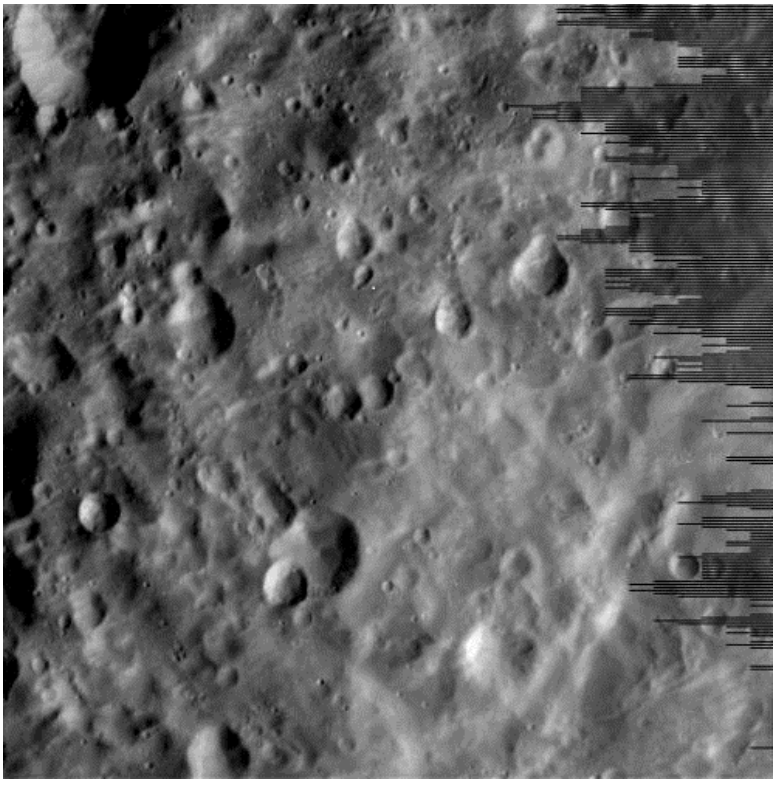


Figure 6.8: Compressed image of Earth (MIMAS) taken by NASA/JPL/Space Science Institute

From Equation (4.3) and (4.4), the compression ratio of 5:1 (5MB/1MB) and 7.5:1 (7.5MB/1MB) are needed respectively for an on-board camera of bit depth of 8 bits/pixel and 12 bits/pixel. To ensure that the still image captured by ZACUBE-02 is relayed to the ground station regardless of bit depth, a minimum compression ratio of 7.5:1 is sufficient. By achieving a compression ratio of 10:1 within the four images, it can be safely concluded that any image taken by the 5MP camera on-board ZACUBE-02 will be successfully compressed to less than 1MB.

An image file of less than 1MB, means the downlink time, bandwidth, and power consumption are optimised.

Table 6.1 provides the evaluation table of the proposed design in comparison with current compression system. The proposed design meets ZACUBE-02 requirements in terms of compression ratio, image quality after compression, and the bit depth storage.

Table 6.1: Evaluation table

		Compression Ratio	**Compression Methods**	**Image quality after compression**	**Meet ZACUBE-02 requirements**
Lossless	**GIF**	85.9%-91%	LZW coding	Very good if the image does not contain more than 256 colours. Can only store 8 bit of information per pixel	No, it can only store 8 bits of information per pixel
	PNG	78.6%	Combination of the LZ77 as well as Huffman coding	Very good and can store more than 8 bit per pixel	No, because of its poor compression ratio
	JPEG-LS	88%-95%	Modelling Prediction	Good and can store more than 8 bit per pixel	No, because of poor image quality
	JPEG2000	86%-91.4%	Discrete wavelet transform Huffman coding Arithmetic coding Run-length coding	Very good and can store more than 8 bit per pixel	Yes, because of very good image quality and can store more than 8 bits per pixel
Lossy	**JPEG**	91.4%-99.3%	Discrete cosine transform Quantization Entropy coding	Not good if the image is going to be edited and saved numerous times, because of loss in quality every time	No, because of loss in image quality
Proposed Design	**Lossless**	10:1	Segmentation Linearization Run-length coding	Very good and can store more than 8 bit per pixel	Yes, because of its good compression ratio, its very good image quality and its ability to store more than 8bit/pixel

6.3 Conclusion

Four images of bit depth 8 bits/pixel (taken by Google satellite), 12 bits/pixel (taken by a CANON digital camera), 8 bits/pixel (taken by a Masat-1 CubeSat), and 8 bits/pixel (taken by NASA/JPL/Space Science Institute) have been used for evaluation. The result has been very positive since the compression ratio obtained (10:1) is better than the minimum compression ratio of 7.5:1 targeted. A high compression ratio of 10:1 as compare to the ones illustrated in the literature review is mainly due to the appropriate hardware and the proposed design.

Since the compression technique provided is lossless, the expected compressed images are identical; therefore one can't see any difference. If the proposed compression technique were lossy, the best way of evaluating the result of the proposed design would have been to show the difference between the two images (compressed and uncompressed).

CHAPTER 7
CONCLUSION

7.1 Addressing the Research Problem

The research study was conducted in a small nano-satellite environment with the main objective of providing an image compression system for a 3U CubeSat. The 3U CubeSat (ZACUBE-02) is equipped with a 5 MP, high resolution on-board camera. The on-board camera will be used to capture images of Earth and relay them to the ground station. Images captured at 5 MP can't be relayed to the ground station in one pass, hence the necessity of an image compression system.

Image compression was studied then classified as lossless image compression and lossy image compression from which lossless image compression was implemented in order to meet the ZACUBE-02 requirements.

The lossless image compression was successfully implemented through the knowledge of image probabilities, image entropy, image bit depth, image segmentation, image linearization, and image entropy coding.

7.2 Achieving the Project Objectives

The main objective as detailed in Chapter 1 is to design and implement an image compression system for ZACUBE-02.

This objective was achieved by providing an image compression system for a 3U CubeSat able to compress lossless image up to a compression ratio of 10:1. This compression ratio of 10:1 is met through:

- A detailed study, while comparison of image compression techniques were conducted in Chapter 2 and Chapter 3.
- Detailed studies of techniques for improving the compression ratio while retaining image quality, were completed in Chapter 2.
- Thorough studies of imagery and compression systems especially catered for the CubeSat environment, were completed in Chapter 4.
- The design criteria and requirements were put forward in Chapter 4.
- A study and choice of industrial hardware components to meet environmental requirements, design requirements and application requirements, were conducted in Chapter 5.
- The evaluation of the proposed design through implementation and testing were completed in Chapter 6.

This research project provided various opportunities and exposures to a wide field of image processing as well as the knowledge about embedded system and different hardware suitable for on-board small Nano-satellites.

With increase in demand for high image resolution from space exploration, the camera specifications (number of pixels) on-board future small Nano-satellites will increase. This will then need some improvement on image compression ratio which will be part of our future research. Achieving a compression ratio of 10:1 on-board ZACUBE-02 which has a high resolution camera of 5 MP is a good starting point for South African CubeSats on-board compression. This compression system will then be used for future South African CubeSats family.

7.3 Challenges

As ZACUBE-02 is only set to be launched in 2014, the system could not be evaluated within a real satellite environment. Since the hardware used to test the proposed compression system runs Linux operating system, the biggest challenge was to have the appropriate kernel as well as the matching Linux version installed on the workstation and the testing hardware.

Most Satellite or CubeSat applications used for space exploration or Earth imaging are commercialised, therefore it was difficult to access images from satellite's companies.

7.4 Recommendations

The proposed design can be implemented on any hardware that has a CPU/DSP processor, but for CubeSat applications the chosen hardware has to meet the CubeSat's restrictions. Beside the CubeSat applications, this proposed design is suitable for medical applications (amongst others) in the field of biomedical, especially in medical imaging, commercial photography and industrial imaging.

8. REFERENCES

AAU CUBESAT. AAUSAT3. 2013. http://www.space.aau.dk/cubesat/ [03 April 2013].

Abramson, N. 1963. *Information theory and coding*. New York: McGraw-Hill: 61-62.

Acharya, T. & Tsai, P. 2005. *JPEG2000 standard for image compression: concepts, algorithm and VLSI architectures*. New Jersey: John Wiley & Sons: 149-151.

Akhtar, M.B., Qureshi, A.M. & Qamar-ul-Islam. 2011. Optimized run length coding for JPEG image compression used in space research project of IST. *In Proceedings of the IEEE International Conference on Computer Networks and Information Technology (ICCNIT-2011)*, Abbottabad, 11-13 July 2011, 81-85.

Al-Mualla, M.E., Canagarajah, C.N. & Bull, D.R. 2002. *Video coding for mobile communications: efficiency, complexity, and resilience*. California: Academic Press.

Antonini, M., Barlaud, M., Mathieu, P. & Daubechies, I. 1992. Image coding using wavelet transform. *IEEE Transactions on Image Processing*, 2(1):205-220.

Barnsley, M. & Hurd, L.P. 1993. *Fractal image compression*. Massachusetts: Peters, A.K., Wellesley.

Barthel, K.U. & Voyé, T. 1994. Adaptive fractal image coding in the frequency domain. *Proceedings of the International Workshop on Image Processing: theory, methodology, systems and applications*, Budapest, 20-22 June 1994, 45:33-38.

Bassiouni, M.A., Tzannes, A.P., Tzannes, M.C. & Tzannes, N.S. 1991. Image compression using integrated lossless/lossy methods. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'91)*, Toronto, 14-17 April 1991, 2817-2820.

Bayat, F., Johansen, T.A. & Jalali, A.A. 2011. Combining truncated binary search tree and direct search for flexible piecewise function evaluation for explicit MPC in embedded microcontrollers. *Proceedings of the Eighteenth International Federation of Automatic Control (IFAC-2011) World Congress*, Milano, 28 August-02 September 2011.

Bertelli, L. 2009. Variational image segmentation based on pixel pairwise similarities. University of California, Santa Barbara.

Bhaskaran, V. & Konstantinides, K. 1995. *Image and video compression Standards: algorithms and architectures*. Dordrecht: Kluwer Academic Publishers, 47-49.

Boghosian, M. and Valerdi, R. 2012. Cost estimating methodology for very small satellites. *First Interplanetary CubeSat Workshop*. Cambridge, Massachusetts, 29-30 May 2012.

Calderbank, A.R., Daubechies, I., Sweldens, W. and Yeo, B. 1998. Wavelet transforms that map integers to integers. *Applied and Computational Harmonic Analysis*, 5(3):332-369.

Carpentieri, B., Weinberger, M.J. & Seroussi, G. 2000. Lossless compression of continuous-tone images. *Proceedings of the IEEE*, 88(11):1797-1809.

Chartier, C., Mackay, M., Ravalico, D., Russell, S. & Wallis, A. 2010. Design, build and launch of a small satellite based on CubeSat standards. Honours project, University of Adelaide, Adelaide.

Cho, Y. & Pearlman, W.A. 2007. Hierarchical dynamic range coding of wavelet subbands for fast and efficient image decompression. *IEEE Transactions on Image Processing*, 16(8):2005-2015.

Christopoulos, C., Skodras, A. & Ebrahimi, T. 2000. The JPEG2000 still image coding system: an overview. *IEEE Transactions on Consumer Electronics*, 46(4):1103-1127.

Clyde Space. 2012. CubeSat EPS and battery NASA GEVS vibration test. http://www.clyde-space.com/about_us/videos/cubesat_nasa_gevs_vibration_test [10 November 2012].

Cosman, P.C., Gray, R.M. & Vetterli, M. 1996. Vector quantization of image sub-bands: A survey. *IEEE Transactions on Processing*, 5(2):202-225.

Cowing, K. 2012. CubeSat-based science missions for geospace and atmospheric research, NSF. http://nasahackspace.com/nsf/ [20 July 2012].

Davoine, F., Antonini, M., Chassery, J.M. & Barlaud, M. 1996. Fractal image compression based on Delaunay triangulation and vector quantization. *IEEE Transactions on Image Processing*, 5(2):338-346.

Davoine, F., Bertin, E. & Chassery, J.M. 1993. From rigidity to adaptive tessellations for fractal image compression: comparative studies. *IEEE Eight Workshop on Image and Multidimensional Signal Processing*, Cannes, September 1993, 56-57.

Davoine, F. & Chassery, J.M. 1994. Adaptive Delaunay Triangulation for Attractor Image Coding. *Proceedings of the twelfth IAPR International Conference on Computer Vision and Image Processing*, Jerusalem, 09-13 October 1994, 1:801-803.

Deshpande, N. & Sane, S.S. 2007. *Data structures and files*. First Edition. Pune: Technical Publications Pune.

Dewitte, S. & Cornelis, J. 1996. Lossless integer wavelet transform. Technical Report IRIS-TR-0041, Royal Meteorological Institute Belguim.

Dia, D., Zeghid, M., Saidani, T., Atri, M., Bouallegue, B., Machhout, M. & Tourki, R. 2009. Multi-level discrete wavelet transform architecture design. *Proceedings of the World Congress on Engineering*, 1-3 July 2009. London: 1:1-5.

Dougherty, G. 2009. *Digital image processing for medical applications*. New York: Cambridge University Press.

Du, K. & Swamy, M.N.S. 2010. *Wireless communication systems: from RF subsystems to 4G enabling technologies*. New York: Cambridge University Press: 723.

Ebrahimpour-Komleh, H., Chandran, V. & Sridharan, S. 2004. Fractal image-set encoding for face recognition. *Proceedings of the International Conference on Computational Intelligence for Modelling Control and Automation (CIMCA-2004)*, Gold Coast, 12-14 July 2004, 664-672.

Elias, P. 1975. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, 21(2):194-203.

Ellis, S. 2012. Streaming video with Gumstix, GStreamer and the DSP. http://www.jumpnowtek.com/index.php?option=com_content&view=article&id=81:gumstix-dsp-gstreamer&catid=35:gumstix&Itemid=67 [28 September 2011].

Erickson, B.J. 2002. Irreversible compression of medical images. *Journal of Digital Imaging*, 15(1):5-14.

ESA. 2013. Berlin Experimental Educational Satellite-1. https://directory.eoportal.org/web/eoportal/satellite-missions/b/beesat-1 [03 April 2013].

Ferreira, P.J.S.G. & Pinho, A.J. 2002. Why does histogram packing improve lossless compression rates? *IEEE Transactions on Signal Processing*, 9(8):259-261.

Fischer, T.R. 1986. A pyramid vector quantizer. *IEEE Transactions on Information Theory*, 32(4):568-583.

Fisher, Y. 1992. A discussion of fractal image compression. In Peitgen, H.O., Jurgens, H. & Saupe, D. (eds). *Chaos and fractals: new frontiers of science*. New York: Springer-Verlag, 903-919.

Fisher, Y. (ed). 1995. *Fractal image compression: theory and application*. New York: Springer-Verlag.

Fisher, Y., Jacobs, E.W. & Boss, R.D. 1992. Fractal image compression using iterated transforms. In Storer, J.A (ed). *Image and text compression*. Massachusetts: Kluwer Academic Publishers, 35-61.

Fu, D., Fan, Y. & Wang, H. 2011. The research and application of electronic signature system based on improved Gif-Shuffle algorithm. *Proceedings of the International Conference on Electrical and Control Engineering (ICECE-2011)*, Yichang, 16-18 September 2011, 2747-2750.

F'SATI. 2011. CubeSat applications towards sustainable socio-economic development. *First International African CubeSat Workshop*, Cape Town, 30 September-02 October 2011.

Gallager, R.G. & Van Voorhis, D. C. 1975. Optimal source codes for geometrically distributed integer alphabets. *IEEE Transactions on Information Theory*, 21(2):228-230.

Gersho, A. 1982. On the structure of vector quantizers. *IEEE Transactions on Information Theory*, 28(2):157-166.

Gersho, A. & Gray, R.M. 1992. *Vector quantization and signal compression.* Massachusetts: Kluwer Academic Publishers.

Gholipour, M. 2011. Design and implementation of lifting based integer wavelet transform for image compression applications. *Digital Information and Communication Technology and Its Applications Part I*, June 2011. Dijon: Springer: 161-172.

GOLIAT. 2012. The first Romanian nanosatellite. http://www.goliat.ro/ [03 April 2013].

Golomb, S.W. 1966. Run-length encodings. *IEEE Transactions on Information Theory*, 12(3):399-401.

Gonzalez, R.C. & Woods, R.E. 2004. *Digital image processing*. Second Edition. New Jersey: Prentice Hall.

Goyal, S. & O'Neal, J. 1975. Entropy coded differential pulse-code modulation systems for television. *IEEE Transactions on Communications*, 23(6):660-666.

Gumstix. 2012a. Overo COMS. http://www.gumstix.com/store/index.php?cPath=33 [25 August 2011].

Gumstix. 2012b. Using the Open Embedded Build System for Overo Series. http://gumstix.org/software-development/open-embedded/61-using-the-open-embedded-build-system.html [20 July 2011].

Gumstix. 2012c. Create a Bootable MicroSD Card. http://www.gumstix.org/create-a-bootable-microsd-card.html [28 August 2011].

Habibi, A. 1971. Comparison of n^{th}-order DPCM encoder with linear transformations and block quantisation techniques. *IEEE Transactions on Communication Technology*, 19(6):948-956.

Heer, V.K. & Reinfelder, H.E. 1990. A comparison of reversible methods for data compression. *SPIE-Med Imaging IV*, 1233:354-365.

Heidt, H., Puig-Suari, J., Moore, A., Nakasuka, S. & Twiggs, R. 2001. CubeSat: A new generation of picosatellite for education and industry low-cost space experimentation. *Proceedings of the fourteenth AIAA/USU Small Satellite Conference*. SSC00-V-5. August 2001.

Horowitz, E., Sahni, S. & Mehta, D. 2006. *Fundamentals of data structures in C++*. New Jersey: Silicon Press.

Howard, P.G. & Vitter, J.S. 1994. Arithmetic coding for data compression. *IEEE Transactions on Information Processing and Management*, 82(6):857-865.

Huang, H.K. 2004. *PACS and imaging informatics: basic principles and applications*. New Jersey: John Wiley & Sons. 119-152.

Huang, S. & Zheng, T. 2008. Hardware design for accelerating PNG decode. *Proceedings of the IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC-2008)*, Hong Kong, 8-10 December 2008, 1-4.

Huffman, D.A 1952. A method for the construction of minimum redundancy codes. *Proceedings of the Institute of Radio Engineers*, 40(9):1098-1101.

ISA. 2012. ASTRID2-The Ultimate Synchrotron Radiation Source. http://www.isa.au.dk/facilities/astrid2/astrid2.asp [03 April 2013].

ITU-T ISO/IEC. 2004. Information technology – JPEG 2000 image coding system: core coding system. ITU-T recommendation T.800 and ISO/IEC international standard 15444-1:2004. http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=37674 [28 May 2013]

ITU-T ISO/IEC. 2011. Information technology - Lossless and near-lossless compression of continuous-tone still images – baseline. IUT-T recommendation T.87 and ISO/IEC international standard 14495-1:2011.

Jacquin, A.E. 1993. Fractal image coding: A review. *IEEE Transactions on Image Processing*, 81(10):1451-1465.

Jayant, N.S. & Noll, P. 1984. *Digital coding of waveforms*. New Jersey: Prentice-Hall.

Jie, Y., Zhongshan, Z., Huiling, Q., Peihuang, G. & Guoning, Z. 2008. An improved method of remote sensing image compression based on fractal and wavelet domain. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XXXVII(B2):487-490.

JPEG2000 Standards. 1997. Call for contributions for JPEG 2000 (JTC 1.29.14 15 444): Image coding systems. Int. Standards Org./Int. Electrotech. Comm. (ISO/IEC), ISO/IEC JTC1/SC29/WG1/N505, March 1997.

Karagiannakis, P., Weiss, S. & Bowman, J. 2012. Solving the digital signal processing problem for CubeSats. *Fourth European CubeSat Symposium*, Ecole Royale Militaire, Brussels, 30 January – 1 February 2012, 44.

Kauderer, A. 2013. FalconSat-6 nurtures collaboration between young cadets and NASA. http://www.nasa.gov/centers/johnson/home/falconsat6_cadets.html [03 April 2013].

Kautz, W. 1965. Fibonacci codes for synchronization control. *IEEE Transactions on Information Theory*, 11(2):284-292.

Keissarian, F. 2008. Using a novel variable block size image compression algorithm for hiding secret data. *Proceedings of the IEEE International Conference on Signal Image Technology and Internet Based Systems (SITIS-2008)*, Bali, 30 November-03 December 2008, 285-292.

Kirk, D. & Hwu, W. 2012. *Programming massively parallel processors: A hands-on approach (Applications of GPU computing series)*. Chicago: Morgan Kaufmann.

Klofas, B., Anderson, J. & Leveque, K. 2008. A survey of CubeSat communication systems. http://www.klofas.com/papers/CommSurvey-Bryan_Klofas.pdf [28 July 2012].

Kobayashi, H. & Bahl, L.R. 1974. Image data compression by predictive coding I: prediction algorithms. *IMB Journal of Research and Development*, 18(2):164-171.

Kou, W. 1995. *Digital image compression: algorithms and standards*. Dordrecht: Kluwer Academic Publishers.

Krebs, G.D. 2013a. QuakeSat 1. http://space.skyrocket.de/doc_sdat/quakesat-1.htm [03 April 2013].

Krebs, G.D. 2013b. UniSat 1, 2, 4, 5. http://space.skyrocket.de/doc_sdat/unisat-1.htm [30 April 2013].

Langdon, G., Gulati, A. & Seiler, E. 1992. On the JPEG model for lossless image compression. *In Proceedings of the Data Compression Conference (DCC'92)*, Snowbird, 24-27 March 1992, 172-180.

Langdon, G.G.Jr. & Haidinyak, C.A. 1995. Experiments in lossless and virtually lossless image-compression algorithms. *IS&T/SPIE's Symposium on Electronic Imaging: Science and Technology, International Society for Optics and Photonics (*21-27).

Laroia, R., Tretter, S.A. & Farvardin, N. 1993. A simple and effective precoding scheme for noise whitening on intersymbol interference channels. *IEEE Transactions on Communications*, 41(10):1460-1463.

Larson, W.J. & Wertz, J.R. (eds). 1999. *Space mission analysis and design*. California: Microcosm & Dordrecht: Kluwer: 44.

Lee, D.T. 2005. JPEG 2000: Retrospective and new developments. *IEEE Transactions on Image Processing*, 93(1): 32-41, January 2005.

LeGall, D.L. & Tabatabai, A. 1988. Sub-band coding of digital Images using symmetric short kernel filters and arithmetic coding techniques. *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP'88)*, New York, 11-14 April 1988, 2:761–764.

Leondes, C.L. 2002. *Database and data communication network systems: Techniques and applications*. California: Academic Press.

Lewis-Beck, M.S., Bryman, A. & Liao, T.F. (eds). 2004. *The Sage Encyclopedia of Social Science Research methods*, volume 1. Sage.

Lyons, R.G (ed). 2007. *Streamlining digital signal processing: A tricks of the trade guidebook*. New Jersey: John Wiley & Sons.

Mandelbrot, B.B. 1977. *The fractal geometry of nature*. New York: W.H. Freedman and Company.

Masud, S. & Canny, J.V. 1998. Finding a suitable wavelet for image compression applications. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'98)*, Seattle, 12-15 May 1998, 5:2581-2584.

M-Cubed. 2008. Subsystems. http://www-personal.umich.edu/~kgmerek/subsystems.html [03 April 2013].

Meher, S. 2010. Color image denoising with multi-channel spatial color filtering. *Proceedings of the twelfth IEEE International Conference on Computer Modelling and Simulation (UKSim-2010)*, Cambridge, 24-26 March 2010, 284-288.

Monro, D.M. 1993. Class of fractal transforms. *IEEE Electronics Letters*, 29(4):362-363.

Monro, D.M. & Dudbridge, F. 1992a. Fractal approximation of image blocks. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'92)*, San Francisco, 23-26 March 1992, 3:485-488.

Monro, D.M. & Dudbridge, F. 1992b. Fractal block coding of images. *IEEE Electronics Letters*, 28(11):1053-1055, May 1992.

Monro, D.M. & Woolley, S.J. 1994a. Fractal image compression without searching. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'94)*, Adelaide, 19-22 April 1994, 5:557-560.

Monro, D.M. & Woolley, S.J. 1994b. Rate/Distortion in fractal compression: Order of transform and block symmetries. *Fractals*, 2(03):395-398.

Murray, J.D., Van Ryper, W. 1996. *Encyclopedia of graphics file formats*. California: O'Reilly & Associates. http://www.fileformat.info/mirror/egff [15 December 2012].

NASA. 2005. General Environmental Verification Standard (GEVS) for GSFC flight programs and projects. http://standards.gsfc.nasa.gov/gsfc-std/gsfc-std-7000.pdf [24 August 2011].

Nasrabadi, N.M. & King, R.A. 1988. Image coding using vector quantization: A review. *IEEE Transactions on Communications*, 36(8):957-971.

Novak, M. 1993. Attractor coding of images. *Proceedings of the International Picture Coding Symposium (PCS'93)*, Lausanne, March 1993, 6-15.

Nunez, J.L. 2003. Run-length coding extensions for high performance hardware data compression. *IEEE Transactions on Computers and Digital Techniques,* 150(6):387-395.

Oberhumer, M.F.X.J. 2011. LZO real-time data compression library. http://www.oberhumer.com/opensource/lzo/ [30 May 2013].

Odenwald, S., Green, J. & Taylor, W. 2006. Forecasting the impact of an 1859-calibre superstorm on satellite resources. *Advances in Space Research*, 38(2):280-297.

Oien, G.E., Lepsoy, S. & Ramstad, T.A. 1991. An inner product space approach to image coding by contractive transformations. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'91)*, Toronto, 14-17 April 1991, 4:2773-2776.

Pande, A & Zambreno, J. 2008. Design and analysis of efficient reconfigurable wavelet filters. *Proceedings of the IEEE International Conference on Electro/Information Technology (EIT-2008)*, Ames, 18-20 May 2008, 327-332.

Pavlov, I. 2013. High compression ratio in 7z format with LZMA and LZMA2 compression. http://www.7-zip.org/ [01 Jun 2013].

Pennebaker, W.B. & Mitchell, J.L. 1993. *JPEG still image data compression standard*. New York: Van Nostrand Reinhold.

Pigeon, S. & Begio, Y. 1998. A memory-efficient adaptive Huffman coding algorithm for very large sets of symbols. *Proceedings of the Data Compession Conference (DCC'98)*, Snowbird, 30 Mar-1 Apr 1998.

Pinho, A.J. 2001. On the impact of histogram sparseness on some lossless image compression techniques. *Proceedings of the IEEE International Conference on Image Processing*, Thessaloniki, 7-10 October 2001, 3:442-445.

Pinho, A.J. 2002a. An online preprocessing technique for improving the lossless compression of images with sparse histograms. *IEEE Signal Processing Letters*, 9(1):5-7.

Pinho, A.J. 2002b. Preprocessing techniques for improving the lossless compression of images with quasi-sparse and locally sparse histograms. *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME-2002)*, Lausanne, 26-29 August 2002, 1:633-636.

Polesel, A., Ramponi, G. & Mathews, V.J. 2000. Image enhancement via adaptive unsharp masking. *IEEE Transactions on Image Processing*, 9(3):505−510.

Portoni, L., Combi, C., Pozzi, G., Pinciroli, F., Fritsch, J.P. & Brennecke, R. 1997. Angiocardiographic digital still images compressed via irreversible methods: concepts and experiments. *International Journal of Medical Informatics*, 46(3):185-204.

Pratt, W., Kane, J. & Andrews, H.C. 1969. Hadamard transform image coding. *IEEE Computer Processing Communications*, 57(1):58-68.

Preparata, F.P. & Shamos, M.I. 1985. *Computational geometry: texts and monographs in computer science*. New York: Springer-Verlag.

Rabbani, M. & Jones, P.W. 1991. Digital image compression techniques. Vol 7, Bellingham, Washington: *SPIE Press*.

Ramamurthi, B. & Gersho, A. 1986. Classified vector quantization of images. *IEEE Transactions on Communications*, 35(11):1105-1115.

Ranganathan, N., Romaniuk, S.G. & Namuduri, K.R. 1995. A lossless image compression algorithm using variable block size segmentation. *IEEE Transactions on Image Processing*, 4(10):1396-1406.

Rasib, A.W., Hashim, M., Omar, A.H. & Tan, A.A. 2011. Geometric rectification technique for high resolution satellite data imagery using new geocentric-based datum, VOT 75037. http://eprints.utm.my/5674/1/75037.pdf [22 September 2012].

Reusens, E. 1994. Partitioning complexity issue for iterated functions systems based image coding. *Proceedings of the seventh European Signal Processing Conference (EUSIPCO'94)*, Edinburgh, September 1994, 1:171-174.

Rice, R.F. 1979. Some practical universal noiseless coding techniques. Jet Propulsion Laboratory, JPL Publication 79-22, Pasadena, California.

Rice, R.F. 1991. Some practical universal noiseless coding techniques-Part III. Module PSI14.K. Jet Propulsion Laboratory, JPL Publication 91-3, Pasadena, California.

Rissanen, J. and Langdon, G. 1981. Universal modeling and coding. *IEEE Transactions on Information Theory*, 27(1):12-23.

Russ, J.C. 2011. *The image processing handbook*. Florida: CRC Press, Taylor & Francis Group.

Said, A. & Pearlman, W.A. 1996a. An image multiresolution representation for lossless and lossy compression. *IEEE Transactions on Image Processing*, 5(9):1303-1310.

Said, A. & Pearlman, W.A. 1996b. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(3):243-250.

Salomon, D., Motta, G. & Bryant, D (ed). 2010. *Handbook of data compression*. London: Springer-Verlag.

Saupe, D. 1994. Breaking the time complexity of fractal image compression. Technical Report 53, Institut fur Informatik, University of Freiburg, Germany.

Saupe, D. & Hamzaoui, R. 1994. Complexity reduction methods for fractal image compression. In Blackledge, J.M. (ed). *Proceedings of the Institute of Mathematics and its Applications (IMA)*

Conference on Image Processing: Mathematical Methods and Applications, Oxford, September 1994, 211-229.

Schefter, J. 1984. How they build high-tech rockets on the cheap. *Popular science*, 224(5):92-95, May.

Se-Kee, K., Jong-Shill, L., Dong-Fan, S. & Je-Goon, R. 2006. Lossless medical image compression using redundancy analysis. *International Journal of Computer Science and Network Security*, 6(1A):50-56.

SFL. 2011. CanX-1: Canada's first nanosatellite. http://www.utias-sfl.net/nanosatellites/CanX1/ [03 April 2013].

Shannon, C.E, 1948. A mathematical theory of communication. *Bell System Technical Journal*, 27:379-423.

Sifuzzaman, M., Islam, M.R. & Ali, M.Z. 2009. Application of wavelet transform and its advantages compared to Fourier transform. *Journal of Physical Sciences*, (13):121-134.

Skodras, A., Christopoulos, C. & Ebrahimi, T. 2001. The JPEG 2000 still image compression standard. *IEEE Transactions on Signal Processing*, 18(5):36-58.

Smith, J.O. 2007. *Mathematics of the Discrete Fourier Transform (DFT): with audio applications*. Second Edition. California: BookSurge.

Space Micro. 2012. Space electronics products. http://www.spacemicro.com/space_div/se_div.htm [10 November 2012].

Starosolski, R. 2007. Simple fast and adaptive lossless image compression algorithm. *Software – Practice and Experience*, 37(1):65-91.

Sweldens, W. 1996. The lifting scheme: A custom-design construction of biorthogonal wavelets. *ScienceDirect, Applied and Computational Harmonic Analysis*, 3(2):186-200.

Takamura, S. & Takagi, M. 1994. Lossless image compression with lossy image using adaptive prediction and arithmetic coding. *Proceedings of the Data Compression Conference (DCC'94)*, Snowbird, 29-31 March 1994, 155-174.

Taleb, S.A.A., Musafa, H.M.J., Khtoom, A.M. & Gharaybih, I.K. 2010. Improving LZW image compression. *European Journal of Scientific Research*, 44(3):502-509.

Taubman, D. 2000. High performance scalable image compression with EBCOT. *IEEE Transactions on Image Processing*, 9(7): 1158–1170.

Taubman, D. & Zakhor, A. 1994. Multirate 3-D sub-band coding of video. *IEEE Transactions on Image Processing*, 3(5):572-588..

Tzanetakis, G., Essl, G. & Cook, P. 2001. Audio analysis using the discrete wavelet transform. *In Proceedings of the WSES International Conference on Acoustics and Music: Theory and Applications (AMTA-2001)*, Skiathos , 26-30 September 2001.

Usevitch, B.E. 2001. A tutorial on modern lossy wavelet image compression: foundations of JPEG 2000. *IEEE Transactions on Signal Processing*, 18(5):22-35.

Vemuri, B.C., Sahni, S., Chen, F., Kapoor, C., Leonard, C. & Fitzsimmons, J. 2007. Lossless image compression. http://www.cise.ufl.edu/~sahni/papers/encycloimage.pdf [15 April 2011].

Wallace, G.K. 1992. The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics*, 38(1): xviii-xxxiv.

Weinberger, M.J., Seroussi, G. & Sapiro, G. 1996. LOCO-I: A low complexity, context-based, lossless image compression algorithm. *Proceedings of the Data Compression Conference (DCC'96)*, Snowbird, 31 March-03 April 1996, 140-149.

Welch, T.A. 1984. A technique for high-performance data compression. *IEEE Transactions on Computer,* 17(6):8-19.

Witten, I.H., Neal, R.M. & Cleary, J.G. 1987. Arithmetic coding for data compression. *IEEE Transactions on Communications*, 30(6):520-540.

Wohlberg, B. & de Jager, G. 1999. A class of multiresolution stochastic models generating self-affine images. *IEEE Transactions on Signal Processing*, 47(6):1739-1742.

Wolf, P.R. 1983. *Elements of photogrammetry*. New York: McGraw-Hill, 495.

Wu, X. & Menon, N. 1996. CALIC-A Context Based Adaptive Lossless Image Codec. *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'96)*, Atlanta, 09 May 1996, 4:1890-1893.

Wu, Q., Schulze M.A. & Castleman, K.R. 1998. Steerable pyramid filters for selective image enhancement applications. *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'98)*, Monterey, 31 May-03 June 1998, 5:325-328.

Xiaoyu, R & Katti, R. 2006. Reducing the length of Shannon-Fano-Elias codes and Shannon-Fano codes. *Proceedings of the IEEE Military Communications Conference (MILCOM-2006)*, Washington, DC, 23-25 October 2006, 1-7.

Yuhaniz, S., Vladimirova, T. & Sweeting, M. 2005. Embedded intelligent imaging on-board small satellites. *Advances in Computer Systems Architecture*, 3740:90-103.

Zandi, A., Boliek, M., Schwartz, E.L. & Gormish, M.J. 1995. CREW lossless/lossy medical image compresion. Technical Report CRC-TR-9526, RICOH California Research Center.

Ziv, J. & Lempel, A. 1977. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337-343.

Ziv, J. & Lempel, A. 1978. Compression of individual sequences via variable-rate. *IEEE Transactions on Information Theory*, 24(5):530-536.

APPENDICES

9. Appendix A: Compilation Details

The "**Build system checkout**" in (Gumstix, 2012b) builds the latest kernel and if needed to build a specific kernel e.g. kernel 2.6.34; the following has to be changed:

```
git checkout --track -b overo-2011.03 origin/overo-2011.03
```

to

```
git checkout --track -b overo origin/overo
```

and has to add

```
PREFERRED_VERSION_linux-omap3="2.6.34"
```

at the end of "local.conf" file located at:

```
$HOME/overo-oe/build/conf/
```

Since the target is on the CPU/DSP, the DSP image is built using the command:

```
overo-oe$ bitbake dsp-console-image
```

In order to avoid a common mistake, the profile file is captured as source before using bitbake.

```
$ source $HOME/overo-oe/build/profile
```

To avoid this, each time a new shell session is running (command line window is opened), the following command line is set as default:

```
$ echo source $HOME/overo-oe/build/profile >> $HOME/.bashrc
```

Before building the "dsp-console-image", a custom "dsp-console-image" (dsp-console-image.bb) has to be created within the following directory:
"**~/overo-oe/org.openembedded.dev/recipes/images/**" (Ellis, 2012).

Download the "ti_cgt_c6000_6.1.17_setup_linux_x86.bin" file from Texas Instruments website (https://www-a.ti.com/downloads/sds_support/CodeGenerationTools.htm) which requires a free registration.
The "ti_cgt_c6000_6.1.17_setup_linux_x86.bin" file is saved within the "$HOME/overo-oe/sources/" directory where the checksum is computed and compared to the one in "overo-oe/org.openembedded.dev/recipes/ti/ti-cgt6x_6.1.17.bb". The checksum is computed as:

```
sources$ md5sum ti_cgt_c6000_6.1.17_setup_linux_x86.bin > ti_cgt_c6000_6.1.17_setup_linux_x86.bin.md5

sources$ sha256sum ti_cgt_c6000_6.1.17_setup_linux_x86.bin > ti_cgt_c6000_6.1.17_setup_linux_x86.bin.sha256sum
```

and the computed results are:

```
sources$ cat ti_cgt_c6000_6.1.17_setup_linux_x86.bin.md5
5ee5c8e573ab0a1ba1249511d4a06c27  ti_cgt_c6000_6.1.17_setup_linux_x86.bin

sources$ cat ti_cgt_c6000_6.1.17_setup_linux_x86.bin.sha256sum
0cb99e755f5d06a74db22d7c814e4dfd36aa5fcb35eeab01ddb000aef99c08c1  ti_cgt_c6000_6.1.17_setup_linux_x86.bin
```

The checksum in “overo-oe/org.openembedded.dev/recipes/ti/ti-cgt6x_6.1.17.bb” has to be replaced with:

```
SRC_URI[cgt6xbin.md5sum] = "5ee5c8e573ab0a1ba1249511d4a06c27"
SRC_URI[cgt6xbin.sha256sum] = "0cb99e755f5d06a74db22d7c814e4dfd36aa5fcb35eeab01ddb000aef99c08c1"
```

The “task-gstreamer-ti” is built using the following command:

```
$ cd ~/overo-oe
$ bitbake task-gstreamer-ti
```

The “dsp-console-image” is built using the following command:

```
$ cd ~/overo-oe
$ dsp-console-image
```

During the building process of the “dsp-console-image”, few challenges have been encountered namely:

1. The recipe “bluez4_4.89.bb failed”, this is solved by replacing its “SRC_URI[md5sum] and SRC_URI[sha256sum]” to:

   ```
   SRC_URI[md5sum] = "bb0a5864b16c911993d2bceb839edb3c"
   SRC_URI[sha256sum] = "b493530e0cf6798e578a6ce322f513d2ab9a9c1d1987e93acb96974e86b52f1e"
   ```

2. The recipe “libtool_2.2.6b.bb failed”, this is solved by comparing the check sum from the sources file package to the one in the recipes:

   ```
   sources$ cat libtool-2.2.6b.tar.gz.md5
   ```

   ```
   sources$ cat libtool-2.2.6b.tar.gz.sha256
   ```

3. The recipe “docbook-sgml-dtd-3.1-native.bb failed”, this is solved by replacing the SRC_URI with:

```
SRC_URI = "http://www.docbook.org/sgml/${DTD_VERSION}/docbook-${DTD_VERSION}.zip"
```

The kernel loader (x-load) is built as follows:

```
$ cd ~/overo-oe
$ bitbake x-load
```

It is also necessary to build the tool chain for cross compilation:

```
$ cd ~/overo-oe
$ bitbake meta-toolchain
```

and is available at “**$HOME/overo-oe/tmp/sysroots/i686-linux/usr/armv7a/bin/**”

At this stage, the most important components are:

1. The “MLO-overo-1.44+r20+gitr24b8b7f41a83540433024854736518876257672c-r20, u-boot-overo-2010.9+r1+git1e4e5ef0469050f014aee1204dae8a9ab6053e49-r1.bin, uImage-2.6.34-r100-overo.bin and dsp-console-image-overo.tar.bz2”, all available at “**$HOME/overo-oe/tmp/deploy/glibc/images/overo/**” which will be used to create a bootable micro SD Card (Gumstix, 2012c).

2. All “.ipk” files available at “**$HOME/overo-oe/tmp/deploy/glibc/ipk/overo/**” which will be installed in the Gumstix hardware compilation environment.

3. All “libopencv” binary files in the directories “**$HOME/overo-oe/tmp/work/armv7a-angstrom-linux-gnueabi/opencv-2.2.0+svnr4462-r3/opencv/lib**” and “**$HOME/overo-oe/tmp/sysroots/armv7a-angstrom-linux-gnueabi/usr/lib**”, which have to be copied to Gumstix hardware compilation environment library directory.

10. Appendix B: Code Listing

Main program (imagecompression)

```
/*
 * This program compresses lossless images
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

#include "openlib.h"
#include "file.h"
#include "imageconversion.h"
#include "dirent.h"
#include "index.h"

#ifndef WIN32
#define stricmp strcasecmp
#define strnicmp strncasecmp
#endif

/* ---------------------------------------------------------------------- */

#define RED   0
#define GREEN 1
#define BLUE  2

#define PXM_DFMT 10
#define PGX_DFMT 11
#define BMP_DFMT 12
#define YUV_DFMT 13
#define TIF_DFMT 14
#define RAW_DFMT 15
#define TGA_DFMT 16


typedef struct dircnt{
        /** Buffer for holding images read from Directory*/
        char *filename_buf;
        /** Pointer to the buffer*/
        char **filename;
}dircnt_t;

typedef struct img_folder{
        /** The directory path of the folder containing input images*/
        char *imgdirpath;
```

```
        /** Output format*/
        char *out_format;
        /** Enable option*/
        char set_imgdir;
        /** Enable Cod Format for output*/
        char set_out_format;

}img_fol_t;

void encode_help_display() {
        fprintf(stdout,"HELP\n----\n\n");
        fprintf(stdout,"Example: executable_file -i input_image.raw -o output_image.j2k\n\n");

}

OPJ_PROG_ORDER give_progression(char progression[4]) {
        if(strncmp(progression, "LRCP", 4) == 0) {
                return LRCP;
        }
        if(strncmp(progression, "RLCP", 4) == 0) {
                return RLCP;
        }
        if(strncmp(progression, "RPCL", 4) == 0) {
                return RPCL;
        }
        if(strncmp(progression, "PCRL", 4) == 0) {
                return PCRL;
        }
        if(strncmp(progression, "CPRL", 4) == 0) {
                return CPRL;
        }

        return PROG_UNKNOWN;
}

int get_num_images(char *imgdirpath){
        DIR *dir;
        struct dirent* content;
        int num_images = 0;

        /*Reading the input images from given input directory*/

        dir= opendir(imgdirpath);
        if(!dir){
                fprintf(stderr,"Could not open Folder %s\n",imgdirpath);
                return 0;
        }

        num_images=0;
        while((content=readdir(dir))!=NULL){
                if(strcmp(".",content->d_name)==0 || strcmp("..",content->d_name)==0 )
                        continue;
                num_images++;
```

```
	}
	return num_images;
}

int load_images(dircnt_t *dirptr, char *imgdirpath){
	DIR *dir;
	struct dirent* content;
	int i = 0;

	/*Reading the input images from given input directory*/

	dir= opendir(imgdirpath);
	if(!dir){
		fprintf(stderr,"Could not open Folder %s\n",imgdirpath);
		return 1;
	}else	{
		fprintf(stderr,"Folder opened successfully\n");
	}

	while((content=readdir(dir))!=NULL){
		if(strcmp(".",content->d_name)==0 || strcmp("..",content->d_name)==0 )
			continue;

		strcpy(dirptr->filename[i],content->d_name);
		i++;
	}
	return 0;
}

int get_file_format(char *filename) {
	unsigned int i;
	static const char *extension[] = {
   "pgx", "pnm", "pgm", "ppm", "bmp", "tif", "raw", "tga", "j2k", "jp2", "j2c"
   };
	static const int format[] = {
   PGX_DFMT, PXM_DFMT, PXM_DFMT, PXM_DFMT, BMP_DFMT, TIF_DFMT, RAW_DFMT,
TGA_DFMT, RED, GREEN, BLUE
   };
	char * ext = strrchr(filename, '.');
	if (ext == NULL)
		return -1;
	ext++;
	for(i = 0; i < sizeof(format)/sizeof(*format); i++) {
		if(strnicmp(ext, extension[i], 3) == 0) {
			return format[i];
		}
	}
	return -1;
}

char * get_file_name(char *name){
	char *fname;
	fname= (char*)malloc(OPJ_PATH_LEN*sizeof(char));
```

```
	fname= strtok(name,".");
	return fname;
}

char get_next_file(int imageno,dircnt_t *dirptr,img_fol_t *img_fol, opj_cparameters_t
*parameters){
	char image_filename[OPJ_PATH_LEN],
infilename[OPJ_PATH_LEN],outfilename[OPJ_PATH_LEN],temp_ofname[OPJ_PATH_LEN];
  char *temp_p, temp1[OPJ_PATH_LEN]="";

	strcpy(image_filename,dirptr->filename[imageno]);
	fprintf(stderr,"File Number %d \"%s\"\n",imageno,image_filename);
	parameters->decod_format = get_file_format(image_filename);
	if (parameters->decod_format == -1)
		return 1;
	sprintf(infilename,"%s/%s",img_fol->imgdirpath,image_filename);
	strncpy(parameters->infile, infilename, sizeof(infilename));

	//Set output file
	strcpy(temp_ofname,get_file_name(image_filename));
	while((temp_p = strtok(NULL,".")) != NULL){
		strcat(temp_ofname,temp1);
		sprintf(temp1,".%s",temp_p);
	}
	if(img_fol->set_out_format==1){
		sprintf(outfilename,"%s/%s.%s",img_fol->imgdirpath,temp_ofname,img_fol-
>out_format);
		strncpy(parameters->outfile, outfilename, sizeof(outfilename));
	}
 return 0;
}

static int initialise_4K_poc(opj_poc_t *POC, int numres){
	POC[0].tile  = 1;
	POC[0].resno0  = 0;
	POC[0].compno0 = 0;
	POC[0].layno1  = 1;
	POC[0].resno1  = numres-1;
	POC[0].compno1 = 3;
	POC[0].prg1 = CPRL;
	POC[1].tile  = 1;
	POC[1].resno0  = numres-1;
	POC[1].compno0 = 0;
	POC[1].layno1  = 1;
	POC[1].resno1  = numres;
	POC[1].compno1 = 3;
	POC[1].prg1 = CPRL;
	return 2;
}



/* ----------------------------------------------------------------------------- */
```

```
int parse_cmdline_encoder(int argc, char **argv, opj_cparameters_t *parameters, img_fol_t
*img_fol, raw_cparameters_t *raw_cp, char *indexfilename) {
	int i, j,totlen;
	option_t long_option[]={

		{"ImgDir",REQ_ARG, NULL ,'z'},
		{"TP",REQ_ARG, NULL ,'v'},
		{"SOP",NO_ARG, NULL ,'S'},
		{"EPH",NO_ARG, NULL ,'E'},
		{"OutFor",REQ_ARG, NULL ,'O'},
		{"POC",REQ_ARG, NULL ,'P'},
	};

	/* parse the command line */
	const char optlist[] = "i:o:hr:q:n:b:c:t:p:s:SEM:x:R:d:T:If:P:C:F:"
#ifdef USE_JPWL
		"W:"
#endif /* USE_JPWL */
		;

	totlen=sizeof(long_option);
	img_fol->set_out_format=0;
	raw_cp->rawWidth = 0;

	while (1) {
   int c = getopt_long(argc, argv, optlist,long_option,totlen);
		if (c == -1)
			break;
		switch (c) {
			case 'i':			/* input file */
			{
				char *infile = optarg;
				parameters->decod_format = get_file_format(infile);
				switch(parameters->decod_format) {
					case PGX_DFMT:
					case PXM_DFMT:
					case BMP_DFMT:
					case TIF_DFMT:
					case RAW_DFMT:
					case TGA_DFMT:
						break;
					default:
						fprintf(stderr,
							"!! Unrecognized format for infile : %s "
	"[accept only *.pnm, *.pgm, *.ppm, *.pgx, *.bmp, *.tif, *.raw or *.tga] !!\n\n",
							infile);
						return 1;
				}
				strncpy(parameters->infile, infile, sizeof(parameters->infile)-1);
			}
			break;
```

```
                /* ----------------------------------------------------- */

            case 'o':                       /* output file */
            {
                char *outfile = optarg;
                parameters->cod_format = get_file_format(outfile);
                switch(parameters->cod_format) {
                    case RED:
                    case GREEN:
                        break;
                    default:
                        fprintf(stderr, "Unknown output format image %s
[only *.j2k, *.j2c or *.jp2]!! \n", outfile);
                        return 1;
                }
                strncpy(parameters->outfile, outfile, sizeof(parameters->outfile)-1);
            }
            break;

                /* ----------------------------------------------------- */
            case 'O':                       /* output format */
                {
                    char outformat[50];
                    char *of = optarg;
                    sprintf(outformat,".%s",of);
                    img_fol->set_out_format = 1;
                    parameters->cod_format = get_file_format(outformat);
                    switch(parameters->cod_format) {
                        case RED:
                        case GREEN:
                            img_fol->out_format = optarg;
                            break;
                        default:
                            fprintf(stderr, "Unknown output format image
[only j2k, j2c, jp2]!! \n");
                            return 1;
                    }
                }
                break;


                /* ----------------------------------------------------- */


            case 'r':                       /* rates/distorsion */
            {
                char *s = optarg;
                while (sscanf(s, "%f", &parameters->tcp_rates[parameters-
>tcp_numlayers]) == 1) {
                    parameters->tcp_numlayers++;
                    while (*s && *s != ',') {
                        s++;
                    }
```

```
                        if (!*s)
                                break;
                        s++;
                }
                parameters->cp_disto_alloc = 1;
        }
        break;

                /* ------------------------------------------------------ */


        case 'F':                       /* Raw image format parameters */
        {
                char signo;
                char *s = optarg;
                if (sscanf(s, "%d,%d,%d,%d,%c", &raw_cp->rawWidth, &raw_cp-
>rawHeight, &raw_cp->rawComp, &raw_cp->rawBitDepth, &signo) == 5) {
                        if (signo == 's') {
                                raw_cp->rawSigned = true;
                                fprintf(stdout,"\nRaw file parameters: %d,%d,%d,%d
Signed\n", raw_cp->rawWidth, raw_cp->rawHeight, raw_cp->rawComp, raw_cp->rawBitDepth);
                        }
                        else if (signo == 'u') {
                                raw_cp->rawSigned = false;
                                fprintf(stdout,"\nRaw file parameters: %d,%d,%d,%d
Unsigned\n", raw_cp->rawWidth, raw_cp->rawHeight, raw_cp->rawComp, raw_cp-
>rawBitDepth);
                        }
                        else {
                                fprintf(stderr,"\nError: invalid raw image parameters:
Unknown sign of raw file\n");
                                fprintf(stderr,"Please use the Format option -F:\n");
                                fprintf(stderr,"-F
rawWidth,rawHeight,rawComp,rawBitDepth,s/u (Signed/Unsigned)\n");
                                fprintf(stderr,"Example: -i lena.raw -o lena.j2k -F
512,512,3,8,u\n");
                                fprintf(stderr,"Aborting\n");
                        }
                }
                else {
                        fprintf(stderr,"\nError: invalid raw image parameters\n");
                        fprintf(stderr,"Please use the Format option -F:\n");
                        fprintf(stderr,"-F
rawWidth,rawHeight,rawComp,rawBitDepth,s/u (Signed/Unsigned)\n");
                                fprintf(stderr,"Example: -i lena.raw -o lena.j2k -F
512,512,3,8,u\n");
                        fprintf(stderr,"Aborting\n");
                        return 1;
                }
        }
        break;

                /* ------------------------------------------------------ */
```

```
                        case 'q':                       /* add fixed_quality */
                        {
                                char *s = optarg;
                                while (sscanf(s, "%f", &parameters->tcp_distoratio[parameters-
>tcp_numlayers]) == 1) {
                                        parameters->tcp_numlayers++;
                                        while (*s && *s != ',') {
                                                s++;
                                        }
                                        if (!*s)
                                                break;
                                        s++;
                                }
                                parameters->cp_fixed_quality = 1;
                        }
                        break;

                                /* dda */
                                /* ------------------------------------------------------ */

                        case 'f':                       /* mod fixed_quality (before : -q) */
                        {
                                int *row = NULL, *col = NULL;
                                int numlayers = 0, numresolution = 0, matrix_width = 0;

                                char *s = optarg;
                                sscanf(s, "%d", &numlayers);
                                s++;
                                if (numlayers > 9)
                                        s++;

                                parameters->tcp_numlayers = numlayers;
                                numresolution = parameters->numresolution;
                                matrix_width = numresolution * 3;
                                parameters->cp_matrice = (int *) malloc(numlayers * matrix_width
* sizeof(int));
                                s = s + 2;

                                for (i = 0; i < numlayers; i++) {
                                        row = &parameters->cp_matrice[i * matrix_width];
                                        col = row;
                                        parameters->tcp_rates[i] = 1;
                                        sscanf(s, "%d,", &col[0]);
                                        s += 2;
                                        if (col[0] > 9)
                                                s++;
                                        col[1] = 0;
                                        col[2] = 0;
                                        for (j = 1; j < numresolution; j++) {
                                                col += 3;
                                                sscanf(s, "%d,%d,%d", &col[0], &col[1], &col[2]);
                                                s += 6;
```

```
                                        if (col[0] > 9)
                                                s++;
                                        if (col[1] > 9)
                                                s++;
                                        if (col[2] > 9)
                                                s++;
                                }
                                if (i < numlayers - 1)
                                        s++;
                        }
                        parameters->cp_fixed_alloc = 1;
                }
                break;

                        /* ----------------------------------------------------- */

                case 't':                       /* tiles */
                {
                        sscanf(optarg, "%d,%d", &parameters->cp_tdx, &parameters-
>cp_tdy);
                        parameters->tile_size_on = true;
                }
                break;

                        /* ----------------------------------------------------- */

                case 'n':                       /* resolution */
                {
                        sscanf(optarg, "%d", &parameters->numresolution);
                }
                break;

                        /* ----------------------------------------------------- */
                case 'c':                       /* precinct dimension */
                {
                        char sep;
                        int res_spec = 0;

                        char *s = optarg;
                        do {
                                sep = 0;
                                sscanf(s, "[%d,%d]%c", &parameters->prcw_init[res_spec],
        &parameters->prch_init[res_spec], &sep);
                                parameters->csty |= 0x01;
                                res_spec++;
                                s = strpbrk(s, "]") + 2;
                        }
                        while (sep == ',');
                        parameters->res_spec = res_spec;
                }
                break;

                        /* ----------------------------------------------------- */
```

```
                        case 'b':                       /* code-block dimension */
                        {
                                int cblockw_init = 0, cblockh_init = 0;
                                sscanf(optarg, "%d,%d", &cblockw_init, &cblockh_init);
                                if (cblockw_init * cblockh_init > 4096 || cblockw_init > 1024
                                        || cblockw_init < 4 || cblockh_init > 1024 || cblockh_init < 4)
{
                                        fprintf(stderr,
                                                "!! Size of code_block error (option -b)
!!\n\nRestriction :\n"
        "    * width*height<=4096\n    * 4<=width,height<= 1024\n\n");
                                        return 1;
                                }
                                parameters->cblockw_init = cblockw_init;
                                parameters->cblockh_init = cblockh_init;
                        }
                        break;

                                /* ------------------------------------------------------ */

                        case 'x':                       /* creation of index file */
                        {
                                char *index = optarg;
                                strncpy(indexfilename, index, OPJ_PATH_LEN);
                        }
                        break;

                                /* ------------------------------------------------------ */

                        case 'p':                       /* progression order */
                        {
                                char progression[4];

                                strncpy(progression, optarg, 4);
                                parameters->prog_order = give_progression(progression);
                                if (parameters->prog_order == -1) {
                                        fprintf(stderr, "Unrecognized progression order "
        "[LRCP, RLCP, RPCL, PCRL, CPRL] !!\n");
                                        return 1;
                                }
                        }
                        break;

                                /* ------------------------------------------------------ */

                        case 's':                       /* subsampling factor */
                        {
                                if (sscanf(optarg, "%d,%d", &parameters->subsampling_dx,
                                 &parameters->subsampling_dy) != 2) {
                                        fprintf(stderr,   "'-s' sub-sampling argument error !  [-s
dx,dy]\n");
                                        return 1;
```

```
                        }
                    }
                    break;

                        /* ------------------------------------------------------ */

                    case 'd':                   /* coordonnate of the reference grid */
                    {
                        if (sscanf(optarg, "%d,%d", &parameters->image_offset_x0,
                         &parameters->image_offset_y0) != 2) {
                            fprintf(stderr,   "-d 'coordonnate of the reference grid'
argument "
          "error !! [-d x0,y0]\n");
                            return 1;
                        }
                    }
                    break;

                        /* ------------------------------------------------------ */

                    case 'h':                   /* display an help description */
                        encode_help_display();
                        return 1;

                        /* ------------------------------------------------------ */

                    case 'P':                   /* POC */
                    {
                        int numpocs = 0;            /* number of progression order
change (POC) default 0 */
                        opj_poc_t *POC = NULL;      /* POC : used in case of Progression
order change */

                        char *s = optarg;
                        POC = parameters->POC;

                        while (sscanf(s, "T%d=%d,%d,%d,%d,%d,%4s",
&POC[numpocs].tile,
                            &POC[numpocs].resno0, &POC[numpocs].compno0,
                            &POC[numpocs].layno1, &POC[numpocs].resno1,
                            &POC[numpocs].compno1, &POC[numpocs].progorder) ==
7) {
                            POC[numpocs].prg1 =
give_progression(POC[numpocs].progorder);
                            numpocs++;
                            while (*s && *s != '/') {
                                s++;
                            }
                            if (!*s) {
                                break;
                            }
                            s++;
                        }
```

```
                parameters->numpocs = numpocs;
            }
            break;

                /* ------------------------------------------------------ */

            case 'S':                   /* SOP marker */
            {
                parameters->csty |= 0x02;
            }
            break;

                /* ------------------------------------------------------ */

            case 'E':                   /* EPH marker */
            {
                parameters->csty |= 0x04;
            }
            break;

                /* ------------------------------------------------------ */

            case 'M':                   /* Mode switch pas tous au point !! */
            {
                int value = 0;
                if (sscanf(optarg, "%d", &value) == 1) {
                    for (i = 0; i <= 5; i++) {
                        int cache = value & (1 << i);
                        if (cache)
                            parameters->mode |= (1 << i);
                    }
                }
            }
            break;

                /* ------------------------------------------------------ */

            case 'T':                   /* Tile offset */
            {
                if (sscanf(optarg, "%d,%d", &parameters->cp_tx0, &parameters-
>cp_ty0) != 2) {
                    fprintf(stderr, "-T 'tile offset' argument error !! [-T X0,Y0]");
                    return 1;
                }
            }
            break;

                /* ------------------------------------------------------ */

            case 'C':                   /* add a comment */
            {
                parameters->cp_comment = (char*)malloc(strlen(optarg) + 1);
                if(parameters->cp_comment) {
```

```
                        strcpy(parameters->cp_comment, optarg);
                }
        }
        break;


                /* ------------------------------------------------------ */

        case 'I':                       /* reversible or not */
        {
                parameters->irreversible = 1;
        }
        break;

        /* ------------------------------------------------------ */

        case 'v':                       /* Tile part generation*/
        {
                parameters->tp_flag = optarg[0];
                parameters->tp_on = 1;
        }
        break;

                /* ------------------------------------------------------ */

        case 'z':                       /* Image Directory path */
        {
                img_fol->imgdirpath = (char*)malloc(strlen(optarg) + 1);
                strcpy(img_fol->imgdirpath,optarg);
                img_fol->set_imgdir=1;
        }
        break;



/* UniPG>> */
#ifdef USE_JPWL
                /* ------------------------------------------------------ */

        case 'W':                       /* JPWL capabilities switched on */
        {
                char *token = NULL;
                int hprot, pprot, sens, addr, size, range;

                /* we need to enable indexing */
                if (!indexfilename) {
                        strncpy(indexfilename, JPWL_PRIVATEINDEX_NAME,
OPJ_PATH_LEN);
                }

                /* search for different protection methods */

                /* break the option in comma points and parse the result */
```

```
                                token = strtok(optarg, ",");
                                while(token != NULL) {

                                        /* search header error protection method */
                                        if (*token == 'h') {

                                                static int tile = 0, tilespec = 0, lasttileno = 0;

                                                hprot = 1; /* predefined method */

                                                if(sscanf(token, "h=%d", &hprot) == 1) {
                                                        /* Main header, specified */
                                                        if (!((hprot == 0) || (hprot == 1) || (hprot ==
16) || (hprot == 32) ||
                                                                ((hprot >= 37) && (hprot <= 128)))) {
                                                                fprintf(stderr, "ERROR -> invalid
main header protection method h = %d\n", hprot);
                                                                return 1;
                                                        }
                                                        parameters->jpwl_hprot_MH = hprot;

                                                } else if(sscanf(token, "h%d=%d", &tile, &hprot) ==
2) {
                                                        /* Tile part header, specified */
                                                        if (!((hprot == 0) || (hprot == 1) || (hprot ==
16) || (hprot == 32) ||
                                                                ((hprot >= 37) && (hprot <= 128)))) {
                                                                fprintf(stderr, "ERROR -> invalid tile
part header protection method h = %d\n", hprot);
                                                                return 1;
                                                        }
                                                        if (tile < 0) {
                                                                fprintf(stderr, "ERROR -> invalid tile
part number on protection method t = %d\n", tile);
                                                                return 1;
                                                        }
                                                        if (tilespec < JPWL_MAX_NO_TILESPECS)
{
                                                                parameters-
>jpwl_hprot_TPH_tileno[tilespec] = lasttileno = tile;
                                                                parameters-
>jpwl_hprot_TPH[tilespec++] = hprot;
                                                        }

                                                } else if(sscanf(token, "h%d", &tile) == 1) {
                                                        /* Tile part header, unspecified */
                                                        if (tile < 0) {
                                                                fprintf(stderr, "ERROR -> invalid tile
part number on protection method t = %d\n", tile);
                                                                return 1;
                                                        }
                                                        if (tilespec < JPWL_MAX_NO_TILESPECS)
{
```

```
                                                parameters-
>jpwl_hprot_TPH_tileno[tilespec] = lasttileno = tile;
                                                parameters-
>jpwl_hprot_TPH[tilespec++] = hprot;
                                        }


                                } else if (!strcmp(token, "h")) {
                                        /* Main header, unspecified */
                                        parameters->jpwl_hprot_MH = hprot;

                                } else {
                                        fprintf(stderr, "ERROR -> invalid protection
method selection = %s\n", token);
                                        return 1;
                                };

                        }

                        /* search packet error protection method */
                        if (*token == 'p') {

                                static int pack = 0, tile = 0, packspec = 0, lastpackno
= 0;

                                pprot = 1; /* predefined method */

                                if (sscanf(token, "p=%d", &pprot) == 1) {
                                        /* Method for all tiles and all packets */
                                        if (!((pprot == 0) || (pprot == 1) || (pprot ==
16) || (pprot == 32) ||
                                                ((pprot >= 37) && (pprot <= 128)))) {
                                                fprintf(stderr, "ERROR -> invalid
default packet protection method p = %d\n", pprot);
                                                return 1;
                                        }
                                        parameters->jpwl_pprot_tileno[0] = 0;
                                        parameters->jpwl_pprot_packno[0] = 0;
                                        parameters->jpwl_pprot[0] = pprot;

                                } else if (sscanf(token, "p%d=%d", &tile, &pprot) ==
2) {
                                        /* method specified from that tile on */
                                        if (!((pprot == 0) || (pprot == 1) || (pprot ==
16) || (pprot == 32) ||
                                                ((pprot >= 37) && (pprot <= 128)))) {
                                                fprintf(stderr, "ERROR -> invalid
packet protection method p = %d\n", pprot);
                                                return 1;
                                        }
                                        if (tile < 0) {
                                                fprintf(stderr, "ERROR -> invalid tile
part number on protection method p = %d\n", tile);
```

```
                                        return 1;
                                    }
                                    if (packspec <
JPWL_MAX_NO_PACKSPECS) {
                                        parameters-
>jpwl_pprot_tileno[packspec] = tile;
                                        parameters-
>jpwl_pprot_packno[packspec] = 0;
                                        parameters->jpwl_pprot[packspec++]
= pprot;
                                    }

                                } else if (sscanf(token, "p%d:%d=%d", &tile, &pack,
&pprot) == 3) {
                                    /* method fully specified from that tile and
that packet on */
                                    if (!((pprot == 0) || (pprot == 1) || (pprot ==
16) || (pprot == 32) ||
                                        ((pprot >= 37) && (pprot <= 128)))) {
                                        fprintf(stderr, "ERROR -> invalid
packet protection method p = %d\n", pprot);
                                        return 1;
                                    }
                                    if (tile < 0) {
                                        fprintf(stderr, "ERROR -> invalid tile
part number on protection method p = %d\n", tile);
                                        return 1;
                                    }
                                    if (pack < 0) {
                                        fprintf(stderr, "ERROR -> invalid
packet number on protection method p = %d\n", pack);
                                        return 1;
                                    }
                                    if (packspec <
JPWL_MAX_NO_PACKSPECS) {
                                        parameters-
>jpwl_pprot_tileno[packspec] = tile;
                                        parameters-
>jpwl_pprot_packno[packspec] = pack;
                                        parameters->jpwl_pprot[packspec++]
= pprot;
                                    }

                                } else if (sscanf(token, "p%d:%d", &tile, &pack) ==
2) {
                                    /* default method from that tile and that
packet on */
                                    if (!((pprot == 0) || (pprot == 1) || (pprot ==
16) || (pprot == 32) ||
                                        ((pprot >= 37) && (pprot <= 128)))) {
                                        fprintf(stderr, "ERROR -> invalid
packet protection method p = %d\n", pprot);
                                        return 1;
```

```
                                                }
                                                if (tile < 0) {
                                                        fprintf(stderr, "ERROR -> invalid tile
part number on protection method p = %d\n", tile);
                                                        return 1;
                                                }
                                                if (pack < 0) {
                                                        fprintf(stderr, "ERROR -> invalid
packet number on protection method p = %d\n", pack);
                                                        return 1;
                                                }
                                                if (packspec <
JPWL_MAX_NO_PACKSPECS) {
                                                        parameters-
>jpwl_pprot_tileno[packspec] = tile;
                                                        parameters-
>jpwl_pprot_packno[packspec] = pack;
                                                        parameters->jpwl_pprot[packspec++]
= pprot;
                                                }

                                        } else if (sscanf(token, "p%d", &tile) == 1) {
                                                /* default from a tile on */
                                                if (tile < 0) {
                                                        fprintf(stderr, "ERROR -> invalid tile
part number on protection method p = %d\n", tile);
                                                        return 1;
                                                }
                                                if (packspec <
JPWL_MAX_NO_PACKSPECS) {
                                                        parameters-
>jpwl_pprot_tileno[packspec] = tile;
                                                        parameters-
>jpwl_pprot_packno[packspec] = 0;
                                                        parameters->jpwl_pprot[packspec++]
= pprot;
                                                }


                                        } else if (!strcmp(token, "p")) {
                                                /* all default */
                                                parameters->jpwl_pprot_tileno[0] = 0;
                                                parameters->jpwl_pprot_packno[0] = 0;
                                                parameters->jpwl_pprot[0] = pprot;

                                        } else {
                                                fprintf(stderr, "ERROR -> invalid protection
method selection = %s\n", token);
                                                return 1;
                                        };

                                }
```

```
                                        /* search sensitivity method */
                                        if (*token == 's') {

                                                static int tile = 0, tilespec = 0, lasttileno = 0;

                                                sens = 0; /* predefined: relative error */

                                                if(sscanf(token, "s=%d", &sens) == 1) {
                                                        /* Main header, specified */
                                                        if ((sens < -1) || (sens > 7)) {
                                                                fprintf(stderr, "ERROR -> invalid
main header sensitivity method s = %d\n", sens);
                                                                return 1;
                                                        }
                                                        parameters->jpwl_sens_MH = sens;

                                                } else if(sscanf(token, "s%d=%d", &tile, &sens) ==
2) {
                                                        /* Tile part header, specified */
                                                        if ((sens < -1) || (sens > 7)) {
                                                                fprintf(stderr, "ERROR -> invalid tile
part header sensitivity method s = %d\n", sens);
                                                                return 1;
                                                        }
                                                        if (tile < 0) {
                                                                fprintf(stderr, "ERROR -> invalid tile
part number on sensitivity method t = %d\n", tile);
                                                                return 1;
                                                        }
                                                        if (tilespec < JPWL_MAX_NO_TILESPECS)
{
                                                                parameters-
>jpwl_sens_TPH_tileno[tilespec] = lasttileno = tile;
                                                                parameters-
>jpwl_sens_TPH[tilespec++] = sens;
                                                        }

                                                } else if(sscanf(token, "s%d", &tile) == 1) {
                                                        /* Tile part header, unspecified */
                                                        if (tile < 0) {
                                                                fprintf(stderr, "ERROR -> invalid tile
part number on sensitivity method t = %d\n", tile);
                                                                return 1;
                                                        }
                                                        if (tilespec < JPWL_MAX_NO_TILESPECS)
{
                                                                parameters-
>jpwl_sens_TPH_tileno[tilespec] = lasttileno = tile;
                                                                parameters-
>jpwl_sens_TPH[tilespec++] = hprot;
                                                        }

                                                } else if (!strcmp(token, "s")) {
```

```
                                        /* Main header, unspecified */
                                        parameters->jpwl_sens_MH = sens;

                                } else {
                                        fprintf(stderr, "ERROR -> invalid sensitivity
method selection = %s\n", token);
                                        return 1;
                                };

                                parameters->jpwl_sens_size = 2; /* 2 bytes for
default size */
                        }

                        /* search addressing size */
                        if (*token == 'a') {

                                static int tile = 0, tilespec = 0, lasttileno = 0;

                                addr = 0; /* predefined: auto */

                                if(sscanf(token, "a=%d", &addr) == 1) {
                                        /* Specified */
                                        if ((addr != 0) && (addr != 2) && (addr != 4)) {
                                                fprintf(stderr, "ERROR -> invalid
addressing size a = %d\n", addr);
                                                return 1;
                                        }
                                        parameters->jpwl_sens_addr = addr;

                                } else if (!strcmp(token, "a")) {
                                        /* default */
                                        parameters->jpwl_sens_addr = addr; /* auto
for default size */

                                } else {
                                        fprintf(stderr, "ERROR -> invalid addressing
selection = %s\n", token);
                                        return 1;
                                };

                        }

                        /* search sensitivity size */
                        if (*token == 'z') {

                                static int tile = 0, tilespec = 0, lasttileno = 0;

                                size = 1; /* predefined: 1 byte */

                                if(sscanf(token, "z=%d", &size) == 1) {
                                        /* Specified */
                                        if ((size != 0) && (size != 1) && (size != 2)) {
```

```
                                        fprintf(stderr, "ERROR -> invalid
sensitivity size z = %d\n", size);
                                        return 1;
                                }
                                parameters->jpwl_sens_size = size;

                        } else if (!strcmp(token, "a")) {
                                /* default */
                                parameters->jpwl_sens_size = size; /* 1 for
default size */

                        } else {
                                fprintf(stderr, "ERROR -> invalid size
selection = %s\n", token);
                                return 1;
                        };

                }

                /* search range method */
                if (*token == 'g') {

                        static int tile = 0, tilespec = 0, lasttileno = 0;

                        range = 0; /* predefined: 0 (packet) */

                        if(sscanf(token, "g=%d", &range) == 1) {
                                /* Specified */
                                if ((range < 0) || (range > 3)) {
                                        fprintf(stderr, "ERROR -> invalid
sensitivity range method g = %d\n", range);
                                        return 1;
                                }
                                parameters->jpwl_sens_range = range;

                        } else if (!strcmp(token, "g")) {
                                /* default */
                                parameters->jpwl_sens_range = range;

                        } else {
                                fprintf(stderr, "ERROR -> invalid range
selection = %s\n", token);
                                return 1;
                        };

                }

                /* next token or bust */
                token = strtok(NULL, ",");
        };


        /* some info */
```

```
                                fprintf(stdout, "Info: JPWL capabilities enabled\n");
                                parameters->jpwl_epc_on = true;

                        }
                        break;
#endif /* USE_JPWL */
/* <<UniPG */

                                /* ---------------------------------------------------- */

                        default:
                                fprintf(stderr, "ERROR -> Command line not valid\n");
                                return 1;
                }
        }

        /* check for possible errors */

        if(img_fol->set_imgdir == 1){
                if(!(parameters->infile[0] == 0)){
                        fprintf(stderr, "Error: options -ImgDir and -i cannot be used together !!\n");
                        return 1;
                }
                if(img_fol->set_out_format == 0){
                        fprintf(stderr, "Error: When -ImgDir is used, -OutFor <FORMAT> must be
used !!\n");
                        fprintf(stderr, "Only one format allowed! Valid formats are j2k and jp2!!\n");
                        return 1;
                }
                if(!((parameters->outfile[0] == 0))){
                        fprintf(stderr, "Error: options -ImgDir and -o cannot be used together !!\n");
                        fprintf(stderr, "Specify OutputFormat using -OutFor<FORMAT> !!\n");
                        return 1;
                }
        }else{
                if((parameters->infile[0] == 0) || (parameters->outfile[0] == 0)) {
                        fprintf(stderr, "Error: One of the options; -i or -ImgDir must be
specified\n");
                        fprintf(stderr, "Error: When using -i; -o must be used\n");
                        fprintf(stderr, "usage: image_compression -i image-file -o j2k/jp2-file (+
options)\n");
                        return 1;
                }
        }

        if (parameters->decod_format == RAW_DFMT && raw_cp->rawWidth == 0) {
                        fprintf(stderr,"\nError: invalid raw image parameters\n");
                        fprintf(stderr,"Please use the Format option -F:\n");
                        fprintf(stderr,"-F rawWidth,rawHeight,rawComp,rawBitDepth,s/u
(Signed/Unsigned)\n");
                                                fprintf(stderr,"Example: -i lena.raw -o lena.j2k -F
512,512,3,8,u\n");
                        fprintf(stderr,"Aborting\n");
```

```
                        return 1;
        }

        if ((parameters->cp_disto_alloc || parameters->cp_fixed_alloc || parameters-
>cp_fixed_quality)
                && (!(parameters->cp_disto_alloc ^ parameters->cp_fixed_alloc ^ parameters-
>cp_fixed_quality))) {
                fprintf(stderr, "Error: options -r -q and -f cannot be used together !!\n");
                return 1;
        }                                       /* mod fixed_quality */

        /* if no rate entered, lossless by default */
        if (parameters->tcp_numlayers == 0) {
                parameters->tcp_rates[0] = 0;           /* MOD antonin : losslessbug */
                parameters->tcp_numlayers++;
                parameters->cp_disto_alloc = 1;
        }

        if((parameters->cp_tx0 > parameters->image_offset_x0) || (parameters->cp_ty0 >
parameters->image_offset_y0)) {
                fprintf(stderr,
                        "Error: Tile offset dimension is unnappropriate -->
TX0(%d)<=IMG_X0(%d) TYO(%d)<=IMG_Y0(%d) \n",
                        parameters->cp_tx0, parameters->image_offset_x0, parameters->cp_ty0,
parameters->image_offset_y0);
                return 1;
        }

        for (i = 0; i < parameters->numpocs; i++) {
                if (parameters->POC[i].prg == -1) {
                        fprintf(stderr,
                                "Unrecognized progression order in option -P (POC n %d) [LRCP,
RLCP, RPCL, PCRL, CPRL] !!\n",
                                i + 1);
                }
        }

        return 0;
}

/* ---------------------------------------------------------------------- */

/**
sample error callback expecting a FILE* client object
*/
void error_callback(const char *msg, void *client_data) {
        FILE *stream = (FILE*)client_data;
        fprintf(stream, "[ERROR] %s", msg);
}
/**
sample warning callback expecting a FILE* client object
*/
void warning_callback(const char *msg, void *client_data) {
```

```
        FILE *stream = (FILE*)client_data;
        fprintf(stream, "[WARNING] %s", msg);
}
/**
sample debug callback expecting a FILE* client object
*/
void info_callback(const char *msg, void *client_data) {
        FILE *stream = (FILE*)client_data;
        fprintf(stream, "[INFO] %s", msg);
}

/* ---------------------------------------------------------------------- */

int main(int argc, char **argv) {
        bool bSuccess;
        opj_cparameters_t parameters;        /* compression parameters */
        img_fol_t img_fol;
        opj_event_mgr_t event_mgr;           /* event manager */
        opj_image_t *image = NULL;
        int i,num_images;
        int imageno;
        dircnt_t *dirptr;
        raw_cparameters_t raw_cp;
        opj_codestream_info_t cstr_info;             /* Codestream information structure */
        char indexfilename[OPJ_PATH_LEN];        /* index file name */

        /*
        configure the event callbacks (not required)
        setting of each callback is optionnal
        */
        memset(&event_mgr, 0, sizeof(opj_event_mgr_t));
        event_mgr.error_handler = error_callback;
        event_mgr.warning_handler = warning_callback;
        event_mgr.info_handler = info_callback;

        /* set encoding parameters to default values */
        opj_set_default_encoder_parameters(&parameters);

        /* Initialize indexfilename and img_fol */
        *indexfilename = 0;
        memset(&img_fol,0,sizeof(img_fol_t));

        /* parse input and get user encoding parameters */
        if(parse_cmdline_encoder(argc, argv, &parameters,&img_fol, &raw_cp, indexfilename)
== 1) {
                return 1;
        }

    }



        /* Read directory if necessary */
```

```
	if(img_fol.set_imgdir==1){
		num_images=get_num_images(img_fol.imgdirpath);
		dirptr=(dircnt_t*)malloc(sizeof(dircnt_t));
		if(dirptr){
			dirptr->filename_buf =
(char*)malloc(num_images*OPJ_PATH_LEN*sizeof(char));		// Stores at max 10 image file
names
			dirptr->filename = (char**) malloc(num_images*sizeof(char*));
			if(!dirptr->filename_buf){
				return 0;
			}
			for(i=0;i<num_images;i++){
				dirptr->filename[i] = dirptr->filename_buf + i*OPJ_PATH_LEN;
			}
		}
		if(load_images(dirptr,img_fol.imgdirpath)==1){
			return 0;
		}
		if (num_images==0){
			fprintf(stdout,"Folder is empty\n");
			return 0;
		}
	}else{
		num_images=1;
	}
	/*Encoding image one by one*/
	for(imageno=0;imageno<num_images;imageno++) {
		image = NULL;
		fprintf(stderr,"\n");

		if(img_fol.set_imgdir==1){
			if (get_next_file(imageno, dirptr,&img_fol, &parameters)) {
				fprintf(stderr,"skipping file...\n");
				continue;
			}
		}
		switch(parameters.decod_format) {
			case PGX_DFMT:
				break;
			case PXM_DFMT:
				break;
			case BMP_DFMT:
				break;
			case TIF_DFMT:
				break;
			case RAW_DFMT:
				break;
			case TGA_DFMT:
				break;
			default:
				fprintf(stderr,"skipping file...\n");
				continue;
		}
```

```
/* decode the source image */
/* ----------------------- */

switch (parameters.decod_format) {
	case PGX_DFMT:
		image = pgxtoimage(parameters.infile, &parameters);
		if (!image) {
			fprintf(stderr, "Unable to load pgx file\n");
			return 1;
		}
		break;

	case PXM_DFMT:
		image = pnmtoimage(parameters.infile, &parameters);
		if (!image) {
			fprintf(stderr, "Unable to load pnm file\n");
			return 1;
		}
		break;

	case BMP_DFMT:
		image = bmptoimage(parameters.infile, &parameters);
		if (!image) {
			fprintf(stderr, "Unable to load bmp file\n");
			return 1;
		}
		break;

	case TIF_DFMT:
		image = tiftoimage(parameters.infile, &parameters);
		if (!image) {
			fprintf(stderr, "Unable to load tiff file\n");
			return 1;
		}
	break;

	case RAW_DFMT:
		image = rawtoimage(parameters.infile, &parameters,
&raw_cp);
		if (!image) {
			fprintf(stderr, "Unable to load raw file\n");
			return 1;
		}
	break;

	case TGA_DFMT:
		image = tgatoimage(parameters.infile, &parameters);
		if (!image) {
			fprintf(stderr, "Unable to load tga file\n");
			return 1;
		}
	break;
```

```
	}

		/* encode the destination image */
		/* ---------------------------- */

		if (parameters.cod_format == RED) {/* J2K format output */
			int codestream_length;
			opj_cio_t *cio = NULL;
			FILE *f = NULL;

			/* get a J2K compressor handle */
			opj_cinfo_t* cinfo = opj_create_compress(CODEC_J2K);

			/* catch events using our callbacks and give a local context */
			opj_set_event_mgr((opj_common_ptr)cinfo, &event_mgr, stderr);

			/* setup the encoder parameters using the current image and user
parameters */
			opj_setup_encoder(cinfo, &parameters, image);

			/* open a byte stream for writing */
			/* allocate memory for all tiles */
			cio = opj_cio_open((opj_common_ptr)cinfo, NULL, 0);

			/* encode the image */
			if (*indexfilename)                                    // If need to
extract codestream information
				bSuccess = opj_encode_with_info(cinfo, cio, image,
&cstr_info);
			else
				bSuccess = opj_encode(cinfo, cio, image, NULL);
			if (!bSuccess) {
				opj_cio_close(cio);
				fprintf(stderr, "failed to encode image\n");
				return 1;
			}
			codestream_length = cio_tell(cio);

			/* write the buffer to disk */
			f = fopen(parameters.outfile, "wb");
			if (!f) {
				fprintf(stderr, "failed to open %s for writing\n",
parameters.outfile);
				return 1;
			}
			fwrite(cio->buffer, 1, codestream_length, f);
			fclose(f);

			fprintf(stderr,"Generated outfile %s\n",parameters.outfile);
			/* close and free the byte stream */
			opj_cio_close(cio);
```

```
                        /* Write the index to disk */
                        if (*indexfilename) {
                                bSuccess = write_index_file(&cstr_info, indexfilename);
                                if (bSuccess) {
                                        fprintf(stderr, "Failed to output index file into [%s]\n",
indexfilename);
                                }
                        }

                        /* free remaining compression structures */
                        opj_destroy_compress(cinfo);
                        if (*indexfilename)
                                opj_destroy_cstr_info(&cstr_info);
                } else {                        /* JP2 format output */
                        int codestream_length;
                        opj_cio_t *cio = NULL;
                        FILE *f = NULL;

                        /* get a JP2 compressor handle */
                        opj_cinfo_t* cinfo = opj_create_compress(CODEC_JP2);

                        /* catch events using our callbacks and give a local context */
                        opj_set_event_mgr((opj_common_ptr)cinfo, &event_mgr, stderr);


                        /* setup the encoder parameters using the current image and using
user parameters */
                        opj_setup_encoder(cinfo, &parameters, image);

                        /* open a byte stream for writing */
                        /* allocate memory for all tiles */
                        cio = opj_cio_open((opj_common_ptr)cinfo, NULL, 0);

                        /* encode the image */
                        if (*indexfilename)                                     // If need to
extract codestream information
                                bSuccess = opj_encode_with_info(cinfo, cio, image,
&cstr_info);
                        else
                                bSuccess = opj_encode(cinfo, cio, image, NULL);
                        if (!bSuccess) {
                                opj_cio_close(cio);
                                fprintf(stderr, "failed to encode image\n");
                                return 1;
                        }
                        codestream_length = cio_tell(cio);

                        /* write the buffer to disk */
                        f = fopen(parameters.outfile, "wb");
                        if (!f) {
                                fprintf(stderr, "failed to open %s for writing\n",
parameters.outfile);
                                return 1;
```

```
                        }
                        fwrite(cio->buffer, 1, codestream_length, f);
                        fclose(f);
                        fprintf(stderr,"Generated outfile %s\n",parameters.outfile);
                        /* close and free the byte stream */
                        opj_cio_close(cio);

                        /* Write the index to disk */
                        if (*indexfilename) {
                                bSuccess = write_index_file(&cstr_info, indexfilename);
                                if (bSuccess) {
                                        fprintf(stderr, "Failed to output index file\n");
                                }
                        }

                        /* free remaining compression structures */
                        opj_destroy_compress(cinfo);
                        if (*indexfilename)
                                opj_destroy_cstr_info(&cstr_info);
                }

                /* free image data */
                opj_image_destroy(image);
        }

        /* free user parameters structure */
  if(parameters.cp_comment) free(parameters.cp_comment);
        if(parameters.cp_matrice) free(parameters.cp_matrice);

        return 0;
}
```

Header files

1. openlib

```
#ifndef OPENLIB_H
#define OPENLIB_H


/*
==========================================================
  Compiler directives
==========================================================
*/

#if defined(OPJ_STATIC) || !(defined(WIN32) || defined(__WIN32__))
#define OPJ_API
#define OPJ_CALLCONV
#else
#define OPJ_CALLCONV __stdcall
/*
The following ifdef block is the standard way of creating macros which make exporting
from a DLL simpler. All files within this DLL are compiled with the OPJ_EXPORTS
symbol defined on the command line. this symbol should not be defined on any project
that uses this DLL. This way any other project whose source files include this file see
OPJ_API functions as being imported from a DLL, wheras this DLL sees symbols
defined with this macro as being exported.
*/
#ifdef OPJ_EXPORTS
#define OPJ_API __declspec(dllexport)
#else
#define OPJ_API __declspec(dllimport)
#endif /* OPJ_EXPORTS */
#endif /* !OPJ_STATIC || !WIN32 */

#ifndef __cplusplus
#if defined(HAVE_STDBOOL_H)
/*
The C language implementation does correctly provide the standard header
file "stdbool.h".
 */
#include <stdbool.h>
#else
/*
The C language implementation does not provide the standard header file
"stdbool.h" as required by ISO/IEC 9899:1999.  Try to compensate for this
braindamage below.
*/
#if !defined(bool)
#define          bool     int
#endif
#if !defined(true)
#define true     1
```

```
#endif
#if !defined(false)
#define	false	0
#endif
#endif
#endif /* __cplusplus */

/*
==========================================================
  Useful constant definitions
==========================================================
*/

#define OPJ_PATH_LEN 4096 /**< Maximum allowed size for filenames */

#define IMAGE_MAXRLVLS 33                                       /**< Number of
maximum resolution level authorized */
#define IMAGE_MAXBANDS (3*IMAGE_MAXRLVLS-2)     /**< Number of maximum
sub-band linked to number of resolution level */

/* UniPG>> */
#define JPWL_MAX_NO_TILESPECS		16 /**< Maximum number of tile parts
expected by JPWL: increase at your will */
#define JPWL_MAX_NO_PACKSPECS		16 /**< Maximum number of packet parts
expected by JPWL: increase at your will */
#define JPWL_MAX_NO_MARKERS		512 /**< Maximum number of JPWL
markers: increase at your will */
#define JPWL_PRIVATEINDEX_NAME "jpwl_index_privatefilename" /**< index file
name used when JPWL is on */
#define JPWL_EXPECTED_COMPONENTS 3 /**< Expect this number of components,
so you'll find better the first EPB */
#define JPWL_MAXIMUM_TILES 8192 /**< Expect this maximum number of tiles, to
avoid some crashes */
#define JPWL_MAXIMUM_HAMMING 2 /**< Expect this maximum number of bit errors
in marker id's */
#define JPWL_MAXIMUM_EPB_ROOM 65450 /**< Expect this maximum number of
bytes for composition of EPBs */
/* <<UniPG */

/*
==========================================================
  enum definitions
==========================================================
*/
/**
Rsiz Capabilities
*/
typedef enum RSIZ_CAPABILITIES {
	STD_RSIZ = 0,			/** Image profile*/
} OPJ_RSIZ_CAPABILITIES;

/**
Progression order
```

```
*/
typedef enum PROG_ORDER {
        PROG_UNKNOWN = -1,      /**< place-holder */
        LRCP = 0,               /**< layer-resolution-component-precinct order */
        RLCP = 1,               /**< resolution-layer-component-precinct order */
        RPCL = 2,               /**< resolution-precinct-component-layer order */
        PCRL = 3,               /**< precinct-component-resolution-layer order */
        CPRL = 4                /**< component-precinct-resolution-layer order */
} OPJ_PROG_ORDER;

/**
Supported image color spaces
*/
typedef enum COLOR_SPACE {
        CLRSPC_UNKNOWN = -1,   /**< place-holder */
        CLRSPC_SRGB = 1,       /**< sRGB */
        CLRSPC_GRAY = 2,       /**< grayscale */
        CLRSPC_SYCC = 3                 /**< YUV */
} OPJ_COLOR_SPACE;

/**
Supported codec
*/
typedef enum CODEC_FORMAT {
        CODEC_UNKNOWN = -1,    /**< place-holder */
        CODEC_J2K = 0,         /**< IMAGE codestream : read/write */
        CODEC_JPT = 1,         /**< JPT-stream (IMAGE, JPIP) : read only */
        CODEC_JP2 = 2          /**< IMAGE file format : read/write */
} OPJ_CODEC_FORMAT;

/**
Limit decoding to certain portions of the codestream.
*/
typedef enum LIMIT_DECODING {
        NO_LIMITATION = 0,                      /**< No limitation for the decoding.
The entire codestream will de decoded */
        LIMIT_TO_MAIN_HEADER = 1,               /**< The decoding is limited to the
Main Header */
        DECODE_ALL_BUT_PACKETS = 2 /**< Decode everything */
} OPJ_LIMIT_DECODING;

/*
==========================================================
  event manager typedef definitions
==========================================================
*/

/**
Callback function prototype for events
@param msg Event message
@param client_data
*/
typedef void (*opj_msg_callback) (const char *msg, void *client_data);
```

```
/**
Message handler object
used for
<ul>
<li>Error messages
<li>Warning messages
<li>Debugging messages
</ul>
*/
typedef struct opj_event_mgr {
        /** Error message callback if available, NULL otherwise */
        opj_msg_callback error_handler;
        /** Warning message callback if available, NULL otherwise */
        opj_msg_callback warning_handler;
        /** Debug message callback if available, NULL otherwise */
        opj_msg_callback info_handler;
} opj_event_mgr_t;


/*
==========================================================
  codec typedef definitions
==========================================================
*/

/**
Progression order changes
*/
typedef struct opj_poc {
        /** Resolution num start, Component num start, given by POC */
        int resno0, compno0;
        /** Layer num end,Resolution num end, Component num end, given by POC */
        int layno1, resno1, compno1;
        /** Layer num start,Precinct num start, Precinct num end */
        int layno0, precno0, precno1;
        /** Progression order enum*/
        OPJ_PROG_ORDER prg1,prg;
        /** Progression order string*/
        char progorder[5];
        /** Tile number */
        int tile;
        /** Start and end values for Tile width and height*/
        int tx0,tx1,ty0,ty1;
        /** Start value, initialised in pi_initialise_encode*/
        int layS, resS, compS, prcS;
        /** End value, initialised in pi_initialise_encode */
        int layE, resE, compE, prcE;
        /** Start and end values of Tile width and height, initialised in
pi_initialise_encode*/
        int txS,txE,tyS,tyE,dx,dy;
        /** Temporary values for Tile parts, initialised in pi_create_encode */
        int lay_t, res_t, comp_t, prc_t,tx0_t,ty0_t;
```

```
} opj_poc_t;

/**
Compression parameters
*/
typedef struct opj_cparameters {
	/** size of tile: tile_size_on = false (not in argument) or = true (in argument) */
	bool tile_size_on;
	/** XTOsiz */
	int cp_tx0;
	/** YTOsiz */
	int cp_ty0;
	/** XTsiz */
	int cp_tdx;
	/** YTsiz */
	int cp_tdy;
	/** allocation by rate/distortion */
	int cp_disto_alloc;
	/** allocation by fixed layer */
	int cp_fixed_alloc;
	/** add fixed_quality */
	int cp_fixed_quality;
	/** fixed layer */
	int *cp_matrice;
	/** comment for coding */
	char *cp_comment;
	/** csty : coding style */
	int csty;
	/** progression order (default LRCP) */
	OPJ_PROG_ORDER prog_order;
	/** progression order changes */
	opj_poc_t POC[32];
	/** number of progression order changes (POC), default to 0 */
	int numpocs;
	/** number of layers */
	int tcp_numlayers;
	/** rates of layers */
	float tcp_rates[100];
	/** different psnr for successive layers */
	float tcp_distoratio[100];
	/** number of resolutions */
	int numresolution;
	/** initial code block width, default to 64 */
	int cblockw_init;
	/** initial code block height, default to 64 */
	int cblockh_init;
	/** mode switch (cblk_style) */
	int mode;
	/** 1 : use the irreversible DWT 9-7, 0 : use lossless compression (default) */
	int irreversible;
	/** region of interest: affected component in [0..3], -1 means no ROI */
	int roi_compno;
	/** region of interest: upshift value */
```

```
        int roi_shift;
        /* number of precinct size specifications */
        int res_spec;
        /** initial precinct width */
        int prcw_init[IMAGE_MAXRLVLS];
        /** initial precinct height */
        int prch_init[IMAGE_MAXRLVLS];

        /**@name command line encoder parameters (not used inside the library) */
        /*@{*/
        /** input file name */
        char infile[OPJ_PATH_LEN];
        /** output file name */
        char outfile[OPJ_PATH_LEN];
        /** DEPRECATED. Index generation is now handeld with the
opj_encode_with_info() function. Set to NULL */
        int index_on;
        /** DEPRECATED. Index generation is now handeld with the
opj_encode_with_info() function. Set to NULL */
        char index[OPJ_PATH_LEN];
        /** subimage encoding: origin image offset in x direction */
        int image_offset_x0;
        /** subimage encoding: origin image offset in y direction */
        int image_offset_y0;
        /** subsampling value for dx */
        int subsampling_dx;
        /** subsampling value for dy */
        int subsampling_dy;
        /** input file format 0: PGX, 1: PxM, 2: BMP 3:TIF*/
        int decod_format;
        /** output file format 0: J2K, 1: JP2, 2: JPT */
        int cod_format;
        /*@}*/

/* UniPG>> */
        /**@name JPWL encoding parameters */
        /*@{*/
        /** enables writing of EPC in MH, thus activating JPWL */
        bool jpwl_epc_on;
        /** error protection method for MH (0,1,16,32,37-128) */
        int jpwl_hprot_MH;
        /** tile number of header protection specification (>=0) */
        int jpwl_hprot_TPH_tileno[JPWL_MAX_NO_TILESPECS];
        /** error protection methods for TPHs (0,1,16,32,37-128) */
        int jpwl_hprot_TPH[JPWL_MAX_NO_TILESPECS];
        /** tile number of packet protection specification (>=0) */
        int jpwl_pprot_tileno[JPWL_MAX_NO_PACKSPECS];
        /** packet number of packet protection specification (>=0) */
        int jpwl_pprot_packno[JPWL_MAX_NO_PACKSPECS];
        /** error protection methods for packets (0,1,16,32,37-128) */
        int jpwl_pprot[JPWL_MAX_NO_PACKSPECS];
        /** enables writing of ESD, (0=no/1/2 bytes) */
        int jpwl_sens_size;
```

```
        /** sensitivity addressing size (0=auto/2/4 bytes) */
        int jpwl_sens_addr;
        /** sensitivity range (0-3) */
        int jpwl_sens_range;
        /** sensitivity method for MH (-1=no,0-7) */
        int jpwl_sens_MH;
        /** tile number of sensitivity specification (>=0) */
        int jpwl_sens_TPH_tileno[JPWL_MAX_NO_TILESPECS];
        /** sensitivity methods for TPHs (-1=no,0-7) */
        int jpwl_sens_TPH[JPWL_MAX_NO_TILESPECS];
        /*@}*/
/* <<UniPG */

        /** Maximum rate for each component. If == 0, component size limitation is not
considered */
        int max_comp_size;
        /** Profile name*/
        OPJ_RSIZ_CAPABILITIES cp_rsiz;
        /** Tile part generation*/
        char tp_on;
        /** Flag for Tile part generation*/
        char tp_flag;
        /** MCT (multiple component transform) */
        char tcp_mct;
} opj_cparameters_t;

/**
Decompression parameters
*/
typedef struct opj_dparameters {
        /**
        Set the number of highest resolution levels to be discarded.
        The image resolution is effectively divided by 2 to the power of the number of
discarded levels.
        The reduce factor is limited by the smallest total number of decomposition levels
among tiles.
        if != 0, then original dimension divided by 2^(reduce);
        if == 0 or not used, image is decoded to the full resolution
        */
        int cp_reduce;
        /**
        Set the maximum number of quality layers to decode.
        If there are less quality layers than the specified number, all the quality layers are
decoded.
        if != 0, then only the first "layer" layers are decoded;
        if == 0 or not used, all the quality layers are decoded
        */
        int cp_layer;

        /**@name command line encoder parameters (not used inside the library) */
        /*@{*/
        /** input file name */
        char infile[OPJ_PATH_LEN];
```

```
        /** output file name */
        char outfile[OPJ_PATH_LEN];
        /** input file format 0: J2K, 1: JP2, 2: JPT */
        int decod_format;
        /** output file format 0: PGX, 1: PxM, 2: BMP */
        int cod_format;
        /*@}*/

/* UniPG>> */
        /**@name JPWL decoding parameters */
        /*@{*/
        /** activates the JPWL correction capabilities */
        bool jpwl_correct;
        /** expected number of components */
        int jpwl_exp_comps;
        /** maximum number of tiles */
        int jpwl_max_tiles;
        /*@}*/
/* <<UniPG */

        /**
        Specify whether the decoding should be done on the entire codestream, or be
limited to the main header
        Limiting the decoding to the main header makes it possible to extract the
characteristics of the codestream
        if == NO_LIMITATION, the entire codestream is decoded;
        if == LIMIT_TO_MAIN_HEADER, only the main header is decoded;
        */
        OPJ_LIMIT_DECODING cp_limit_decoding;

} opj_dparameters_t;

/** Common fields between IMAGE compression and decompression master structs. */

#define opj_common_fields \
        opj_event_mgr_t *event_mgr;          /**< pointer to the event manager */\
        void * client_data;                  /**< Available for use by application */\
        bool is_decompressor;                /**< So common code can tell which is which
*/\
        OPJ_CODEC_FORMAT codec_format;       /**< selected codec */\
        void *j2k_handle;                    /**< pointer to the IMAGE compression */\
        void *jp2_handle;                    /**< pointer to the JP2 codec */\
        void *mj2_handle                     /**< pointer to the MJ2 codec */

/* Routines that are to be used by both halves of the library are declared
 * to receive a pointer to this structure.  There are no actual instances of
 * opj_common_struct_t, only of opj_cinfo_t and opj_dinfo_t.
 */
typedef struct opj_common_struct {
  opj_common_fields;            /* Fields common to both master struct types */
  /* Additional fields follow in an actual opj_cinfo_t or
   * opj_dinfo_t.  All three structs must agree on these
   * initial fields!  (This would be a lot cleaner in C++.)
```

```
   */
} opj_common_struct_t;

typedef opj_common_struct_t * opj_common_ptr;

/**
Compression context info
*/
typedef struct opj_cinfo {
        /** Fields shared with opj_dinfo_t */
        opj_common_fields;
        /* other specific fields go here */
} opj_cinfo_t;

/**
Decompression context info
*/
typedef struct opj_dinfo {
        /** Fields shared with opj_cinfo_t */
        opj_common_fields;
        /* other specific fields go here */
} opj_dinfo_t;

/*
==========================================================
  I/O stream typedef definitions
==========================================================
*/

/*
 * Stream open flags.
 */
/** The stream was opened for reading. */
#define OPJ_STREAM_READ         0x0001
/** The stream was opened for writing. */
#define OPJ_STREAM_WRITE 0x0002

/**
Byte input-output stream (CIO)
*/
typedef struct opj_cio {
        /** codec context */
        opj_common_ptr cinfo;

        /** open mode (read/write) either OPJ_STREAM_READ or
OPJ_STREAM_WRITE */
        int openmode;
        /** pointer to the start of the buffer */
        unsigned char *buffer;
        /** buffer size in bytes */
        int length;

        /** pointer to the start of the stream */
```

```
        unsigned char *start;
        /** pointer to the end of the stream */
        unsigned char *end;
        /** pointer to the current position */
        unsigned char *bp;
} opj_cio_t;

/*
==========================================================
  image typedef definitions
==========================================================
*/

/**
Defines a single image component
*/
typedef struct opj_image_comp {
        /** XRsiz: horizontal separation of a sample of ith component with respect to the
reference grid */
        int dx;
        /** YRsiz: vertical separation of a sample of ith component with respect to the
reference grid */
        int dy;
        /** data width */
        int w;
        /** data height */
        int h;
        /** x component offset compared to the whole image */
        int x0;
        /** y component offset compared to the whole image */
        int y0;
        /** precision */
        int prec;
        /** image depth in bits */
        int bpp;
        /** signed (1) / unsigned (0) */
        int sgnd;
        /** number of decoded resolution */
        int resno_decoded;
        /** number of division by 2 of the out image compared to the original size of
image */
        int factor;
        /** image component data */
        int *data;
} opj_image_comp_t;

/**
Defines image data and characteristics
*/
typedef struct opj_image {
        /** XOsiz: horizontal offset from the origin of the reference grid to the left side of
the image area */
        int x0;
```

```
        /** YOsiz: vertical offset from the origin of the reference grid to the top side of the
image area */
        int y0;
        /** Xsiz: width of the reference grid */
        int x1;
        /** Ysiz: height of the reference grid */
        int y1;
        /** number of components in the image */
        int numcomps;
        /** color space: sRGB, Greyscale or YUV */
        OPJ_COLOR_SPACE color_space;
        /** image components */
        opj_image_comp_t *comps;
} opj_image_t;

/**
Component parameters structure used by the opj_image_create function
*/
typedef struct opj_image_comptparm {
        /** XRsiz: horizontal separation of a sample of ith component with respect to the
reference grid */
        int dx;
        /** YRsiz: vertical separation of a sample of ith component with respect to the
reference grid */
        int dy;
        /** data width */
        int w;
        /** data height */
        int h;
        /** x component offset compared to the whole image */
        int x0;
        /** y component offset compared to the whole image */
        int y0;
        /** precision */
        int prec;
        /** image depth in bits */
        int bpp;
        /** signed (1) / unsigned (0) */
        int sgnd;
} opj_image_cmptparm_t;

/*
==========================================================
  Information on the IMAGE codestream
==========================================================
*/

/**
Index structure : Information concerning a packet inside tile
*/
typedef struct opj_packet_info {
        /** packet start position (including SOP marker if it exists) */
        int start_pos;
```

```
        /** end of packet header position (including EPH marker if it exists)*/
        int end_ph_pos;
        /** packet end position */
        int end_pos;
        /** packet distorsion */
        double disto;
} opj_packet_info_t;

/**
Index structure : Information concerning tile-parts
*/
typedef struct opj_tp_info {
        /** start position of tile part */
        int tp_start_pos;
        /** end position of tile part header */
        int tp_end_header;
        /** end position of tile part */
        int tp_end_pos;
        /** start packet of tile part */
        int tp_start_pack;
        /** number of packets of tile part */
        int tp_numpacks;
} opj_tp_info_t;

/**
Index structure : information regarding tiles
*/
typedef struct opj_tile_info {
        /** value of thresh for each layer by tile cfr. Marcela   */
        double *thresh;
        /** number of tile */
        int tileno;
        /** start position */
        int start_pos;
        /** end position of the header */
        int end_header;
        /** end position */
        int end_pos;
        /** precinct number for each resolution level (width) */
        int pw[33];
        /** precinct number for each resolution level (height) */
        int ph[33];
        /** precinct size (in power of 2), in X for each resolution level */
        int pdx[33];
        /** precinct size (in power of 2), in Y for each resolution level */
        int pdy[33];
        /** information concerning packets inside tile */
        opj_packet_info_t *packet;
        /** add fixed_quality */
        int numpix;
        /** add fixed_quality */
        double distotile;
        /** number of tile parts */
```

```
        int num_tps;
        /** information concerning tile parts */
        opj_tp_info_t *tp;
} opj_tile_info_t;

/* UniPG>> */
/**
Marker structure
*/
typedef struct opj_marker_info_t {
        /** marker type */
        unsigned short int type;
        /** position in codestream */
        int pos;
        /** length, marker val included */
        int len;
} opj_marker_info_t;
/* <<UniPG */

/**
Index structure of the codestream
*/
typedef struct opj_codestream_info {
        /** maximum distortion reduction on the whole image (add for Marcela) */
        double D_max;
        /** packet number */
        int packno;
        /** writing the packet in the index with t2_encode_packets */
        int index_write;
        /** image width */
        int image_w;
        /** image height */
        int image_h;
        /** progression order */
        OPJ_PROG_ORDER prog;
        /** tile size in x */
        int tile_x;
        /** tile size in y */
        int tile_y;
        /** */
        int tile_Ox;
        /** */
        int tile_Oy;
        /** number of tiles in X */
        int tw;
        /** number of tiles in Y */
        int th;
        /** component numbers */
        int numcomps;
        /** number of layer */
        int numlayers;
        /** number of decomposition for each component */
        int *numdecompos;
```

```
/* UniPG>> */
        /** number of markers */
        int marknum;
        /** list of markers */
        opj_marker_info_t *marker;
        /** actual size of markers array */
        int maxmarknum;
/* <<UniPG */
        /** main header position */
        int main_head_start;
        /** main header position */
        int main_head_end;
        /** codestream's size */
        int codestream_size;
        /** information regarding tiles inside image */
        opj_tile_info_t *tile;
} opj_codestream_info_t;

#ifdef __cplusplus
extern "C" {
#endif


==========================================================
  image functions definitions
==========================================================
*/

/**
Create an image
@param numcmpts number of components
@param cmptparms components parameters
@param clrspc image color space
@return returns a new image structure if successful, returns NULL otherwise
*/
OPJ_API opj_image_t* OPJ_CALLCONV opj_image_create(int numcmpts,
opj_image_cmptparm_t *cmptparms, OPJ_COLOR_SPACE clrspc);

/**
Deallocate any resources associated with an image
@param image image to be destroyed
*/
OPJ_API void OPJ_CALLCONV opj_image_destroy(opj_image_t *image);

/*
==========================================================
  stream functions definitions
==========================================================
*/

/**
Open and allocate a memory stream for read / write.
On reading, the user must provide a buffer containing encoded data. The buffer will be
```

```
wrapped by the returned CIO handle.
On writing, buffer parameters must be set to 0: a buffer will be allocated by the library
to contain encoded data.
@param cinfo Codec context info
@param buffer Reading: buffer address. Writing: NULL
@param length Reading: buffer length. Writing: 0
@return Returns a CIO handle if successful, returns NULL otherwise
*/
OPJ_API opj_cio_t* OPJ_CALLCONV opj_cio_open(opj_common_ptr cinfo, unsigned
char *buffer, int length);

/**
Close and free a CIO handle
@param cio CIO handle to free
*/
OPJ_API void OPJ_CALLCONV opj_cio_close(opj_cio_t *cio);

/**
Get position in byte stream
@param cio CIO handle
@return Returns the position in bytes
*/
OPJ_API int OPJ_CALLCONV cio_tell(opj_cio_t *cio);
/**
Set position in byte stream
@param cio CIO handle
@param pos Position, in number of bytes, from the beginning of the stream
*/
OPJ_API void OPJ_CALLCONV cio_seek(opj_cio_t *cio, int pos);

/*
==========================================================
  event manager functions definitions
==========================================================
*/

OPJ_API opj_event_mgr_t* OPJ_CALLCONV opj_set_event_mgr(opj_common_ptr
cinfo, opj_event_mgr_t *event_mgr, void *context);

/*
==========================================================
  codec functions definitions
==========================================================
*/
/**
Creates a J2K/JPT/JP2 decompression structure
@param format Decoder to select
@return Returns a handle to a decompressor if successful, returns NULL otherwise
*/
OPJ_API opj_dinfo_t* OPJ_CALLCONV
opj_create_decompress(OPJ_CODEC_FORMAT format);
/**
Destroy a decompressor handle
```

```
@param dinfo decompressor handle to destroy
*/
OPJ_API void OPJ_CALLCONV opj_destroy_decompress(opj_dinfo_t *dinfo);
/**
Set decoding parameters to default values
@param parameters Decompression parameters
*/
OPJ_API void OPJ_CALLCONV
opj_set_default_decoder_parameters(opj_dparameters_t *parameters);
/**
Setup the decoder decoding parameters using user parameters.
Decoding parameters are returned in image->cp.
@param dinfo decompressor handle
@param parameters decompression parameters
*/
OPJ_API void OPJ_CALLCONV opj_setup_decoder(opj_dinfo_t *dinfo,
opj_dparameters_t *parameters);
/**
Decode an image from an IMAGE codestream
@param dinfo decompressor handle
@param cio Input buffer stream
@return Returns a decoded image if successful, returns NULL otherwise
*/
OPJ_API opj_image_t* OPJ_CALLCONV opj_decode(opj_dinfo_t *dinfo, opj_cio_t *cio);

/**
Decode an image from an IMAGE codestream and extract the codestream information
@param dinfo decompressor handle
@param cio Input buffer stream
@param cstr_info Codestream information structure if needed afterwards, NULL
otherwise
@return Returns a decoded image if successful, returns NULL otherwise
*/
OPJ_API opj_image_t* OPJ_CALLCONV opj_decode_with_info(opj_dinfo_t *dinfo,
opj_cio_t *cio, opj_codestream_info_t *cstr_info);
/**
Creates a J2K/JP2 compression structure
@param format Coder to select
@return Returns a handle to a compressor if successful, returns NULL otherwise
*/
OPJ_API opj_cinfo_t* OPJ_CALLCONV opj_create_compress(OPJ_CODEC_FORMAT
format);
/**
Destroy a compressor handle
@param cinfo compressor handle to destroy
*/
OPJ_API void OPJ_CALLCONV opj_destroy_compress(opj_cinfo_t *cinfo);
/**
Set encoding parameters to default values, that means :
<ul>
<li>Lossless
<li>1 tile
<li>Size of precinct : 2^15 x 2^15 (means 1 precinct)
```

Size of code-block : 64 x 64
Number of resolutions: 6
No SOP marker in the codestream
No EPH marker in the codestream
No sub-sampling in x or y direction
No mode switch activated
Progression order: LRCP
No index file
No ROI upshifted
No offset of the origin of the image
No offset of the origin of the tiles
Reversible DWT 5-3

@param parameters Compression parameters
*/
OPJ_API void OPJ_CALLCONV
opj_set_default_encoder_parameters(opj_cparameters_t *parameters);
/**
Setup the encoder parameters using the current image and using user parameters.
@param cinfo Compressor handle
@param parameters Compression parameters
@param image Input filled image
*/
OPJ_API void OPJ_CALLCONV opj_setup_encoder(opj_cinfo_t *cinfo,
opj_cparameters_t *parameters, opj_image_t *image);
/**
Encode an image into an IMAGE codestream
@param cinfo compressor handle
@param cio Output buffer stream
@param image Image to encode
@param index Depreacted -> Set to NULL. To extract index, used opj_encode_wci()
@return Returns true if successful, returns false otherwise
*/
OPJ_API bool OPJ_CALLCONV opj_encode(opj_cinfo_t *cinfo, opj_cio_t *cio,
opj_image_t *image, char *index);
/**
Encode an image into an IMAGE codestream and extract the codestream information
@param cinfo compressor handle
@param cio Output buffer stream
@param image Image to encode
@param cstr_info Codestream information structure if needed afterwards, NULL
otherwise
@return Returns true if successful, returns false otherwise
*/
OPJ_API bool OPJ_CALLCONV opj_encode_with_info(opj_cinfo_t *cinfo, opj_cio_t *cio,
opj_image_t *image, opj_codestream_info_t *cstr_info);
/**
Destroy Codestream information after compression or decompression
@param cstr_info Codestream information structure
*/
OPJ_API void OPJ_CALLCONV opj_destroy_cstr_info(opj_codestream_info_t
*cstr_info);

```
#ifdef __cplusplus
}
#endif

#endif /* OPENLIB_H */
```

2. file

```
#ifndef _FILE_H_
#define _FILE_H_

typedef struct option
{
        char *name;
        int has_arg;
        int *flag;
        int val;
}option_t;

#define         NO_ARG      0
#define REQ_ARG     1
#define OPT_ARG     2

extern int opterr;
extern int optind;
extern int optopt;
extern int optreset;
extern char *optarg;

extern int getopt(int nargc, char *const *nargv, const char *ostr);
extern int getopt_long(int argc, char * const argv[], const char *optstring,
                        const struct option *longopts, int totlen);


#endif                          /* _FILE_H_ */
```

3. imageconversion

```
#ifndef __IMAGE_IMAGECONVERSION_H
#define __IMAGE_ IMAGECONVERSION_H

/**@name RAW image encoding parameters */
/*@{*/
typedef struct raw_cparameters {
        /** width of the raw image */
        int rawWidth;
        /** height of the raw image */
```

```
        int rawHeight;
        /** components of the raw image */
        int rawComp;
        /** bit depth of the raw image */
        int rawBitDepth;
        /** signed/unsigned raw image */
        bool rawSigned;
        /*@}*/
} raw_cparameters_t;

/* TGA conversion */
opj_image_t* tgatoimage(const char *filename, opj_cparameters_t *parameters);
int imagetotga(opj_image_t * image, const char *outfile);

/* BMP conversion */
opj_image_t* bmptoimage(const char *filename, opj_cparameters_t *parameters);
int imagetobmp(opj_image_t *image, const char *outfile);

/* TIFF to image conversion*/
opj_image_t* tiftoimage(const char *filename, opj_cparameters_t *parameters);
int imagetotif(opj_image_t *image, const char *outfile);
/**
Load a single image component encoded in PGX file format
@param filename Name of the PGX file to load
@param parameters *List ?*
@return Returns a greyscale image if successful, returns NULL otherwise
*/
opj_image_t* pgxtoimage(const char *filename, opj_cparameters_t *parameters);
int imagetopgx(opj_image_t *image, const char *outfile);

opj_image_t* pnmtoimage(const char *filename, opj_cparameters_t *parameters);
int imagetopnm(opj_image_t *image, const char *outfile);

/* RAW conversion */
int imagetoraw(opj_image_t * image, const char *outfile);
opj_image_t* rawtoimage(const char *filename, opj_cparameters_t *parameters,
raw_cparameters_t *raw_cp);

#endif /* __IMAGE_ IMAGECONVERSION_H */
```

4. <u>dirent</u>

```
#ifndef DIRENT_H

#define DIRENT_H

#define DIRENT_H_INCLUDED


/* find out platform */

#if defined(MSDOS)                  /* MS-DOS */
```

```
#elif defined(__MSDOS__)                  /* Turbo C/Borland */
# define MSDOS
#elif defined(__DOS__)                    /* Watcom */
# define MSDOS
#endif

#if defined(WIN32)                        /* MS-Windows */
#elif defined(__NT__)                     /* Watcom */
# define WIN32
#elif defined(_WIN32)                     /* Microsoft */
# define WIN32
#elif defined(__WIN32__)                  /* Borland */
# define WIN32
#endif

/*
 * See what kind of dirent interface we have unless autoconf has already
 * determinated that.
 */
#if       !defined(HAVE_DIRENT_H)       &&       !defined(HAVE_DIRECT_H)       &&
!defined(HAVE_SYS_DIR_H)          &&          !defined(HAVE_NDIR_H)          &&
!defined(HAVE_SYS_NDIR_H) && !defined(HAVE_DIR_H)
# if defined(_MSC_VER)                    /* Microsoft C/C++ */
   /* no dirent.h */
# elif defined(__BORLANDC__)              /* Borland C/C++ */
#   define HAVE_DIRENT_H
#   define VOID_CLOSEDIR
# elif defined(__TURBOC__)                /* Borland Turbo C */
   /* no dirent.h */
# elif defined(__WATCOMC__)               /* Watcom C/C++ */
#   define HAVE_DIRECT_H
# elif defined(__apollo)                  /* Apollo */
#   define HAVE_SYS_DIR_H
# elif defined(__hpux)                    /* HP-UX */
#   define HAVE_DIRENT_H
```

```
# elif defined(__alpha) || defined(__alpha__)  /* Alpha OSF1 */
#   error "not implemented"
# elif defined(__sgi)                       /* Silicon Graphics */
#   define HAVE_DIRENT_H
# elif defined(sun) || defined(_sun)          /* Sun Solaris */
#   define HAVE_DIRENT_H
# elif defined(__FreeBSD__)                   /* FreeBSD */
#   define HAVE_DIRENT_H
# elif defined(__linux__)                    /* Linux */
#   define HAVE_DIRENT_H
# elif defined(__GNUC__)                     /* GNU C/C++ */
#   define HAVE_DIRENT_H
# else
#   error "not implemented"
# endif
#endif

/* include proper interface headers */
#if defined(HAVE_DIRENT_H)
# include <dirent.h>
# ifdef FREEBSD
#   define NAMLEN(dp) ((int)((dp)->d_namlen))
# else
#   define NAMLEN(dp) ((int)(strlen((dp)->d_name)))
# endif

#elif defined(HAVE_NDIR_H)
# include <ndir.h>
# define NAMLEN(dp) ((int)((dp)->d_namlen))

#elif defined(HAVE_SYS_NDIR_H)
# include <sys/ndir.h>
# define NAMLEN(dp) ((int)((dp)->d_namlen))

#elif defined(HAVE_DIRECT_H)
```

```
# include <direct.h>
# define NAMLEN(dp) ((int)((dp)->d_namlen))

#elif defined(HAVE_DIR_H)
# include <dir.h>
# define NAMLEN(dp) ((int)((dp)->d_namlen))

#elif defined(HAVE_SYS_DIR_H)
# include <sys/types.h>
# include <sys/dir.h>
# ifndef dirent
#   define dirent direct
# endif
# define NAMLEN(dp) ((int)((dp)->d_namlen))

#elif defined(MSDOS) || defined(WIN32)

  /* figure out type of underlaying directory interface to be used */
# if defined(WIN32)
#   define DIRENT_WIN32_INTERFACE
# elif defined(MSDOS)
#   define DIRENT_MSDOS_INTERFACE
# else
#   error "missing native dirent interface"
# endif

  /*** WIN32 specifics ***/
# if defined(DIRENT_WIN32_INTERFACE)
#   include <windows.h>
#   if !defined(DIRENT_MAXNAMLEN)
#     define DIRENT_MAXNAMLEN (MAX_PATH)
#   endif


  /*** MS-DOS specifics ***/
```

```
# elif defined(DIRENT_MSDOS_INTERFACE)
#   include <dos.h>

   /* Borland defines file length macros in dir.h */
#  if defined(__BORLANDC__)
#    include <dir.h>
#    if !defined(DIRENT_MAXNAMLEN)
#      define DIRENT_MAXNAMLEN ((MAXFILE)+(MAXEXT))
#    endif
#    if !defined(_find_t)
#      define _find_t find_t
#    endif

   /* Turbo C defines ffblk structure in dir.h */
#  elif defined(__TURBOC__)
#    include <dir.h>
#    if !defined(DIRENT_MAXNAMLEN)
#      define DIRENT_MAXNAMLEN ((MAXFILE)+(MAXEXT))
#    endif
#    define DIRENT_USE_FFBLK

   /* MSVC */
#  elif defined(_MSC_VER)
#    if !defined(DIRENT_MAXNAMLEN)
#      define DIRENT_MAXNAMLEN (12)
#    endif

   /* Watcom */
#  elif defined(__WATCOMC__)
#    if !defined(DIRENT_MAXNAMLEN)
#      if defined(__OS2__) || defined(__NT__)
#        define DIRENT_MAXNAMLEN (255)
#      else
#        define DIRENT_MAXNAMLEN (12)
#      endif
```

```
#     endif

#   endif
# endif

  /*** generic MS-DOS and MS-Windows stuff ***/
# if !defined(NAME_MAX) && defined(DIRENT_MAXNAMLEN)
#   define NAME_MAX DIRENT_MAXNAMLEN
# endif
# if NAME_MAX < DIRENT_MAXNAMLEN
#   error "assertion failed: NAME_MAX >= DIRENT_MAXNAMLEN"
# endif


  /*
   * Substitute for real dirent structure.  Note that `d_name' field is a
   * true character array although we have it copied in the implementation
   * dependent data.  We could save some memory if we had declared `d_name'
   * as a pointer refering the name within implementation dependent data.
   * We have not done that since some code may rely on sizeof(d_name) to be
   * something other than four.  Besides, directory entries are typically so
   * small that it takes virtually no time to copy them from place to place.
   */
  typedef struct dirent {
    char d_name[NAME_MAX + 1];

    /*** Operating system specific part ***/
# if defined(DIRENT_WIN32_INTERFACE)       /*WIN32*/
    WIN32_FIND_DATA data;
# elif defined(DIRENT_MSDOS_INTERFACE)     /*MSDOS*/
#   if defined(DIRENT_USE_FFBLK)
    struct ffblk data;
#   else
    struct _find_t data;
#   endif
```

```
# endif
  } dirent;

  /* DIR substitute structure containing directory name.  The name is
   * essential for the operation of ``rewinndir'' function. */
  typedef struct DIR {
    char          *dirname;                 /* directory being scanned */
    dirent        current;                  /* current entry */
    int           dirent_filled;            /* is current un-processed? */

  /*** Operating system specific part ***/
#  if defined(DIRENT_WIN32_INTERFACE)
    HANDLE        search_handle;
#  elif defined(DIRENT_MSDOS_INTERFACE)
#  endif
  } DIR;

# ifdef __cplusplus
extern "C" {
# endif

/* supply prototypes for dirent functions */
static DIR *opendir (const char *dirname);
static struct dirent *readdir (DIR *dirp);
static int closedir (DIR *dirp);
static void rewinddir (DIR *dirp);

/*
 * Implement dirent interface as static functions so that the user does not
 * need to change his project in any way to use dirent function.  With this
 * it is sufficient to include this very header from source modules using
 * dirent functions and the functions will be pulled in automatically.
 */
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
#include <assert.h>
#include <errno.h>

/* use ffblk instead of _find_t if requested */
#if defined(DIRENT_USE_FFBLK)
# define _A_ARCH   (FA_ARCH)
# define _A_HIDDEN (FA_HIDDEN)
# define _A_NORMAL (0)
# define _A_RDONLY (FA_RDONLY)
# define _A_SUBDIR (FA_DIREC)
# define _A_SYSTEM (FA_SYSTEM)
# define _A_VOLID  (FA_LABEL)
# define _dos_findnext(dest) findnext(dest)
# define _dos_findfirst(name,flags,dest) findfirst(name,dest,flags)
#endif

static int _initdir (DIR *p);
static const char *_getdirname (const struct dirent *dp);
static void _setdirname (struct DIR *dirp);

/*
 * <function name="opendir">
 * <intro>open directory stream for reading
 * <syntax>DIR *opendir (const char *dirname);
 *
 * <desc>Open named directory stream for read and return pointer to the
 * internal working area that is used for retrieving individual directory
 * entries.  The internal working area has no fields of your interest.
 *
 * <ret>Returns a pointer to the internal working area or NULL in case the
 * directory stream could not be opened.  Global `errno' variable will set
 * in case of error as follows:
 *
 * <table>
```

```
 * [EACESS  |Permission denied.
 * [EMFILE  |Too many open files used by the process.
 * [ENFILE  |Too many open files in system.
 * [ENOENT  |Directory does not exist.
 * [ENOMEM  |Insufficient memory.
 * [ENOTDIR |dirname does not refer to directory.  This value is not
 *          reliable on MS-DOS and MS-Windows platforms.  Many
 *          implementations return ENOENT even when the name refers to a
 *          file.]
 * </table>
 * </function>
 */
static DIR *opendir(const char *dirname)
{
  DIR *dirp;
  assert (dirname != NULL);

  dirp = (DIR*)malloc (sizeof (struct DIR));
  if (dirp != NULL) {
    char *p;

    /* allocate room for directory name */
    dirp->dirname = (char*) malloc (strlen (dirname) + 1 + strlen ("\\*.*"));
    if (dirp->dirname == NULL) {
      /* failed to duplicate directory name.  errno set by malloc() */
      free (dirp);
      return NULL;
    }
    /* Copy directory name while appending directory separator and "*.*".
     * Directory separator is not appended if the name already ends with
     * drive or directory separator.  Directory separator is assumed to be
     * '/' or '\' and drive separator is assumed to be ':'. */
    strcpy (dirp->dirname, dirname);
    p = strchr (dirp->dirname, '\0');
    if (dirp->dirname < p  &&
```

```
        *(p - 1) != '\\'  &&  *(p - 1) != '/'  &&  *(p - 1) != ':')
    {
      strcpy (p++, "\\");
    }
# ifdef DIRENT_WIN32_INTERFACE
    strcpy (p, "*"); /*scan files with and without extension in win32*/
# else
    strcpy (p, "*.*"); /*scan files with and without extension in DOS*/
# endif

    /* open stream */
    if (_initdir (dirp) == 0) {
      /* initialization failed */
      free (dirp->dirname);
      free (dirp);
      return NULL;
    }
  }
  return dirp;
}


/*
 * <function name="readdir">
 * <intro>read a directory entry
 * <syntax>struct dirent *readdir (DIR *dirp);
 *
 * <desc>Read individual directory entry and return pointer to a structure
 * containing the name of the entry.  Individual directory entries returned
 * include normal files, sub-directories, pseudo-directories "." and ".."
 * and also volume labels, hidden files and system files in MS-DOS and
 * MS-Windows.   You might want to use stat(2) function to determinate which
 * one are you dealing with.  Many dirent implementations already contain
 * equivalent information in dirent structure but you cannot depend on
 * this.
```

```
 *
 * The dirent structure contains several system dependent fields that
 * generally have no interest to you.  The only interesting one is char
 * d_name[] that is also portable across different systems.  The d_name
 * field contains the name of the directory entry without leading path.
 * While d_name is portable across different systems the actual storage
 * capacity of d_name varies from system to system and there is no portable
 * way to find out it at compile time as different systems define the
 * capacity of d_name with different macros and some systems do not define
 * capacity at all (besides actual declaration of the field). If you really
 * need to find out storage capacity of d_name then you might want to try
 * NAME_MAX macro. The NAME_MAX is defined in POSIX standard althought
 * there are many MS-DOS and MS-Windows implementations those do not define
 * it.  There are also systems that declare d_name as "char d_name[1]" and
 * then allocate suitable amount of memory at run-time.  Thanks to Alain
 * Decamps (Alain.Decamps@advalvas.be) for pointing it out to me.
 *
 * This all leads to the fact that it is difficult to allocate space
 * for the directory names when the very same program is being compiled on
 * number of operating systems.  Therefore I suggest that you always
 * allocate space for directory names dynamically.
 *
 * <ret>
 * Returns a pointer to a structure containing name of the directory entry
 * in `d_name' field or NULL if there was an error.  In case of an error the
 * global `errno' variable will set as follows:
 *
 * <table>
 * [EBADF  |dir parameter refers to an invalid directory stream.  This value
 *         is not set reliably on all implementations.]
 * </table>
 * </function>
 */
static struct dirent *
readdir (DIR *dirp)
```

```
{
  assert(dirp != NULL);
  if (dirp == NULL) {
    errno = EBADF;
    return NULL;
  }

#if defined(DIRENT_WIN32_INTERFACE)
  if (dirp->search_handle == INVALID_HANDLE_VALUE) {
    /* directory stream was opened/rewound incorrectly or it ended normally */
    errno = EBADF;
    return NULL;
  }
#endif

  if (dirp->dirent_filled != 0) {
    /*
     * Directory entry has already been retrieved and there is no need to
     * retrieve a new one.  Directory entry will be retrieved in advance
     * when the user calls readdir function for the first time.  This is so
     * because real dirent has separate functions for opening and reading
     * the stream whereas Win32 and DOS dirents open the stream
     * automatically when we retrieve the first file.  Therefore, we have to
     * save the first file when opening the stream and later we have to
     * return the saved entry when the user tries to read the first entry.
     */
    dirp->dirent_filled = 0;
  } else {
    /* fill in entry and return that */
#if defined(DIRENT_WIN32_INTERFACE)
    if (FindNextFile (dirp->search_handle, &dirp->current.data) == FALSE) {
      /* Last file has been processed or an error occured */
      FindClose (dirp->search_handle);
      dirp->search_handle = INVALID_HANDLE_VALUE;
      errno = ENOENT;
```

```
      return NULL;
    }

# elif defined(DIRENT_MSDOS_INTERFACE)
    if (_dos_findnext (&dirp->current.data) != 0) {
      /* _dos_findnext and findnext will set errno to ENOENT when no
       * more entries could be retrieved. */
      return NULL;
    }
# endif

    _setdirname (dirp);
    assert (dirp->dirent_filled == 0);
  }
  return &dirp->current;
}



/*
 * <function name="closedir">
 * <intro>close directory stream.
 * <syntax>int closedir (DIR *dirp);
 *
 * <desc>Close directory stream opened by the `opendir' function.  Close of
 * directory stream invalidates the DIR structure as well as previously read
 * dirent entry.
 *
 * <ret>The function typically returns 0 on success and -1 on failure but
 * the function may be declared to return void on same systems.  At least
 * Borland C/C++ and some UNIX implementations use void as a return type.
 * The dirent wrapper tries to define VOID_CLOSEDIR whenever closedir is
 * known to return nothing.  The very same definition is made by the GNU
 * autoconf if you happen to use it.
 *
 * The global `errno' variable will set to EBADF in case of error.
```

```
 * </function>
 */
static int
closedir (DIR *dirp)
{
  int retcode = 0;

  /* make sure that dirp points to legal structure */
  assert (dirp != NULL);
  if (dirp == NULL) {
    errno = EBADF;
    return -1;
  }

  /* free directory name and search handles */
  if (dirp->dirname != NULL) free (dirp->dirname);

#if defined(DIRENT_WIN32_INTERFACE)
  if (dirp->search_handle != INVALID_HANDLE_VALUE) {
    if (FindClose (dirp->search_handle) == FALSE) {
      /* Unknown error */
      retcode = -1;
      errno = EBADF;
    }
  }
#endif

  /* clear dirp structure to make sure that it cannot be used anymore*/
  memset (dirp, 0, sizeof (*dirp));
# if defined(DIRENT_WIN32_INTERFACE)
  dirp->search_handle = INVALID_HANDLE_VALUE;
# endif

  free (dirp);
  return retcode;
```

```
}


/*
 * <function name="rewinddir">
 * <intro>rewind directory stream to the beginning
 * <syntax>void rewinddir (DIR *dirp);
 *
 * <desc>Rewind directory stream to the beginning so that the next call of
 * readdir() returns the very first directory entry again.  However, note
 * that next call of readdir() may not return the same directory entry as it
 * did in first time.  The directory stream may have been affected by newly
 * created files.
 *
 * Almost every dirent implementation ensure that rewinddir will update
 * the directory stream to reflect any changes made to the directory entries
 * since the previous ``opendir'' or ``rewinddir'' call.  Keep an eye on
 * this if your program depends on the feature.  I know at least one dirent
 * implementation where you are required to close and re-open the stream to
 * see the changes.
 *
 * <ret>Returns nothing.  If something went wrong while rewinding, you will
 * notice it later when you try to retrieve the first directory entry.
 */
static void
rewinddir (DIR *dirp)
{
  /* make sure that dirp is legal */
  assert (dirp != NULL);
  if (dirp == NULL) {
    errno = EBADF;
    return;
  }
  assert (dirp->dirname != NULL);
```

```
  /* close previous stream */
#if defined(DIRENT_WIN32_INTERFACE)
  if (dirp->search_handle != INVALID_HANDLE_VALUE) {
    if (FindClose (dirp->search_handle) == FALSE) {
      /* Unknown error */
      errno = EBADF;
    }
  }
#endif

  /* re-open previous stream */
  if (_initdir (dirp) == 0) {
    /* initialization failed but we cannot deal with error.  User will notice
     * error later when she tries to retrieve first directory enty. */
    /*EMPTY*/;
  }
}



/*
 * Open native directory stream object and retrieve first file.
 * Be sure to close previous stream before opening new one.
 */
static int
_initdir (DIR *dirp)
{
  assert (dirp != NULL);
  assert (dirp->dirname != NULL);
  dirp->dirent_filled = 0;

# if defined(DIRENT_WIN32_INTERFACE)
  /* Open stream and retrieve first file */
  dirp->search_handle = FindFirstFile (dirp->dirname, &dirp->current.data);
  if (dirp->search_handle == INVALID_HANDLE_VALUE) {
    /* something went wrong but we don't know what.  GetLastError() could
```

```
     * give us more information about the error, but then we should map
     * the error code into errno. */
    errno = ENOENT;
    return 0;
  }

# elif defined(DIRENT_MSDOS_INTERFACE)
  if (_dos_findfirst (dirp->dirname,
          _A_SUBDIR | _A_RDONLY | _A_ARCH | _A_SYSTEM | _A_HIDDEN,
          &dirp->current.data) != 0)
  {
    /* _dos_findfirst and findfirst will set errno to ENOENT when no
     * more entries could be retrieved. */
    return 0;
  }
# endif

  /* initialize DIR and it's first entry */
  _setdirname (dirp);
  dirp->dirent_filled = 1;
  return 1;
}



/*
 * Return implementation dependent name of the current directory entry.
 */
static const char *
_getdirname (const struct dirent *dp)
{
#if defined(DIRENT_WIN32_INTERFACE)
  return dp->data.cFileName;

#elif defined(DIRENT_USE_FFBLK)
  return dp->data.ff_name;
```

```
#else
  return dp->data.name;
#endif
}


/*
 * Copy name of implementation dependent directory entry to the d_name field.
 */
static void
_setdirname (struct DIR *dirp) {
  /* make sure that d_name is long enough */
  assert (strlen (_getdirname (&dirp->current)) <= NAME_MAX);

  strncpy (dirp->current.d_name,
      _getdirname (&dirp->current),
      NAME_MAX);
  dirp->current.d_name[NAME_MAX] = '\0'; /*char d_name[NAME_MAX+1]*/
}

# ifdef __cplusplus
}
# endif
# define NAMLEN(dp) ((int)(strlen((dp)->d_name)))

#else
# error "missing dirent interface"
#endif


#endif /*DIRENT_H*/
```

5. index

```
#ifndef __IMAGE_INDEX_H
#define __IMAGE_INDEX_H

#ifdef __cplusplus
extern "C" {
#endif

/**
Write a structured index to a file
@param cstr_info Codestream information
@param index Index filename
@return Returns 0 if successful, returns 1 otherwise
*/
int write_index_file(opj_codestream_info_t *cstr_info, char *index);

#ifdef __cplusplus
}
#endif

#endif /* __IMAGE_INDEX_H */
```