University of Stuttgart
Germany

Institute of
Robust Power
Semiconductor Systems

Student Research Project

FPGA implementation of an energy-efficient real-time image compression algorithm for the EIVE satellite mission

Florian Wiewel

Supervisor:	M. Sc. Laura Manoliu
	Prof. Dr.-Ing. Ingmar Kallfass
Period:	05.10.2020 – 31.03.2021

Stuttgart, 31.03.2021

Postanschrift: Institut für Robuste Leistungshalbleitersysteme
Pfaffenwaldring 47

D-70569 Stuttgart

Tel.: +49 (0) 711 685 68700
Fax.: +49 (0) 711 685 58700
E-Mail: sekretariat@ilh.uni-stuttgart.de
Web: www.ilh.uni-stuttgart.de

Declaration

I hereby declare that this thesis is my own work and effort and follows the regulations related to good scientific practice of the University of Stuttgart in its latest form. All sources cited or quoted are indicated and acknowledged by means of a comprehensive list of references.

Stuttgart, 31.03.2021

Florian Wiewel

Executive Abstract

In this thesis three commonly used image compression algorithms are analyzed in terms of computational complexity, rate-distortion performance and execution time in order to find the most suitable basis for the implementation of an energy-efficient and real time image compression algorithm for the EIVE satellite mission. The selected algorithm is than modified to reduce its complexity while keeping its performance at a comparable level to the base algorithm. Afterwards the algorithm is implemented in the programmable logic part of a Xilinx Zynq UltraScale+ MPSoC device. Finally, the performance of the algorithm implemented on hardware is determined and compared to the performance of an implementation using a high-level scripting language and the performance of its base algorithm.

Zusammenfassung

In dieser Projektarbeit werden drei häufig verwendete Bildkompressionsalgorithmen in Bezug auf ihre Berechnungskomplexität, Rate-Distortion Leistung und Energieeffizienz untersucht, um einen guten Ausgangspunkt für die darauf aufbauende Implementierung eines Bildkompressionsalgorithmus für die EIVE Satellitienmission zu finden. Der ausgewählte Algoritmnus wird anschließend so angepasst, dass der Implementierungsaufwand reduziert aber die Leistungsfähigkeit auf einem vergleichbaren Niveau zum ursprünglichen Algorithmus gehalten wird. Anschließend wird der Algorithmus auf einem Xilinx Zynq UltraScale+ MPSoC Gerät implementiert und seine Leistungsfähigkeit bestimmt. Die auf der Hardware gemessenen Ergebnisse werden abschließend mit der Leistung einer Implementierung des Algorithmus in einer Skriptsprache und der Leistung des als Ausgangspunkt genutzten Algorithmus verglichen.

Contents

List of Figures

List of Tables

1. Introduction

The exponential growth in digital communications requires new communication systems that can transmit data with ever-increasing data rates and nearly global coverage. This trend led to the development of new satellite constellations like SpaceX' Starlink or OneWeb. These satellite constellations consist of a large number of small satellites operating at frequencies in the range of K_u- to K_a-Band. Since the frequency space in those bands is limited and already partially allocated, investigating new frequency ranges for the use of broadband satellite communication is obvious. Due to the maximum in the absorption of atmospheric gases caused by oxygen at around 60 GHz [10], the next higher frequency band of interest is the E-band ranging from 60 GHz to 90 GHz. The University of Stuttgart started the Exploratory In-orbit Verification of an E-band link (EIVE) project together with a consortium of commercial and academic partners. Within the scope of this project a CubeSat will be developed and launched for demonstrating broadband satellite communication in the frequency range of 71 GHz to 76 GHz [13].

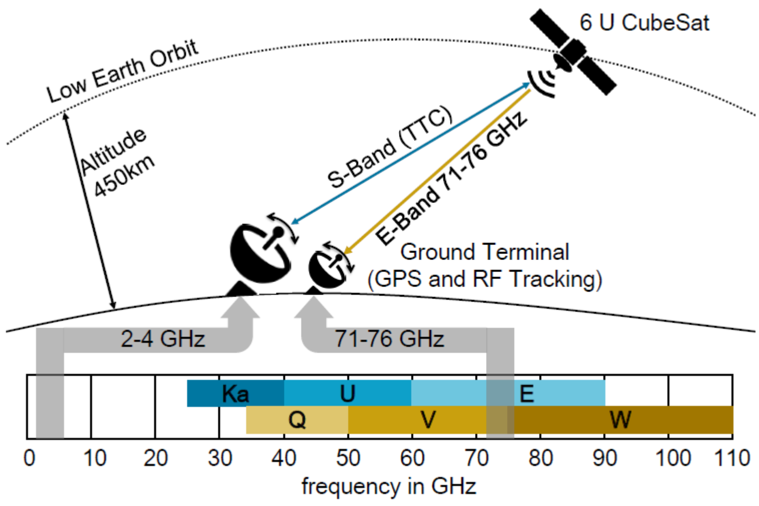


Figure 1.1.: EIVE - Mission concept and communication channels with frequency allocation regarding to wave guide bands and corresponding ITU bands [13]

The CubeSat will fly in a sun-synchronous low earth orbit as shown in the EIVE mission concept in figure 1.1. It will use the S-band frequency range for telemetry and telecommand and the E-band frequency range for broadband data downlink. Besides the primary objective, demonstrating broadband satellite communication, the EIVE mission will also provide earth observation capabilities. For this purpose a high-resolution camera will be included in the satellite bus as a payload.

The satellite bus including the payload components is shown in figure 1.2.

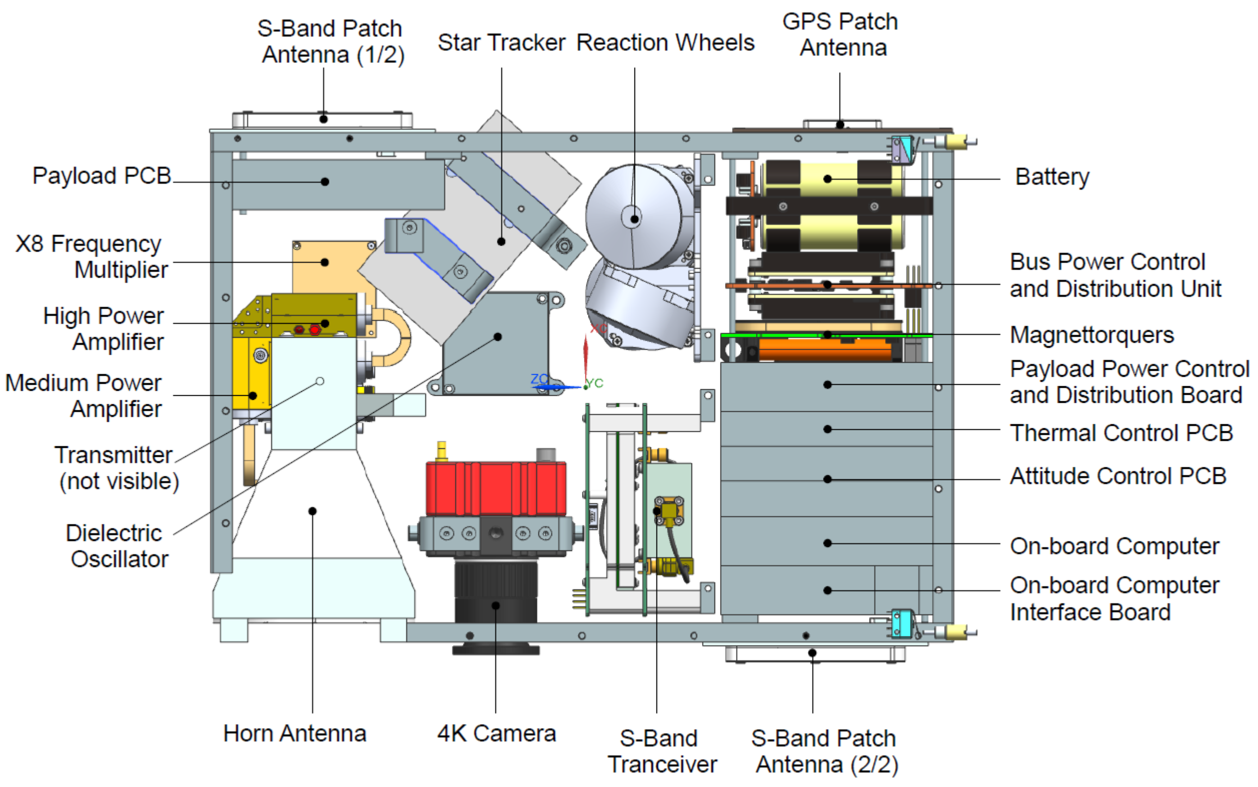


Figure 1.2.: EIVE - Satellite bus [13]

The camera will be used for streaming uncompressed live video data while the satellite is in contact with a ground station and also for taking images and storing them in memory for subsequent download while the satellite is not in contact with a ground station. Both live video data and stored images will be transmitted using the E-band link.

1.1. Motivation

Since images of natural scenes like trees on a field or the earth from space contain a lot of redundant and irrelevant information, storing them without removing this unnecessary information is very inefficient in terms of storage space. In some sections of an image neighboring pixels tend to have very similar values, because these clusters of pixels represent homogeneous regions of the depicted object. For example, in an image of the earth taken from space the clouds will be represented by mostly white pixels with only small differences in between them. These pixels are highly correlated and therefore contain redundancy. Because of the fact that in the end the purpose of these images is to be looked at by humans and the perception of the human eye is limited in certain aspects, there is also some irrelevant information contained in those images. The redundant and irrelevant information contained in the image can be removed for reducing the space needed to store it without impairing the quality too much. This process is called image compression. Since the resources available on the CubeSat platform are limited, compressing the stored images in the EIVE mission is reasonable.

1.2. Problem Formulation

There is no built-in image compressor inside the high-resolution camera nor the payload computer, which is connected to the camera and in charge of storing the images. Hence it is necessary to design and implement a custom image compressor inside the payload computer. There are several constraints to be taken into consideration while designing the image compressor, which are all related to the CubeSat platform:

- **Power consumption:** Most satellites use solar panels for their power supply. These solar panels can only have a limited size and this limits the maximum average power consumption of the satellite. The peak power consumption is limited by the maximum current provided by the satellites batteries. CubeSats are especially limited in their power consumption because their small size only allows for small solar panels and batteries. On top of that the E-band transmitter and the high-speed DAC with the corresponding serial transceivers inside the payload computer already use a lot of the power available on the CubeSat. Therefore the image compressor should use as little power as possible.
- **Storage capacity:** The capacity of the non-volatile storage device used by the payload computer will be 4 GByte. This storage device will be used for storing the images and also other mission specific data. This means the compression ratio achieved by the image compressor should be as high as possible.
- **Execution time:** Since earth observation is not the primary objective of the EIVE mission, the execution time for capturing and compressing an image should be as short as possible so that the primary objective is not affected. The CubeSat is moving relatively fast (approximately 84.4 minutes per revolution) and the terrestrial surface covered by an image is small (a few hundred square kilometers), therefore a shorter execution time equals better coverage.
- **Image quality:** In the end the stored images will be used for identifying objects on the surface of the earth. Recognizing different objects may require different levels of quality in the image. For example, a continent or tropical cyclone can be recognized with a much lower quality level than a city. As a result the image compressor should support different quality levels including lossless compression.

In this thesis the problem of finding and implementing an image compression algorithm for the EIVE mission, which is optimum in the sense of the above mentioned constraints, is discussed.

2. Theory and Background

2.1. System Overview

A communications system can be modeled as an information source, a modulator, a channel, a demodulator and an information sink. The data produced by the source is transmitted over the channel to the sink. Limitations imposed by the characteristics of the channel (e.g. noise, interference, availability, capacity, ...) lead to several problems. To overcome these problems the communications system needs to be supplemented with additional components: The source encoder removes redundant information from the data produced by the source. This ensures an efficient use of the channel in terms of capacity. The channel encoder introduces some redundant information again to allow the correction of potential errors caused by the channel. This error correction is performed by the channel decoder. Finally the source decoder reconstructs the original data. In this abstract model the channel can not only be an actual transmission line or wireless link for transmitting data through space. It can also be thought of as a storage medium for transmitting data through time. The basic model of the communications system for the downlink of images taken by the high-resolution camera within the scope of the EIVE satellite mission is depicted in figure 2.1.

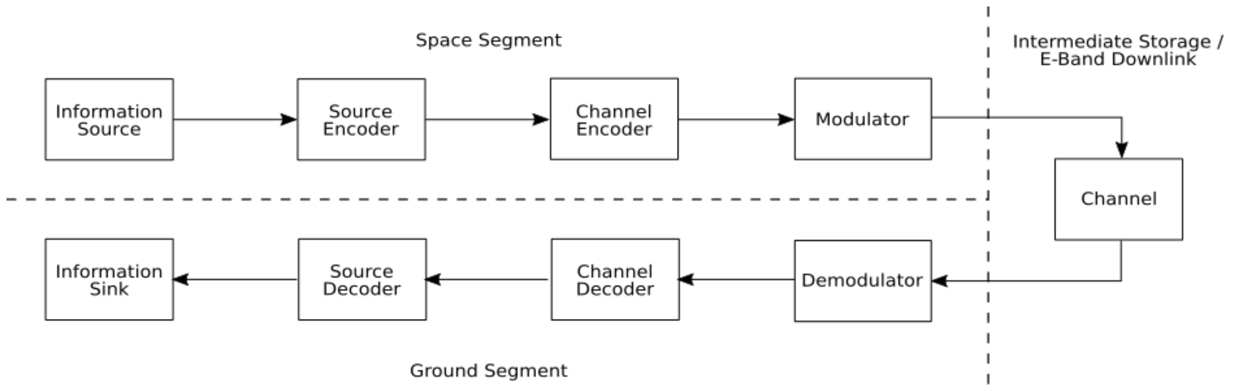


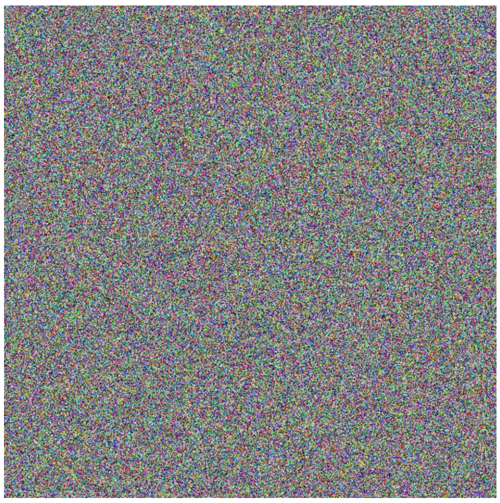
Figure 2.1.: Communications system - basic model for E-band downlink

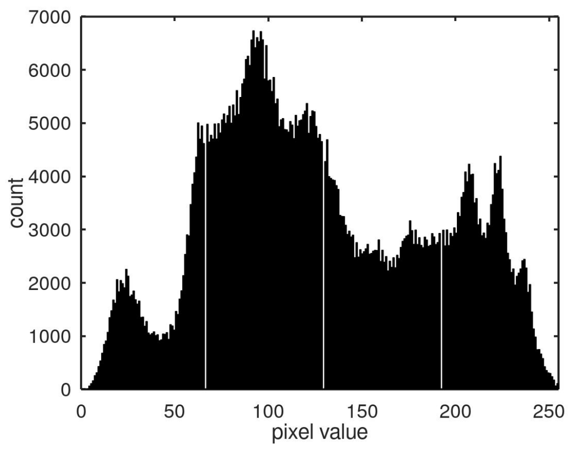
In this case the information source is represented by the high-resolution camera, the channel is a combination of intermediate storage and transmission over the E-band downlink and the information sink is the user in the ground station. This means that source encoder, channel encoder and modulator are part of the space segment whereas source decoder, channel decoder and demodulator are part of the ground segment. The focus of this thesis is on the encoding process performed by the source encoder in the space segment. The decoding process, which is performed by the source decoder in the ground segment, will not be discussed in this thesis. Note that this model includes intermediate storage of the images and therefore does not represent the transmission of live video data, which will not be discussed in this thesis.

2.2. Image Compression

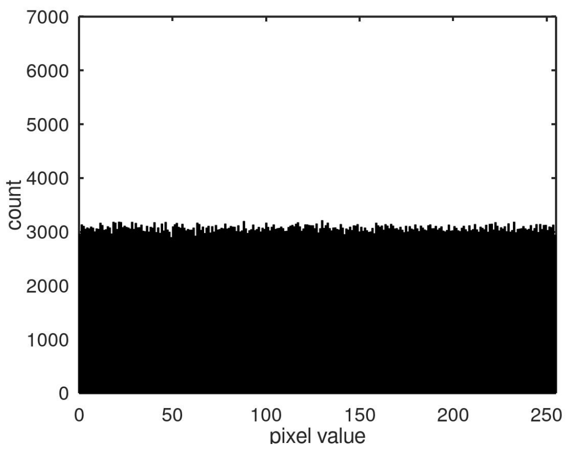
The process of removing redundant and irrelevant information from image data is called image compression. It is performed by the source encoder and exploits the structure in the image data and also the characteristics of the user. The final goal of image compression is to represent the original image by using fewer bits while keeping the subjective image quality at an acceptable level. In figure 2.2 the exploitation of the structure in the image data is illustrated.

(a) Natural image [12], size = 37.121 kBytes


(b) Random image, size 150.191 kBytes



(c) Natural image - histogram



(d) Random image - histogram

Figure 2.2.: Compressed images

Figure 2.2 (a) shows a natural image containing a lot of structure. In contrast a random image

with less structure is shown in figure 2.2 (b). Both images have the same size of 512 x 512 pixels, the same 24 bit-RGB color space and have been compressed using the same image compression algorithm (JPEG). However, the size of the compressed images differs significantly. The size of the random image is approximately 4 times the size of the natural image. This difference is caused by the absence of statistical structure in the random image as can be seen by comparing the histograms of both images in figure 2.2 (c) and (d).
The field of image compression includes a lot of methods for exploiting many different types of structure in the data to be compressed or characteristics of the user. Some image compression algorithms are more general and can be applied to a wide range of images while others are deigned only for the use with certain types of images (e.g. images containing text or lines). Many of these algorithms use a combination of different methods to achieve better compression performance. In this chapter only those methods used in the image compression algorithms, which will be considered in the decision process for selecting a suitable algorithm for the EIVE mission, will be discussed.

2.2.1. Lossless Compression

Methods for compression that remove redundant information from data are called lossless, because the reconstructed data is an exact copy of the original data with no loss of information.

2.2.1.1. Run-Length Encoding

Run-length encoding is a rather simple approach to data compression. The main idea is to search for runs of consecutive symbols with the same value in the data to be compressed and replacing them with the corresponding run-length and value. The following example demonstrates run-length encoding:
The data to be compressed is a sequence of integers

$$(12,12,12,12,12,12,12,12,12,35,76,112,67,87,87,87,5,5,5,5,5,5). \tag{2.1}$$

This sequence of integers has a length of 22 symbols and contains runs of consecutive symbols with the same value. The run-length encoded sequence is given by

$$(\underline{9},12,35,76,112,67,\underline{3},87,\underline{6},5), \tag{2.2}$$

where an underline indicates a run-length for the following symbol. The encoded sequence has a length of only 10 symbols. But additional information is needed to identify the run-length symbols. A simple way of including this additional information is to simply add an extra bit for each symbol indicating if the corresponding symbol contains a run-length or a value [6]. Using this approach the number of bits required to represent the original sequence is 154 bits, whereas the run-length encoded sequence can be represented with only 80 bits.

2.2.1.2. Entropy Encoding

In information theory a measure for the information contained in a symbol outputted by a source is called entropy. It represents the average information contained in one symbol. When binary symbols are used, the entropy is defined as

$$H(S) = -\sum_{n=1}^{N} P(s_n) log_2(P(s_n)), \tag{2.3}$$

where S is the set of symbols s_n outputted by the source, $N = |S|$ is the cardinality of the set S and $P(s_n)$ is the probability of the corresponding symbol [14]. For binary symbols the unit of the entropy is bits. Note that this definition only holds when the source outputs a sequence of independent and identically distributed (i.i.d.) symbols. As can be seen from equation 2.3 the optimum length of each symbol in bits is given by

$$l_{opt} = -log_2(P(s_n)). \tag{2.4}$$

The concept of entropy encoding is mapping each symbol s_n to a new symbol with this optimum length and thereby reducing the number of bits used to represent the symbol. In practice, the length of each symbol has to be an integer number of bits whereas the optimum length is a real number. Therefore the next higher integer number is being used in practice as an approximation to the optimum length. Furthermore, the probability mass function (PMF) $P(s_n)$ of the source is also unknown. A histogram is often used in practice to estimate the PMF.

Huffman Coding

A popular entropy coding algorithm is called Huffman coding. It produces a variable-length prefix code for encoding the source symbols based on an estimated PMF. The first step of the algorithm is to estimate the PMF. Based on the PMF each source symbol s_n is assigned a unique variable-length code word. The procedure of creating a binary Huffman tree is depicted in 2.3. It can be used to assign each source symbol a variable-length code word. $s_1, \dots, s_6$ are the source symbols with their corresponding probability of occurrence shown in the brackets. In each step of the procedure the two symbols with the lowest probability of occurrence are selected and each symbol is assigned a binary value (0 or 1), represented by the red values. The combined probability of occurrence for these two symbols is the sum of both individual probabilities and will be used in the next step. The steps repeat until the combined probability of the last two symbols is equal to 1. The variable-length code word can be read out from the tree by starting at its root and following the branches to a leaf. The binary values on this path represent the corresponding code word [11]. The code obtained for the Huffman tree shown in figure 2.3 is given in table 2.1. Assuming that the samples of the image shown in figure 2.2 (a) are i.i.d., using equation 2.3 its entropy is

$$H(S_{img}) = 7.7502\ bit. \tag{2.5}$$

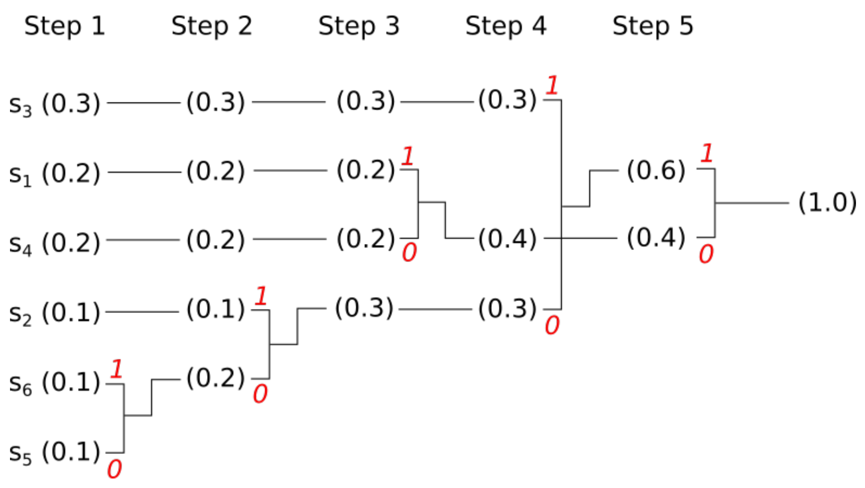


Figure 2.3.: Huffman coding procedure

Source Symbol	Code Word
s_1	01
s_2	101
s_3	11
s_4	00
s_5	1000
s_6	1001

Table 2.1.: Variable length code

This means that under this assumption $8\ bit - 7.7502 = 0.2498\ bit$ of every pixel in the image contains redundant information, which can be removed without loss of information. After applying Huffman coding on this image to remove the redundant information its entropy increases to

$$H(S_{img\_HFE}) = 7.9778\ bit. \tag{2.6}$$

As a result each pixel contains less redundancy and the number of bits required for storing the Huffman encoded image has been reduced.

Golomb / Rice Coding

Golomb codes follow the same idea of assigning long code words to source symbols with low probability of occurrence and short code words to source symbols with high probability of occurrence. But they are based on the assumption that the source samples with small values occur more often than source samples with large values. The set of source samples S is restricted to the natural numbers including zero. An example for such a distribution is given in figure 2.4. Note that the count is displayed on a logarithmic scale.

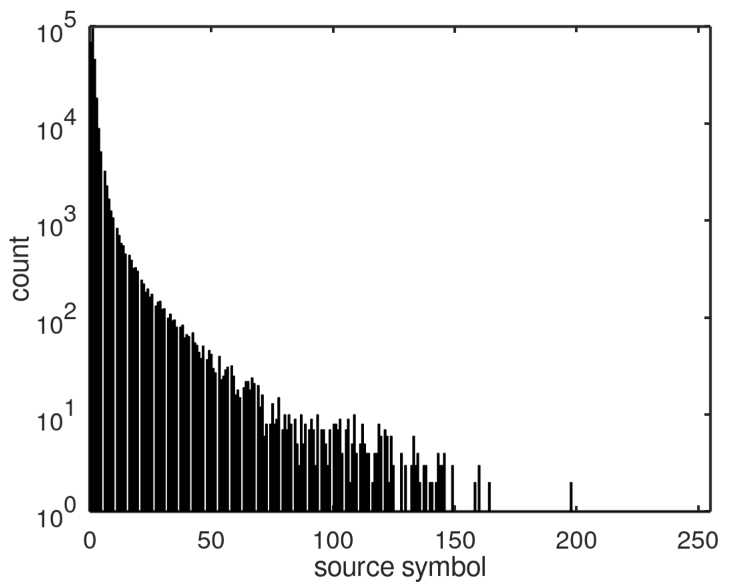


Figure 2.4.: Rice coding - example input sample distribution

A parameter K is used to divide the source sample s_n into two parts

$$q = \left\lfloor \frac{s_n}{K} \right\rfloor \tag{2.7}$$

and

$$r = s_n - qK. \tag{2.8}$$

The rounded quotient of the source sample is q and the remainder is r. The output code word is then constructed by taking the quotient q represented in unary representation followed by a zero used for separation and appending the remainder in binary representation [11]. An example for a Golomb code with parameter $K = 5$ is given in table 2.2.

Source Symbol	q	r	Code Word	Source Symbol	q	r	Code Word
0	0	0	000	8	1	3	10110
1	0	1	001	9	1	4	10111
2	0	2	010	10	2	0	11000
3	0	3	0110	11	2	1	11001
4	0	4	0111	12	2	2	11010
5	1	0	1000	13	2	3	110110
6	1	1	1001	14	2	4	110111
7	1	2	1010	15	3	0	111000

Table 2.2.: Golomb code - example [11]

Rice codes can be seen as a modification of Golomb codes, where the parameter K is a power of two. This is particular useful for the implementation of this coding algorithm on a computer or a FPGA, since division by two can be implemented as a simple shift operation. Furthermore, the set of source symbols is extended to the integer numbers. This leads to a minor modification of the coding procedure: in the first step, the sign is extracted from the source symbol and used as the most significant bit of the code word. Then, the Golomb coding procedure is executed on the absolute value of the source symbol [6].

2.2.1.3. Differential Encoding

Differential encoding is based on the idea of predicting a source symbol based on previous source symbols and only encoding the difference between predicted symbol and actual symbol. For the prediction step, the correlation between the symbols is being exploited and the difference step removes this correlation from the data. The simplest way of predicting a source symbol is taking the previous symbol s_{n-1}. This leads to the expression

$$d_n = s_n - s_{n-1} \tag{2.9}$$

for the difference value to be encoded. The difference d_n is then encoded using an entropy coding algorithm, e.g. Huffman coding or Rice Coding. There are many different methods available for prediction. Some of them are not only static but also adapting to the statistical characteristics of the source. The prediction and difference steps can be thought of as a preprocessing for the subsequent entropy coding, which reduces the correlation between the source symbols respectively redundancy.

2.2.1.4. Transform Coding

Another method of removing correlation from a data is to apply an invertible linear transform to it. If a suitable transform is selected, the transformed data is less correlated and is represented in form, where the model for representing the data contains more information than before. This means that less information is contained in the transformed data itself and thus it can be represented by fewer bits after encoding. Furthermore, the transform may be used as a preprocessing step in a lossy compression algorithm to allow a more targeted quantization. This raises the question what a suitable transform is. There exist many different transforms that are more or less suitable for a given application, which makes the process of selecting the correct transform a difficult task [6].

Two-Dimensional Discrete Cosine Transform

A popular transform used in image compression is the two-dimensional discrete cosine transform (DCT) of type 2. It is a linear invertible transform that uses cosine functions for generating a basis of an N-dimensional vector space. Type 2 describes one particular choice of selecting N equally spaced angles for generating the basis vectors using the cosine function. The transform is applied to a N x N block of input samples s_{xy} and is given by

$$G1_{x,j} = \sqrt{\frac{2}{N}} C_j \sum_{y=0}^{N-1} s_{xy} \, cos\left(\frac{(2y+1)j\pi}{2N}\right) \tag{2.10}$$

$$G_{ij} = \sqrt{\frac{2}{N}} C_i \sum_{x=0}^{N-1} G1_{x,j} \, cos\left(\frac{(2x+1)j\pi}{2N}\right), \tag{2.11}$$

where

$$C_x = \begin{cases} \frac{1}{\sqrt{2}}, & if \; x = 0 \; or \; x = N \\ 1, & otherwise \end{cases} \tag{2.12}$$

is the coefficient for handling the boundary conditions of the two dimensional data [6]. The two-dimensional DCT can be divided into two steps: In the first step a one-dimensional DCT is applied to all rows of the input sample block and in the second step a one-dimensional DCT is applied to all columns. As a result, the low frequency content of the input sample block is represented by the samples in the upper left and the high frequency content by the lower right samples of the output block. The basis images of the two-dimensional DCT are shown in figure 2.5.

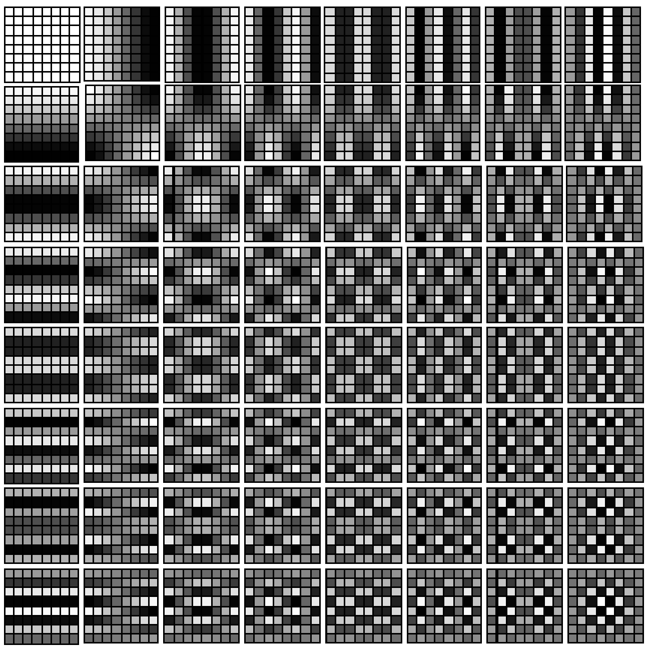


Figure 2.5.: 64 basis images of two-dimensional DCT [6]

The DCT can be interpreted as two rotations in N dimensions. The first rotation rotates each row of the input sample block and the second rotates all columns. The rotation results in a concentration of large coefficients in the upper left corner of the output sample block and small coefficients elsewhere. Figure 2.6 shows the output of the two-dimensional DCT in terms of coefficient magnitudes (L equals large, S and s equal small). Figure 2.6 (a) shows the magnitude of coefficients of the 8*x*8 input sample block, (b) the result after the first rotation and (c) the result after the second rotation.

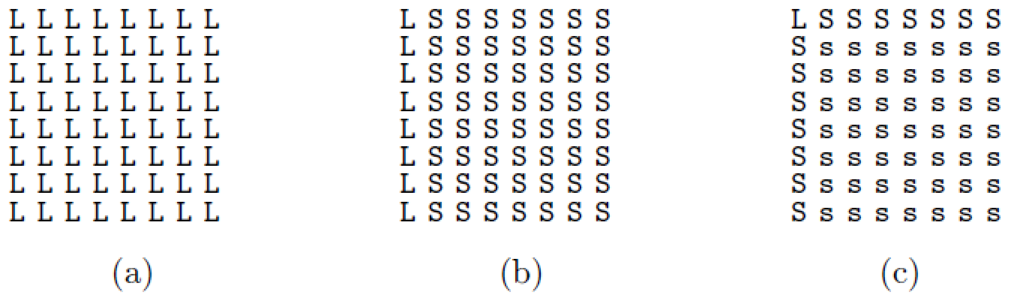


Figure 2.6.: DCT as two rotations [6]

2.2.1.5. Subband Coding

Many transform coding methods divide the data to be transformed in an arbitrary block structure, i.e. the DCT. This has the drawback of generating coding artifacts at the block edges. The problem becomes particularly visible when high compression ratios are targeted. Subband coding represents a different way of decomposing data into separate frequency components without applying a block structure. It is based on digital filters and allows for applying the most suitable encoding and/or quantization method to each frequency component separately [11].

Discrete Wavelet Transform

One popular transform used in subband coding is the discrete wavelet transform (DWT). The main idea behind the DWT

$$X_\psi(m,n) = \langle x(t), \psi_{m,n} \rangle = \int x(t)\overline{\psi_{m,n}}dt \tag{2.13}$$

is the inner product of the input sequence $x(t)$ with scaled and shifted versions of an analysis function, the so called wavelet $\psi_{m,n}$. The DWT uses wavelets of the following form

$$\psi_{m,n}(t) = a_0^{-\frac{m}{2}} \psi\left(\frac{t - nb_0 a_0^m}{a_0^m}\right) = a_0^{-\frac{m}{2}} \psi(a_0^{-m} t - nb_0). \tag{2.14}$$

In contrast to many other frequency analysis methods like the DCT or DFT, the DWT does not only output the frequency components of the input signal but also the temporal (or spatial) changes of the frequency components. This makes the DWT suitable for analyzing signals with non-stationary frequency components like images. In practice the scale factors are negative powers of two and the shift factors positive powers of two as shown in figure 2.7.

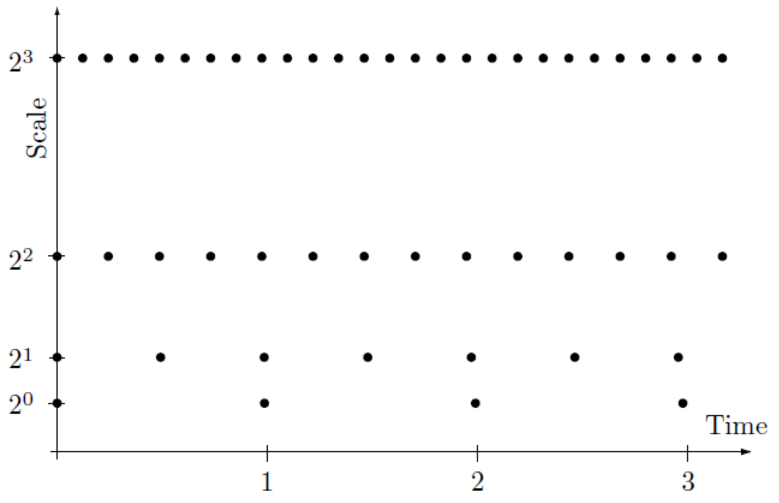


Figure 2.7.: DWT - relationship between scale factors and time used in practice [6]

The wavelets used form othonormal (or biorthogonal) wavelet bases and have compact support, what allows for the implementation of the DWT using pairs of special finite impulse response (FIR) filters. These pairs of special FIR filters are called quadrature mirror filters (QMF) and are characterized by the fact that one filter is a low-pass and the other one is a high-pass. The frequency normalized magnitude transfer functions of the QMF are mirrored around the frequency of $\frac{\pi}{2}$. An example for the frequency responses of QMFs is given in figure 2.8.

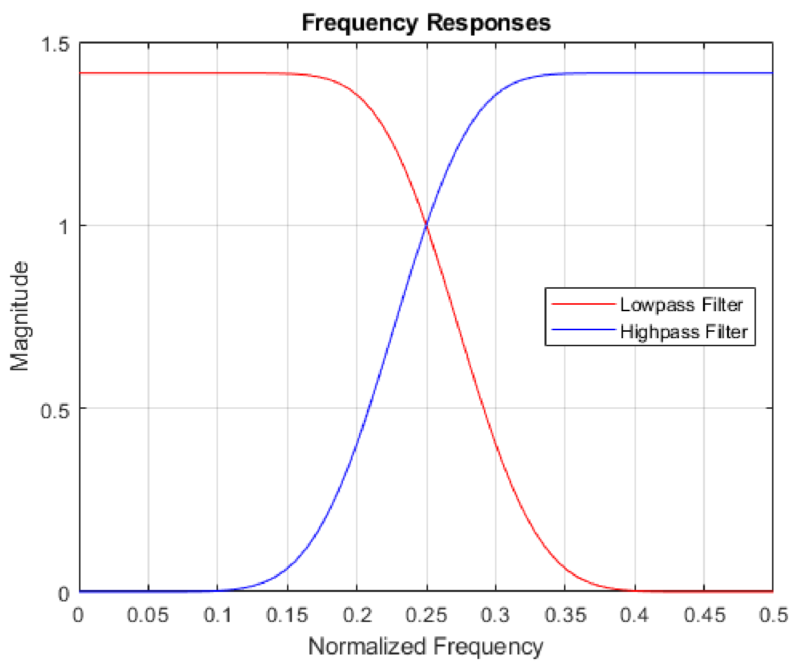


Figure 2.8.: Frequency response of QMFs [16]

In practice, a filter bank structure using QMF is commonly used to implement the DWT. An example for such a filter bank is shown in figure 2.9.

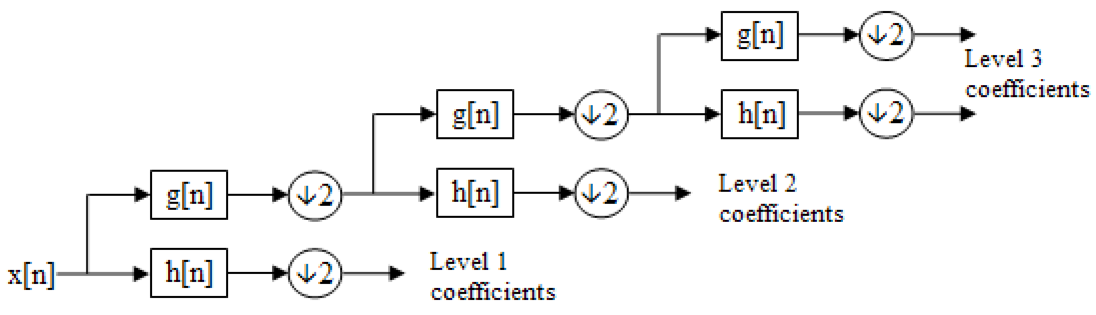


Figure 2.9.: 3 level DWT filter bank [1]

Note that the bandwidth of the output signal of each filter is reduced by a factor of two and therefore it is possible to downsample the output signal [11][6][2].

2.2.2. Lossy Compression

In contrast to lossless compression methods, lossy compression methods remove irrelevant information. The reconstructed data is only an approximation of the original data and some information is lost.

2.2.2.1. Scalar Quantization

The most simple method of quantization is scalar quantization. Assuming a sequence of continuous source symbols s_n the quantization is performed by mapping these symbols into intervals, which divide the set of all source source symbols $S = \mathbb{R}$. Each interval contains infinitely many source symbols and is represented by a unique code word. Figure shows an example of scalar 3 bit quantization of the real numbers. If the number of intervals is finite, the infinite precision source symbol can be represented by a finite-length code word. This process is also known as analog-to-digital conversion.

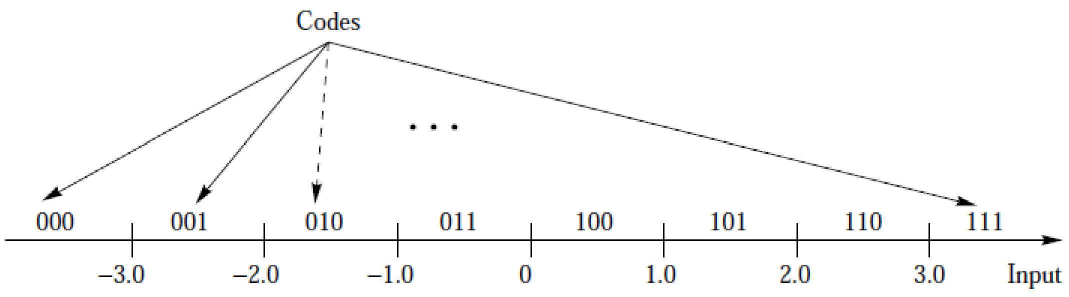
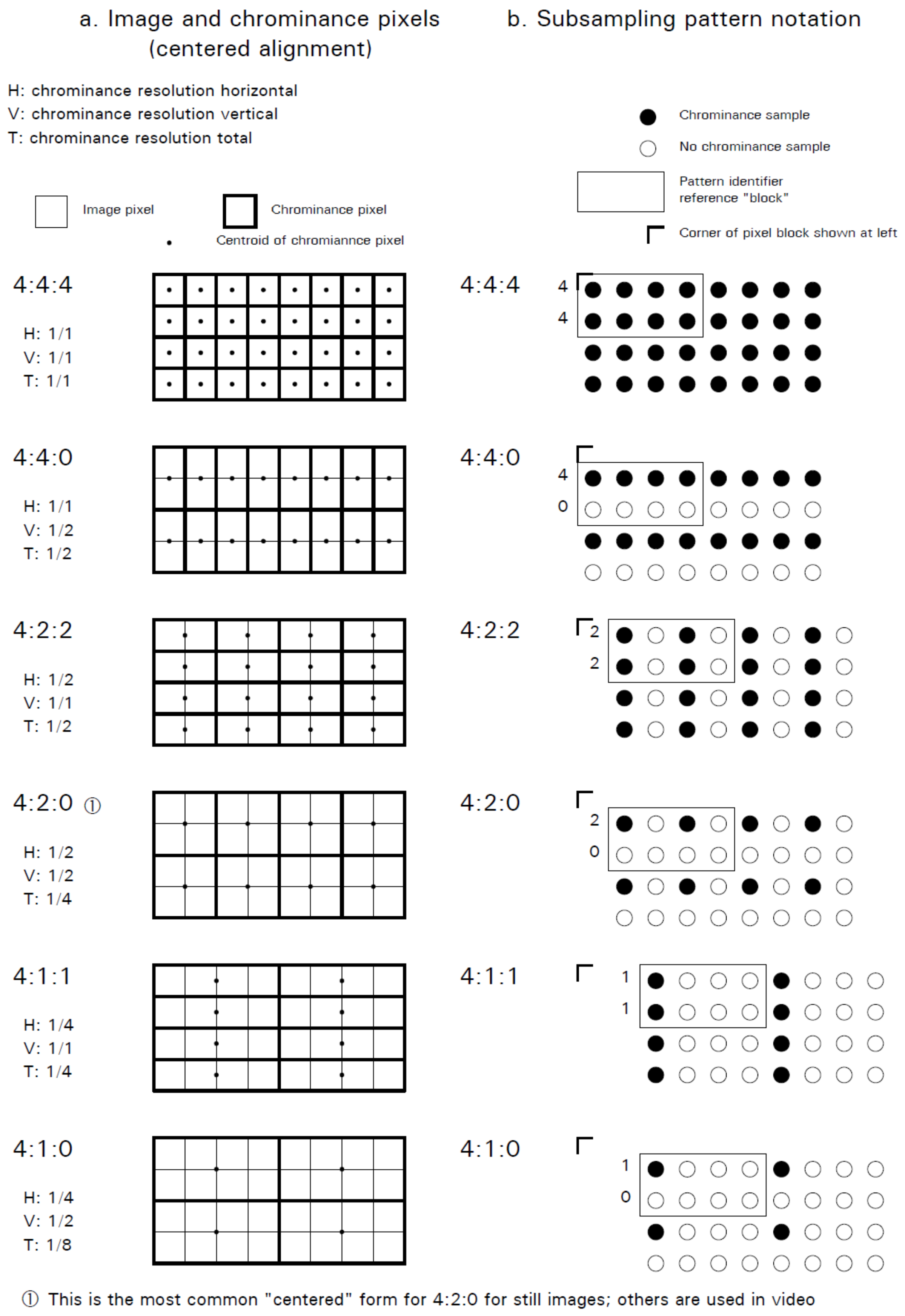


Figure 2.10.: 3 bit quantization of real numbers [11]

Scalar quantization can also be applied to a sequence of discrete source symbols. If suitable intervals are chosen, the output code word can be represented with less bits than the source symbols. Due to the fact that the mapping of multiple source symbols into the same interval is not uniquely invertible, information is lost during the process of quantization. The amount of information, that is lost during quantization, is determined by the interval size and needs to be chosen properly based on the characteristics and requirements of the user of the data. The quantization is called uniform, if the set of source symbols is divided into intervals with the same size.

2.2.2.2. Chrominance Subsampling

For digital images there exists a so called luma-chrominance color space, which is derived from the RGB color space by applying a nonlinear transform to it. In this color space the color of a pixel is described by two components: a Y component, which describes the luminance of the pixel and a two-dimensional component C_r and C_b, which describes the chrominance of the pixel.
The human eye is more sensitive to differences in the luminance than to differences in the chrominance components. This fact allows the use of subsampling on the chrominance components while keeping the subjective quality of the image at an acceptable level. Due to the subsampling, not all of the chrominance values are used but only every second or fourth depending on the pattern of subsampling being used. Figure 2.11 shows different patterns of chrominance subsampling. Depending on the subsampling pattern the relative total resolution of the chrominance component can be as low as $\frac{1}{8}$ for the 4:1:0 pattern. Since the chrominance component is represented by $\frac{2}{3}$ of the bits of a pixel, this means that only $\frac{1}{3} + \frac{1}{8} \cdot \frac{2}{3} = \frac{5}{12}$ of the original number of bits are required to represent a pixel.



a. Image and chrominance pixels (centered alignment)
b. Subsampling pattern notation
H: chrominance resolution horizontal
V: chrominance resolution vertical
T: chrominance resolution total
Image pixel
Chrominance pixel
Centroid of chromiannce pixel
Chrominance sample
No chrominance sample
Pattern identifier reference "block"
Corner of pixel block shown at left
4:4:4 H: 1/1 V: 1/1 T: 1/1
4:4:0 H: 1/1 V: 1/2 T: 1/2
4:2:2 H: 1/2 V: 1/1 T: 1/2
4:2:0 ① H: 1/2 V: 1/2 T: 1/4
4:1:1 H: 1/4 V: 1/1 T: 1/4
4:1:0 H: 1/4 V: 1/2 T: 1/8
① This is the most common "centered" form for 4:2:0 for still images; others are used in video

Figure 2.11.: Chrominance subsampling patterns [7]

2.2.3. Performance Metrics

In order to quantify the effectiveness of an image compressor several different metrics can be used:

- **Bit rate:** A metric for quantifying the compressed data volume is the *bit rate*. It is measured in $\frac{bit}{pixel}$ and is defined as the number of bits used in the compressed representation of the image divided by the number of pixels in the image. When compared to the number of bits used to represent a pixel in the original image, the achieved compression becomes obvious.

When lossy compression is used and the original image is available, quality metrics can be used to quantify the distortions introduced by the image compressor. Some commonly used metrics include:

- **Mean Squared Error (MSE):**

$$MSE = \frac{1}{w \cdot h} \sum_i \sum_j (x_{i,j} - \hat{x}_{i,j})^2 \tag{2.15}$$

- **Peak Signal to Noise Ration (PSNR):**

$$PSNR = 20 \, log_{10} \left(\frac{2^B - 1}{\sqrt{MSE}} \right) \; [dB] \tag{2.16}$$

- **Maximum Absolute Error (MAE):**

$$MAE = \max_{i,j} |(x_{i,j} - \hat{x}_{i,j})| \tag{2.17}$$

Here w and h denote the width and height of the image, $x_{i,j}$ the pixel in the i^{th} row and j^{th} column of the original image, $\hat{x}_{i,j}$ the pixel in the i^{th} row and j^{th} column of the compressed image and B the dynamic range of the original image in bits [3].
If the original image is not available, so called no-reference image quality metrics have to be used:

- **Perception based image quality Evaluator (PIQE):** This quality metric is based on the extraction of local features of the image. These features are then used to calculate a quality score, which is derived from applying the PIQE on the image dataset contained in the LIVE Image Quality Assessment Database (Release 2). For more details see [17] and [8].

Quality Scale	Score Range
Excellent	[0, 20]
Good	[21, 35]
Fair	[36, 50]
Poor	[51, 80]
Bad	[81, 100]

Table 2.3.: PIQE - Score ranges [15]

2.3. Image Compression Algorithms

A large variety of image compression algorithms is available for many different kinds of applications. In this section three image compression algorithms will be presented. These algorithms haven been selected based on their simplicity and fields of application. They represent a starting point for the development of an image compression algorithm specific for the EIVE mission.

2.3.1. CCITT T.81 (JPEG)

One of the most commonly used image compression algorithms is defined by the CCITT Recommendation T.81. It has been developed by the CCITT Study Group VIII and the Joint Photographic Experts Group (JPEG). The abbreviation JPEG also refers this algorithm. This algorithm is designed for the sequential progressive encoding of continuous tone grayscale and color images. Although it supports lossless and lossy compression, only the lossy DCT-based coding process will be presented here.

Figure 2.12 depicts the block diagram of the DCT-based encoder.

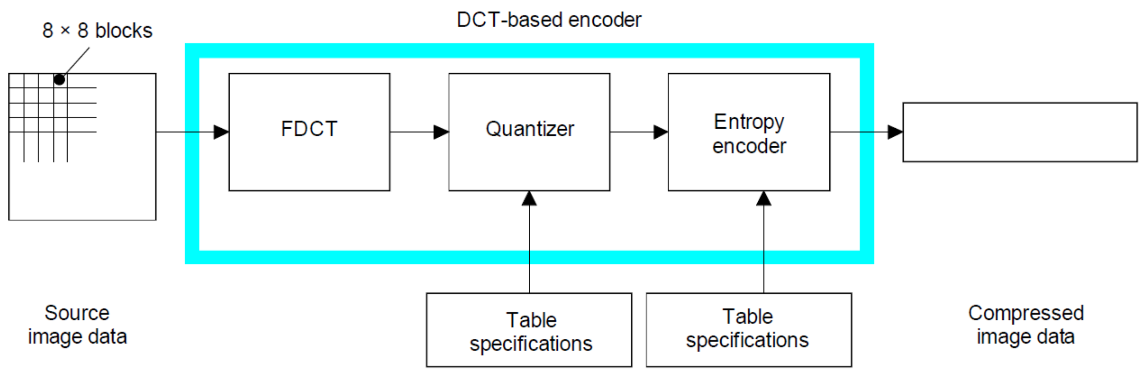


Figure 2.12.: JPEG - DCT-based encoder [9]

The input to the encoder is the source image data. In case of a color image each color component is processed individually by the same encoding process. The source image data is grouped into non-overlapping blocks with a size of 8 x 8 pixels. Each pixel is represented by an integer number with p bits.

The sample of the input blocks are then level shifted to a signed representation and sequentially feed into the forward DCT (FDCT) component, which performs a FDCT on each block. The output of the FDCT is a set of 64 DCT coefficients. One of these coefficients represents the DC component of the input block and is called DC coefficient whereas the remaining 63 coefficients are called the AC coefficients.

Each of the 64 coefficients is then quantized based on a corresponding value specified in the table specifications. The uniform scalar quantizer is defined as

$$sq_{vu} = round\left(\frac{s_{vu}}{Q_{vu}}\right), \tag{2.18}$$

where s_{vu} denotes the input sample of the quantizer and Q_{vu} the corresponding value defined in the quantization table.
After quantization the 64 coefficients are prepared for entropy encoding. The DC coefficient is differentially encoded and the AC coefficients are rearranged in a zig-zag sequence. The preparation process is shown in figure 2.13.

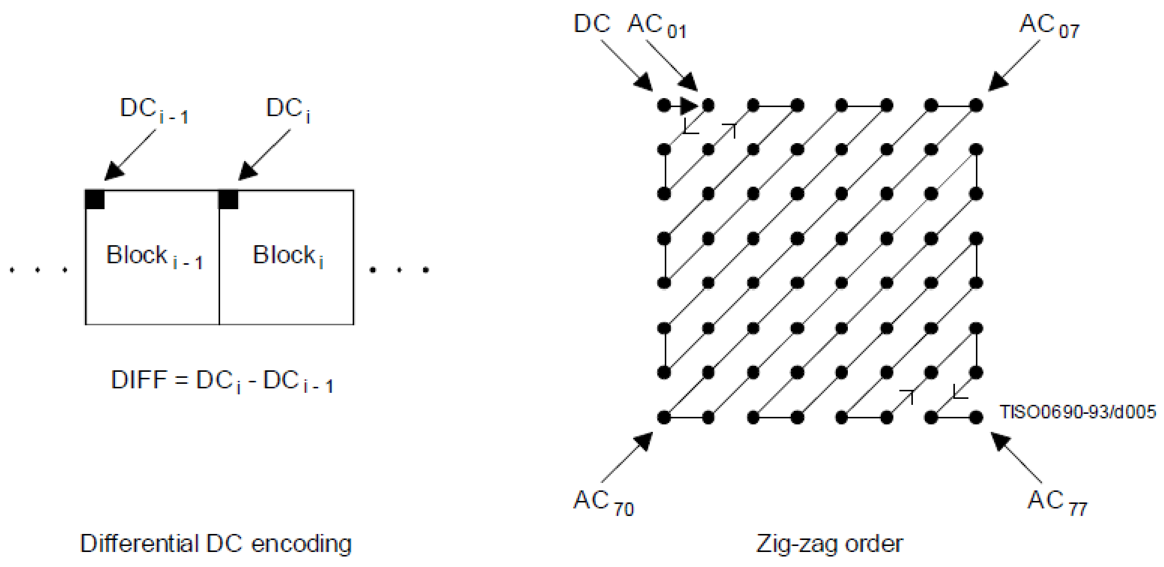


Figure 2.13.: JPEG - Preparation for entropy encoding [9]

For entropy encoding one of two methods can be used: a combination of run-length coding and Huffman coding or arithmetic coding. Arithmetic coding is more complex to implement than Huffman coding and only provides 5% to 10% better compression than Huffman coding. Therefore, arithmetic coding is rarely used in practice and will not be considered here. Typically only a few of the quantized DCT coefficients are non-zero with runs of zero in between them. The differentially encoded DC coefficient is directly Huffman encoded based on a Huffman coding table. The encoder then proceeds to the first non-zero AC coefficient in the zig-zag sequence. The number of preceding zeros and the AC coefficient are than Huffman encoded based on second Huffman code table.

2.3.2. CCSDS 122.0-B-2

The Consultative Committee for Space Data Systems (CCSDS) has defined an image compression standard that is based on the JPEG2000 standard and can be used for lossy and lossless compression. It is called CCSDS 122.0-B-2 and differs from JPEG2000 in the following aspects:

- specifically designed for use with high-rate instruments on board of spacecraft
- trade-off between compression performance and complexity with particular emphasis on spacecraft applications
- lower complexity allowing fast and low-power hardware implementation
- limited set of options, supporting its successful application without in-depth algorithm knowledge

The compressor consists of two main blocks: a DWT and a bit-plane encoder. The block diagram of the compressor depicted in figure 2.14. The image (input data) is fed into the DWT block,

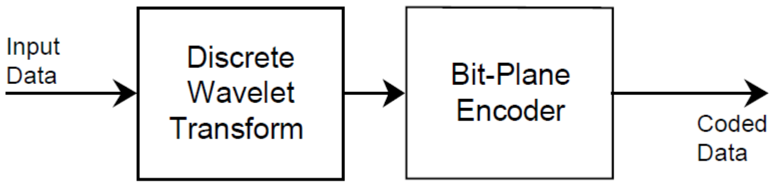


Figure 2.14.: CCSDS 122.0-B-2 block diagram [4]

which decorrelates the data. The CSSDS 122.0-B-2 standard offers two different DWTs: the 9/7 float DWT and the 9/7 integer DWT. The 9/7 float DWT uses a FIR filter bank implementation with 9 respectively 7 floating point coefficients for the high-pass and low-pass filter. The 9/7 integer DWT is non-linear approximation to the 9/7 float DWT, which uses rounding operations to ensure that only integer values are outputted.

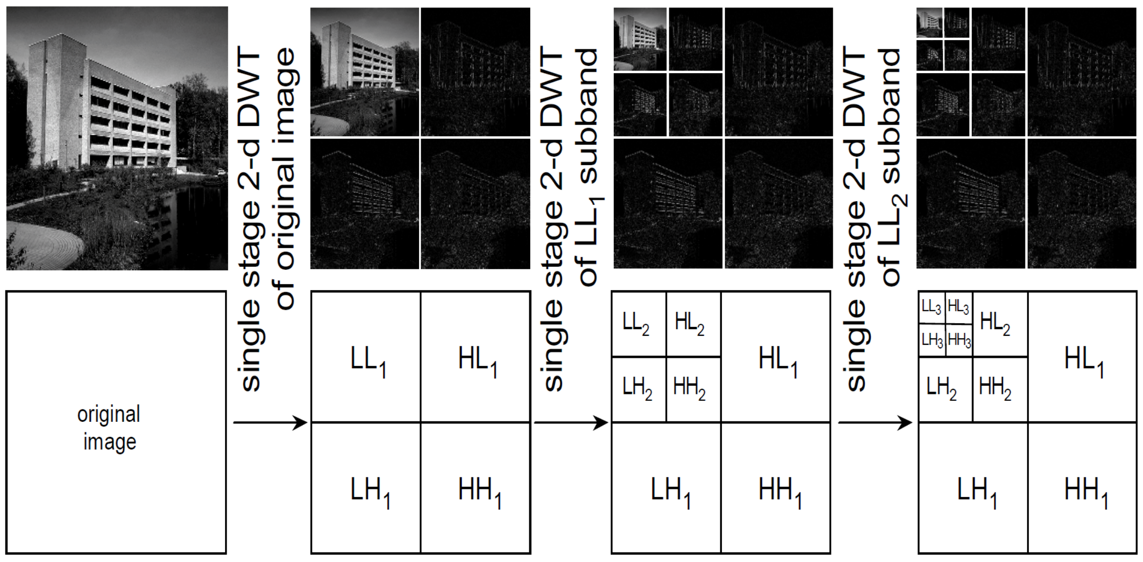

single stage 2-d DWT of original image
single stage 2-d DWT of LL1 subband
single stage 2-d DWT of LL2 subband
original image
LL1
HL1
LH1
HH1
LL2
HL2
LH2
HH2
LL3
HL3
LH3
HH3

Figure 2.15.: Three level two-dimensional DWT decomposition of an image [4]

A two-dimensional DWT is performed on three levels: On the first level the complete image is transformed by first applying a one dimensional DWT to each row and than applying a one

dimensional DWT again to each column. The output data of this process consists of four blocks representing the horizontal low-pass and vertical low-pass data (LL_1), the horizontal low-pass and vertical hig-hpass data (LH_1), the horizontal high-pass and vertical low-pass data (HL_1) and the horizontal high-pass and vertical high-pass data (HH_1). The two-dimensional DWT is than applied two more times to the LL-data of the previous level. The three level two-dimensional DWT decomposition of an image is shown in figure 2.15
The decorrelated data is grouped into blocks of 64 coefficients, which correspond a localized region in the original image. The structure of a block is shown in figure 2.16. The blocks are again grouped into segments consisting of a least 16 blocks and than passed into the bit-plane encoder. The bit-plane encoder processes the pixels contained in each segment in binary representation. It starts grouping the most significant bit (MSB) of each pixel into one bit-plane. If all of the bits in one bit-plane have the same value, the bit-plane encoder marks this bit-plane as constant and outputs its value. If the bit-plane is not constant, the bits are grouped into small words of two to four bits and than entropy encoded using a variable-length code. Once encoding of one bit-plane is finished the bit-plane encoder proceeds with the next lower bit-plane until a file data volume or quality limit has been reached. Note that the DC coefficients are quantized and coded separately [4].

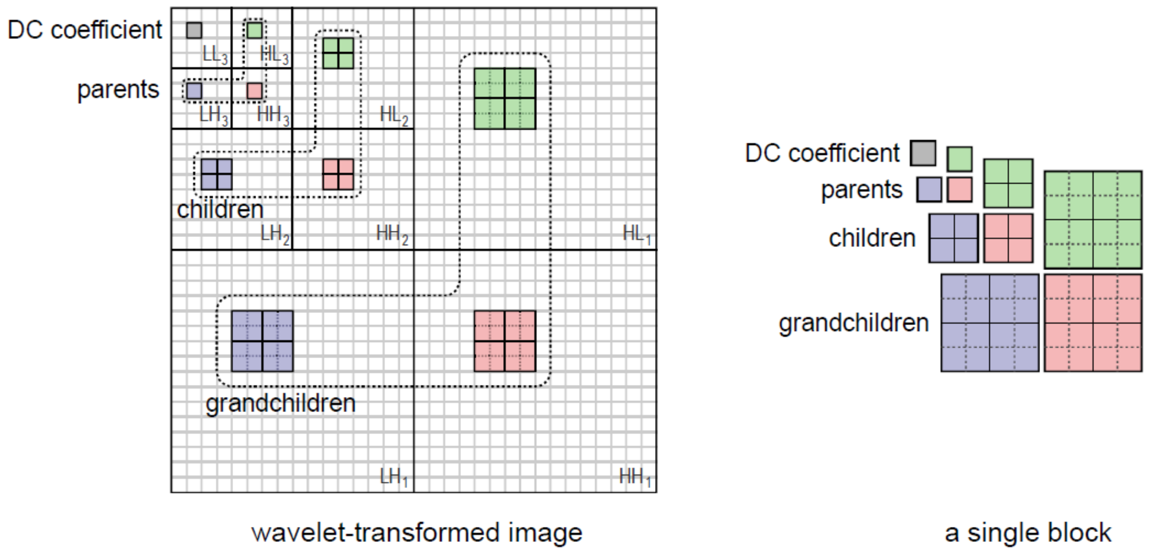


Figure 2.16.: Block of DWT coefficients [3]

2.3.3. CCSDS 123.0-B-2

CCSDS has defined another image compression algorithm called CCSDS 123.0-B-2, which is applicable for multispectral and hyperspectral images [5]. It is capable of performing lossless or near-lossless compression with a user defined maximum error in the reconstructed image. The algorithm is specifically designed for low-complexity to allow fast and low-power hardware implementations. The block diagram of the compressor is shown in figure 2.17. It consists of two

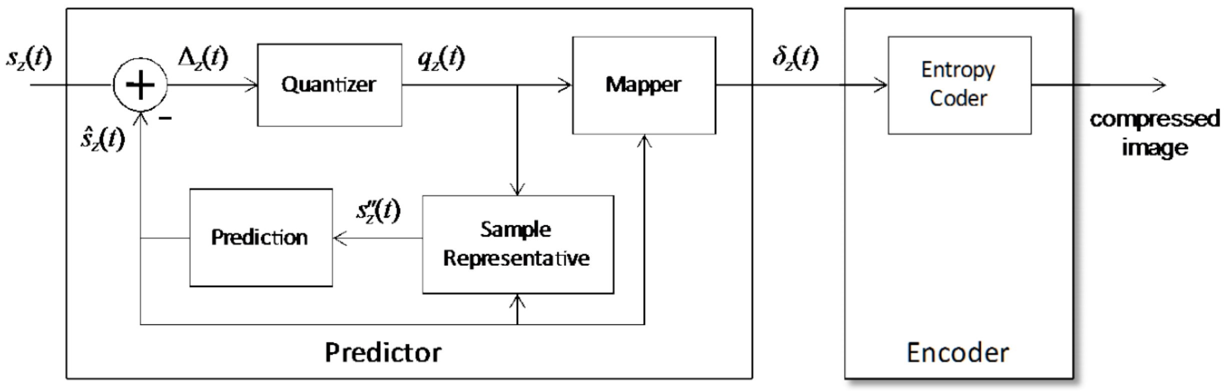


Figure 2.17.: CCSDS 123.0-B-2 block diagram [5]

main blocks: a predictor and an encoder.
The predictor uses an adaptively weighted prediction algorithm to predict the value of each pixel based on the values of nearby pixels in a small three-dimensional region around the current pixel. Assuming that the image data is inputted row by row, all pixels used for prediction have already been inputted before prediction for the current pixel takes places. This allows a memory efficient implementation, since no large buffers are needed for storing the complete image. Figure 2.18 shows a typical prediction region. The prediction step is performed sequentially in a single pass

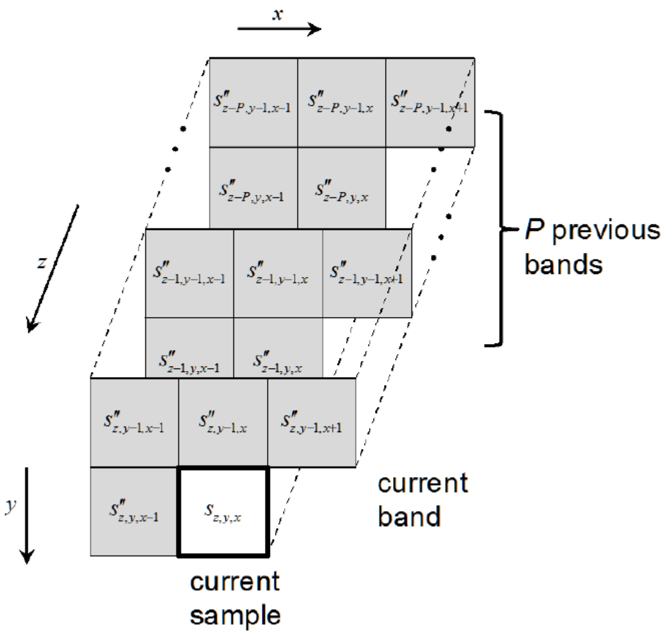


Figure 2.18.: CCSDS 123.0-B-2 - typical prediction region [5]

over the complete image. After each prediction step the weights used for prediction are updated. Since the decompressor does not have access to the original pixels in near-lossless mode, the predic-

tion is performed using sample representatives $s''_z(t)$, which are also available to the decompressor. The predicted pixel value $\hat{s_z}(t)$ is then subtracted from the actual pixel value and quantized using a uniform quantizer. The quantizer limits the maximum quantization error to a user specified value. The quantized difference value $q_z(t)$ is than used together with the predicated pixel value to calculate the next sample representative. Furthermore, the quantized difference value is mapped to an unsigned integer value in preparation for entropy encoding.
Entropy encoding is performed using one of three different methods:

- Sample-adaptive entropy encoding, which is basically length-limited rice encoding, whose length limit is determined sample by sample.
- Hybrid entropy encoding, which uses a variable-length binary code from a family of codes for high-entropy input data and a variable-to-variable length code for low entropy input data.
- Block-adaptive entropy encoding, which is defined in CCSDS 121.0-B-2.

3. Comparison of Image Compression Algorithms

In this chapter the main concepts of the three image compression algorithms presented in chapter "2.3 Image Compression Algorithms" are being used to create modified versions of these algorithms and evaluate them for the constraints mentioned in chapter "1.2 Problem Formulation".

3.1. Modifications

An image compressor typically consists of three main components: a decorrelation block, a quantizer and an encoder as shown in figure 3.1.

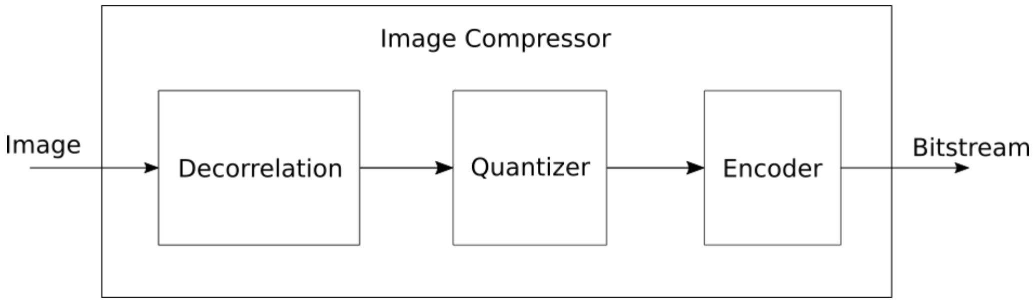


Figure 3.1.: High level block diagram of a typical image compressor

The decorrelation block uses prediction methods or transforms (linear or non-linear) to reduce the correlation between the image pixels. The quantizer removes irrelevant information and the encoder then compresses the decorrelated and quantized image data into a bitstream consisting of less bits than the original image. In order to reduce complexity and ease the implementation the quantizer and encoder of the algorithms have been modified.
An overview of these modifications is shown in figure 3.2. As defined in the corresponding standards the decorrelation block for CCITT T.81 (JPEG) consists of the two-dimensional DCT, for CCSDS 122.0-B-2 it consists of the 3-level two-dimensional DWT and for CCSDS 123.0-B-2 it consists of the three-dimensional difference based prediction. These decorrelation blocks are also inherited by the modified versions of the algorithms. CCITT T.81 (JPEG) and CCSDS 123.0-B-2 use a simple uniform quantizer but CSSDS 122.0-B-2 uses a more sophisticated quantization scheme incorporated in the bit-plane encoder. This more complex quantizer has been changed to a simple uniform quantizer. Since each algorithm also has its own kind of encoder and implementing

all these different encoders would have gone beyond the scope of this thesis, already available encoder implementations for Huffman encoding, Rice encoding and also run-length encoding have been selected to replace the encoders defined in the corresponding standards. Including different encoders in the comparison allows for finding the encoder, which is best suited for the corresponding algorithm. The modified image compression algorithms have been implemented

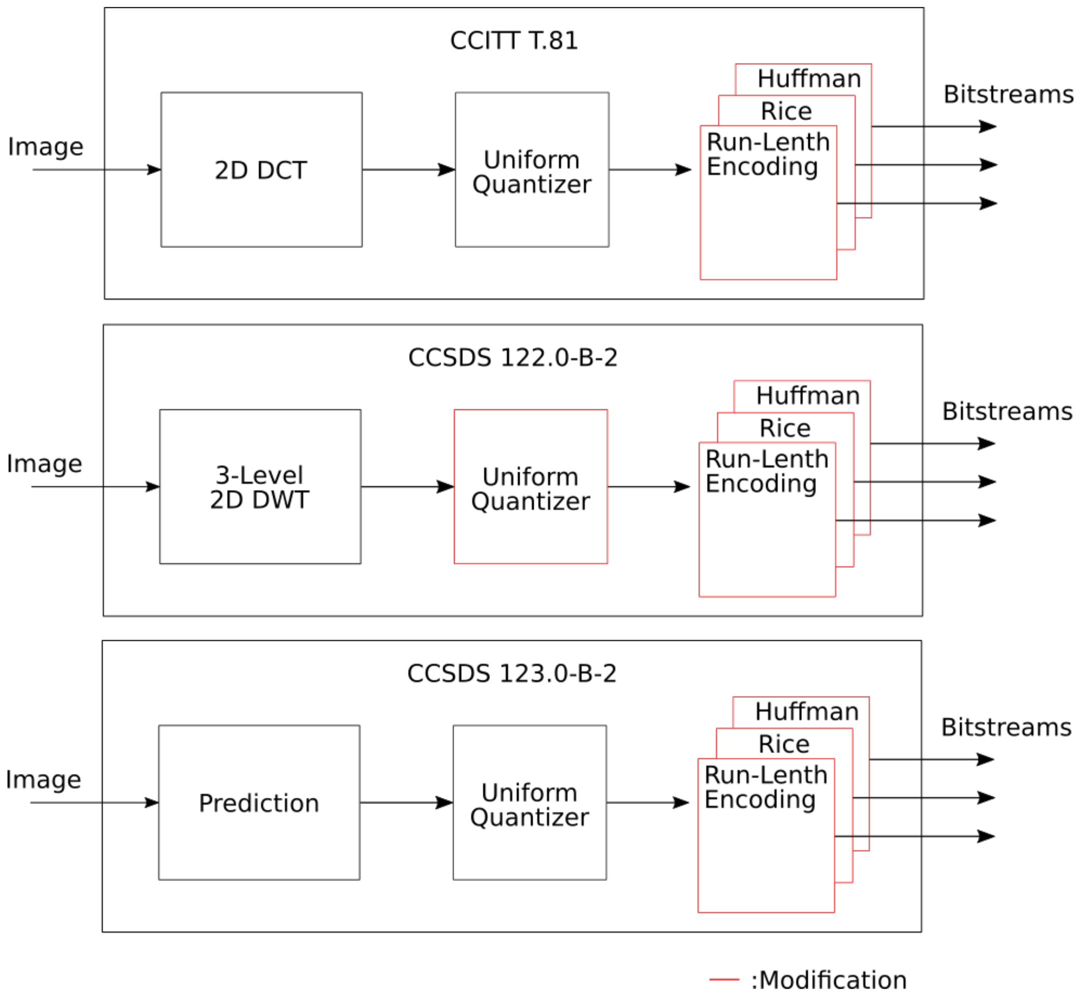


Figure 3.2.: Modified image compression algorithms (including multiple encoder options)

and evaluated in Octave. During the comparison several different test images have been inputted into the modified algorithms. The size of the outputted bitstreams has been determined and an image has been reconstructed from the bitstreams. The reconstructed image has then been compared with the original image by using the performance metrics given in equations 2.15 to 2.17.

3.2. Power Consumption

In order to compare the modified algorithms in terms of power consumption a complexity analysis has been performed on the decorrelation parts of the algorithms. The comparison is based on the idea that an algorithm with high computational and/or memory complexity will use more power than an algorithm with lower complexity, since it utilizes more logic and/or memory. For the following analysis an input image with HD resolution (1920 x 1080 pixel), three bands (YC_rC_b color space, 4:2:2 chrominance subsampling) and a row-by-row input order is assumed.

CCITT T.81(JPEG)

In CCITT T.81 (JPEG) the two-dimensional DCT is performed on blocks with a size of 8 x 8 pixel. In preparation for the DCT each sample is level shifted to a signed representation, resulting in 64 additions. For the DCT of each block $8^3 = 512$ multiplications and $8^2(8-1) = 448$ additions have to be performed. For the complete input image consisting of 1920 x 1080 pixel and one band a total of

$$n_{mul} = \frac{1920 \cdot 1080}{8^2} 512 = 1.65888 \cdot 10^7 \tag{3.1}$$

multiplications and

$$n_{add} = \frac{1920 \cdot 1080}{8^2}(448 + 64) = 1.65888 \cdot 10^7 \tag{3.2}$$

additions are necessary. Note that except for the level shifting all operations need to be implemented as floating point operations.
Due to the fact that the DCT is performed on 8 rows and the input image is inputted in row-by-row order, it is necessary to store 7 previous rows and the latest 8 pixels in the current row before all data is available to perform the DCT. This is illustrated in figure 3.3.

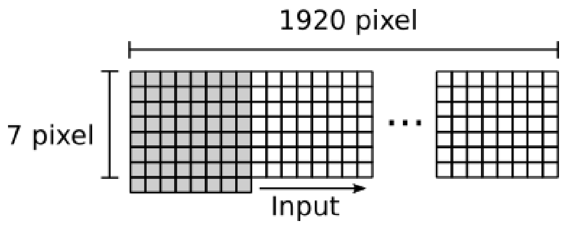


Figure 3.3.: Required memory for DCT

This results in a memory complexity of

$$n_{mem} = (7 \cdot 1920 + 8)\ pixel = 1.3448 \cdot 10^4\ pixel. \tag{3.3}$$

CCSDS 122.0-B-2

The two-dimensional integer DWT defined in [5] consists of two one-dimensional integer DWTs. One is performed on each row of the input image and the second one on each column. Each one-dimensional DWT outputs two individual sets of coefficients, high-pass coefficients

$$\begin{aligned}
D_0 &= x_1 - \left\lfloor \frac{9}{16}(x_0 + x_2) - \frac{1}{16}(x_2 + x_4) + \frac{1}{2} \right\rfloor \\
D_j &= x_{2j+1} - \left\lfloor \frac{9}{16}(x_{2j} + x_{2j+2}) - \frac{1}{16}(x_{2j-2} + x_{2j+4}) + \frac{1}{2} \right\rfloor \quad for j = 1, ..., N-3 \\
D_{N-2} &= x_{2N-3} - \left\lfloor \frac{9}{16}(x_{2N-4} + x_{2N-2}) - \frac{1}{16}(x_{2N-6} + x_{2N-2}) + \frac{1}{2} \right\rfloor \\
D_{N-1} &= x_{2N-1} - \left\lfloor \frac{9}{8}(x_{2N-2}) - \frac{1}{8}(x_{2N-4}) + \frac{1}{2} \right\rfloor
\end{aligned} \tag{3.4}$$

and low-pass coefficients

$$\begin{aligned}
C_0 &= x_0 - \left\lfloor -\frac{D_0}{2} + \frac{1}{2} \right\rfloor \\
C_j &= x_{2j} - \left\lfloor -\frac{D_{j-1} + D_j}{4} + \frac{1}{2} \right\rfloor \quad for j = 1, ..., N-1
\end{aligned} \tag{3.5}$$

where x_n denotes the $n-th$ sample of the input sequence of length $2N$. Calculating the high-pass coefficients needs N multiplications, $5N-2$ additions, $3N$ shift operations and N floor operations. Calculating the lox-pass coefficients needs $3N-1$ additions, $2N$ shift operations and N floor operations. Performing the two-dimensional DWT therefore needs $N+M$ multiplications, $8(N+M)-6$ additions, $5(N+M)$ shift operations and $2(N+M)$ floor operations assuming an input image with $2N$ rows and $2M$ columns and one band.

The two-dimensional DWT is performed on three levels with different sized input images: first on the complete input image, secondly on the subband coefficients LL_1 outputted by the first level and last on the subband coefficients LL_2 outputted by the second level. Each set of subband coefficients LL_x has one fourth of the size of the input image of the next higher level, which means that each column and row are only half as long as the corresponding row respectively column of the next higher level. As a result for the complete input image (1920x1080 pixel, $N = 960, M = 540$) a total of

$$n_{mul} = (960 + 540)(1 + \frac{1}{2} + \frac{1}{4}) = 2.625 \cdot 10^3 \tag{3.6}$$

multiplications,

$$n_{add} = 14(960 + 540) - 18 = 2.0982 \cdot 10^4 \tag{3.7}$$

additions,

$$n_{shift} = 5(960 + 540)(1 + \frac{1}{2} + \frac{1}{4}) = 1.3125 \cdot 10^4 \tag{3.8}$$

shift operations and

$$n_{floor} = 2(960 + 540)(1 + \frac{1}{2} + \frac{1}{4}) = 5.25 \cdot 10^3 \tag{3.9}$$

floor operations are necessary. Note that the input sequence consists of integers and the denominators are all powers of two, hence all operations can be implemented as fixed point operations.

The memory complexity for calculating the two-dimensional integer DWT is similar to that of the

DCT: 7 input samples need to be buffered for calculating the high-pass coefficients D_j and one high-pass coefficient needs to be buffered for calculating the low-pass coefficients C_j. This results in a memory complexity of

$$n_{mem} = (7.5 \cdot 1920)(1 + \frac{1}{2} + \frac{1}{4})\,pixel = 2.52 \cdot 10^4\,pixel. \tag{3.10}$$

CCSDS 123.0-B-2

In contrast to CCITT T.81(JPEG) and CCSDS 122.0-B-2 the prediction defined in CCSDS 123.0-B-2 [5] is performed not only on samples from the current band of the image but also on samples from previous bands. This means that chrominance subsampling is not supported by this algorithm, since each band needs to have the same dimension as the previous band. In order to predict the next sample of the current band a local sum

$$\sigma_{z,y,x} = \begin{cases} s''_{z,y,x-1} + s''_{z,y-1,x-1} + s''_{z,y-1,x} + s''_{z,y-1,x+1}, & y > 0,\ 0 < x < N_x - 1 \\ 4s''_{z,y,x-1}, & y = 0,\ x > 0 \\ 2(s''_{z,y-1,x} + s''_{z,y-1,x+1}), & y > 0,\ x = 0 \\ s''_{z,y,x-1} + s''_{z,y-1,x-1} + 2s''_{z,y-1,x}, & y > 0,\ x = N_x - 1 \end{cases} \tag{3.11}$$

consisting of neighboring sample representatives $s''_{z,y,x}$ is calculated (see figure 2.18). N_x is the number of samples/pixels in one row of the input image and N_y is the number of samples/pixels in one column. This results in $3(N_x - 2)(N_y - 1) + 3(N_y - 1)$ additions and $N_x - 2 + N_y - 1$ shift operations for each band. Based on the local sum and sample representatives a central difference

$$d_{z,y,x} = 4s''_{z,y,x} - \sigma_{z,y,x} \tag{3.12}$$

and directional local differences

$$d^N_{z,y,x} = \begin{cases} 4s''_{z,y-1,x} - \sigma_{z,y,x}, & y > 0 \\ 0 & y = 0 \end{cases} \tag{3.13}$$

$$d^W_{z,y,x} = \begin{cases} 4s''_{z,y,x-1} - \sigma_{z,y,x}, & x > 0,\ y > 0 \\ 4s''_{z,y-1,x} - \sigma_{z,y,x}, & x = 0,\ y > 0 \\ 0 & y = 0 \end{cases} \tag{3.14}$$

$$d^{NW}_{z,y,x} = \begin{cases} 4s''_{z,y-1,x-1} - \sigma_{z,y,x}, & x>0,\ y>0 \\ 4s''_{z,y-1,x} - \sigma_{z,y,x}, & x=0,\ y>0 \\ 0 & y=0 \end{cases} \tag{3.15}$$

are calculated. Resulting in $4N_xN_y - 3N_x$ additions and $4N_xN_y - 3N_x$ shift operations for each band. For the following steps a new variable

$$t = yN_x + x \tag{3.16}$$

for indexing the samples in the current band is introduced. Furthermore, the local differences are collected in a column vector

$$\mathbf{U}_z(t) = \begin{bmatrix} d^N_z(t) \\ d^W_z(t) \\ d^{NW}_z(t) \\ d_{z-1}(t) \\ d_{z-2}(t) \\ \dots \\ d_1(t) \end{bmatrix} \tag{3.17}$$

and a weight vector

$$\mathbf{W}_z(t) = \begin{bmatrix} w^N_z(t) \\ w^W_z(t) \\ w^{NW}_z(t) \\ w^{(1)}_z(t) \\ w^{(2)}_z(t) \\ \dots \\ w^{(3)}_z(t) \end{bmatrix} \tag{3.18}$$

is introduced. The predicted central local difference

$$\hat{d}_z(t) = \mathbf{W}^T_z(t)\mathbf{U}_z(t) \tag{3.19}$$

is then calculated for each element $t > 0$ resulting in $15(N_xN_y - 1)$ multiplications and $12(N_xN_y - 1)$ additions for an input image with 3 bands. A high-resolution predicted sample value

$$\breve{s}_z(t) = clip\Big(mod_R\left[\hat{d}_z(t) + 2^{\Omega}(\sigma_z(t) - 4s_{mid})\right] + 2^{\Omega+2}s_{mid} + 2^{\Omega+1},\ \{2^{\Omega+2}s_{min},\ 2^{\Omega+2}s_{max} + 2^{\Omega+1}\}\Big) \tag{3.20}$$

is derived from the predicted central local difference, where Ω and R are user-selectable parameters and s_{min}, s_{mid} and s_{max} represent the minimum, midpoint and maximum value a sample can have. The clip function forces the output value to be located in certain thresholds and the mod_R function truncates the input value to a value, which can be represented with R bits. This step needs $3N_xN_y$ additions, N_xN_y shift operations, N_xN_y modulo operations and N_xN_y clipping operations for each band. The algorithm continues with calculating a double-resolution predicted sample

$$\tilde{s}_z(t) = \begin{cases} \left\lfloor \frac{\check{s}_z(t)}{2^{\Omega+1}} \right\rfloor, & t > 0 \\ 2s_{z-1}(t), & t = 0,\ P > 0,\ z > 0 \\ 2s_{mid}, & t = 0 \textit{ and } (P = 0 \textit{ or } z = 0) \end{cases} \tag{3.21}$$

and the predicted sample value

$$\hat{s}_z(t) = \left\lfloor \frac{\tilde{s}_z(t)}{2} \right\rfloor, \tag{3.22}$$

where P is again a user-selectable parameter, which defines the number of preceding bands to be used for prediction. This step only needs $4N_xN_y$ shift operations and $2N_xN_y - 2$ floor operations for each band. This value is then used to calculate sample representative

$$s''_z(t) = \begin{cases} s_z(0), & t = 0 \\ \left\lfloor \frac{\tilde{s}''_z(t)+1}{2} \right\rfloor, & t > 0 \end{cases} \tag{3.23}$$

from the the double-resolution sample representative

$$\tilde{s}''_z(t) = \left\lfloor \frac{4(2^{\Theta} - \phi_z) \cdot (s'_z(t)2^{\Omega} - sgn(q_z(t)) \cdot m_z(t) \cdot \psi_z \cdot 2^{\Omega-\Theta}) + \phi_z \cdot \check{s}_z(t) - \phi_z \cdot 2^{\Omega+1}}{2^{\Omega+\Theta+1}} \right\rfloor \tag{3.24}$$

and the clipped version of the quantizer bin center

$$s'_z(t) = clip\left(\hat{s}_z(t) + q_z(t)(2m_z(t) + 1),\ \{s_{min},\ s_{max}\}\right), \tag{3.25}$$

where Θ, ϕ_z and ψ_z are user-specified integers, $q_z(t)$ is the quantized difference value and $m_z(t)$ is the quantization step size. Calculation of the sample representative needs $4N_xN_y$ multiplications, $5N_xN_y - 1$ additions, $2N_xN_y - 1$ shift operations, $2N_xN_y - 1$ floor operations and N_xN_y clipping operations for each band. The final step of the predication is the update of the weight vector, which is defined as

$$w_z^{(i)}(t+1) = clip\left(w_z^{(i)}(t) + \left\lfloor \frac{1}{2}(sgn[e_z(t)] \cdot 2^{-(\rho(t)+\zeta_z^{(i)})} \cdot d_{z-1}(t) + 1) \right\rfloor,\ \{w_{min},\ w_{max}\}\right), \tag{3.26}$$

$$w_z^{N}(t+1) = clip\left(w_z^{N}(t) + \left\lfloor \frac{1}{2}(sgn[e_z(t)] \cdot 2^{-(\rho(t)+\zeta_z^{*})} \cdot d_z^{N}(t) + 1) \right\rfloor,\ \{w_{min},\ w_{max}\}\right), \tag{3.27}$$

$$w_z^{W}(t+1) = clip\left(w_z^{W}(t) + \left\lfloor \frac{1}{2}(sgn[e_z(t)] \cdot 2^{-(\rho(t)+\zeta_z^{*})} \cdot d_z^{W}(t) + 1) \right\rfloor,\ \{w_{min},\ w_{max}\}\right), \tag{3.28}$$

$$w_z^{NW}(t+1) = clip\left(w_z^{NW}(t) + \left\lfloor \frac{1}{2}(sgn[e_z(t)] \cdot 2^{-(\rho(t)+\zeta_z^{*})} \cdot d_z^{NW}(t) + 1) \right\rfloor,\ \{w_{min},\ w_{max}\}\right), \tag{3.29}$$

with the prediction error

$$e_z(t) = 2s_z^{'}(t) - \tilde{s}_z(t) \tag{3.30}$$

and the weight update scaling exponent

$$\rho(t) = clip\left(v_{min} + \left\lfloor \frac{t - N_x}{t_{inc}} \right\rfloor, \{v_{min}, v_{max}\}\right) + D - \Omega. \tag{3.31}$$

The value t_{inc} is a power of 2 and defines how fast the update should be changed, $\zeta_z^{(i)}$, ζ_z^{*}, w_{min}, w_{max} v_{min} and v_{max} are user-specified integers. This leads to 16 additions, 6 shift operations, 5 floor operations and 5 clipping operations for each band.
For an input image with 1920 columns and 1080 rows and 3 bands this finally sums up to

$$\begin{aligned} n_{mul} &= 15(1920 \cdot 1080 - 1) + 12 \cdot 1920 \cdot 1080 \\ &= 5.5987185 \cdot 10^7 \end{aligned} \tag{3.32}$$

multiplications,

$$\begin{aligned} n_{add} = &[3(1920-2)(1080-1) + 3(1080-1) + 4 \cdot 1902 \cdot 1080 \\ &- 3 \cdot 1920 + 3 \cdot 1920 \cdot 1080 + 5 \cdot 1920 \cdot 1080 - 1 + 16] \cdot 3 \\ &+ 12(1920 \cdot 1080 - 1) \\ = &6.4269413 \cdot 10^7 \end{aligned} \tag{3.33}$$

additions,

$$\begin{aligned} n_{shift} = &[1920 - 2 + 1080 - 1 + 4] \cdot 1920 \cdot 1080 - 3 \cdot 1920 \\ &+ 5 \cdot 1920 \cdot 1080 \cdot 3 + 2 \cdot 1920 \cdot 1080 - 1 + 6 \\ = &6.0126117 \cdot 10^7 \end{aligned} \tag{3.34}$$

shift operations,

$$\begin{aligned} n_{floor} &= [2 \cdot 1920 \cdot 1080 - 2 + 2 \cdot 1920 \cdot 1080 - 1] \cdot 3 + 5 \\ &= 2.4883196 \cdot 10^7 \end{aligned} \tag{3.35}$$

floor operations,

$$\begin{aligned} n_{clip} &= [2 \cdot 1920 \cdot 1080] \cdot 3 + 5 \\ &= 1.2441605 \cdot 10^7 \end{aligned} \tag{3.36}$$

clipping operations and

$$\begin{aligned} n_{mod} &= [1920 \cdot 1080] \cdot 3 \\ &= 6.2208 \cdot 10^6 \end{aligned} \tag{3.37}$$

modulo operations.
Assuming that all 3 bands of the input image are available at the same time and outputted row-by-row, only one line of each band of the input image needs to be stored in memory for the performing the prediction (see figure 2.18). This results in a memory complexity of

$$n_{mem} = 3 \cdot 1920\, pixel = 5.76 \cdot 10^3\, pixel. \tag{3.38}$$

Summary

In order to compare all three algorithms in terms of complexity and therefore also power consumption it is necessary to take the 4:2:2 chrominance subsample into account. All numbers of operations to be performed by the algorithms CCIT T.81 (JPEG) and CCSDS 122.0-B-2 are only valid for an input image with 1920 x 1080 pixels and one band. Both algorithms can be performed individually on each of the three bands with different dimensions for the input image. This is a necessary condition to support 4:2:2 chrominance subsampling on the input image, since the length of each row of the C_r and C_b band is only half of that of the Y band. As a result the numbers given for CCIT T.81 (JPEG) and CCSDS 122.0-B-2 need to be scaled by a factor of 2. On the other hand CCSDS 123.0-B-2 can not be applied to each band of the input image individually, since it requires the bands to have the same dimensions for the prediction step, which is also based on the data in preceding bands. As a consequence 4:2:2 chrominance subsampling is not supported by CCSDS 123.0-B-2 and all 3 bands need to have the same size. The number of operations needed to perform each algorithm is given in table 3.1.

	CCIT T.81 (JPEG)	**CCSDS 122.0-B-2**	**CCSDS 123.0-B-2**
n_{mul}	$3.31776 \cdot 10^7$	$5.25 \cdot 10^3$	$5.5987185 \cdot 10^7$
n_{add}	$3.31776 \cdot 10^7$	$4.1964 \cdot 10^4$	$6.4269413 \cdot 10^7$
n_{shift}	0	$2.625 \cdot 10^4$	$6.0126117 \cdot 10^7$
n_{floor}	0	$1.05 \cdot 10^4$	$2.4883196 \cdot 10^7$
n_{clip}	0	0	$1.2441605 \cdot 10^7$
n_{mod}	0	0	$6.2208 \cdot 10^6$
n_{mem}	$2.6896 \cdot 10^4$	$5.04 \cdot 10^4$	$5.76 \cdot 10^3$

Table 3.1.: Complexity of algorithms

When comparing all three algorithms it becomes obvious that CCSDS 122.0-B-2 requires by far the least number of multiplications. This is a big advantage in terms of complexity, because the implementation of the multiplication operation is in general the most complex one of the operations considered in the comparison. In a FPGA typically dedicated logic cells, so called DSP slices, are used for multiplication and these cells are only available in a limited amount. CCSDS 122.0-B-2 needs shift and floor operations, which are not necessary for implementing CCIT T.81 (JPEG). However, implementing these operations in a FPGA is simple and can be done by using LUTs. Only in terms of memory complexity CCSDS 122.0-B-2 lags behind in the comparison with the other two algorithms. On the other hand, implementation of CCSDS 123.0-B-2 needs the highest number of operations and is therefore the most expensive one to implement. To sum up, implementation of the CCSDS 122.0-B-2 algorithm is the least expensive option in terms of complexity and therefore also power consumption.

3.3. Rate-Distortion Performance

The process of image compression always involves making a compromise between the memory space needed to store the compressed image and the quality of the reconstructed image. The higher the compression of the image, the lower is the quality of the reconstructed image. In order to compare the three modified algorithms and also the different encoder options the so called rate-distortion performance is used. It is a function that represents the distortion in the reconstructed image for a given bit rate. The distortion is introduced by the compression process and is quantified by the MSE. It is even possible to see the distortions directly in the reconstructed images when compression is very high.
The test image used for comparing the rate-distortion performance has a resolution of $512 \cdot 512$ pixels, three bands (YC_bC_r color space), a resolution of 8 bit per pixel in each band and 4:2:2 chrominance subsampling. This results in a bit rate of 16 *bit*/*pixel* for the uncompressed test image.

CCITT T.81 (JPEG)

In order to determine the rate-distortion performance of the modified CCITT T.81 (JPEG) algorithm the quantization tables for the luminance and chroma channels given as a reference in the standard have been changed. This change is necessary to adapt the level of quantization and consists of rounding each value in the quantization table to the next power of two and multiplying or dividing the complete table again element-wise by a power of two. Using powers of two in the quantization table is necessary due to the fact, that division is in general hard to implement in a FPGA whereas dividing by a power of two can be simply replaced by a shift operation. The following values of the changed quantization tables (**Q**) have been used:

- **Q** = *all ones* (lossless)
- **Q**./8
- **Q**./4
- **Q**./2
- **Q**
- **Q**. ∗ 2
- **Q**. ∗ 4

where .∗ means element-wise multiplication and ./ means element-wise division. The resulting rate-distortion performance is given in figure 3.4. Lossless corresponds to the data points with minimum MSE and **Q**. ∗ 4 to the data points with maximum MSE.

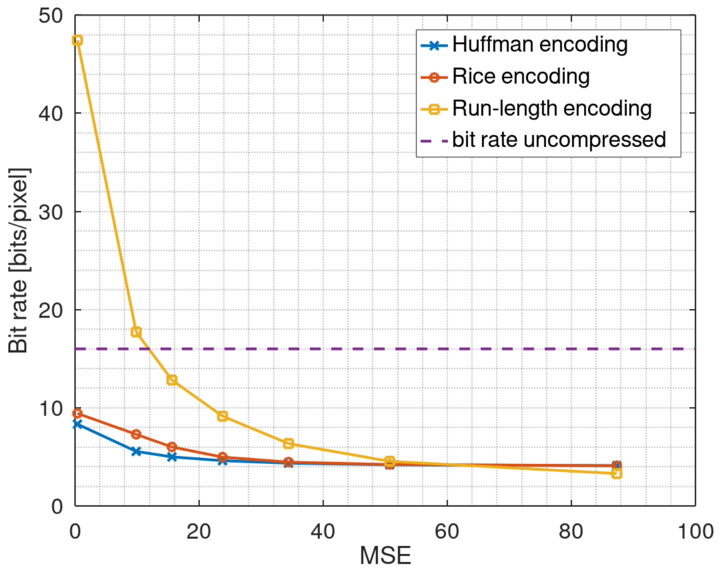


Figure 3.4.: Rate-distortion performance - CCITT T.81 (JPEG)

A general observation that can be made from the results is that when the bit-rate decreases the MSE increases. This coincides with the statement about image quality and achieved compression from the introduction of this chapter. Although it is not clearly visible in figure 3.4, the results also show that for the lossless setting (all values of **Q** are 1) the MSE is contrary to the expectation not equal to zero. This can be explained by the fact that the DCT and inverse DCT are implemented with limited precision and therefore some information is lost during the transformation process.

The rate-distortion performance of the run-length encoder is the worst. It starts at around 47.5 *bits*/*pixel*, quickly drops below the bit rate of 16 *bits*/*pixel* for the uncompressed image but stays well above the bit rate of the other encoders until a MSE of around 50 is reached. From that point onward run-length encoding is slightly better than Huffman and Rice encoding but the MSE in that region is already very high and image quality very low. Note that a bit rate above 16 *bits*/*pixel* (indicated by the purple dashed line) means that the compressed image needs more memory space to be stored than the uncompressed image.

Rice encoding has a bit rate of 9.4 *bits*/*pixel* for the lossless case, which corresponds to 58.75% of the size of the uncompressed image. Its bit rate decreases with increasing MSE until it approaches a limit value slightly above 4 *bits*/*pixel*, which represents the limit value for the rice encoding. When the limit of 4 *bits*/*pixel* is reached, only one sign bit and one stop bit for the variable length code is being used to represent each pixel in the three bands. Note that in this case each sample has a value of zero.

Huffman encoding seems to be the best option. It has the lowest bit rate for the lossless case of only 8.3 *bits*/*pixel* and approaches the limit of 4 *bits*/*pixel* even faster than Rice encoding. The plot does not include the information contained in the Huffman tree, which is created during the encoding process based on the probability distribution of the input data and is necessary for decoding the bitstream. But the increase of the bit rate due to the information contained in the Huffman tree is negligible. For the lossless case it only increases the bit rate to 8.4 *bits*/*pixel*.

CCSDS 122.0-B-2

The rate-distortion performance of the modified CCSDS 122.0-B-2 algorithm is shown in figure 3.5. The quantizer of the modified algorithm works by simply diving the coefficients outputted by the 3-level two-dimensional DWT by a power of two. This change has the same reason as the change of the quantization tables in the modified CCITT T.81 (JPEG) algorithm: ease of implementation. The following divisor settings have been used:

- L1 = 1, L2 = 1, L3 = 1
- L1 = 2, L2 = 1, L3 = 1
- L1 = 4, L2 = 1, L3 = 1
- L1 = 2, L2 = 2, L3 = 1
- L1 = 4, L2 = 2, L3 = 1
- L1 = 8, L2 = 1, L3 = 1
- L1 = 8, L2 = 4, L3 = 2
- L1 = 8, L2 = 8, L3 = 4
- L1 = 16, L2 = 8, L3 = 4
- L1 = 32, L2 = 16, L3 = 8

The setting L1 = 1, L2 = 1, L3 = 1 (lossless) corresponds to the data points with minimum MSE and L1 = 32, L2 = 16, L3 = 8 to the data points with maximum MSE.

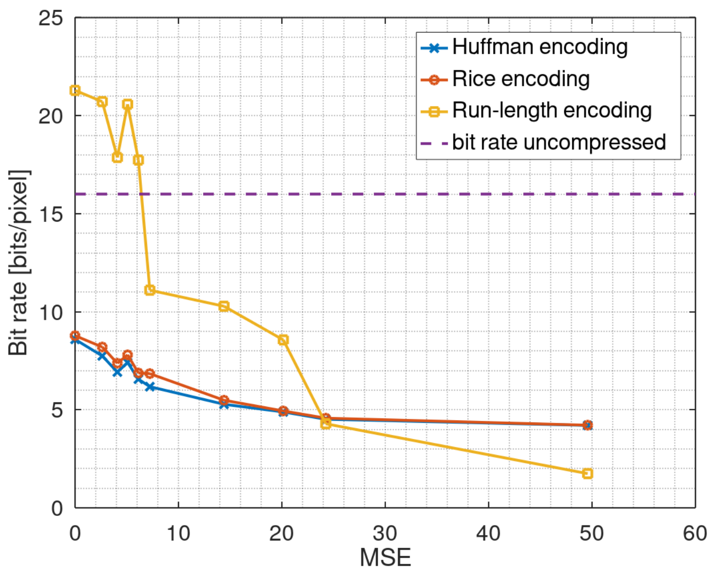


Figure 3.5.: Rate-distortion performance - CCSDS 122.0-B-2

The rate-distortion performance of the modified CCSDS 122.0-B-2 algorithm is in terms of waveform similar to that of the modified CCITT T.81 (JPEG) algorithm but in terms of absolute values it performs better. In contrast to the modified CCITT T.81 (JPEG) algorithm the MSE for the lossless case is as expected equal to zero and no information is lost. Another important thing to notice is that the rate-distortion performance depends on the selected quantization settings. This can be seen at the fourth data point, which corresponds to the quantization setting L1 = 2, L2 = 2, L3 = 1.

For this setting the bit rate is higher than the bit rate of the previous setting L1 = 4, L2 = 1, L3 = 1 although its MSE is higher than that of the previous setting.
The rate-distortion performance of the run-length encoder is again the worst. But here it starts at around 22 *bits/pixel*, quickly drops below the bit rate of 16 *bits/pixel* for the uncompressed image but stays well above the bit rate of the other encoders until a MSE of around 25 is reached. From that point onward run-length encoding performs better than Huffman and Rice encoding.
The rate-distortion performance of Rice and Huffman encoding is close to each other. They both start at just below 9 *bits/pixel* and than approach a limit value of slightly above 4 *bits/pixel*. But again Huffman encoding performs better than Rice encoding.

CCSDS 123.0-B-2

For the modified CCSDS 123.0-B-2 algorithm the rate-distortion performance is given in figure 3.6. Unfortunately the decorrelation block of the CCSDS 123.0-B-2 algorithm does not support input images with different resolutions in each band. This means that 4:2:2 chroma subsampling is not supported and therefore the bit rate of the uncompressed image is 24 *bits/pixel*.

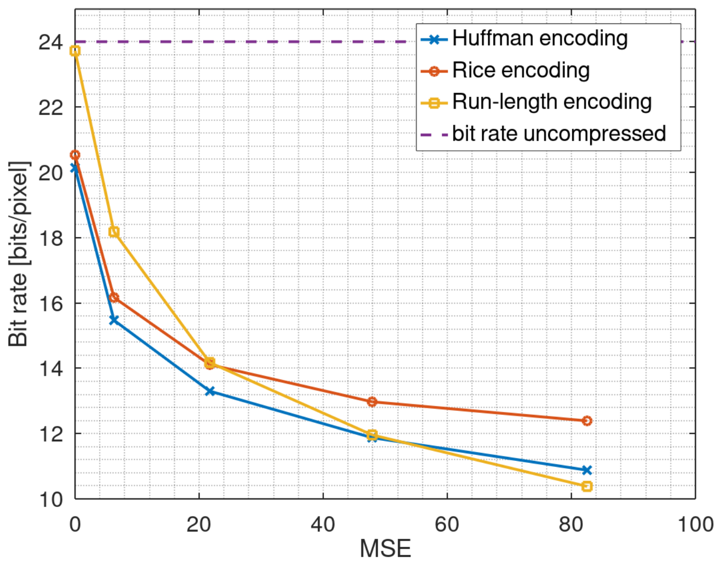


Figure 3.6.: Rate-distortion performance - CCSDS 123.0-B-2 (downscaled)

Furthermore, the runtime of this algorithm in Octave was very high for a test image with a resolution of $512 \cdot 512$ pixels. Therefore the test image had to be scaled down by a factor of 0.5. The following quantization settings have been used:

- $err_{max} = 0$ (lossless)
- $err_{max} = 1$
- $err_{max} = 2$
- $err_{max} = 3$
- $err_{max} = 4$

where err_{max} represents a user-selectable parameter for limiting the MAE in the reconstructed image.
The rate-distortion performance of this algorithm is different in terms of waveform compared to the other algorithms. The performance of all three encoding options is much closer to each other. Run-length encoding starts just below the bit rate of the uncompressed image of 24 *bits/pixel* in the lossless case. It performs even better than Rice encoding from about a MSE of 21 onward and better than Huffman encoding from about a MSE of 58 onward.
Rice and Huffman encoding start at around 20 *bits/pixel* for the lossless case. The bit rate then drops fast for both the encoding options but Huffman encoding is always better than Rice encoding. This difference in performance increases with the MSE.

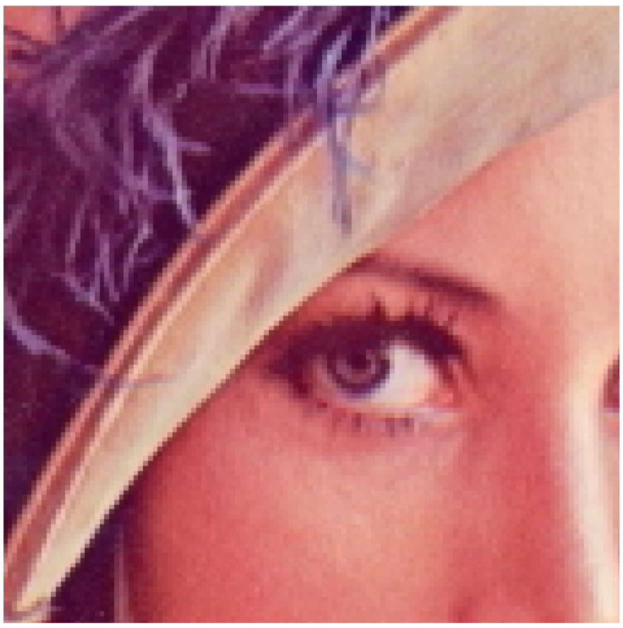
Image quality

In order to get an impression of what the rate-distortion performances actually look like in the reconstructed images and what kind of distortion is introduced by the decorrelation block of each of the three modified algorithms, some reconstructed images are shown in figure 3.7 to 3.8.
In figure 3.7 the uncompressed test image and reconstructed images for the three different algorithms are shown. The bit rate for the compressed images is around 50 % of the bit rate of the uncompressed image. The following quantization settings and MAE, MSE and PSNR values belong the corresponding images:

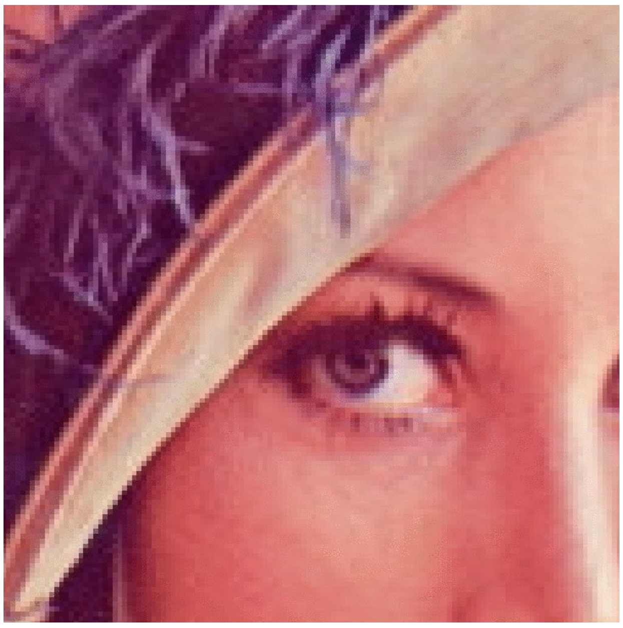
subfigure	quantization	MAE	MSE	PSNR
(a)	-	-	-	-
(b)	**Q**./8	21	9.85	38.20
(c)	L1 = 4, L2 = 1, L3 = 1	11	4.09	42.01
(d)	$err_{amx} = 4$	96	82.54	28.96

Table 3.2.: Parameters for figure 3.7

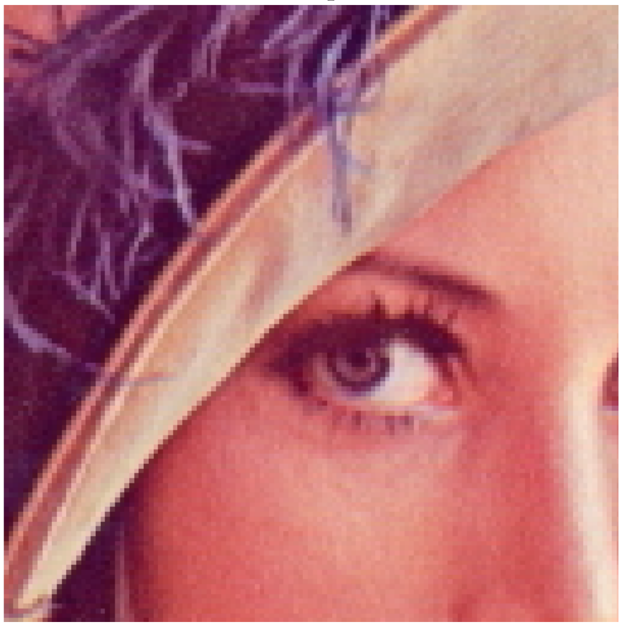
Based on the parameters given in table 3.2 CCSDS 122.0-B-2 clearly performs best. But looking at the actual reconstructed images it is hard to spot any difference between the subfigure (b) and (c). Whereas subfigure (d) already shows compression artifacts in the form of horizontal lines with different colors than the uncompressed image.



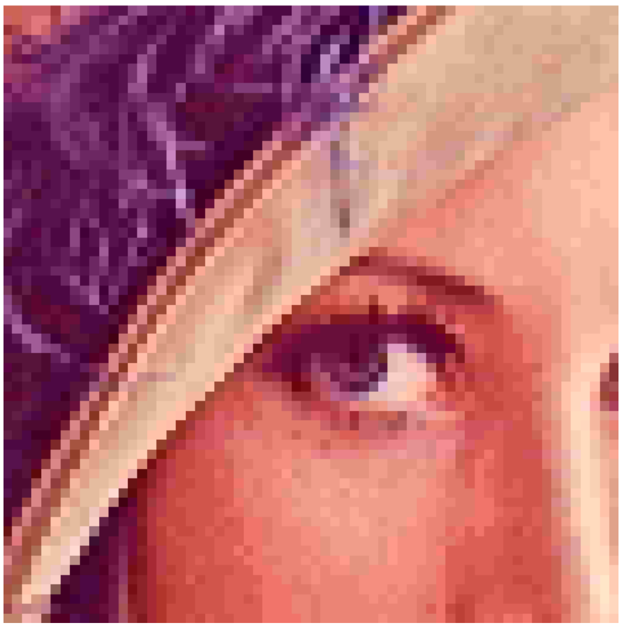
(a) Uncompressed



(b) CCIT T.81 (JPEG)



(c) CCSDS 122.0-B-2



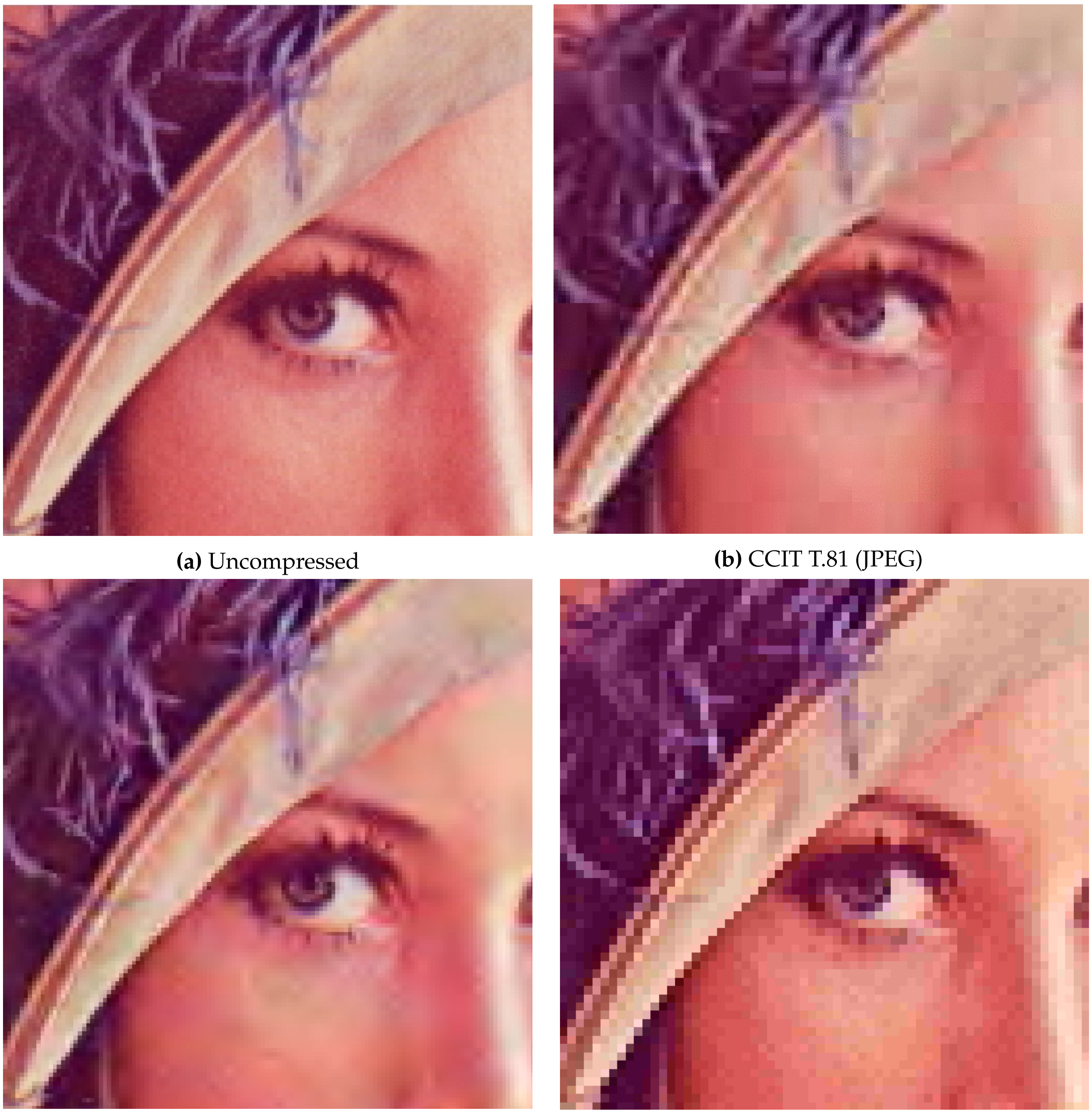
(d) CCSDS 123.0-B-2 (downscaled 0.5)

Figure 3.7.: Distortion in compressed images (*bitrate* ≈ 50%)

In order to show the compression artifacts of each of the three algorithms the bit rate of the compressed images has been reduced until the MSE in the reconstructed image is approximately 50. This lead to compression artifacts in all of the three reconstructed images as can be seen in figure 3.8. The following quantization settings, MAE, MSE and PSNR values belong the corresponding images:

subfigure	quantization	MAE	MSE	PSNR
(a)	-	-	-	-
(b)	$\mathbf{Q}.*2$	78	50.72	31.080
(c)	L1 = 32, L2 = 16, L3 = 8	52	49.58	31.18
(d)	$err_{amx} = 3$	41	47.88	31.33

Table 3.3.: Parameters for figure 3.8



(a) Uncompressed **(b)** CCIT T.81 (JPEG)

(c) CCSDS 122.0-B-2 **(d)** CCSDS 123.0-B-2 (downscaled 0.5)

Figure 3.8.: Compression artifacts ($MSE \approx 50$)

CCITT T.81 (JPEG) shows typical block artifacts. These artifacts are a result of grouping the input image into 8 x 8 pixel blocks. Additionally the color of some of the blocks is different from the uncompressed image. This creates sharp edges between neighboring blocks.
CCSDS 122.0-B-2 does not show any blocks but instead the image becomes blurry and also contains sections with colors different from the uncompressed image. The blur is a result of the quantization of the high frequency components whereas the change in color seems to be caused by the quantization of the low frequency components, especially the DC coefficients.
The CCSDS 123.0-B-2 compressed image shows again a different kind of artifact. Here, the image contains clusters of neighboring pixels with very similar colors. This is a result of the prediction method used in the decorrelation block of the algorithm. With higher quantization the difference value between the actual pixel and the predicted mean value becomes zero and therefore, all pixels in the cluster have a similar color.

Summary

Comparing the three encoding options shows that Huffman encoding is the best in terms of rate-distortion performance for each of the three considered algorithms. Rice encoding performs nearly as good as Huffman coding and run-length encoding only performs good for highly compressed data, which results in poor quality of the reconstructed image. Would be obvious to select Huffman encoding as the encoder for the EIVE image compressor, since it provides the lowest bit rates in all of the evaluated encoder options. But for other reasons, which will be explained later, Rice encoding represents a less complex but still very good performing solution.
A comparison of the rate-distortion performance of the three different decorrelation methods in combination with Rice encoding is shown in figure 3.9.

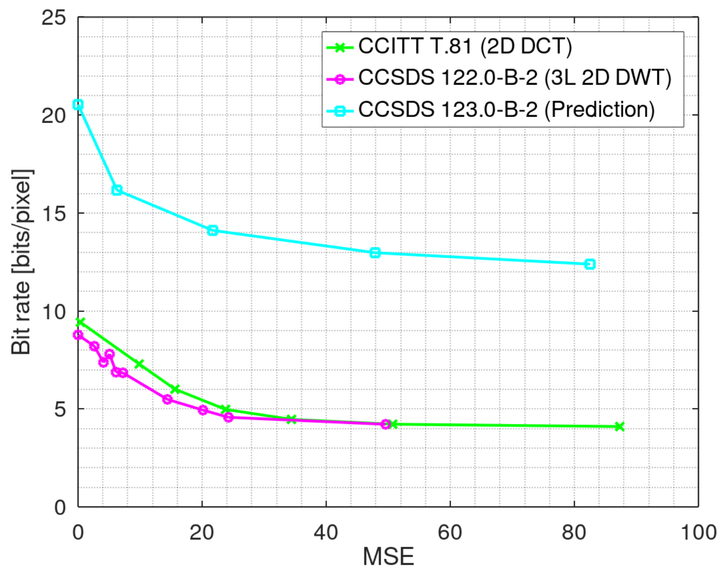


Figure 3.9.: Rate-distortion performance - comparison of Rice encoding

The decorrelation method of the CCDS 122.0-B-2 algorithm, the 3-level two-dimensional DWT, shows the best rate-distortion performance. For the lossless case and using Huffman or Rice encoding it reduces the bit rate to approximately 50% compared to the bit rate of the uncompressed image. The decorrelation method of CCITT T.81(JPEG) shows a similar but slightly worse performance and CCSDS 123.0-B-2 shows a poor performance.
To sum up, the 3-level two-dimensional DWT in combination with Huffman encoding performs best in terms of rate-distortion performance. But it is hard to make a decision between CCITT T.81 (JPEG) and CCSDS 122.0-B-2 regarding the impacts on image quality due to compression artifacts for low bit rates. Based on the performance metrics given in table 3.3 all algorithms perform equally worse. The PSNR value, which is a quality metric used in image compression, is nearly the same for all algorithms and as a result image quality at low bit rates cannot be used for deciding which algorithm should be used for the EIVE image compressor.

3.4. Execution Time

Another aspect of the comparison of the three modified algorithms is the execution time. Since taking images with the 4k camera is not the primary mission objective and consumes limited resources such as energy and time for performing more important task, the execution time for compressing an image should be as short as possible. In the following a row-by-row input order is assumed.
As already mentioned in the analysis of the memory complexity of each of the three decorrelation blocks of the corresponding algorithms, none of them operates on more than 8 consecutive rows of input data. This means that execution of the decorrelation process can start earliest when 8 rows of input data have been received. After the first 8 rows have been received each of the decorrelation processes can continue as new input data is received. Assuming that the data throughput is constant and each execution step produces a new output sample, the decorrelation process finishes with a delay equivalent to the duration of inputting 8 rows of the input image and a execution time, which is equivalent to the duration of receiving the complete image.
Run-length and Rice encoding also only operate on the current input data sample and are not dependent on any previous or subsequent data samples. Also each execution step produces a new output sample. Therefore these two encoders can operate with a delay of only one execution step.
For the Huffman encoder the situation is different. It depends on the probability distribution of the input data, which is unknown before the input data has been received completely. This means that Huffman encoding can not start before the complete input image has been received and therefore in the best case Huffman encoding adds an additional execution time equivalent to receiving the complete image to the execution time of the decorrelation process. There also exists an adaptive Huffman encoding, which does not need to read the complete input data prior to processing it but it is more complex than Huffman encoding and not considered in this comparison [6].

3.5. Conclusion

To sum up, the findings of the comparison show that the proposed modified CCSDS 122.0-B-2 image compression algorithm performs best in terms of computational complexity respectively power consumption and rate-distortion performance. While in terms of execution time no significant difference exists between the three algorithms. With Huffman coding being the most efficient but also complex encoder considered in the comparison, Rice encoding represents the best trade-off between implementation effort and compression effectiveness. As a result, the following modified CCSDS 122.0-B-2 image compressor architecture (figure 3.10) has been selected for implementation in the EIVE mission.

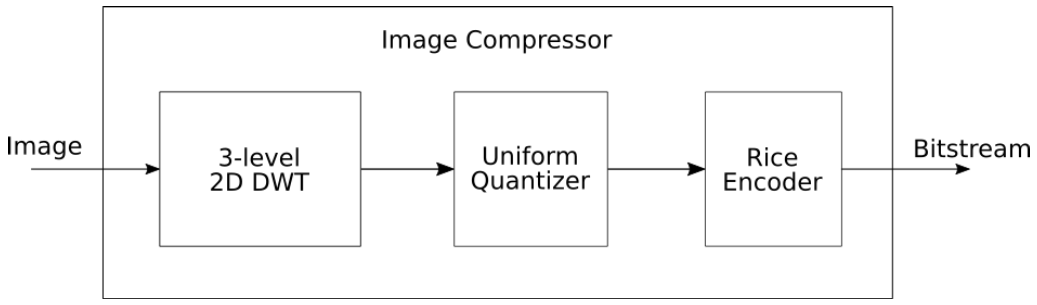


Figure 3.10.: EIVE image compressor - Architecture

It uses the 3-level 2D DWT for decorrelation of the image pixels, a simple uniform quantizer, which can perform quantization independently on each of the 10 subbands outputted by the DWT and a Rice encoder for entropy encoding of the decorrelated and quantized data.

4. Implementation

This chapter describes the implementation of the image compression algorithm, which has been selected in chapter 3 as a result of the modification and comparison of three already existing algorithms. First an overview of the hardware setup is given. Then the actual FPGA implementation is described on a high level of abstraction.

4.1. EIVE Payload On-board Computer

The EIVE mission includes a payload on-board computer (PLOC), which is responsible for controlling the operation and monitoring the status of all the payload components on the satellite bus. One of the uses cases for the EIVE is taking a snapshot of the earth by capturing a frame from a 4K video camera, which is one of the payload components. An overview of all the system components involved in this use case is given in figure 4.1.

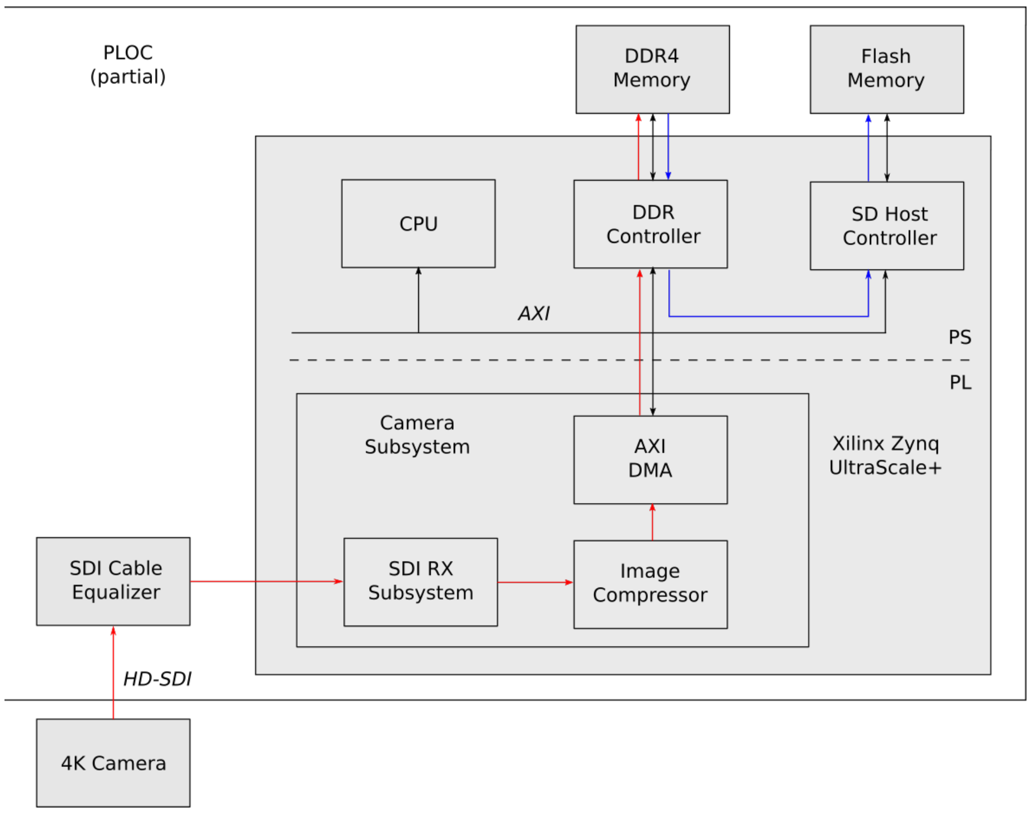


Figure 4.1.: EIVE camera subsystem - block diagram

The PLOC is implemented on a Xilinx Zynq UltraScale+ MPSoC. This system on a chip (SoC) consists of two major parts. One of them is the processing system (PS), which contains several CPUs and other peripheral components such as DDR or SD host controller. The other one is the programmable logic (PL), which essentially contains a FPGA with freely programmable logic cells and hard IP cores for specific purposes. Both parts of the SoC are closely connected by a high-performance bus architecture.
The 4K camera is connected to the PLOC by means of a high-speed serializer/deserializer (SerDes) interface. The data transmitted on this interface is formatted according to the SMPTE 292M standard (HD-SDI). The HD-SDI data is then processed by the camera subsystem implemented in the PL of the PLOC. The SDI RX Subsystem takes care of the protocol handling and extracts the image frames from the continuously incoming data stream. On reception of a command, the process of taking a snapshot is started by passing one single frame from the SDI RX subsystem to the image compressor. The image compressor compresses the image into a bitstream and outputs it to a DMA engine, which then transfers the data into RAM using DMA transfers. Once all data has been transferred to RAM, the CPU copies it to the non-volatile flash memory for subsequent downlink.

4.2. 4K Camera

The 4K camera is a compact and light-weight video camera, which is capable of producing high-quality and uncompressed video data. It provides several different configurations for outputting the video data on multiple interfaces. Unfortunately not all of the output interfaces can be configured to output video data at 4K resolution (3840 x 2160 pixel). Therefore the video data sent to the PLOC over the HD-SDI interface only has HD resolution (1920 x 1080 pixel). The most important parameters of the video data are summarized in table 4.2.

parameter	**value**
output mode	HD-SDI
optical format	1920 x 1080 pixel, 16:9
frame rate	30 fps
dynamic range	10 bit
color space	YC_bC_r
chroma subsampling	4:2:2

Table 4.1.: Video data parameters

The video data on the HD-SDI interface is outputted frame by frame. Each frame is again outputted line by line.

4.3. HDL Design

The image compressor has been divided into multiple smaller modules. Each of these modules fulfills one specific task in the compression process and only relevant information is passed from one module to the next. This division of the design was necessary in order to simplify the implementation and keep code complexity low.
The design of the image compressor allows it to work on input images with a generic number of rows and columns. However it only supports pixels with one component. Since the image data is provided in the YC_bC_r color space with 4:2:2 chrominance subsampling, three instances of the image compressor are used for compressing each of the color and luminance channels independently. Due to the subsampling, the image on the luminance channel has a resolution of 1920 x 1080 pixel, whereas the image on the chrominance channels only has a resolution of 960 x 1080 pixel. An overview of the HDL design is depicted in figure 4.2.

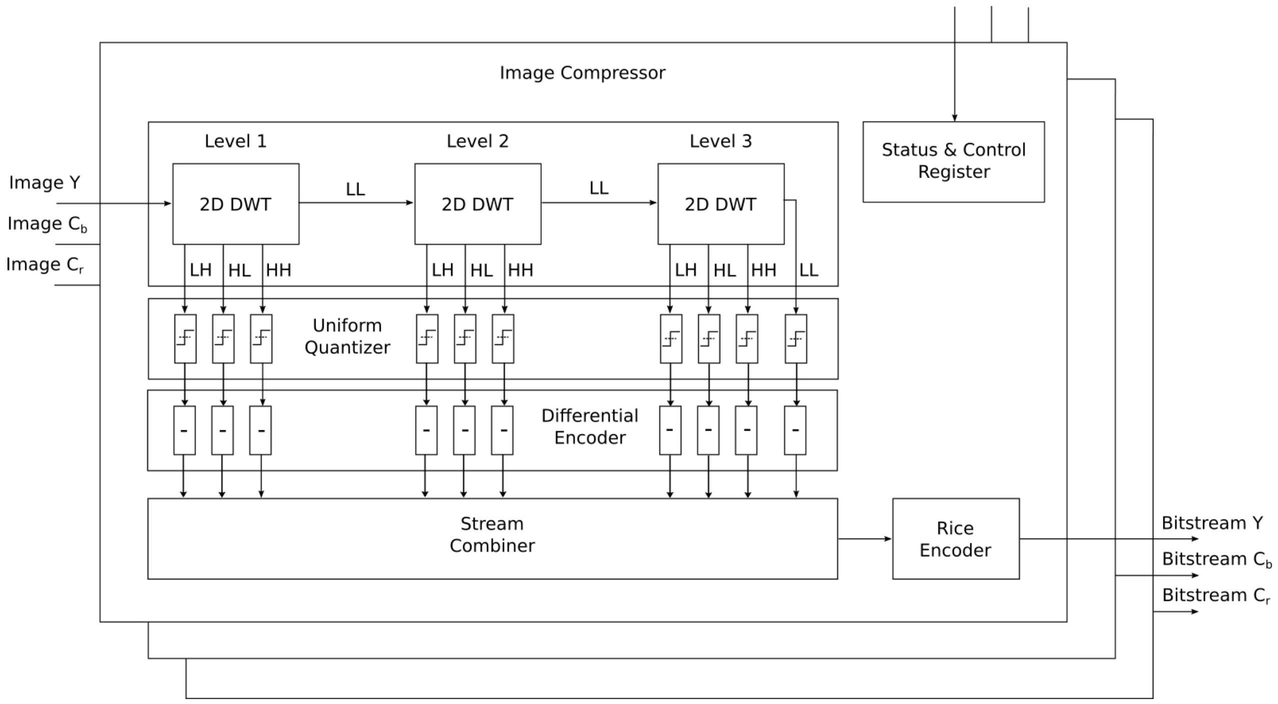


Figure 4.2.: HDL design - overview

The image is fed into the 2D DWT module on level 1, which outputs data for the four subbands (LL, LH, HL and HH). The LL data from level 1 is forwarded to the 2D DWT module on level 2. And finally the LL data from level 2 is forwarded to the 2D DWT module on level 3. After the 3-level two-dimensional DWT has been performed on the image each of the resulting 10 subbands is passed through an instance of a uniform quantizer. The 10 quantized subbands are differentially encoded and recombined into a single data stream, which is then processed by the Rice encoder. Status and control registers are implemented in an additional module, which is connected to the CPU via the internal bus architecture.

4.3.1. 2D DWT

Since the image is inputted in a row-by-row order and buffering the complete image inside the on-chip memory of the FPGA or the DDR RAM connected to the PS is not possible respectively compliant with the constraints on power consumption and execution time, a line-based 2D DWT has been implemented. A block diagram of the 2D DWT module is shown in figure 4.3.

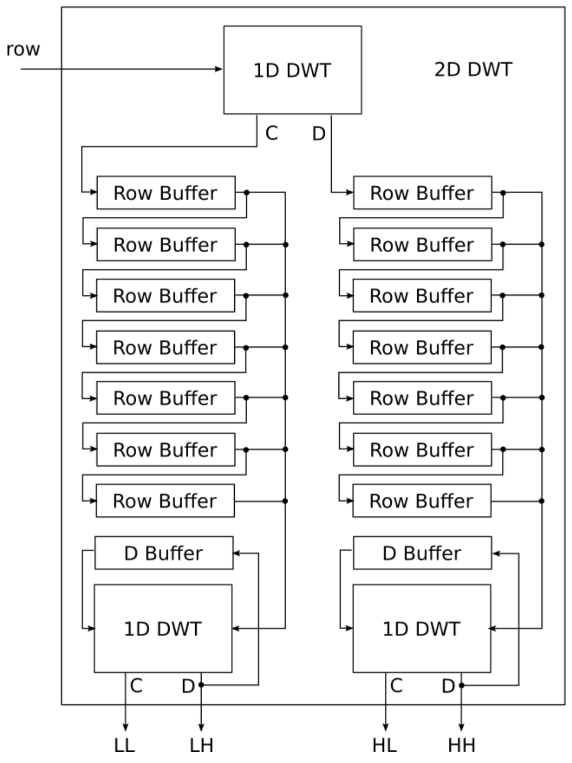


Figure 4.3.: Line based 2D DWT

It consists of three instances of the 1D DWT module, which is a direct implementation of the equations for the DWT coefficients given equations 3.4 and 3.5 and several FIFO buffers. The row buffers are being used to buffer the last seven rows of DWT coefficients outputted by the first 1D DWT module. This module performs a one-dimensional DWT on each row. The buffered 7 rows and the D coefficient of the previous row are then used for calculating the one-dimensional DWT for each column. Remember that this is possible, since the DWT only depends on the previous seven samples of the input sequence. Since the output sequences of low-pass (C) and high-pass (D) coefficients have a length equal to half the length of the input sequence $2N$, each FIFO buffer needs to have at least a capacity of N. Due to implementation constraints, the capacity of a FIFO also has to be a power of two. This results a minimum FIFO capacity of

$$c_{FIFO} = 2^{\lceil ln_2(1920/2) \rceil} = 1024 \tag{4.1}$$

entries. This results in a total memory complexity of 8192 entries. The actual required memory capacity additionally depends on the width of each of the entries and this again depends on the width of the input data.

4.3.2. Uniform Quantizer

The uniform quantizer implementation is kept as simple as possible. It divides the input sample by a power of 2 respectively shifts the input sample by the corresponding number of bits to the right. This approach prevents the necessity of complex logic for the implementation of usual division.

4.3.3. Differential Encoder

The differential encoder is used for transforming the distribution of the quantized DWT coefficients into a distribution, which is characterized by the fact that values with small magnitude occur more often than values with higher magnitude. The differential encoding is performed as a preprocessing step for the Rice encoder. It is implemented by a simple difference calculation between the current input sample and the previous sample.

4.3.4. Stream Combiner

The 3-level tow-dimensional DWT outputs 10 subband data streams in parallel. This represents a problem in terms of data rate, since it means that for a short period the peak data rate could be 10 times as high as the average data rate of one sample per clock cycle, which is the targeted throughput of the complete design. For solving this issue the stream combiner includes FIFO buffers for each of the subbands, which have enough capacity to buffer the periods with peak data rate. Also each level of the two-dimensional DWT outputs data at a different rate. This is due to the fact, that each two-dimensional DWT divides the input image into four subbands respectively output images with only one fourth of the size of the input image. On top of that, the delay introduced by the row-by-row input order of the image as mentioned in section 3.4 propagates through the three levels of the two-dimensional DWT and also delays the output data of the different levels.
In order to compensate for the different data rates and output delays of the 3-level two-dimensional DWT and also to provide a known structure of the data words in the compressed image respectively bitstream, which is a necessary condition for the image reconstruction process, an arbiter has been implemented in the stream combiner. This arbiter has five different states and starts with outputting only L1 data as shown in figure 4.4. In state out1 only L1 data is outputted with equal priority. Once data equal to five rows of L1 data has been outputted, the arbiter changes into state out2 (shown in figure 4.5. In this state not only L1 data but also L2 data is available and will be outputted. Since the data rate of the L2 data is one fourth of that of the L1 data, the arbiter outputs four words of L1 data, then one word of L2 and again four words of L1 data. State out2 lasts until an additional 10 rows L1 data and 5 rows of L2 data have been outputted. The arbiter then changes into state out3, which outputs L1, L2 and L3 data. The arbitration scheme of state out3 is shown in figure 4.6. Since L3 has again only one fourth of the data rate of L2 data, the arbiter outputs 16 L1 data

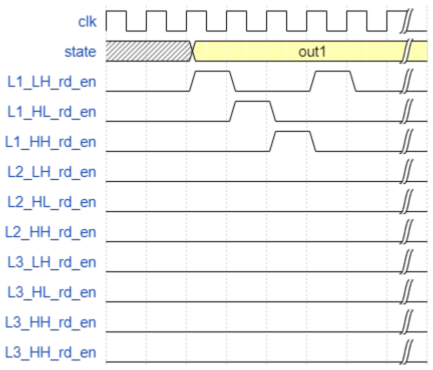


Figure 4.4.: Stream Combiner - Arbitration state 1

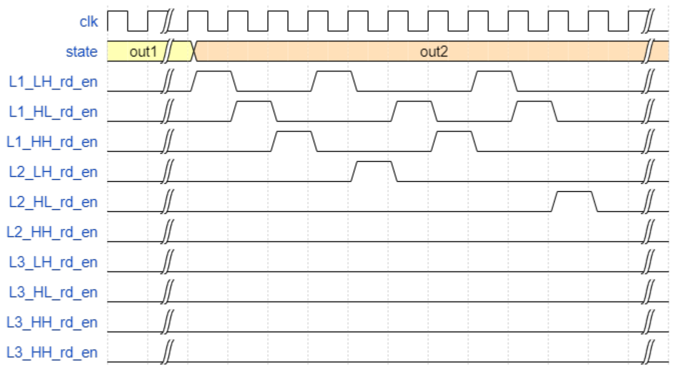


Figure 4.5.: Stream Combiner - Arbitration state 2

words, then four L2 data words and lastly one L3 data word. An additional fourth subband LL is present in the L3 data stream, which is not present in the L1 and L2 data streams. This requires that after the L3 HL data word has been outputted an addition L3 LL data words is outputted before the described arbitration scheme is continued. This ensures that the data rate of the L3 data stream is properly incorporated by the arbiter. State out3 is used for outputting the majority of the data. For the last two rows of L2 data and second last row of L3 data the arbiter changes to state out4,

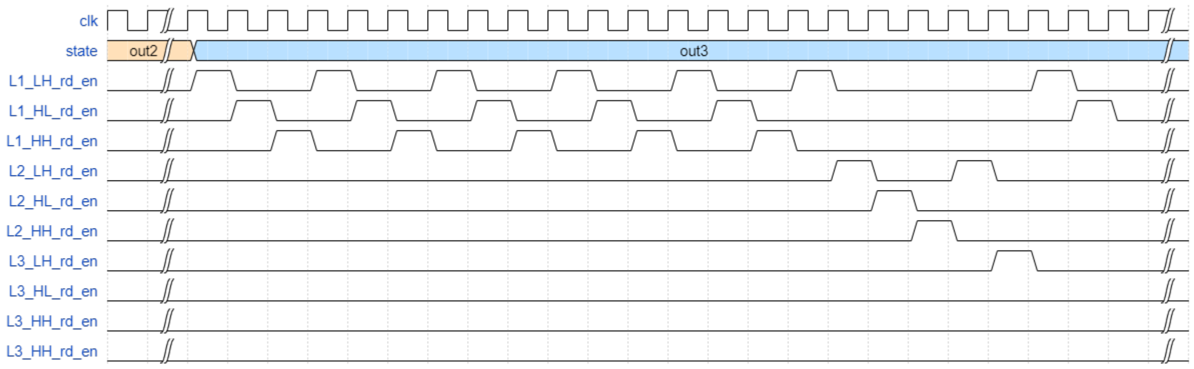


Figure 4.6.: Stream Combiner - Arbitration state 3

which has the same arbitration scheme as state out2. The fifth and last state is out5. In this state the last row of L3 data is outputted with the same arbitration scheme as in state out1 expect now for four subbands instead of three.

4.3.5. Rice Encoder

The Rice encoder is a direct implementation of equations 2.7 and 2.8 for the absolute value of the input data word. A subsequent logic block uses the quotient and remainder values for constructing the variable length code word. One important consequence of using a variable length code is, that one code word can be longer than the available output port width and thus requires more than one clock cycle to be outputted. This blocks the processing of the next incoming data word. As a result a FIFO buffer has been included at the input of the Rice encoder to allow for outputting longer code words with more than one clock cycle. In order to prevent a FIFO overflow, the Rice encoder operates at a clock frequency, which is twice as high as the input clock frequency. This allows the average output code word to take two clock cycles for outputting.

4.3.6. Resource Utilization

One of the three instances of the image compressor architecture has been implemented on a Xilinx Zynq UltraScale+ (xczu9eg-ffvc900-2-i) device using the standard Vivado 2017.4 tool chain with Flow_PerfOptimized_high and Performance_Explore strategies for synthesis respectively implementation. This instance is configured for an input image with 1920 x 1080 pixel resolution, a dynamic range of 10 bit for each pixel and is connected to the luminance (Y) output channel of the SDI RX subsystem. Frequency constraint for the input clock is 74.25 MHz and for the output clock (used for Rice encoder) is 148.5 MHz. The following resource utilization was found:

parameter	**value**
CLB LUTs	16275 (5.9%)
CLB REGs	9443 (1.7%)
CARRY8	801 (2.3%)
CLB	2899 (8.5%)
LUT as Logic	16275 (5.9%)
LUT Flip Flop Pairs	6753 (2.5%)
Block RAM Tile	46.5 (5.1%)
Global Clock Buffers	1 (0.02%)

Table 4.2.: EIVE Image Compressor - resource utilization

A naive approach for estimating the resource utilization of three instances of the image compressor, as needed for compressing all three YC_bC_r channels, would be to simply multiply the utilization given in table 4.2 by a factor of two. The factor two stems from the fact that each of the C_b and C_r channels only has half the resolution of the Y channel. But additional logic, which does not scale with a factor of two but three, would not be incorporated in this estimation. Therefore, the actual utilization might be somewhere between two and three times the utilization given in table 4.2.

4.4. Verification

In order to verify that the implementation of the image compression algorithm in the HDL design is correct a behavioral simulation has been performed. The test bench used in this simulation generates random input data and feeds it into the image compressor. The input data as well as the produced output data is recorded in an individual file for further processing using the Octave scripts used for the comparison in chapter 3. The test bench and associated verification processes are shown in in figure 4.7. The recorded input data is processed by the script for

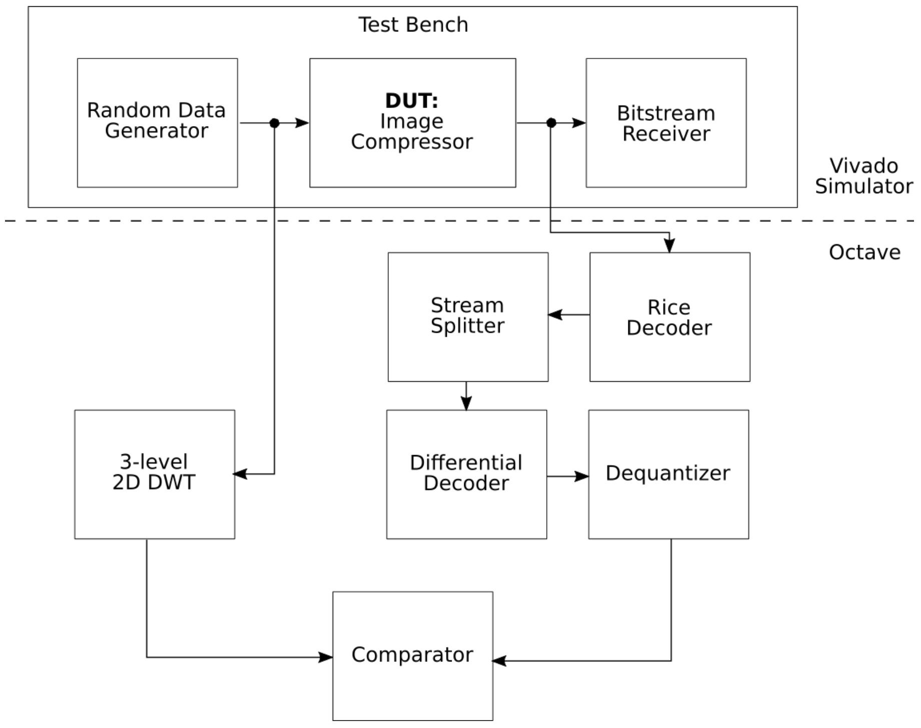


Figure 4.7.: Verification process for HDL design

the 3-level two-dimensional DWT. On the other side the recorded output data passes through a Rice decoder, a stream splitter, a differential decoder and a dequantizer, which each reverse the corresponding processing step performed in the image compressor HDL design and at the end output the reconstructed coefficients of the 3-level two-dimensional DWT. Finally a comparator ensures that each bit of the DWT coefficients belonging to the input data and the output data is the same.

5. Evaluation

In this chapter the performance of the implemented image compression algorithm is evaluated. For this purpose the simulation results are compared to the performance results given in an informational report for the CCSDS 122.0-B-2 image compression algorithm. Aside from that, the performance of the actual hardware implementation is determined and compared to the simulation results.

5.1. Comparison to CCSDS 122.0-B-2

The informational report for the CCSDS 122.0-B-2 image compression algorithm [3] provides additional information on compression options, implementation issues and performance results. The performance results given in the report have been generated by running a software implementation of the CCSDS 122.0-B-2 image compression algorithm on a set of black and white test images. The simulation of the implemented image compression algorithm is limited to lossless test cases, since detailed performance data for lossy compression with CCSDS 122.0-B-2 is only available for very low bit rates (between 0.25 and 2 bit/pixel). Due to the design of the Rice encoder the minimum bit rate, which can be achieved with the implemented image compressor, is 2 bit/pixel and therefore comparison between lossy compression is not reasonable. On top of that lossless compression represents the most important and probably most often used setting in context of the EIVE mission. In order to get a more general result, test images with different resolutions, bit depths and content have been used. The used test images can be found in the appendix B. The results from the report as well as the simulation results for the implemented image compression algorithm are given in table 5.1.

		compressed bit rate [bit/pixel]	
bit depth [bit]	**image**	**CCSDS 122.0-B-2**	**EIVE image compressor**
8	`spot-la_panchr`	4.27	4.98
8	`marstest`	4.78	5.68
8	`lunar`	4.58	5.77
10	`india_2kb1`	4.77	6.45
10	`marseille_G6_10b`	6.74	7.40
10	`ice_2kb1`	4.78	5.93

Table 5.1.: Lossless compression performance

As can be seen from the results, the lossless compression performance of the EIVE image compressor is between 17% to 35% worse than that of the CCSDS 122.0-B-2 compressor, but compared to the bit depth of the test image (8 bit or 10 bit), the lossless compression performance ranges between 62.22% to 74%. However, it is more dependent on the the type of input data. This becomes obvious when the results for the images `marstest` and `ice_2kb1` are compared. Considering the reduced complexity of the Rice encoder a worse performance was expected but the savings in terms of implementation time and resources justify this approach.

5.2. Performance of Hardware Implementation

In order to evaluate the compression performance of the actual hardware implementation, a test scene has been setup and images with different compression settings have been recorded. The test scene is shown in figure 5.1. Only one instance of the image compressor has been implemented

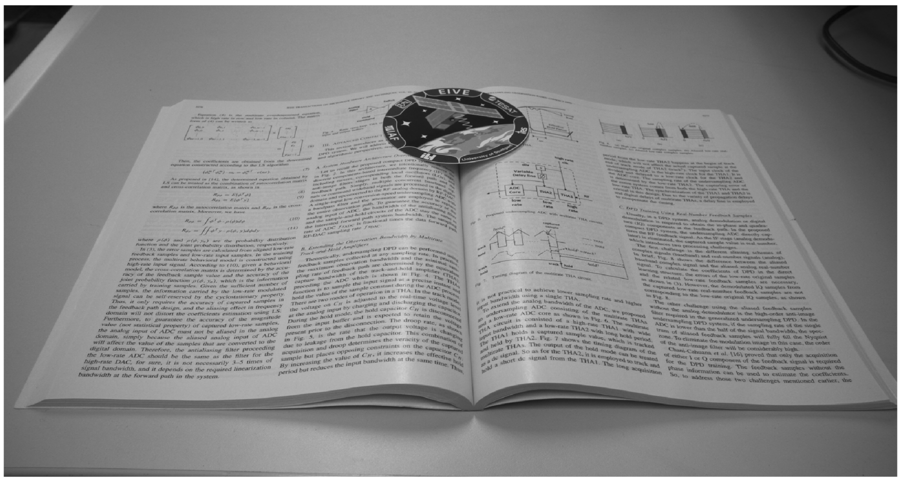


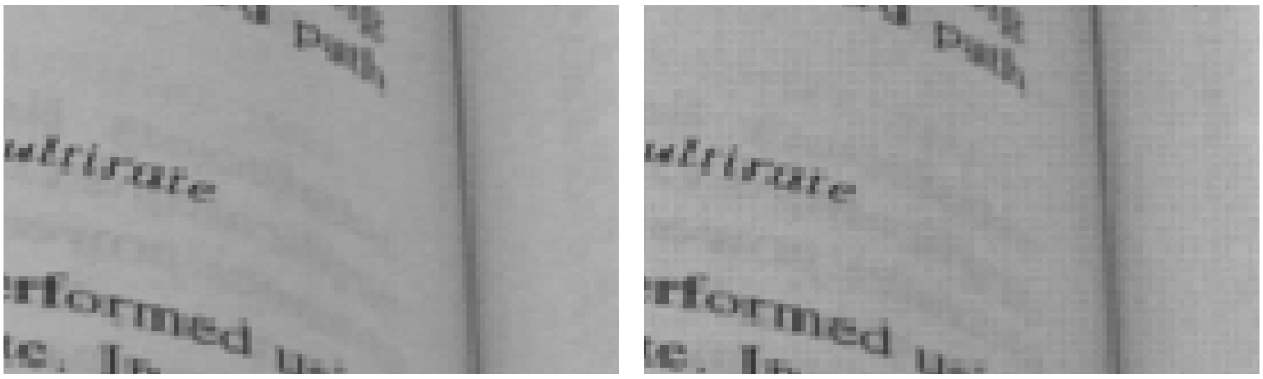
Figure 5.1.: Test scene - reconstructed image (lossless)

in hardware and connected to the luminance channel of the 4K camera, therefore only black and white images have been recorded. The bit rate and perception based image quality evaluator (PIQE) values for the recorded images are given in table 5.2.

bit depth [bit]	compression settings	bit rate [bit/pixel]	PIQE
10	K=4, L1=1, L2=1, L3=1	6.86	19.54
10	K=3, L1=2, L2=1, L3=1	6.29	19.38
10	K=2, L1=4, L2=2, L3=1	5.52	18.35
10	K=1, L1=8, L2=4, L3=2	4.64	16.87
10	K=1, L1=16, L2=4, L3=2	4.28	16.37
10	K=1, L1=16, L2=8, L3=4	3.70	15.08

Table 5.2.: Compression performance - EIVE image compressor

The bite rate of 6.86 *bit*/*pixel* for the lossless case (K=4, L1=1, L2=1, L3=1) is as expected in the value range of the simulation results given in table 5.1. With increased quantization on the levels L1, L2 and L3 the bit rate can be further reduced. The image quality indicated by the PIQE values is excellent ($PIQE \in [0, 20]$). However, the image quality seems to increase with a reduction of the bit rate. A possible reason for this might be an inconsistency in the PIQE metric or a reduction of the noise contained in the image by increased quantization. Comparing the lossless and most quantized images (see figure 5.2) it becomes obvious that the noise has increased with the reduction of the bit rate.



(a) Test Scene (zoomed) - K=4, L1=1, L2=1, L3=1 **(b)** Test Scene (zoomed) - K=1, L1=16, L2=8, L3=4

Figure 5.2.: Noise in lossless and quantized image

This means that the PQIE does not reflect the increased noise in the quantized image. The reason for this behavior is hard to determine, because the PQIE is a complicated metric, which is not only based on simple calculations as for example the MSE or PSNR are, but it relies on a rather complex algorithm and also a reference image set. As a consequence, the observation of increasing image quality with reduced bit rate is doubtful and should not be interpreted as a general feature of the implemented image compression algorithm. All images recorded during the hardware evaluation can be found in the appendix.

6. Conclusion and Outlook

Based on the analysis of power consumption, rate-distortion performance and execution time one out of three commonly used image compression algorithms has been selected as a basis for the implementation of the EIVE image compressor. The encoder part of this base algorithm has been modified in order to ease implementation while keeping the performance at a level comparable to the base algorithm. The modified image compression algorithm has been implemented on a Xilinx Zynq UltraScale+ MPSoC device and its performance was evaluated.
The results show that the EIVE image compressor can keep up with its base algorithm (CCSDS 122.0-B-2) in terms of compression effectiveness by being just 17% to 35% less effective in lossless compression depending on the input image. Due to the fact that the bit plane encoder of the base algorithm is replaced by a Rice encoder in the EIVE image compressor, its computational complexity is significantly less compared to its base algorithm resulting in reduced power consumption and implementation effort. But this comes at the cost of a worse rate-distortion performance, since the Rice encoder is by far not as efficient as the bit plane encoder in terms of compression/bit rate. This becomes obvious when the minimum achievable bit rates are compared: the Rice encoder has a minimum asymptotic bit rate of 4 *bit/pixel* (YC_bC_r color space, 4:2:2 chroma subsampling) while the bit plane encoder's minimum asymptotic bit rate approaches 0 *bit/pixel* when the number of compressed pixels tends to infinity. The execution time is mainly determined by the time it takes to receive the input image over the SDI interface from the 4K camera and only the equivalent time of receiving about 8 image lines on the SDI interface is added to the execution time due to internal buffering.

To sum up, the EIVE image compressor represents a good compromise between complexity and efficiency but there is still some room for further improvements:
Quantizer: The current implementation of the EIVE image compressor uses a simple uniform quantizer. Unfortunately no proper rounding is performed by this quantizer since it simply shifts each x bits to the right. This means, that for an assumed quantization factor of 4 the values (20, 21, 22, 23) would all be mapped to the same quantized value (5, 5, 5, 5). This introduces additional noise to the image and therefore impairs the rate-distortion performance of the EIVE image compressor. A simple fix for this problem is to simply add half the quantization factor to the input values before they are shifted to the right. This would result in the following mapping for the previous example sequence: (20 + 2, 21 + 2, 22 + 2, 23 + 2) mapped to (5, 5, 6, 6).
Rice encoder: As already mentioned, the minimum achievable asymptotic bit rate of the Rice encoder is 4 *bit/pixel*. This limits also the minimum bit rate for a compressed image and is due to

the fact, that each image component (*YCbCr*) can only be encoded with a minimum of two bits: One bit for the remainder and one stop bit for the variable-length code. One way to further reduce the minimum achievable bit rate of the entropy encoder would be using a combination of a Rice encoder and a run-length encoder, where the run-length encoder is used to encode runs of zeros with a length of more then three symbols and all other symbols are encoded by the Rice encoder. This can be done without adding any additional control symbols for indicating that the current value in the bitstream does not represent an encoded symbol but a run-length simply by the convention that after each run of three consecutive zeros the number of following zeros is written to the bitstream even if it is a zero length run. By doing so the bit rate can be further decreased especially for highly quantized images.

FIFO buffers: It might be possible to further reduce the depth of some of the FIFO buffers without risking overflow conditions. This would reduce the resource utilization and also the susceptibility for radiation effects of the HDL design.

DWT implementation: The DWT has been directly implemented based on equations 3.4 and 3.5. This might not be the optimal way to do it. Maybe using a dedicated FIR filter architecture is more efficient in terms of resource utilization. The synthesis tool might not be able to recognize the structure described in the equations as a FIR filter, since it also includes the boundary conditions for the DWT filter as dedicated equations. However, implementing the logic for the boundary conditions of filters might eat up any improvements gained by the use of an FIR filter.

Signed/Unsigned input of DWT: In order to keep the HDL design as generic as possible, the input of the DWT has not been restricted to unsigned integers only. The current implementation also supports signed integers as an input for the DWT. This leads to a reduction of the overall compression performance as early hardware test revealed. The problem is based on the two's complement interpretation of the unsigned input data, which potentially results in much greater difference values between two consecutive input samples. For example, the 10 bit input sample sequence (512, 511) has a corresponding difference of $1000000000(b) - 0111111111(b) = 512(dec) - 511(dec) = 1$ between the two samples when the 10 bit input samples are interpreted as unsigned integers but the difference $-512 - 511 = -1023$ is much bigger when the 10 bit input samples are interpreted as signed integers. This problem has already been confirmed by using the Octave simulation scripts and behavioral simulation of the HDL design. A quick fix has been implemented by simply extending the input range of the DWT modules to 11 bit signed integer but this of course is not the optimal solution to this problem. The DWT modules should be changed to only accept unsigned integers as input values.

Choosing the optimum K value: The Rice encoder implemented in the EIVE image compressor needs to be configured with a divisor value K. This divisor has to be a power of two and is used to divide the input sample into the remainder and rounded quotient. Depending on the distribution of the input data an optimum value for the K parameter exists, which minimizes the size of the compressed image/bitstream. Currently no algorithm has been implemented in order to determine the optimum K value for an input image. The most simple solution would be executing the image compression algorithm multiple times with different K values and only keeping the bitstream containing the compressed image, if it is smaller than the previous one. But this method is most likely also not the most efficient method to deal with this problem.

Bibliography

[1] *3 Level DWT - Filter Bank.* online: https://upload.wikimedia.org/wikipedia/commons/2/22/Wavelets_-_Filter_Bank.png, 2005. – accessed 12.01.2021

[2] Allen, R.: *Signal Analysis : Time, Frequency, Scale, and Structure.* 2004. – ISBN 0471234419

[3] CCSDS: *Image Data Compression - Informational Report: CCSDS 120.1-G-2.* online: https://public.ccsds.org/Pubs/120x1g2.pdf, 02 2015. – accessed 12.01.2021

[4] CCSDS: *Image Data Compression - Recommended Standard: CCSDS 122.0-B-2.* online: https://public.ccsds.org/Pubs/122x0b2.pdf, 92 2017. – accessed 12.01.2021

[5] CCSDS: *Low-Complexity Lossless and Near-Lossless Multispectral and Hyperspectral Image Compression- Recommended Standard: CCSDS 123.0-B-2.* online: https://public.ccsds.org/Pubs/123x0b2c2.pdf, 92 2017. – accessed 12.01.2021

[6] David, Salomon: *Data compression - The Complete Reference, 4th Edition.* 2007. – ISBN 978–1–84628–602–5

[7] Douglas, Issue: Chrominance Subsampling in Digital Images. In: *The Pumpkin* 1 (2009), 01

[8] Image, The U. f. ; Engineering, Video: *LIVE Image Quality Assessment Database Release 2.* https://live.ece.utexas.edu/research/quality. Version: 2005. – accessed 28.03.2021

[9] ITU: *ISO/IEC 10918-1 : 1993(E) CCIT Recommendation T.81.* http://www.w3.org/Graphics/JPEG/itu-t81.pdf. Version: 1993

[10] ITUT-R: *Recommendation ITU-R P.676-10 Attenuation by atmospheric gases.* online: https://www.itu.int/dms_pubrec/itu-r/rec/p/R-REC-P.676-10-201309-S!!PDF-E.pdf, 09 2013. – accessed 08.12.2020

[11] Khalid, Sayood: *Introduction to Data Compression.* Third. San Francisco, CA, USA : Morgan Kaufmann Publishers, 2005

[12] Mike Wakin, University of M.: *lena512color.tiff.* online: https://www.ece.rice.edu/~wakin/images/lena512color.tiff, 01 2013. – accessed 03.01.2021

[13] Schoch, B. ; Chartier, S. ; Mohr, U. ; Koller, M. ; Klinkner, S. ; Kallfass, I.: Towards a CubeSat Mission for a Wideband Data Transmission in E-Band. In: *2020 IEEE Space Hardware and Radio Conference (SHaRC)*, 2020, S. 16–19

[14] Shannon, C. E.: A mathematical theory of communication. In: *The Bell System Technical Journal* 27 (1948), Nr. 3, S. 379–423. http://dx.doi.org/10.1002/j.1538-7305.1948.tb01338.x. – DOI 10.1002/j.1538–7305.1948.tb01338.x

[15] The MathWorks, Inc.: *PIQE Function Description*. `https://www.mathworks.com/help/images/ref/piqe.html`. – accessed 28.03.2021

[16] The MathWorks, Inc.: *QMF Function Description*. online: `https://www.mathworks.com/help/wavelet/ref/qmf.html`, 2021. – accessed 31.03.2021

[17] Venkatanath N ; Praneeth D ; Maruthi Chandrasekhar Bh ; Channappayya, S. S. ; Medasani, S. S.: Blind image quality evaluation using perception based features. In: *2015 Twenty First National Conference on Communications (NCC)*, 2015, S. 1–6

7. Appendix

A. List of Abbreviations

Abbreviation	Meaning
AXI	Advanced Extensible Interface
CCITT	Comité Consultatif International Télégraphique et Téléphonique
CCSDS	Consultative Committee for Space Data Systems
CPU	Central Processing Unit
DCT	Discrete Cosine Transform
DDR	Double Data Rate
DFT	Discrete Fourier Transform
DMA	Direct Memory Access
DWT	Discrete Wavelet Transform
EIVE	Exploratory In-orbit Verification of an E-band link
FDCT	Forward DCT
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
HD	High Definition
I.I.D.	Independent Identically Distributed
JPEG	Joint Photographic Experts Group
LUT	Look Up Table
MAE	Mean Absolute Error
MSE	Mean Squared Error
PLOC	Payload On-Board Computer
PMF	Probability Mass Function
PIQE	Perception based image quality Evaluator
PL	Programmable Logic
PS	Processing System
PSNR	Peak Signal to Noise Ratio
QMF	Quadrature Mirror Filter
RGB	Red Green Blue
RX	Receiver
SDI	Serial Digital Interface
SoC	System on Chip
TX	Transmitter

B. Test Images

The following test images have been used for performance evaluation of the implemented image compression algorithm in section 5.1:

image	**bit depth [bit]**	**resolution**
`marstest`	8	512 x 512
`lunar`	8	512 x 512
`spot-la_panchr`	8	1000 x 1000
`ice_2kb1`	10	2048 x 2048
`india_2kb1`	10	2048 x 2048
`marseille_G6_10b`	10	528 x 1856

Table B.1.: Test images - parameters

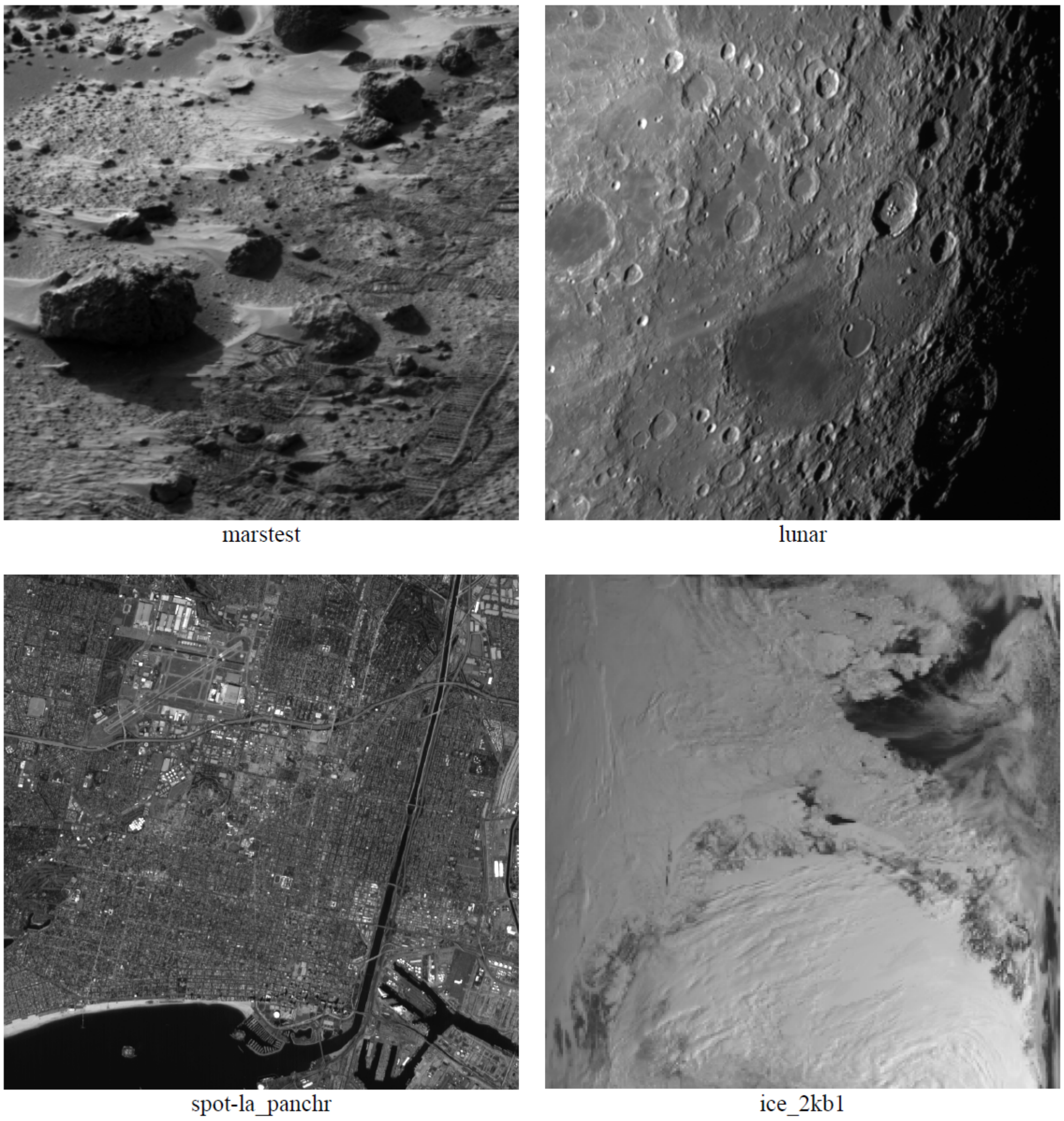


Figure B.1.: Test images CCSDS 122.0-B-2, 1

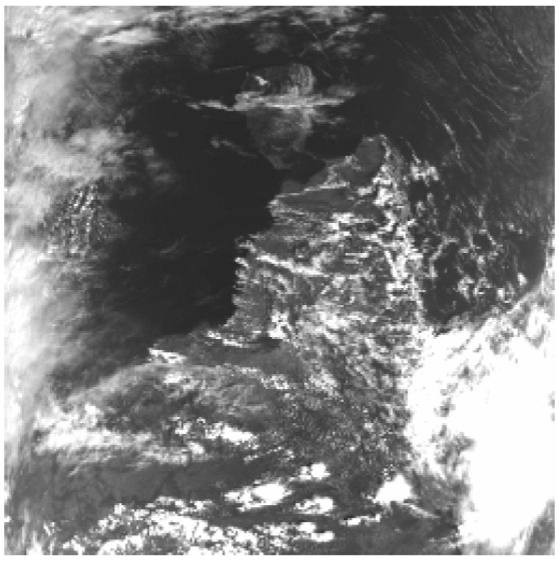


Figure B.2.: Test images CCSDS 122.0-B-2, 2

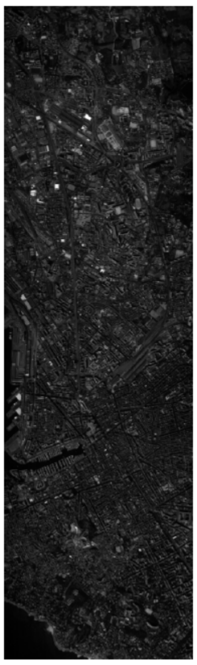


Figure B.3.: Test images CCSDS 122.0-B-2, 3

C. Performance of Hardware Implementation

This section contains all images recorded during the performance evaluation of the hardware implementation of the EIVE image compressor (see next pages).

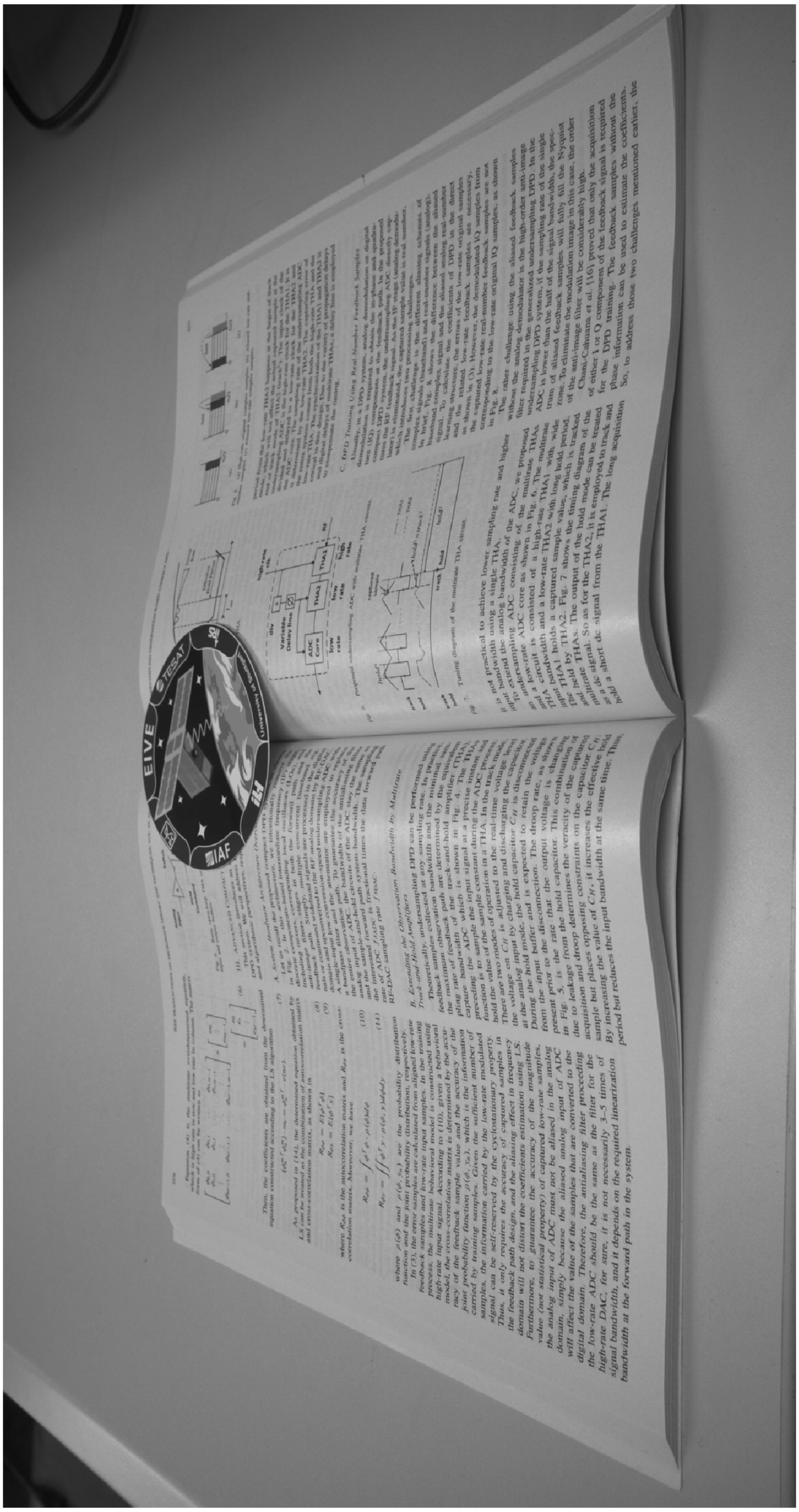


Figure C.4.: Test scene - 10bit, K=4, L1=1, L2=1, L3=1

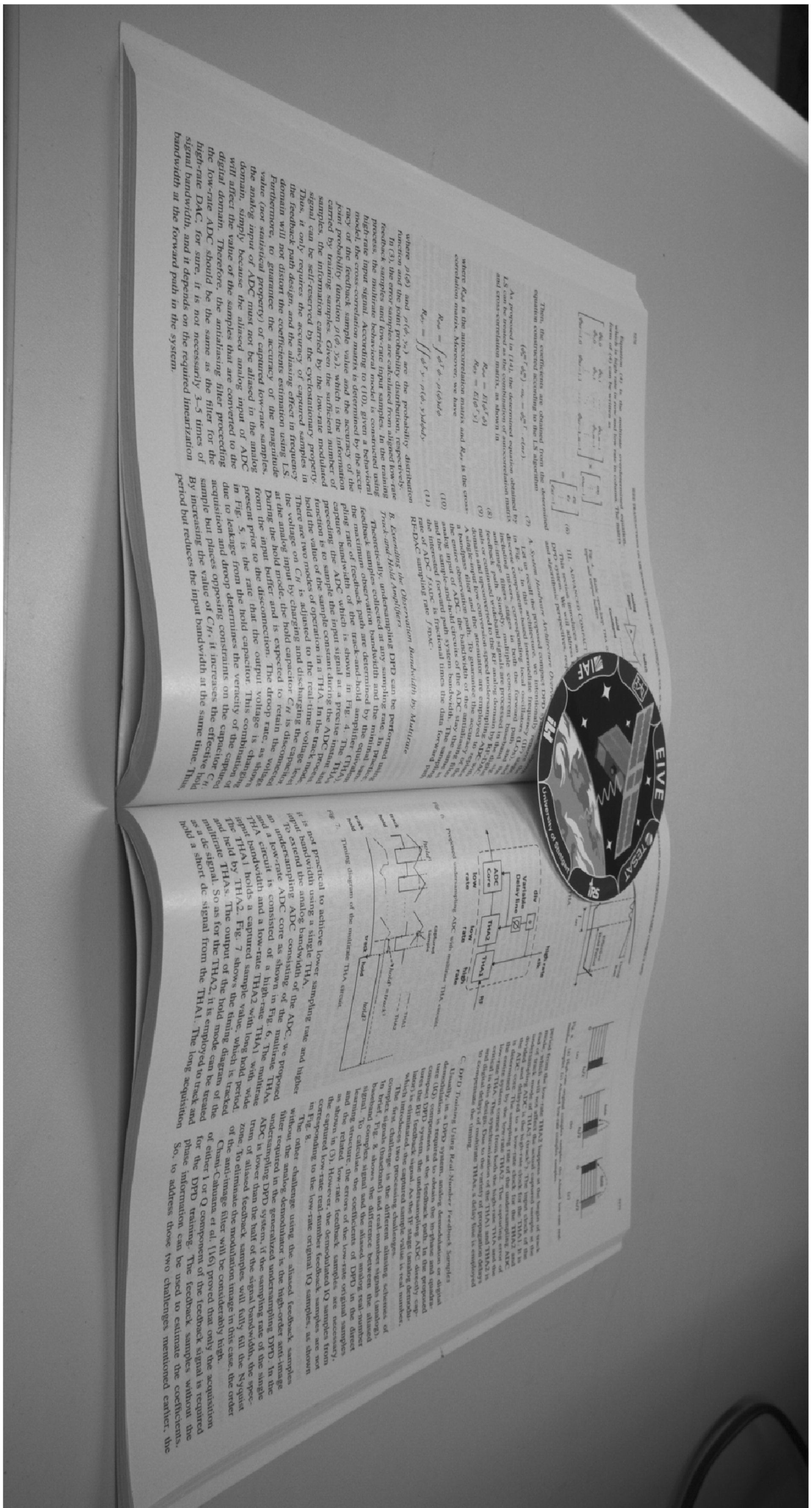


Figure C.5.: Test scene - 10bit, K=3, L1=2, L2=1, L3=1

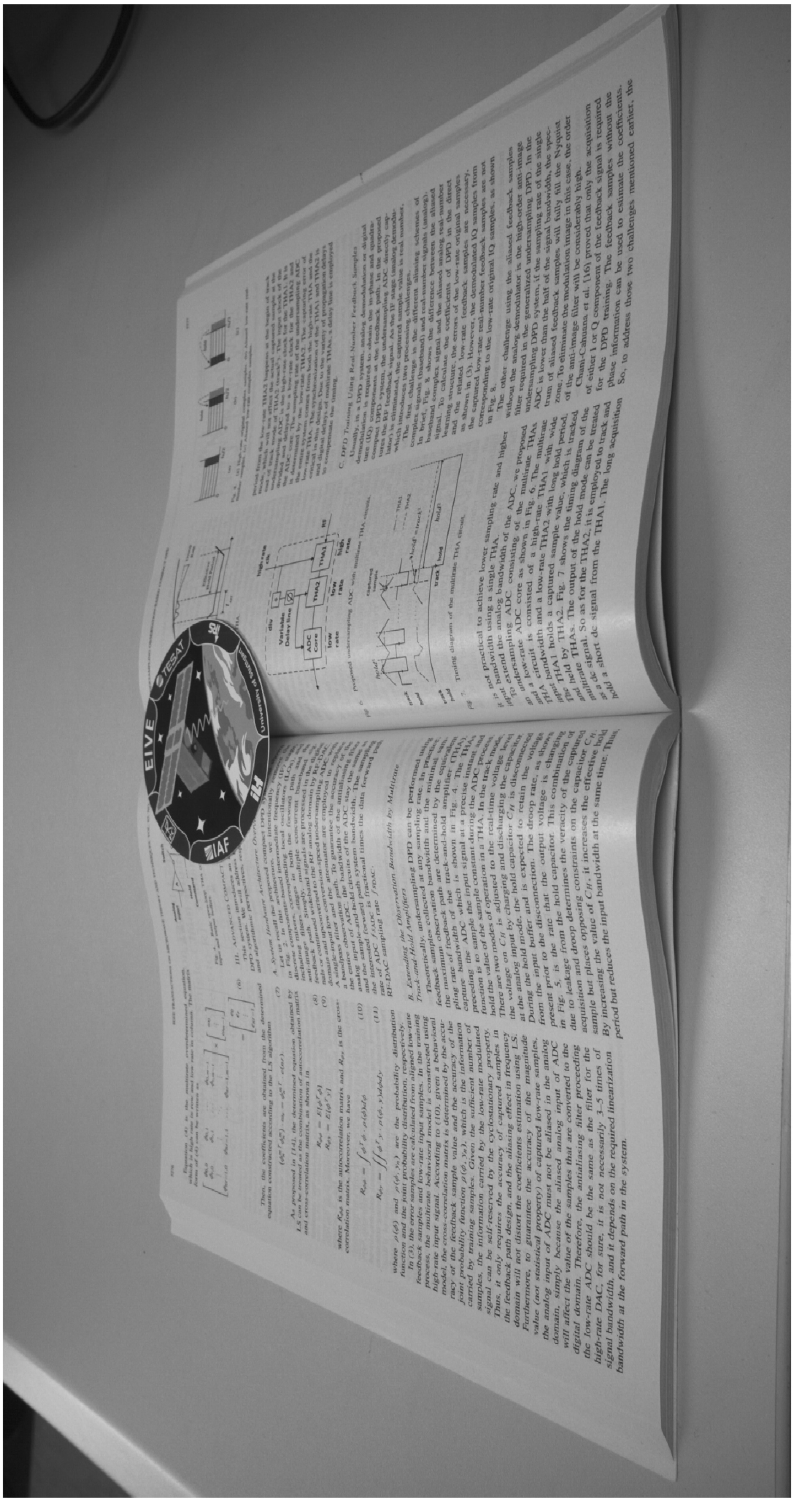


Figure C.6.: Test scene - 10bit, K=2, L1=4, L2=2, L3=1

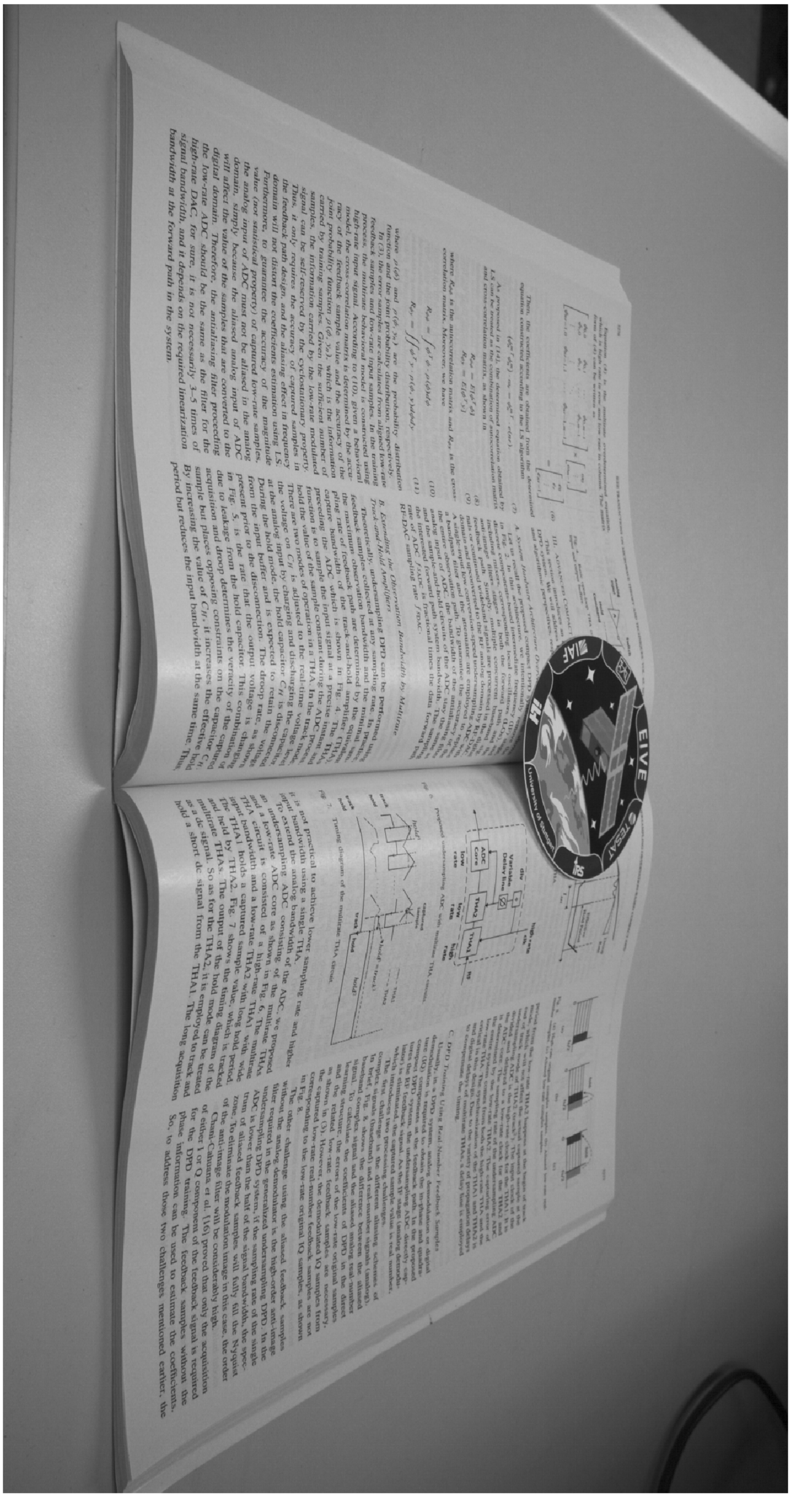


Figure C.7.: Test scene - 10bit, K=1, L1=8, L2=4, L3=2

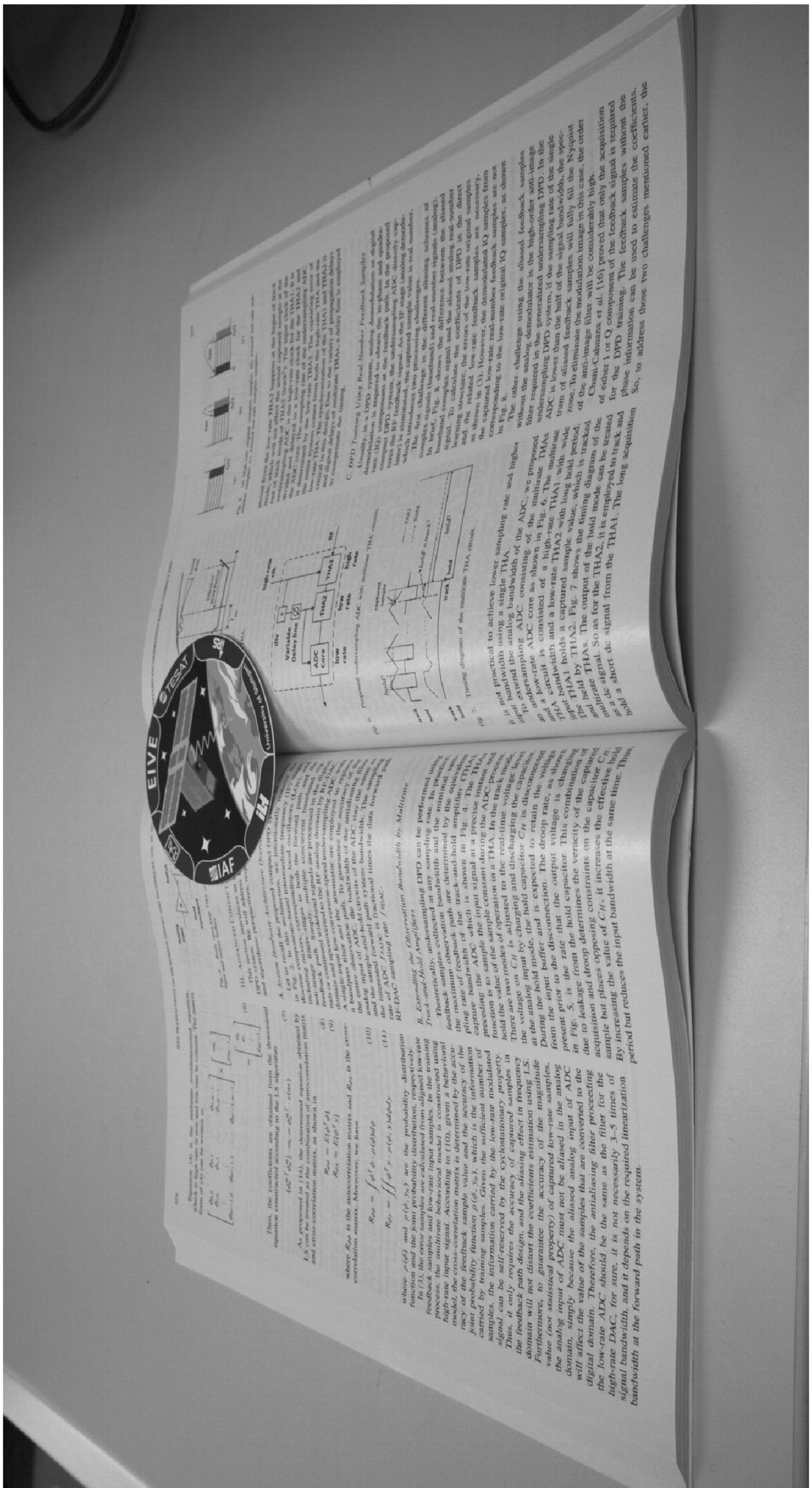


Figure C.8.: Test scene - 10bit, K=1, L1=16, L2=4, L3=2

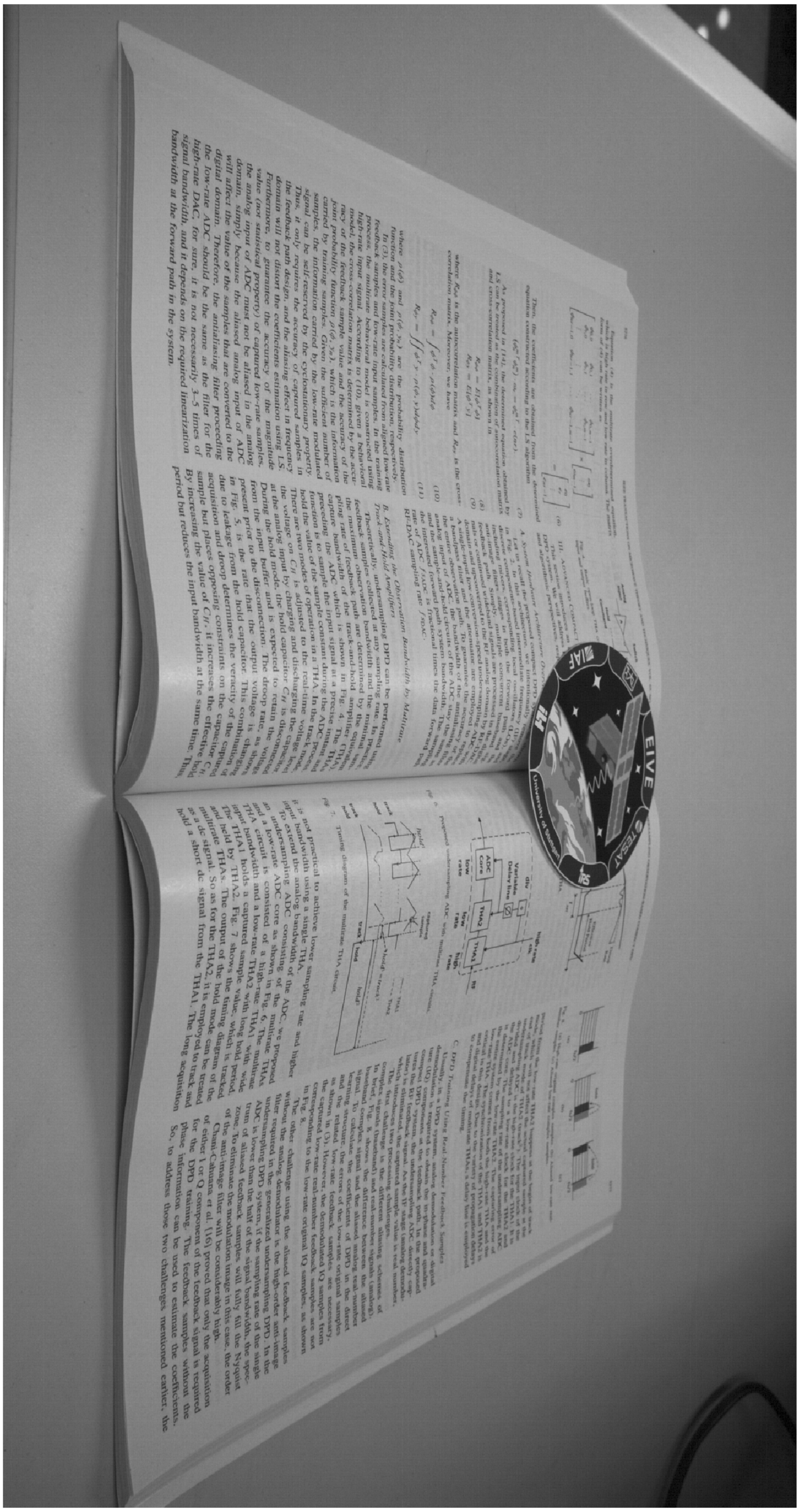


Figure C.9.: Test scene - 10bit, K=1, L1=16, L2=8, L3=4