POLITECNICO DI TORINO

Master degree course in Electronic Engineering

Master Degree Thesis

Optimized VLSI architectures for bilateral filtering in future video coding

Supervisor
prof. Maurizio Martina

Candidate
Marco Lavena

Accademic Year 2017-2018

Summary

By 2021 it is expected that 82% of total IP traffic will be IP video traffic. This percentage includes live internet video, video surveillance, video-on-demand and internet TV. It must also be considered that, following the current trend, the demand for higher video resolutions and a better overall quality is expected in the future. As well as expected an increase in traffic on mobile devices. All this can not be sustained without considerable technological progress in video encoding in order to maximize data compression.

To address this problem over the years, different research groups like ISO/IEC and ITU-T have proposed innovative video coding standards. In 2013 an ISO/IEC and ITU-T joint group called JCTVC released HEVC standard which marked a new frontier in video coding. Since then, the two groups have joined again in a new workgroup named the Joint Video Exploration Team to lay the foundations for the study of a new standard capable of achieving even higher performance and overcoming the technological challenge. This is how the Joint Exploration Model (JEM) was born, which conveys all the efforts and ideas for improvement.

The inclusion of a particular processing unit in the filtering stage has been proposed for the elimination of artefacts due to the compression of the information. This level of frame processing is called Bilateral Filter, which performs an average of the intensity of the pixels, weighed on the geometric and photometric distance among them and preserves the edges. Numerically it acts as an High-Pass filter, suppressing small intensity variations and preserving bigger ones. It is a topic that has only recently been considered to be implemented in the new model, although its original formulation dates back to 1998. As a consequence, it has aspects that are still unexplored, for which this work undertakes to analyze.

Indeed this thesis aims to study an architectural solution for the filter in order to obtain the target performance that allows real-time video processing of an at least 60 frames per second sequence at FHD (1920 x 1080) and UHD4K (3840 x 2160) resolutions. Starting from previous work, the design has been optimized in terms of used resources reduction. In particular the task has been focused on memories involved into the architecture with the purpose of better values compression and area freeing. The main result has been to evolve from a value-based LUT for every filter weights, to an index-based one that replaces numerical values with addresses.

Final weights are then obtained through comparisons between memory input indexes and cells from the new one. This suggests a theoretical size compression of 63.9%. Some further actions have been done in order to decrease resources usage in other smaller LUTs present inside the architecture.

Found solutions, synthesized with commercial purpose UMC Low Leakage 65 nm library, led to a memory area reduction up to 61.7% and a full architectural one up to 42.9%. On the other hand, maximum frequency reached 1.47 GHz (1.47 GPixel/s), with an increment of 11.8% respect to original performance, widely satisfying FHD and UHD4K requirements. An advanced filter architecture able to double the throughput has been also proposed, with a maximum clock of 1.37 GHz (2.74 GPixel/s) and and area just 54.6% bigger.

Acknowledgements

Al professore Maurizio Martina, per l'infinita cortesia e disponibilità offertami durante l'intero lavoro di questa tesi, per tutta la pazienza dimostrata e per avermi indirizzato tutte le volte sulla giusta strada nella ricerca delle soluzioni.

Ai miei genitori, per avermi concesso l'opportunità di intraprendere e portare a coronamento questo percorso senza avermi fatto mai mancare il sostegno e la fiducia, per essermi stati sempre accanto nonostante la distanza e per avermi fatto sentire veramente a casa ogni volta che sono tornato.

A mia sorella, per tutta la sensibilità e il supporto in questi cinque lunghi anni, per avermi saputo dare costantemente i giusti consigli e per avermi guidato nel prendere le decisioni più importanti.

A mio cugino, per aver condiviso con me questa avventura lontano da casa e dalla nostra terra dandomi la possibilità di avere sempre vicino qualcuno su cui contare.

A tutti i miei amici più stretti, per avermi immeritatamente sopportato e per essere stati la migliore via di fuga dallo stress dello studio.

Ai colleghi, ai parenti e a chiunque mi abbia aiutato nel corso di questi anni.

Un sentito grazie.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Since Tim Berners-Lee proposed the World Wide Web in March 1989 [1], he gave birth to a revolution in the information world. Internet was born and all the connected data traffic with it. The world was beginning to connect and the demand for access to the new technological reality was increasing more and more.

Nowadays we have come to a lot of data exchanged through the network that needs 21 digits to be sized. According to the latest Zettabyte Cisco Report [2] in 2016 the incredible threshold of 1.2 ZB of IP traffic was reached through the Internet, and this number is expected to almost triple by 2021. The diagram in figure 1.1 illustrates the growth rate of IP traffic per month.

Figure 1.1. Growth rate of IP traffic per month. [2]

The document also claims that by 2021 it is expected that 82% of total IP traffic will be IP video traffic. This percentage includes live internet video, video surveillance, video-on-demand and internet TV. It must also be considered that,

following the current trend, the demand for higher video resolutions and a better overall quality is expected in the future. As well as expected an increase in traffic on mobile devices.

All this can not be sustained without considerable technological progress in video encoding in order to maximize data compression. To address this problem over the years, different research groups have proposed innovative video coding standards. In the last two decades, for example, the MPEG-1 and 2 issued by the ISO/IEC group, and the H.261 and 263 standards issued by ITU-T, deserve to be mentioned. Great progress has been made with the joint work of these two teams, starting with the AVC up to the 2013 HEVC standard which marked a new frontier in video coding.

Since then the two groups have joined a new workgroup called the Joint Video Exploration Team to lay the foundations for the study of a new standard capable of achieving even higher performance and overcoming the technological challenge. This is how the Joint Exploration Model (JEM) was born, which conveys all the efforts and ideas for improvement. As happened in the past, a software test model of the studied algorithm has always been realized. For this thesis the last available version corresponding to JEM 7.1 has been considered [3].

1.2 Motivations

For the JEM model the insertion of a particular element in the filtering stage has been proposed for the elimination of artefacts due to the compression of the information. To the already present Deblocking Filter and Sample Adaptive Offset, a previous level of frame processing called Bilateral Filter has been added, which performs an average of the intensity of the pixels weighed on the geometric distance between them and preserves the edges.

It is a topic that has only recently been considered to be implemented in the new model, although its original formulation dates back to 1998, in fact the HEVC standard does not include it. As a consequence, it has aspects that are still unexplored, for which this work undertakes to analyze. In particular, its application to the world of image processing has frequently been studied, where the limits due to real-time performance and the amount of data are not present or extremely stringent. As a result, many of the approaches presented in the past have favoured aspects related to the final quality of the output or to the scalability of the filter among the resolutions that an image can reach. Now it is allow to sacrifice the achievement of a total defects absence but it is not possible to go below certain processing speeds, in order to avoid total incompatibility with the purpose of use of and therefore the trivial technological uselessness.

It is also important to have a thorough study on the optimization of this element as it is a mandatory stage for loop decoding. Every time a frame is reconstructed, the Bilateral Filter is activated even before all other filters. Even in some cases it

is applied before the end of the prediction algorithm, and in particular every time frames already decoded are necessary for what is currently being decoded.

1.3 Work Organization

The thesis presented here is divided into six chapters that deal with the essential topics of the discussion. They are presented in the following order.

1. *Introduction* Brief description of the background and the purpose of the presented work.

2. *Video Coding* This chapter summarizes the JEM coding algorithm working principles. The fundamental steps in the encoder and decoder chain are described, along with their characteristics and the path pursued by the ITU-T and ISO/IEC research groups in the previous standards presentation.

3. *Bilateral Filter* Being the main topic of this thesis, an entire chapter was assigned to its mathematical formulation and then to operational development, as well as the approaches taken for JEM arrangement.

4. *Architecture Development* If in the former chapter the theoretical characteristics of the filter have been discussed, in this case a valid hardware implementation is developed. Starting from an architecture already proposed, we worked on optimizing the resources used.

5. *Results* The results achieved by the actual synthesis of the studied architecture are illustrated.

6. *Conclusions* Finally, the conclusions on the performance that a device can achieve using the presented architecture are discussed.

At essay end there are also appendices showing the listing of the realized VHDL codes starting from what described in the architectures chapter, and the JEM Common Test Condition (CTC).

Chapter 2

Video Coding

This chapter summarizes the essential topics of the JEM (Joint Exploration Model) encoding and decoding algorithm. Starting from a brief chronicle of the standards adopted so far, it continues with the main features of the model.

2.1 Standards

In [4] it is well shown by G. Sullivan how the standards of video coding arose over time from the work of two specific research groups: the ITU-T Video Coding Experts Group (VCEG) [5] and the ISO/IEC Moving Picture Experts Group (MPEG) [6].

The ITU-T published for the first time in 1988 the H.261 [7] standard, to be finally updated in 1993 [8]. It has been designed to reach speeds between 40 kbit/s and 2 Mbit/s. In 1996 the same group released the standard H.263 [9], with subsequent modifications in the following years. It was mainly thought for low-bitrate video conferencing, but has also found wide use in the compression of multimedia content on the web.

Meanwhile, ISO/IEC has been the author in 1991 of the MPEG-1 [10] standard, and in 1998 of MPEG-4 Visual [11].

But the standards that have found the most diffusion over time are the H.262/MPEG-2 Video [12] and the H.264/MPEG-4 Advanced Video Coding (AVC) [13]. They was born from the joint work of both the ITU-T and ISO/IEC research groups.

The H.262/MPEG-2 Video was released for the first time in 1995, and then it was upgraded to a second version in 2000. It has been fundamental for TV and satellite broadcasting thanks to interlaced video support. It was also the main compression format for DVD-Video media due to the ability to reach 9.8 Mbit/s.

Eventually, H.264/MPEG-4 has been initially released in 2003 and received major updates and amendments until today. It has almost completely covered every area by replacing the old standards thanks to the superior performance. It has

found application in high definition TV and satellite broadcasting, cable transmissions, digital video capture and editing, as well as in most online multimedia and video chat applications, and in high density media like the Blue-Ray Disc [4]. However, due to the increasing amount of data traffic caused by video-streams at resolutions that are often beginning to outgrow HD, and which are now increasingly concentrated on mobile devices, has led to the need to develop a new standard level.

Hence from the joint work commitment of the ITU-T and MPEG groups which has been realized in a partnership under the name of Joint Collaborative Team on Video Coding (JCT-VC) [14], the High Efficiency Video Coding (HEVC) [15] [16] has been released in 2013. The goal is to improve the performance allowing higher resolutions and to increase the processing parallelization. The attempt is to be able to halve the bit-rate of previous H.264/MPEG-4. In order to give maximum exploration capacity for optimizations, it has been decided to standardize only the syntax, the structure of the bit-stream and the mapping for the generation of the decoded images [4].

Since the release of the last standard, an exploration model has been defined in order to try innovative and experimental approaches. A new group called JVET (Joint Video Exploration Team) [17] was founded in October 2015 by the usual ITU-T VCEG and ISO/IEC MPEG. In this way a new test model called JEM was elaborated. After a short time and some proposals, in 2018 the group was renamed the Joint Video Experts Team, with the aim of design a new standard.

2.2 Overview

The JEM video encoding algorithm can be divided into intermediate stages, each defined by a specific operation. It largely reflects the model of the HEVC, the differences are made to the characteristics of the operations carried out.

The goal pursued in coding is to remove any possible redundancy from the information processed [18]. This redundancy can be of type:

- *spatial* Linked to the local correlation of intensity and color saturation that can occur between adjacent or close pixels, for example in uniform areas of the video frame;
- *temporal* Linked to the correlation of intensity and color saturation that can take place between pixels belonging to consecutive frames in a sequence, as they may have been acquired by the video source in short time intervals;
- *statistic* Linked to the probability that certain symbols of the bit-stream have to appear on most real occasions, and the encoder should work with the target of assigning a limited number of bits to the most common ones;
- *visual* Linked to the limits of human vision.

Assuming to have a non-coded input signal, it is initially divided into regions called Coding Tree Units (CTU). Subsequently, information is collected and is then used to perform a prediction of the picture with the aim of compressing similar contents due to one of the mentioned redundancies. Finally the Discrete Cosine Transform (DCT) and its quantization, which allow to achieve high performance in the compression of the input signal, is operated. This last action, however, determines particular artefacts that can ruin the actual quality of the output signal. This is why a special stage called In-Loop Filters able to specifically remove these defects is inserted in the decoding section of the algorithm.

In figure 2.1 is shown an algorithm simplified diagram.

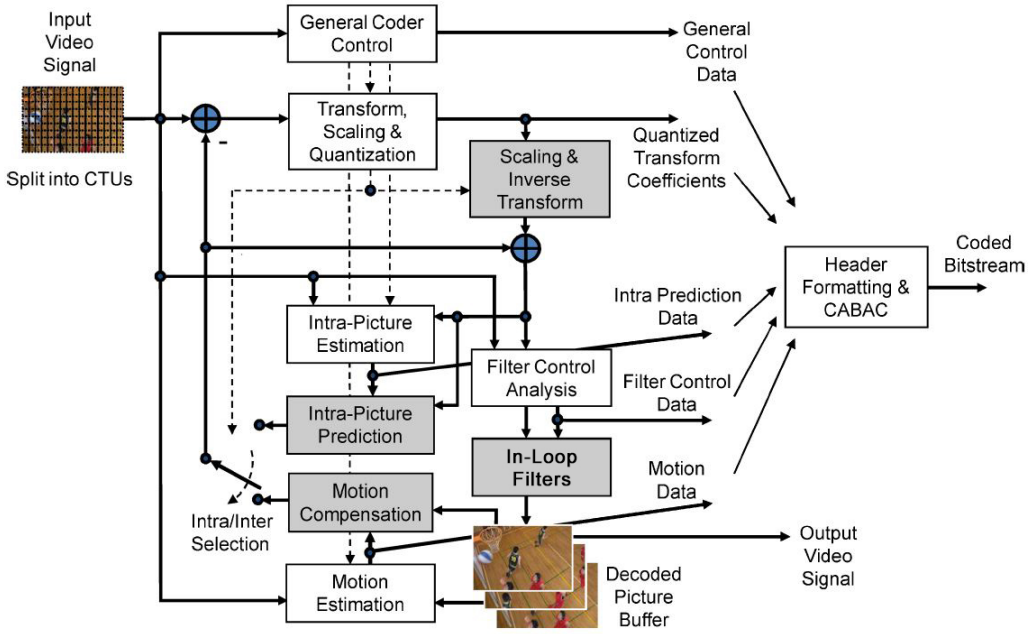

Figure 2.1. JEM simplified diagram. All main stages are individually represented. Gray ones are related to the decoding chain. [4]

The blocks that in the figure 2.1 are called Motion Estimation, Motion Compensation and Intra-Picture Estimation contribute to the prediction operation. The CABAC stage, on the other hand, is used to estimate decoding entropy.

2.2.1 Video Frame

A frame of a video sequence is defined as a matrix of pixels each one having its own value and which together make a picture. Typically it is rectangular with dimensions W (width) × H (height). The most known pairs of values, which determine the resolution of the frame, are typically 1280 × 720 (HD), 1920 × 1080 (FHD) and

3840 × 2160 (UHD4K). The JEM is designed to achieve high performance with the last two resolutions.

Commonly the YCbCr color space is used and with a progressive 4:2:0 sampling [4]. The color space is divided into three separate channels: luma (Y) component that identifies the shade of gray assumed by a pixel in the entire tonal range that goes from white (maximum) to black (minimum); chroma (Cb and Cr) components that define the how much the color deviates from gray in blue and red respectively. Luminance and chrominance are treated separately within the algorithm. The 4:2:0 sub-sampling indicates the relationship between the samples of the luma and chroma components of the frame. More precisely, a single value of Cb and Cr is sampled every squared block 2×2 of Y values. Consequently, the chrominance channels will have dimensions W/2 × H/2. This choice depends on the greater sensitivity that the Human Visual System (HVS) has towards the luma, thus allowing to reduce the bit-rate of the color channels. In figure 2.2 a visual example of 4:2:0 sub-sampling is illustrated.

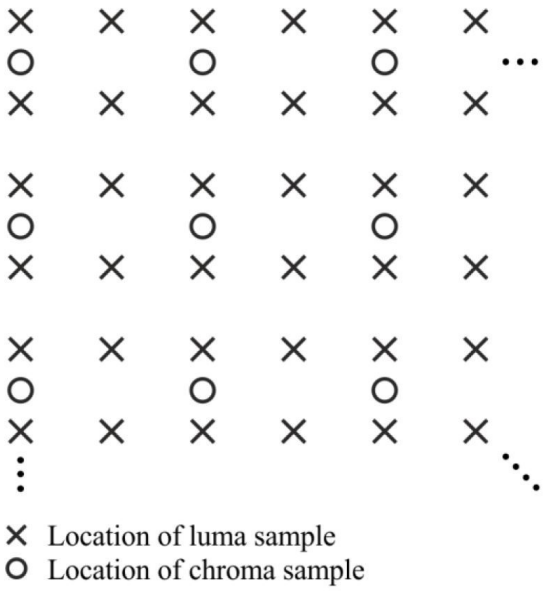

Figure 2.2. Visual example of 4:2:0 subsampling. A single value of Cb and Cr is sampled every squared block 2×2 of Y values. Therefore the chrominance channels have dimensions W/2 × H/2. [16]

JEM typically uses a 10-bit depth, resulting in a range from 0 to 1023 of values assumable by each pixel and for each channel.

2.2.2 Picture Partitioning

In HEVC the input signal frames are partitioned into macroblocks called Coding Tree Units (CTUs) with dimension defined by the encoder. A CTU consists of a luma coding tree block (CTB) and two chroma CTBs. The CTBs are squared structures of samples with side L that can have a value equal to 16, 32 or 64, depending on the compression to be obtained. Each CTB is then further subdivided into a quadtree-like structure containing luma Coding Block (CB) and chroma CBs. One luma CB and two chroma CBs form a Coding Unit (CU). As a consequence a CTB can contain one or more CUs. Each CU is also composed of Prediction Units (PUs) and Transform Units (TUs). The PUs can be from 4×4 to 64×64 samples. TUs can be from 4×4 to 32×32 samples [4]. In figure 2.3 an explanatory scheme of the HEVC picture partitioning is shown.

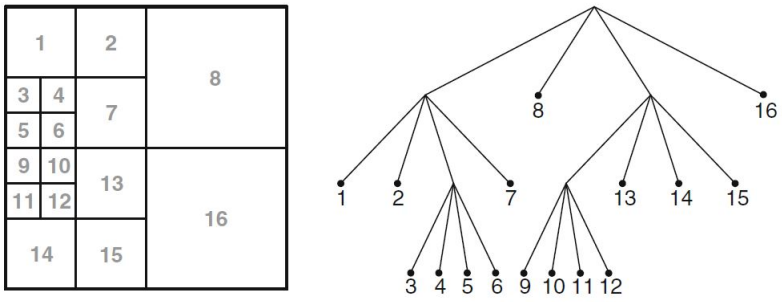

Figure 2.3. HEVC picture partitioning scheme. Example for the partitioning of a 64×64 CTU into CUs of 8×8 to 32×32 luma samples. On the rigth the quadtree structure is shown. The numbers indicate the coding order of the CUs.[15]

With JEM the separation of CU, PU and TU is removed and with them the concept of multiple partition types. This allows greater flexibility for the size and shape of a CU, which can now also assume a rectangular shape. The introduction of a new block structure called QuadTree-plus-Binary-Tree (QTBT), which divides each CTU into a quadtree structure and each leaf further into a binary tree structure. The binary tree allows a horizontal or vertical subdivision based on the value assumed by the division node, 0 for the first case and 1 for the second. An example is shown in figure 2.4. The leaves of the binary tree are CUs. A QTBT partitioning scheme defines the following parameters: CTU size, the root node size of a quadtree; MinQTSize, the minimum allowed quadtree leaf node size; MaxBTSize, the maximum allowed binary tree root node size; MaxBTDepth, the maximum allowed binary tree depth; MinBTSize, the minimum allowed binary tree leaf node size [19]. The maximum size of a CTU is 256×256. JEM and HEVC also define the slice as a set of CTUs processed in raster scans for which a single encoding

method is defined.

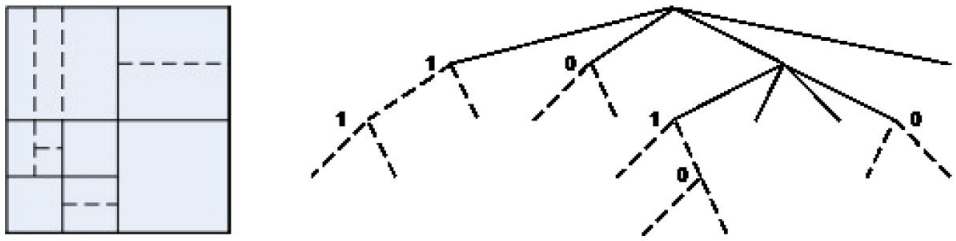

Figure 2.4. Example of QuadTree-plus-Binary-Tree partitioning structure in JEM. Zero value node determines a symmetric vertical division, whereas one value node signal horizontal split. [19]

2.2.3 Prediction

The prediction within the encoding algorithm of a video sequence arises from the need to eliminate the information redundancy. The operation carried out is to gather clues about correlation from frames in order to obtain a more reliable prediction, then a subtraction of this with the original content is performed and only the difference, or residual, obtained is actually sent so that it can be added again to the prediction and precisely reconstruct the information in the decoding phase [4]. There are two types of prediction: *intra-prediction* and *inter-prediction*.

Intra-prediction works in the perspective of reducing spatial redundancy, and therefore the correlation between pixels belonging to the same frame. To do this it is necessary to identify any more or less geometrical ordered arrangement of pixels within each block. Pre-defined directional modes have therefore been introduced, which amount to a total of 67 for the JEM. In figure 2.5 these modes are shown.

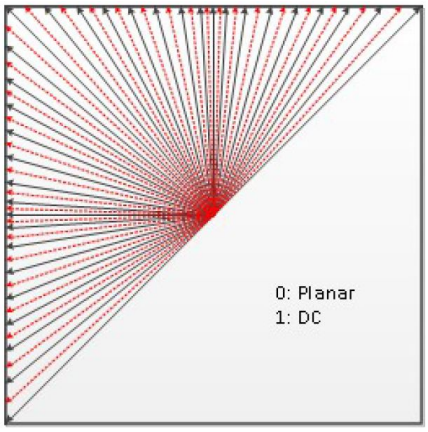

Figure 2.5. Pre-defined directional modes used to capture the arbitrary edge directions presented in natural video when intra-prediction is enabled. [19]

Inter-prediction instead acts on the removal of the temporal redundancy and tries to eliminate the correlation between pixels belonging to consecutive frames. The complexity of this operation requires estimating the movement of the pixels over time and collecting the information in a structure called motion vector, which is then sent together with the residues and allows to perform a motion compensation. In JEM it is possible to estimate the motion at sub-PU level and with adaptive resolution, up to an accuracy equal to 1/16 of luma sample and 1/32 of chroma sample [19].

Inter-prediction typically allows to achieve greater compression factors than intra-prediction, thereby reducing the bandwidth required to transmit the encoded bit-stream. However, it takes more encoder and decoder computational time because the algorithm must consider multiple frames simultaneously.

Based on which prediction paradigm it has been chosen, slices [4] are classified in:

- *I-slice* Slice in which all CUs are coded using only intra-prediction;
- *P-slice* Slice in which some CUs can be coded with inter-prediction method, as well as intra, using information from previous frames;
- *B-slice* Slice in which some CUs are coded with inter-prediction, as well as intra, using frames that precede and succeed the one to which the slice belongs.

2.2.4 Transformation and Quantization

Already from the H.261 standard it was considered to apply to the algorithm a stage able to performing the transformation the prediction residues in order to reduce the statistical correlation. In fact, converting the working domain into that

of the transformed, through the use of the Discrete Cosine Transform (DCT) [20], it is possible to reduce to a limited set of coefficients the amount of data necessary to represent the information. Furthermore, this transformation does not lose any information as it is a lossless compression.

In order to further reduce the amount of data we can proceed with a quantization of the transform coefficients. It is done a normalization of each value of the coefficients to the Quantization Step (Q_{step}) defined as

$$Q_{step} = 2^{\frac{QP-4}{6}}$$

and where QP is the Quantization Parameter that can vary in the interval [0,51]. Therefore, a high QP implies a high compression and therefore an imprecise reconstruction by the decoder. It is necessary to find a compromise between quality and transmission band. For this reason, since the implementation x264 [21] of standard H.264, the Variance-based Adaptive Quantization (VAQ) has been introduced.

The VAQ allows to adjust the quantization on the basis of a given block pixels variance. In particular, if the variance is low, so the block appears sufficiently uniform, QP is limited in order to avoid the onset of compression artefacts. If instead the variance of the block is high, it is possible to increase QP without compromising the quality. To signal to the decoder which corrections to make to QP it is calculated

$$QP_{offset} = QP_{CU} - QP_{frame}$$

where QP_{CU} is the quantization calculated for a single CU starting from the variance of its CB, and QP_{frame} it is actually the value defined a priori for the sequence.

2.2.5 In-Loop filters

In the decoding phase, further operations are needed in order to correctly reconstruct the encoded information and to remove all the artefacts introduced by the quantization and by the division into blocks of the frame. In this way, the stage called In-Loop Filters is introduced into the algorithm, which in the JEM is composed of a chain of four different filters: the Bilateral Filter (BF), the Deblocking Filter (DBF), the Sample Adaptive Offset (SAO) and eventually the Adaptive Loop Filter (ALF). In figure 2.6 a diagram of the components in the In-Loop Filters stage is shown.

The individual filters are briefly described below.

- *BF* The bilateral filter is a non-linear filter with spatial weighted mean but with edge preservation, and it works in the removal of quantization artefacts. It substantially removes, in the areas where it is applied, the low frequencies preserving the high ones. Being the main topic of this thesis is better explained in its personal chapter 3.

- *DBF* The deblocking filter [22] acts on the pixels at the edges of adjacent CUs in order to mediate differences related to different parameters between partitioning or other encoding artefacts. The result must be a smooth profile instead of strong transitions.

- *SAO* The sample adaptive offset [23] acts to remove a constant difference in intensity between the reconstructed and the original sample. In order to do this, these offsets are evaluated for each CTB during the coding phase and some flags that determine which value must be added or subtracted from the decoded samples are signalled to the decoder.

- *ALF* The adaptive loop filter is the last and most complex of those present. It also acts to reduce artifacts such as ringing and blurring through the multiplication of coefficients determined through Wiener-Hopf equations. In order to avoid having to calculate them each time, 25 classes of coefficients are predefined and in the coding phase the most suitable for each block is chosen based on evaluations on horizontal, vertical and diagonal gradients, and n signalled. After identifying the class that best matches the entity of the defect to be removed, any geometric transformations can be applied to make the ALF more accurate [19].

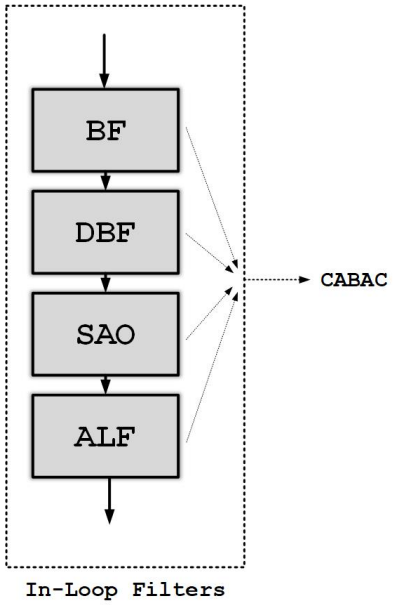

Figure 2.6. Diagram of the components in the In-Loop Filters stage. The first one is the Bilateral Filter, then the Deblocking and Sample Adaptive Offset filters, and the Adaptive Loop Filter at the end of the chain.

2.2.6 Entropy Coding

In this last stage of the JEM algorithm a compression of the data containing all the information necessary to reconstruct the sequence stream actually signalled by the encoder to the decoder is performed. For example, the partitioning block structure and the calculated residuals, the coefficients related to the transformation and to the correction filters applied, data about quantization and prediction, and a whole series of general controls are sent. In order to reduce the amount of bits and therefore the occupied band, an encoding is performed through the Context-based Adaptive Binary Arithmetic Coding (CABAC) system.

Chapter 3

Bilateral Filter

In this chapter it is explained and discussed the theoretical approach of the bilateral filter and its role in the processing and decoding of pictures and video. It shows an in-depth description of the model that best reconciles with the use context and the objectives to be achieved, examining all the applicable improvements to get closer to the state of the art of the topic. The entire work of this research thesis is indeed oriented to the development of a computing architecture that maximizes the performance and minimizes the resources used, therefore the same theoretical approach is carried out focusing on an approach *"hardware-friendly"*.

3.1 Purpose and Role

The bilateral filter arises from the need to have a versatile and reasonably simple tool to perform a smoothing action of the information contained in a matrix of numerical values, or in this case pixels, while preserving the edges between high-contrast regions. Filtering is indeed a hinge process in all contexts involving operations on pictures or video, as it is necessary that the final product is in accordance with the natural human perception. The problem arises from the presence of spurious information superimposed on the original one, and that can be treated as a type of noise originated by mechanisms of acquisition of the image itself or as a collateral consequence of previous operations on it.

Particularly in the video decoding field it is essential to remove the artifacts produced by the quantization in the transform domain during the source stream coding phase, applied since the H.261 standard [8]. The effect that most of all needs to be suppressed is the ringing. In this way the filter tries to reconstruct the original signal as accurately as possible. This produces a final product higher quality as a first and obvious effect, but if we think about the decoding algorithm *inter prediction* which uses temporally related information present on already decoded frames, it is natural to will that the absence of noise on them should be guaranteed

in order to avoid propagation, or worse, unwanted amplification.

Several solutions to the problem have been proposed over time, being a subject of similar interest in pictures and videos. For example, the process of Anisotropic diffusion [25] allows to obtain an effective smoothing with good preservation of the high contrast edges, but which, however, exploits a mathematical approach that includes the resolution of differential equations through iterative methods. This makes the stability of the filter and its efficiency precarious. Another technique often mentioned and reporting good results is presented by Nosratinia [26], who proposes an average between a compressed image information and the same one shifted, exploiting the idea that the original signal is sufficiently preserved in the two cases while the artefacts do not show appreciable correlation. The defect of this process is the high time required for processing which becomes unsustainable with video target.

3.2 Classical Definition

The classical definition of the bilateral filter originally presented in [24] is that of a non-linear method based on a location principle that combines intensity values of the individual elements in the application domain. This allows to mediate the small variations between neighboring points, that is to filter low frequencies, preserving instead strong edges and avoiding therefore to reduce the photometric contrast. It is formulated as the combination of two filters in the spatial domain and in the range domain.

The *spatial domain* relates to the geometric closeness between the central pixel and the neighbouring pixels involved. The *spatial filter* is formulated as:

$$h_d(x) = k_d^{-1}(x) \int_{\infty}^{-\infty} \int_{\infty}^{-\infty} \omega_d(\xi, x)\ I(\xi)\ d\xi. \tag{3.1}$$

With $\omega_d(\xi, x)$ function linked to the geometric distance between x and the near pixel ξ, with $I(\xi)$ intensity value for ξ, and with

$$k_d(x) = \int_{\infty}^{-\infty} \int_{\infty}^{-\infty} \omega_d(\xi, x)\ d\xi. \tag{3.2}$$

But if the filter is shift-invariant (discrete case for time-invariant) it is possible to assert that $\omega_d(\xi, x) = \omega_d(\xi - x)$.

The *range domain* relates instead to the photometric closeness between the central pixel and the neighbouring pixels involve. The *range filter* is formulated as:

$$h_r(x) = k_r^{-1}(x) \int_{\infty}^{-\infty} \int_{\infty}^{-\infty} \omega_r(\xi, x)\ I(\xi)\ d\xi. \tag{3.3}$$

With $\omega_r(\xi, x)$ function linked to intensity difference between x and the near pixel ξ, and with

$$k_r(x) = \int_{\infty}^{-\infty} \int_{\infty}^{-\infty} \omega_r(\xi, x)\ d\xi. \tag{3.4}$$

So, the spatial filter is exclusively based on the relative position of the considered pixels, a quantity that does not depend on what is actually represented in the frame. While the range filter is introduced to preserve the high contrasts and to modulate the ω_r coefficients such that give a greater weight if there is a high photometric similarity between the pixels involved.

Combining them we have the *bilateral filter*:

$$h(x) = k^{-1}(x) \int_{\infty}^{-\infty} \int_{\infty}^{-\infty} \omega_d(\xi, x)\ \omega_r(\xi, x)\ I(\xi)\ d\xi \tag{3.5}$$

with

$$k(x) = \int_{\infty}^{-\infty} \int_{\infty}^{-\infty} \omega_d(\xi, x)\ \omega_r(\xi, x)\ d\xi. \tag{3.6}$$

3.2.1 Gaussian Case

The most common and generally used case is the Gaussian filter, both for the *spatial filter* and the *range filter*. In this way the functions ω_d e $\omega_r(\xi, x)$ are formulated as the Gaussian function of the Euclidean distance of the arguments, and can be written respectively:

$$\omega_d(\xi, x) = e^{-\frac{1}{2}(\frac{d(\xi,x)}{\sigma_d})^2} \tag{3.7}$$

with

$$d(\xi, x) = d(\xi - x) = ||\xi - x||, \tag{3.8}$$

$$\omega_r(\xi, x) = e^{-\frac{1}{2}(\frac{r(I(\xi),I(x))}{\sigma_r})^2} \tag{3.9}$$

with

$$r(I(\xi), I(x)) = r(I(\xi), I(x)) = ||I(\xi) - I(x)||. \tag{3.10}$$

Figure 3.1 shows a graphical example of the application of a bilateral filter with 23x23 square geometry starting from input (a) to the processed output (e).

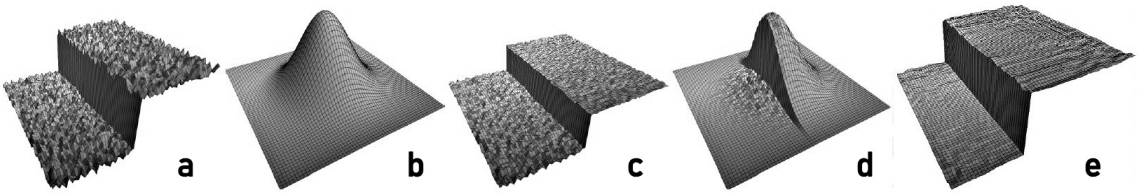

Figure 3.1. Example of a demonstration chart for the application of the bilateral 23x23 filter. (a) Input. (b) Spatial filter function. (c) Range filter function. (d) Bilateral filter function (a · b). (e) Output. [27]

From the parameters σ_d and σ_r at the exponent denominator depends on the value of the Gaussian functions, and it is through them that the filter strength is defined. Specifically, with σ_d the width of the averaging window is determined. The greater the value, the greater the distance from the center where the last pixel considered by the filter is located. It is also dependent on the size of the filter application support, if it increases it is necessary to change the value of the parameter in order to have the same results. The value of σ_r is instead related to the maximum photometric difference to be considered. In fact, if this is greater than twice σ_r, the exponent in (3.9) assumes an absolute value greater than one and consequently the Gaussian itself result is very small. This means that the pixel involved does not contribute significantly to the filter product. Vice versa when the difference in intensity between the two pixels considered is lower than the denominator. This is consistent with the purpose of edges preserving of the bilateral filter.

3.3 JEM

The bilateral filter implemented in the JEM algorithm [19][28][3] is very basic. It has a diamond-shaped geometry that only involves five pixels: one central and the other four distributed in a cross, one above, one below, one on the right and one on the left. The filter geometry is shown in figure 3.2.

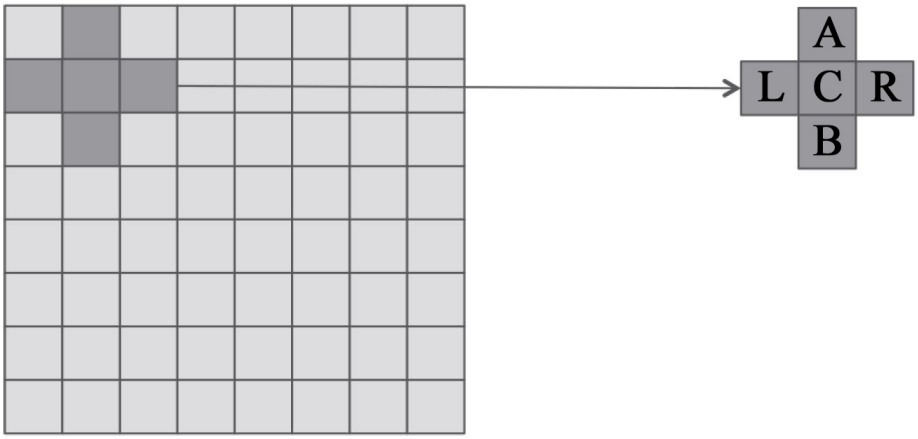

Figure 3.2. Geometry of the bilateral filter in the JEM model on the right. The pixels name defines their position: *C*entral, *A*bove, *B*elow, *R*ighr, *L*eft. On the left a TU 8x8 is shown as a classical application support for the filter. [19]

The support of the filter is always a rectangular or squared TU of side L equal to 4, 8 or 16. Grater structure are not allowed. There is not a pre-defined or imposed filter directional flow, any choice depends on target performance. Typically the raster scan mode, starting from the top left pixel, is the most efficient way.

Also it is to be considered that the integration domain (3.5) it is discretized, and so formulation becomes:

$$h(x) = I_F(i,j) = \frac{\sum_{k,l}\ \omega((i,j)(k,l)) \cdot I(k,l))}{\sum_{k,l}\ \omega((i,j)(k,l))} \tag{3.11}$$

with (i,j) coordinates of the central pixel subject to the filtering operation; (k,l) coordinates of a generic pixel belonging to the spatial domain of the filter, or in other words, element of the diamond geometry in figure 3.2; with $I_F(i,j)$ output value of the coordinate filter (i,j); and with

$$\omega((i,j)(k,l)) = e^{-\frac{(i-k)^2-(j-l)^2}{2\sigma_d^2} - \frac{(I(i,j)-I(k,l))^2}{2\sigma_r^2}} = e^{-\frac{1}{2\sigma_d^2} - \frac{\Delta I^2((i,j)(k,l))}{2\sigma_r^2}}. \tag{3.12}$$

In the JEM model the spatial and range parameters of the Gaussian denominator are defined as:

$$\sigma_d = p - \frac{1}{40} min(H, W, 16) \tag{3.13}$$

$$\sigma_r = max(\frac{QP - 17}{2}, \frac{1}{100}) \tag{3.14}$$

with p depending on the type of prediction, it is equal to 0.92 for *intra prediction* and 0.72 for *inter prediction*; with H and W respectively height and width of the TU wherein the filter is applied, 16 represents the maximum value that the two dimensions can assume; with QP quantization parameter, included in the range of integer values [18, 51].

If QP assumes a value below 18, the bilateral filter is not applied at all. The motivation behind the possibility that p can take two values is related to the filter strength for the two predictions: in fact it is necessary to reduce its intensity in the *inter prediction* case since the frames used by the prediction itself have already been previously filtered and over-filtering needs to be avoided.

Looking at figure 3.2, equation (3.11) can be rewritten as:

$$I_F = \frac{\omega_C \cdot I_C + \omega_A \cdot I_A + \omega_B \cdot I_B + \omega_R \cdot I_R + \omega_L \cdot I_L}{\omega_C + \omega_A + \omega_B + \omega_R + \omega_L} \tag{3.15}$$

Clearly ω_C, which represents the coefficient associated with the central pixel, has a unitary value because geometric and photometric distances are null. This is consistent with what is expected from the behaviour of the filter.

The algorithm shown, however, presents a discretization exclusively of the spatial domain, preserving continuity for the range one. This hardly allows numerical development on digital devices, that's why some specifications have been better defined. Intensity values $I(x)$ can only assume integer quantities greater than or equal to zero, within a contingent limit depending on target performance. Also coefficients undergo an equal discretization, which requires a scale factor in order to represent the values lower than one. It should be noted that this last consideration does not increase the complexity of the method just because in (3.15) coefficients ω are present in both members of the fraction, thus involving an automatic simplification of the scale factor.

More precisely, 65 has been chosen to represent one. This means that the maximum value assumed by the coefficients is 31. In figure 3.3 the trend of the coefficients for each QP is shown.

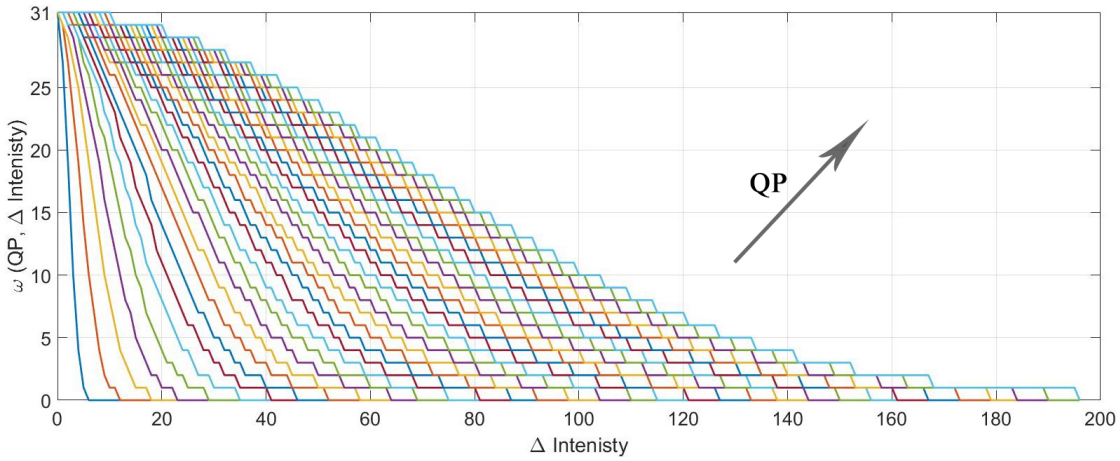

Figure 3.3. Trend of the coefficients value for each QP. As can be seen, they assume a limited quantity in the interval between 0 and 31. Beyond the abscissa 197 all the coefficients are equal to zero.

3.4 Improving Classical Approach

The bilateral filter contained in the JEM model has been improved over time to meet efficiency and resource saving requirements looking to an implementation on digital support. In this section are presented improvement techniques prosed in [28] and [29]. More precisely, the effect of the integer rounding error introduced by the discretization of the range domain was first considered. Starting from the equation (3.15) we get that by rounding up we have

$$I_F = round(\frac{\omega_C \cdot I_C + \omega_A \cdot I_A + \omega_B \cdot I_B + \omega_R \cdot I_R + \omega_L \cdot I_L}{\omega_C + \omega_A + \omega_B + \omega_R + \omega_L}), \tag{3.16}$$

but so that it is possible to obtain a correct result even in integer arithmetic, we must rewrite it as

$$I_F = floor(\frac{\omega_C \cdot I_C + \omega_A \cdot I_A + \omega_B \cdot I_B + \omega_R \cdot I_R + \omega_L \cdot I_L + \frac{\omega_C+\omega_A+\omega_B+\omega_R+\omega_L}{2}}{\omega_C + \omega_A + \omega_B + \omega_R + \omega_L}). \tag{3.17}$$

Algebraic Variation

Also, observing once again (3.15), it can be noted that with a few algebraic steps it is possible to reformulate it as

$$I_F = I_C + \frac{\omega_A \cdot (I_A - I_C) + \omega_B \cdot (I_B - I_C) + \omega_R \cdot (I_R - I_C) + \omega_L \cdot (I_L - I_C)}{\omega_C + \omega_A + \omega_B + \omega_R + \omega_L}. \tag{3.18}$$

The main advantage of this variation, which at first sight might appear to worsen the computational complexity, is that now the numerator of the fraction is in almost every case smaller than the previous formulation. This is due to the fact that $\Delta I_x = (I_x - I_C)$ is statistically lower than I_x and I_C. The only situation in which we do not have an advantage is when I_C assumes the smallest representable value, that is zero, and I_x the greatest one.

However, this change forces to reconsider (3.17). To have a correct rounding, we have to keep in mind that now the numerator of the fraction can also take negative values. As a result it is no longer sufficient to operate as before but must be calculated instead

$$I_F = I_C + s \cdot floor(\frac{s \cdot N + \frac{D+m}{2}}{D}). \tag{3.19}$$

where

$$N = \omega_A \cdot (I_A - I_C) + \omega_B \cdot (I_B - I_C) + \omega_R \cdot (I_R - I_C) + \omega_L \cdot (I_L - I_C) \tag{3.20}$$

$$D = \omega_C + \omega_A + \omega_B + \omega_R + \omega_L \tag{3.21}$$

$$s = sign(N)$$

$$m = \begin{cases} -1, & \text{if } N < 0 \\ 0, & \text{else} \end{cases}$$

Coefficients Table Reduction

It has previously been shown (3.12) that the value of the coefficients is dependent on the parameter σ_d. But the latter is related to the geometric dimension of the filter support itself, as well as the type of prediction (3.13). This should imply the need to have a three-dimensional memory, having QP, ΔI and σ_d as inputs. This situation needs to be well researched in order to limit the use of resources that would otherwise be unsustainable.

Fortunately, a solution has been presented to solve the problem. If, starting from the equation (3.15) and multiplying both members of the fraction by the value

$$s_x = \frac{e^{-\frac{1}{2 \cdot 0.82^2}}}{e^{-\frac{1}{2\sigma_d^2}}} \tag{3.22}$$

and thus obtaining

$$I_F = I_C + \frac{s_x \cdot \omega_A \cdot (I_A - I_C) + s_x \cdot \omega_B \cdot (I_B - I_C) + s_x \cdot \omega_R \cdot (I_R - I_C) + s_x \cdot \omega_L \cdot (I_L - I_C)}{s_x \cdot \omega_C + s_x \cdot \omega_A + s_x \cdot \omega_B + s_x \cdot \omega_R + s_x \cdot \omega_L}, \tag{3.23}$$

we have that all the coefficients have the value normalized to a single and constant σ_d equal to 0.82 as can easily be verified

$$s_x \cdot \omega_x = \frac{e^{-\frac{1}{2 \cdot 0.82^2}}}{e^{-\frac{1}{2\sigma_d^2}}} \cdot \omega_x = \frac{e^{-\frac{1}{2 \cdot 0.82^2}}}{e^{-\frac{1}{2\sigma_d^2}}} \cdot e^{-\frac{1}{2\sigma_d^2} - \frac{\Delta I^2}{2\sigma_r^2}} = e^{-\frac{1}{2 \cdot 0.82^2} - \frac{\Delta I^2}{2\sigma_r^2}}. \tag{3.24}$$

Now the problem previously exposed is transferred on s_x. And on closer inspection, in (3.23) this parameter is multiplied by ω_C that has a constant value of 65, so it becomes

$$\omega_C \cdot s_x = 65 \frac{e^{-\frac{1}{2 \cdot 0.82^2}}}{e^{-\frac{1}{2\sigma_d^2}}}. \tag{3.25}$$

The final effect is that instead of having a memory of the three-dimensional coefficients, we have a separate memory for ω_C that is now function of σ_d. Taking a few considerations, there are two prediction modes, and the values assumable by the width and height of the filter support are three (the dimensions are prefixed and can be 4, 8 o 16). This brings to 6 the count of the possible ω_C. However, H and W can not assume the size 16 for *inter prediction*, lowering the count to 5. The final table of the values of ω_C is shown in 3.1.

prediction	**H, W**		
	4	***8***	***16***
intra	65	81	196
inter	113	196	/

Table 3.1. Table of values assumable by ω_C for H and W dimensions of the filter support, and the prediction mode.

As can also be seen in the figure 3.3, the value of the coefficients is a decreasing monotone function, being a branch of a Gaussian, as well as piecewise constant, due to the discretization. This means that if the address of the first occurrence of a zero is stored separately, it can be compared with the actual calculated difference in intensity and select the smallest one, since the associated coefficient would be identical in any case. In doing so, there is a considerable saving of otherwise useless resources. More precisely, the new memory must have a total of 3468 cells, compared to an original value of 34816, implying an efficiency of the $\sim 90\%$.

Division-Free

An operation that typically requires a great deal of time to complete is the division, because both in hardware and in software it is often done with an iterative technique. There are some architectures that allow to execute it in a single cycle without needing to memorize intermediate steps, but they are also very slow and far too expensive in terms of occupied area.

This is why the division is often replaced with a multiplication operation, in which the second factor is nothing but the inverse of the original denominator. In order to do this, a memory of dimensions equal to the entire range of values that the denominator can take is naturally necessary.

In the case of the JEM the denominator (3.21) can be at most equal to $max(\omega_C) + 4 \cdot max(\omega_x) = 196 + 4 \cdot 31 = 320$. As a result a 321 elements table would be required. But if we remember that ω_c is constant because it is always equal to one, net of the scale factor, and that the minimum value it can take is 65, memory can be reduced to 256 elements ($\frac{1}{65}$ to $\frac{1}{320}$).

But working with integer arithmetic it is not possible to represent quantities smaller than one, so in this case it is also necessary to apply a scaling factor to the second member of the substitutive multiplication. The product must then be rescaled by the same value. It has been decided to apply a scale factor of 2^{14}. In this way the division becomes:

$$\frac{N}{D} = N \cdot \frac{2^{14}}{D} \cdot << 14 \tag{3.26}$$

with $<<$ arithmetical shift operator.

However, again due to rounding, it may result in a non-constant error for each element of the memory, since the smallest values suffer a greater information loss. Reason why we can think of having a dynamic scale factor, which grows proportionally to the denominator. To do this the algorithm from [3] has been used and it is shown in Alg.1. Where $Multiplier\_factor\_Lut$ is the multiplication factor memory while $Scaling\_factor\_Lut$ is the dynamic scaling factor one.

Algorithm 1 Division-Free memories values algorithm

1: $one \leftarrow 2^{14}$
2: **for** $n \leftarrow [1, 320]$ **do**
3: $\quad tryLut \leftarrow one/n$
4: $\quad tryshift \leftarrow 0$
5: $\quad$ **while** $tryLut \leq one$ **do**
6: $\quad\quad Multiplier\_factor\_Lut(n) \leftarrow tryLut$
7: $\quad\quad Scaling\_factor\_Lut(n) \leftarrow trySHift$
8: $\quad\quad trySHift \leftarrow trySHift + 1$
9: $\quad\quad tryLut \leftarrow (one/2^{trySHift})/n$
10: $\quad$ **end while**
11: $\quad$ **if** $Multiplier\_factor\_Lut(n) \cdot n/(one/2^{Scaling\_factor\_Lut(n)})$ **then**
12: $\quad\quad Multiplier\_factor\_Lut(n) \leftarrow Multiplier\_factor\_Lut(n) + 1$
13: $\quad$ **end if**
14: **end for**

The scale factor is a value in the range $14 + [6, 8]$, or better in $[20, 22]$. Therefore the division is now replaced by

$$\frac{N}{D} = N \cdot \frac{2^{20+k}}{D} << (20 \ + \ k) \tag{3.27}$$

with $k \in [0, 2]$.

The side effect that this approach involves is the need for a second memory that contains the scale factor applied to every possible multiplication factor.

Chapter 4

Architecture Development

After analyzing the theoretical aspects of the bilateral filter, the time to study an architectural solution that allows to obtain the desired results has come. The reference target is a level of performance that allows real-time video processing of an at least 60 frames per second sequence at resolutions FHD (1920 × 1080) and UHD4K (3840 × 2160). The possibility of presenting a working architecture for the UHD8K (7680 × 4320) is also considered, although actually it is still a fact far from the market standard.

Taking in account that reference video streams are composed of three channels (Luma, ChromaB and ChormaR) and encoded according to the chrominance subsampling of the type 4:2:0, theoretical counts can be made: the minimum pixel-rate to be observed is equal to $W \cdot H \cdot 60 + 2 \cdot (\frac{W}{2} \cdot \frac{H}{2} \cdot 60)$, so for the three resolutions we have

- *FHD* 186.624 MPixel/s
- *UHD4K* 746.496 MPixel/s
- *UHD8K* 2985.984 MPixel/s

Assuming to realize an architecture capable of processing a pixel by clock, it is necessary to reach the thresholds of the 190MHz for FHD and the 750MHz for UHD4K. In order for this to be possible, it is mandatory to think of an ASIC-type solution due to the fact that it is the only one currently capable of easily achieving such performance. FPGA and μP alternatives would be respectively limited by the difficulty in achieving these frequencies and by the instructions overhead needed to process even one pixel.

The basic scheme of the filter, shown in the chapter 3, is taken from what is described in [28] and [3]. However, good architecture also takes into account the actual resources used. This is why the main objective of this work has been to

maximize efficiency and minimize the occupied area. The task has been aimed at the memories present, being the first items in the final resources count.

4.1 First Order Bilateral Filter

The architecture of the bilateral filter faithfully follows the algorithm described in the previous chapter. With *Order* we want to classify the pixel-rate of the filter: *First Order* implies the processing of a pixel per clock, *Second Order* of two per clock, and so on.

The architecture consists of two parts: the first calculates the differences in intensity of the pixels involved in order to address the coefficients memory and thus to obtain the quantities in the numerator (3.20) and in the denominator (3.21); the second one makes the division between the two members of the fraction through a multiplication, as already illustrated above.

Coefficients and ΔI evaluation

Assuming to be able to simultaneously obtain the intensity values of all five pixels involved in the filter which are shown in the figure 3.2, the differences can be now calculated through four separate adders. ΔI_A, ΔI_B, ΔI_R and ΔI_L are so obtained. Remembering that they have a threshold for every QP such that the value of the associated coefficient is zero, it is correct to think to make a direct comparison with this limit and use the lesser between the two as the address of the memory. This allows to reduce its dimensions, as previously shown. By using the coefficients obtained from the memory, the value of the product $\omega_x \Delta I_x$ can be calculated. Finally, compressors are used to calculate the sums $\sum_{(A,B,R,L)} \omega_x \Delta I_x$ and $\sum_{(A,B,R,L,C)} \omega_x$. In this last summation there is also the contribution of ω_C which is obtained from its own memory holding values shown in the table 3.1.

However, there is a problem with applying the filter to the edges of the support window. Looking at figure 3.2 the margins are considered as the outer ring of pixels. In these cases, in fact, there is no information for each four pixels surrounding the central one. Whatever the strategy one choose to use, the filter needs an assigned value for the missing pixels. This value must however be carefully evaluated as it must not compromise the final result. That is, it is necessary that the contributions arriving at the two final sum compressors are zero. Although it is known that a coefficient has an inverse proportionality with the associated intensity difference, there is no economic way to evaluate a priori what value the input should assume to obtain a zero weight. The most convenient and almost obligatory choice is to place a multiplexer that dynamically selects whether to pass the real coefficient value or zero. With a counter that iterates for each pixel of the support, it is possible to evaluate if the filter is pointing to the outer ring or not, and accordingly act

on the control signal of the multiplexers. In this way both $\sum \omega_x \Delta I_x$ and $\sum \omega_x$ do not contribute if the center pixel belongs to a margin. At this point it is not even more important what is the actual intensity of outside pixels. One can think, for example, to add an additional virtual ring with of predetermined values, . The architecture described is shown in figure 4.1.

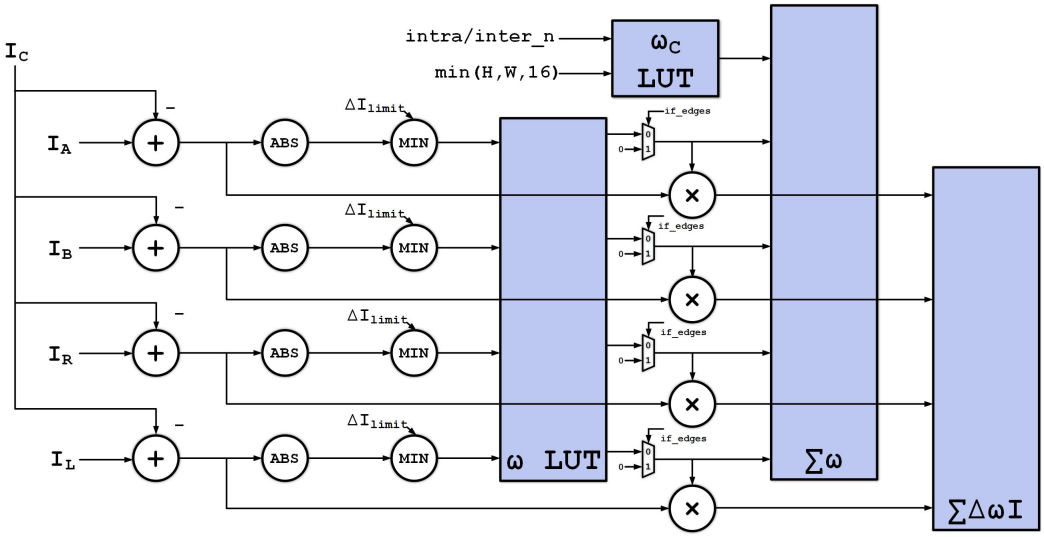

Figure 4.1. Architecture of the first section of the bilateral filter. On the left are the adders that make the difference between the values of I_x and I_C. The coefficients memory is in the center. Finally, on the right are the two final compressors that perform $\sum \omega_x \Delta I_x$ and $\sum \omega_x$.

But focusing better, we can think of using the spatial correlation principle to avoid having to make four evaluations for all four pixels around the central one each time. For two adjacent cells belonging to the same row it is true that

$$I_R(i) - I_C(i) = -(I_L(i+1) - I_C(i+1)),$$

where i is the absolute position of the central pixel within the row. This is shown by the fact that

$$I_R(i) = I_C(i+1)$$

and

$$I_C(i) = I_L(i+1),$$

as illustrated in the figure 4.2. The same consideration can be made for two adjacent cells belonging to the same column, for which it can be argued that

$$I_B(j) - I_C(j) = -(I_A(j+1) - I_C(j+1))$$

due to

$$I_B(j) = I_C(j+1)$$

and

$$I_C(j) = I_A(j+1),$$

where j this time identifies the position inside a column, as can be seen in the figure 4.3.

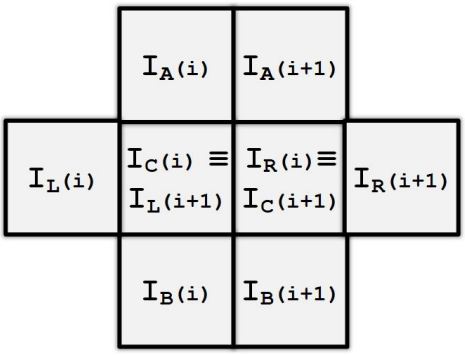

Figure 4.2. Graphical example of how spatial correlation can be used to avoid having to make four evaluations when raster scan processing is adopted. Indeed $I_R(i) = I_C(i+1)$ and $I_C(i) = I_L(i+1)$, where i is the absolute position of the central pixel within the row, due to the fact that the two considered pixels belong to the same row.

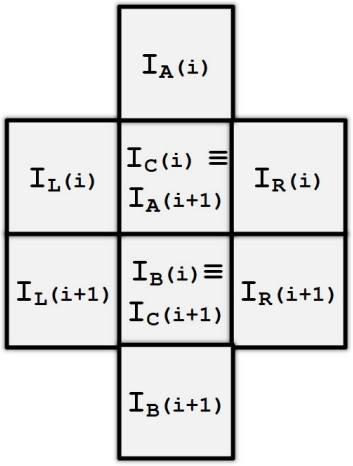

Figure 4.3. Graphical example of how spatial correlation can be used to avoid having to make four evaluations when raster scan processing is adopted. Indeed $I_B(i) = I_C(i+1)$ and $I_C(i) = I_A(i+1)$, where i this time identifies the position inside a column, due to the fact that the two considered pixels belong to the same column.

Assuming to apply the filter starting from the top left position of the support and proceeding in raster scan line by line, it is possible to exploit the advantage

given by this condition by using registers that save ω_R and $\omega_R \Delta I_R$ values for each iteration, and a registers array of length equal to that of the support (W) in order to save ω_B and $\omega_B \Delta I_B$ from the processed line. It is not necessary to perform a check when these values are evaluated for pixels outside the support because the previously introduced multiplexers automatically solve the problem. The new architecture is shown in figure 4.4.

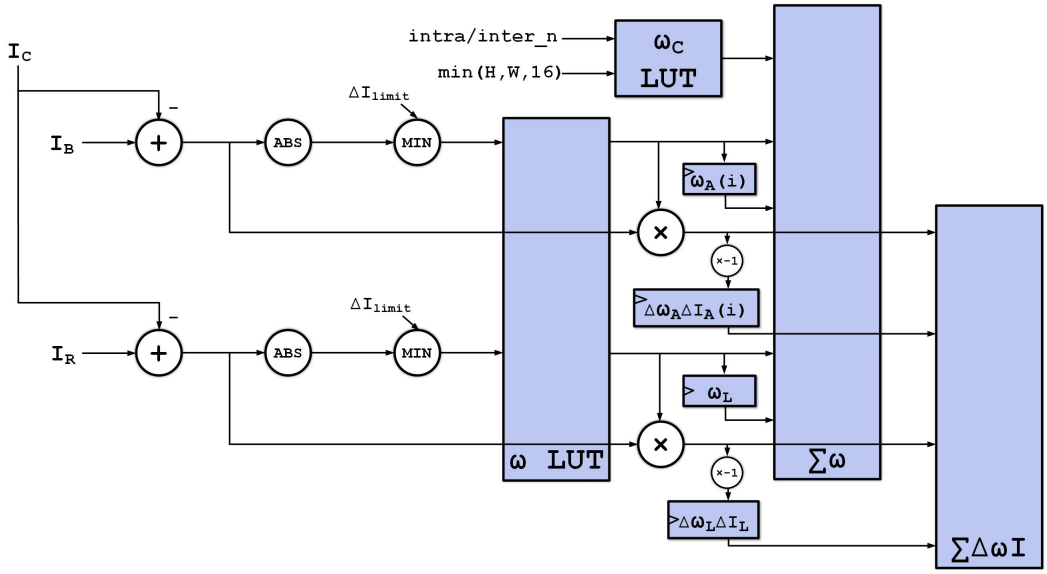

Figure 4.4. Architecture of the first section of the bilateral filter with introduction of registers ω_L, $\omega_L \Delta I_L$, ω_A and $\omega_A \Delta I_A$ that save respectively ω_R, $\omega_R \Delta I_R$, ω_B and $\omega_B \Delta I_B$ in order to reduce complexity.

Division through Multiplication

The division in (3.19) can be transformed into a multiplication for the reciprocal of the denominator as shown in(3.27). The resources necessary to perform the new calculation are first of all a multiplier and a memory containing all the possible values of the reciprocals. As already illustrated, this memory consists of 256 elements ($\frac{1}{65}$ to $\frac{1}{320}$). A third fundamental element is the right shifter which allows to remove the scale factor from the multiplication product. However, the factor is not constant in order to limit the error caused by rounding. This implies the need for additional memory to know how many times to perform the shift. It is also composed of 256 elements. The just discussed architecture of the second section of the filter is shown in figure 4.5. While the complete architecture for a first order filter can be seen in figure 4.6.

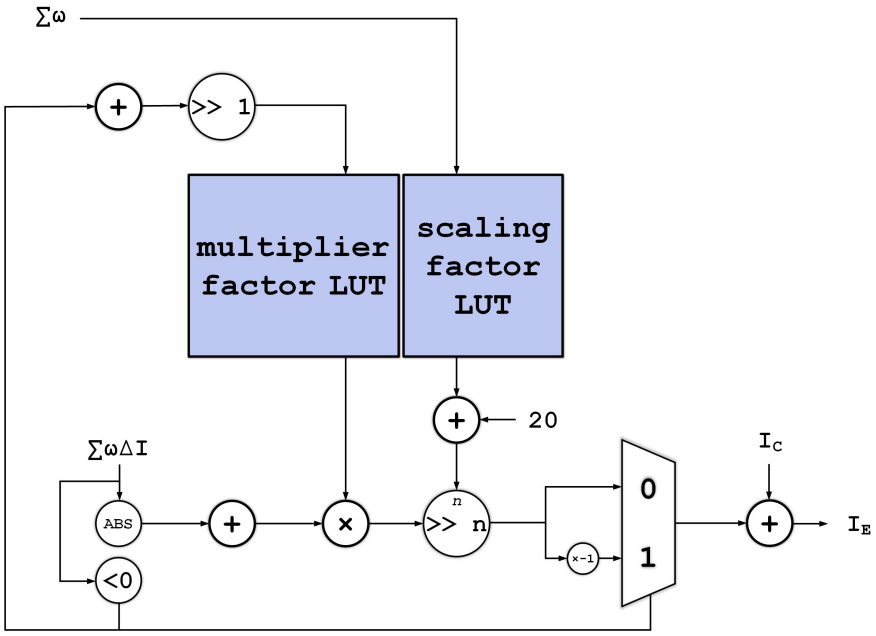

Figure 4.5. Architecture of the second section of the bilateral filter with divider replaced by multiplier. divLUT and shiftLUT are memories respectively for denominator reciprocals and for scale factors.

4.2 Higher Order Bilateral Filter

Naturally, it is possible to realize higher filters order to increase the pixel-rate, for example to achieve the performance required for very high resolutions.

Not all possible orders are allowed. If the geometrical dimensions of the filter support have side equal to the powers of the two (4, 8, 16), it is clear that even the number of pixels processed per clock must be an identical power. Otherwise there would be no support subdivision in equivalent sub-section.

The architecture of a filter of this type would result in a repetition of the one presented in 4.6 as many times as the chosen order. The case of a second order filter is illustrated as an example.

Second Order Bilateral Filter

The filter is applied simultaneously on two pixels, therefore a total of ten intensity values should be involved at most. But based on the relative positions of the two central pixels considered, this number can be reduced. If in fact we always proceed with a raster scan processing and the pair of central points is such that they are two adjacent cells belonging to the same column, the amount of pixels actually involved is equal to eight. Figure 4.7 shows which pixels are considered and illustrates that

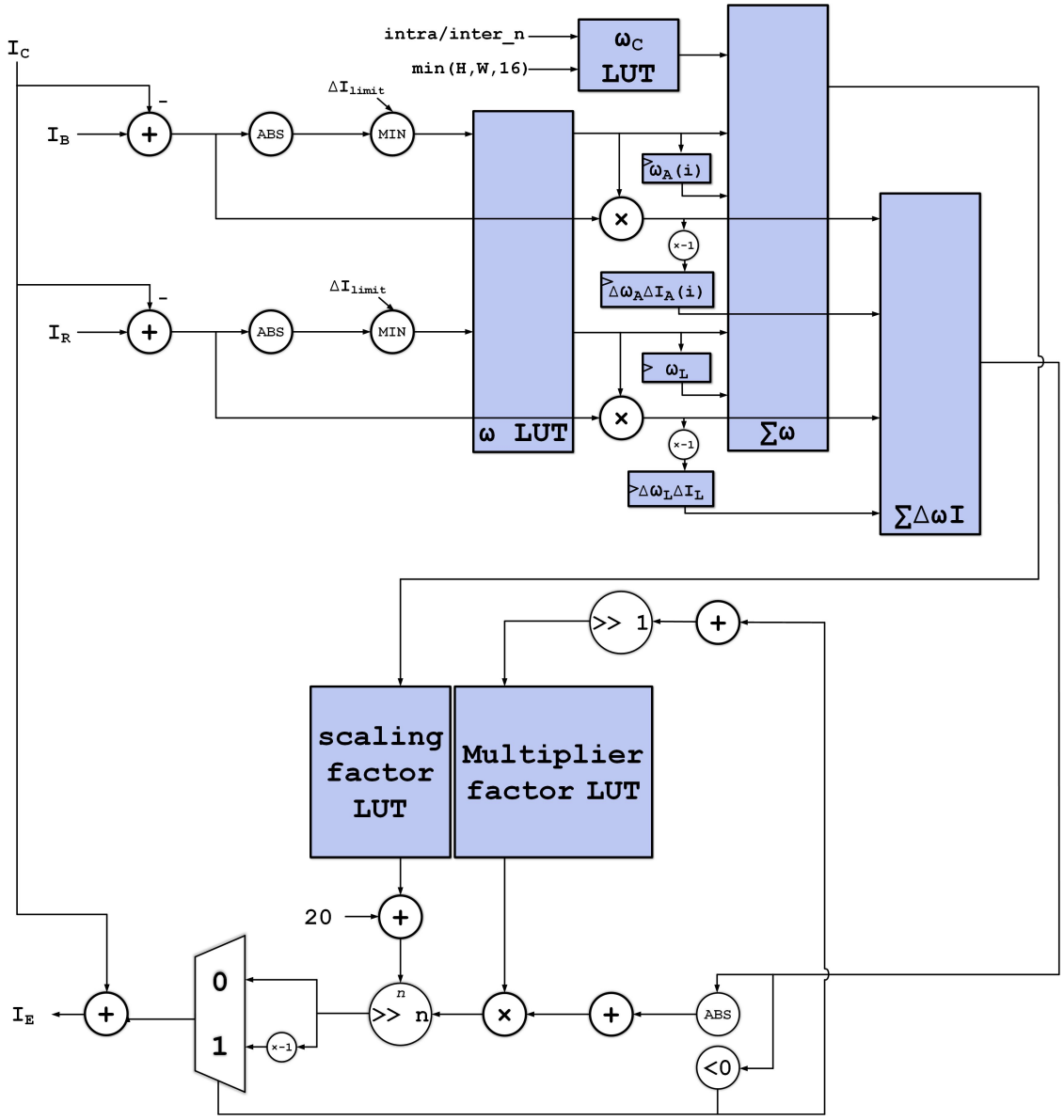

Figure 4.6. Complete architecture of the bilateral filter.

reduction is possible. The advantage is given by the fact that the intensity values of the central pixels are actually shared between the upper and the lower geometry, indeed

$$I_C(j) = I_A(j+1)$$

and

$$I_C(j+1) = I_B(j),$$

with j absolute position of the central pixels within a single column.

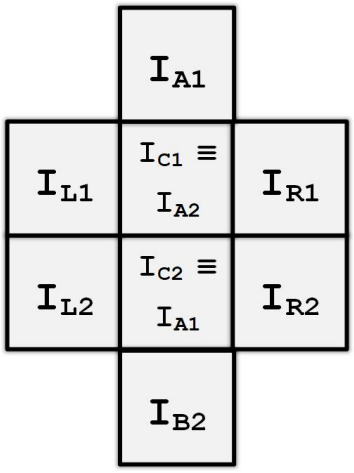

Figure 4.7. Geometry of the second order bilateral filter. It is the sum of two simple cross scheme of the single order filter. Central pixels overlap a below and an above cell, reducing the total number of considered pixels to eight.

Obviously there is only one shared coefficient memory and same situation for memories related to division-free multiplication. Introducing also the same registers of the first order case we obtain that the intensity values to be analyzed for each iteration are only five(I_{C_1}, I_{R_1}, I_{C_2}, I_{R_2} and I_{B_2}).The architecture of the second-order bilateral filter is shown in figure 4.8.

4.3 Improving Memories

The memories relating to the coefficients and to the division-free multiplication are the elements that occupy more area within the architecture, and which can be investigated for the reduction of resources used. This operation has been the hinge work of this thesis.

4.3.1 Coefficients LUT

The memory of the coefficients contains the weights associated with each intensity, or more precisely, the difference in intensity, which makes it possible to modulate the filter strength and preserve the micro-contrast of the image.

There are several ways to store this information. The classic one based on the extensive storage of the weight values is presented here, with a total number of cells equal to the set of all the possible combinations of QP and ΔI. Subsequently it is shown how it is possible to apply a compression of the occupied area thanks to a memory containing the indices within the coefficients are constant.

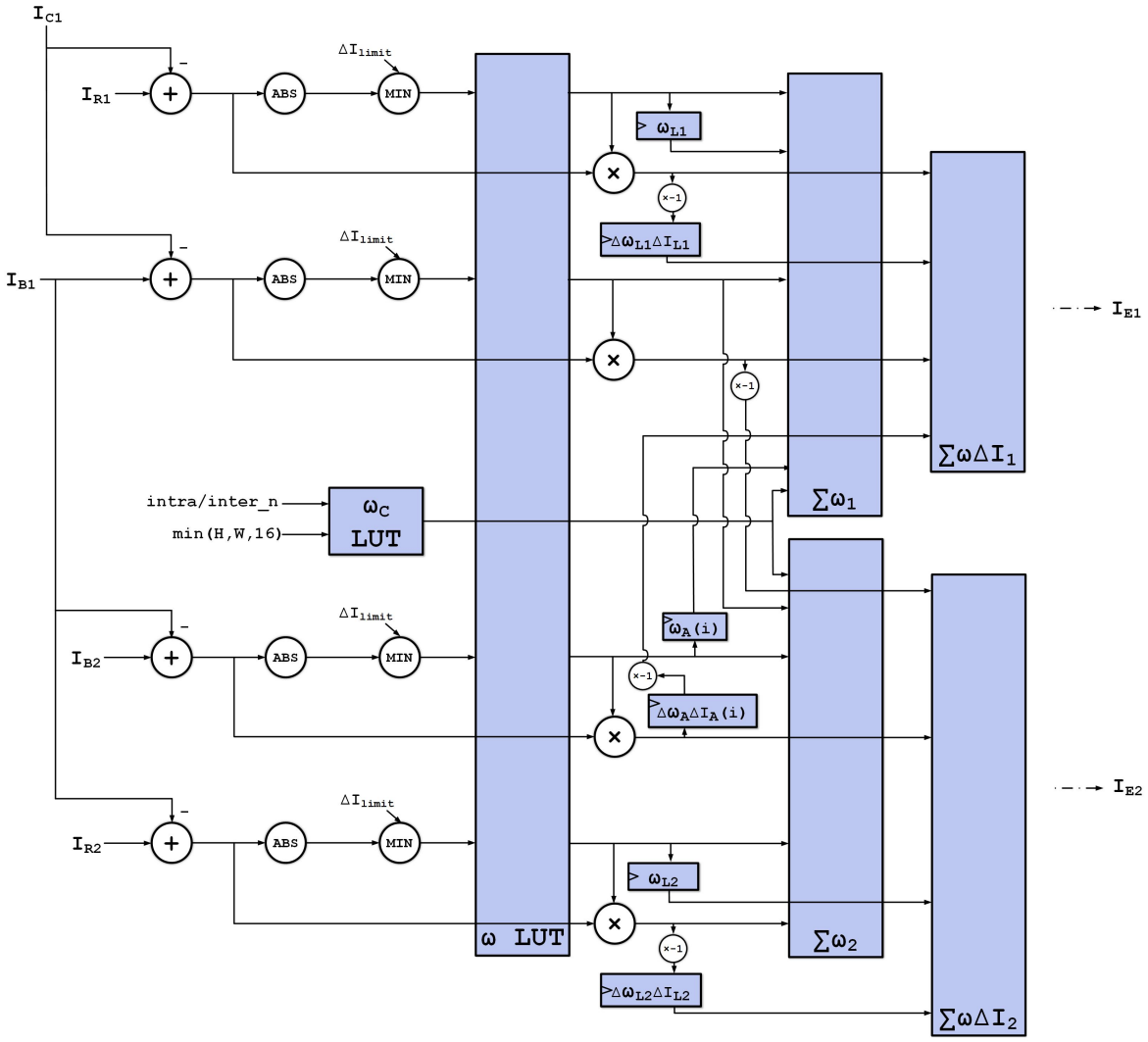

Figure 4.8. First half of the second-order bilateral filter architecture.

4.3.2 Value-Based

The memory originally proposed by [29] and [28] provides a row addressing for QP and columned one for ΔI. It contains exactly the values of the coefficients for each pair of coordinates.

Considering that 34 values of QP and 1024 of ΔI are allowed, we would have in theory a memory of 34816 cells. Knowing also that the coefficient range goes from 0 to a maximum of 31, it can be deduced that 5 bits are necessary for the representation. This leads to a memory of 174080 bits (21.76kB). But as already illustrated, each line contains null values from a certain position onwards, suggesting

that it is possible to implement a comparison system of the column address with the limit value in which the first zero of each row appears. This stratagem allows to significantly reduce the size of the memory, bringing it to a total of 3468 cells and therefore 17340 bits (2168 B).

One aspect that should not be underestimated in this solution is that not all rows have the same number of elements, because the first occurrence of a zero appears at a growing address the larger is QP. The result is a trapezoidal memory, hard to manage and implement. Addressing decoders are not classic.

Least Square Regression

An idea to reduce the area occupied by the memory is to perform a polynomial fitting of the values contained in each of its lines. In this way it would be necessary to memorize only the fitting numeric parameters and recalculate the weight each time through ΔI value. In the first degree polynomial case the fitting would be

$$\omega(QP) = \alpha(QP) \cdot \Delta I + \beta(QP). \tag{4.1}$$

The total memory occupied in this case would be equal to $3 \cdot 34 = 102$ cells, the parallelism is not of immediate deduction, but it can be demonstrated experimentally that it can not exceed 6 bits. The final bill leads to a total of 612 bits (77 B). Obviously the area overhead introduced by the arithmetic operators must be considered, which however would confirm the hypothesis as advantageous.

However, the results of the linear fitting in (4.1) approximate the exact value of the coefficients, so an additional memory is required for the residuals. Furthermore, by operating in an integer arithmetic condition, the fitting parameters require a rounding which inevitably leads to the value of residues being increased. Figures 4.9 shows an example of a linear fitting of the coefficients for QP equal to 51 and the relative residuals using a full precision arithmetic, while 4.10 illustrate the same fitting with rounding caused by integer arithmetic.

It is shown that replacing the memory of the coefficients with that of the residuals, in the best case does not determine reductions in area occupancy because at least 5 bits are always necessary for the numerical representation. Rather, it increases if we consider having to save the fitting parameters as well. Furthermore, the delay and the complexity, although small, of the arithmetic operators are added.

Fitting with greater degree polynomials presents even more disappointing results.

Reduced Parallelism with Sub-memories

A more promising solution than polynomial fitting is to act by subdividing the memory into sub-units with less parallelism. The advantage of this strategy lies in the distribution of the coefficients in each row. They have a decreasing monotone

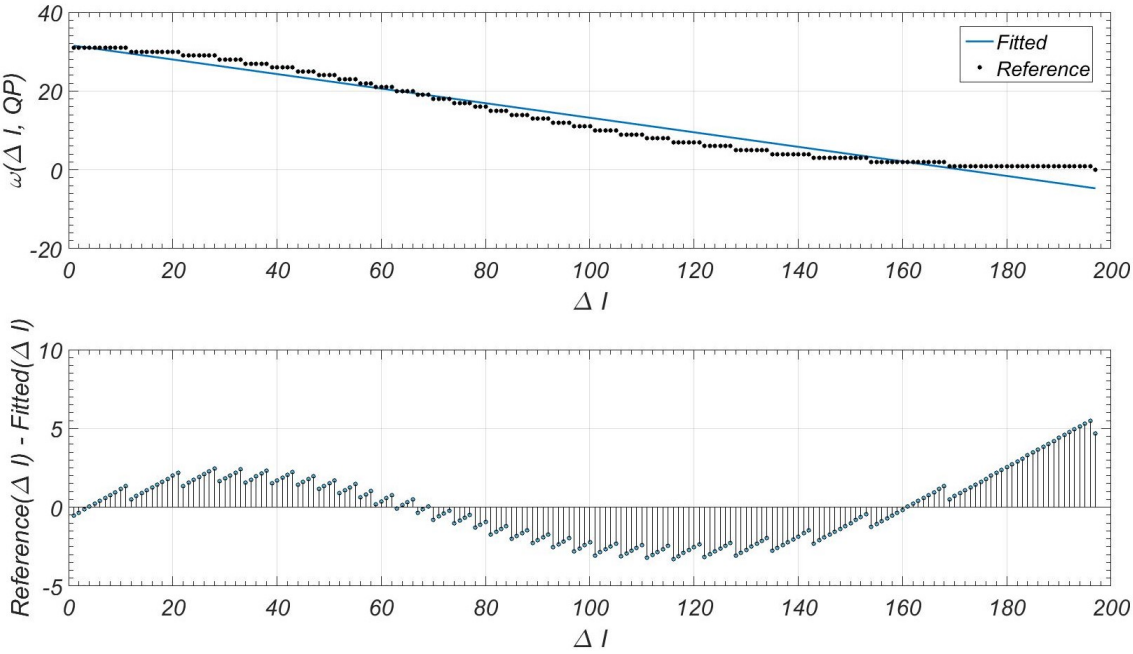

Figure 4.9. Example of a linear fitting of the coefficients for QP equal to 51 and the related residuals using a full precision arithmetic.

trend, and being only integer values, they are also piecewise constant. It is certainly not surprising if we remember that they originate from a branch of a Gaussian function.

As the numeric value of each cell decreases, the number of bits necessary to represent it decreases as well. It can be said that each line can be divided into 5 segments of unequal length, each having an ever smaller parallelism, starting from the 5 bits necessary for the value 31 up to the single bit for the 0. If this concept is extended to the entire memory and the segments of equal parallelism are grouped, five sub-units are obtained. This simple trick leads to a total of 12347 bits (against the original 17340) equal to a saving of 28.8%. But in order to have a transparent addressing of these memories to the rest of the bilateral filter, we need to add some more elements. More precisely, we need to know from which sub-memory to extract the coefficient, and to do this the idea is to compare ΔI with the ends of the constant parallelism intervals. Again, thanks to the decreasing monotone trend, it is sufficient to make a comparison with the address of the first occurrence of a number with a bit length smaller than the previous ones. Four registers where this address is stored for each QP are defined:

- *R_4bit*: register containing the address of first number less than 16 occurrence for each QP;

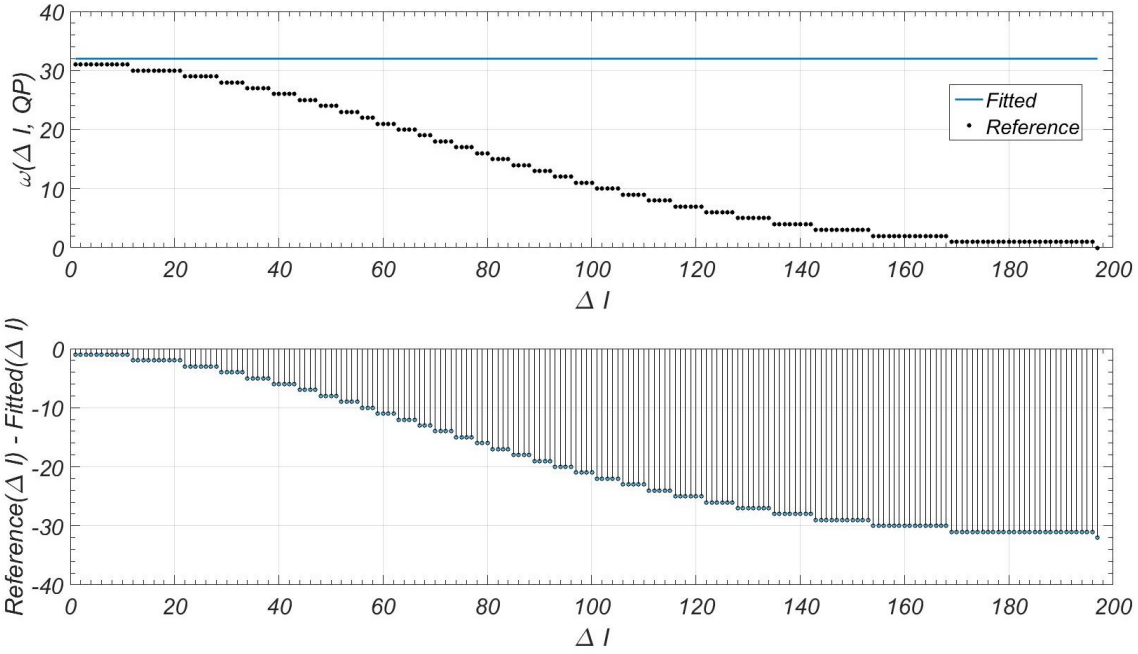

Figure 4.10. Example of a linear fitting of the coefficients for QP equal to 51 and the related residuals using an integer arithmetic.

- *R_3bit*: register containing the address of first number less than 8 occurrence for each QP;
- *R_2bit*: register containing the address of first number less than 4 occurrence for each QP;
- *R_1bit*: register containing the address of first number less than 2 occurrence for each QP;

The overhead of these additional registers is 1020 bits, which brings the actual savings to 22.9%.

Weights ω can be got addressing the right sub-memory through the following comparisons:

$$\omega = \begin{cases} M_{5bit}(QP, \Delta I), & \text{if } 0 \leq \Delta I < R_{4bit}(QP) \\ M_{4bit}(QP, \Delta I - R_{4bit}(QP)), & \text{if } R_{4bit}(QP) \leq \Delta I < R_{3bit}(QP) \\ M_{3bit}(QP, \Delta I - R_{3bit}(QP)), & \text{if } R_{3bit}(QP) \leq \Delta I < R_{2bit}(QP) \\ M_{2bit}(QP, \Delta I - R_{2bit}(QP)), & \text{if } R_{2bit}(QP) \leq \Delta I < R_{1bit}(QP) \\ M_{1bit}(QP, \Delta I - R_{1bit}(QP)), & \text{if } \Delta I \geq R_{1bit}(QP) \end{cases} \tag{4.2}$$

Subtractions $\Delta I - R(QP)$ act as an adjustment of the address offset caused by the subdivision. Otherwise, addresses would be outside the effective permitted range. Also QP should undergo a rescaling equal to 18, this being the first value that can be assumed, but it has been chosen not to represent it in (4.2) in order to avoid more complex formulation. Figure 4.11 shows the sub-memories architecture described.

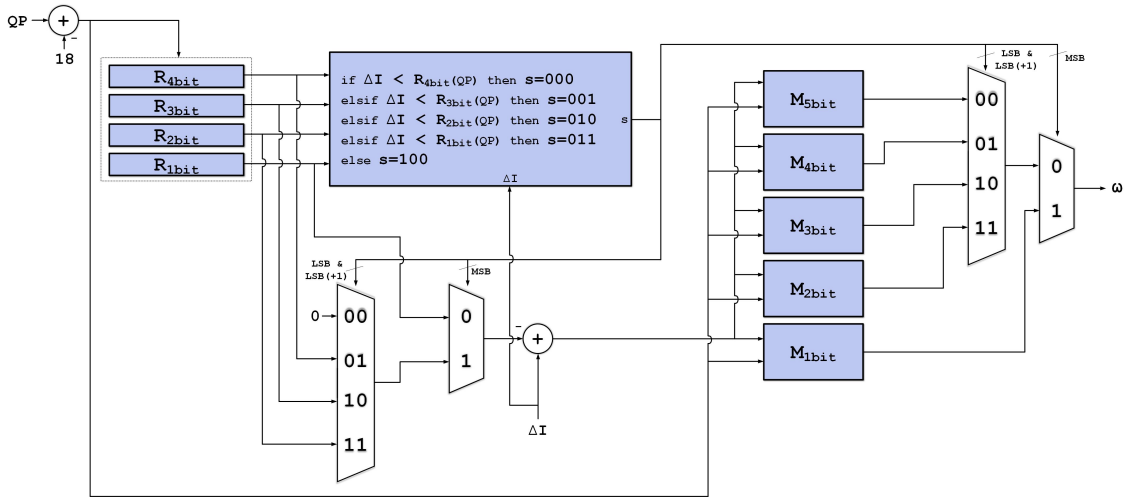

Figure 4.11. Coefficients memory architecture with five reduced parallelism sub-memories. On the left there are the four registers containing the address of first lower parallelism number for each QP. The central block computes the comparisons and signals to multiplexers which sub-memory must be addressed. Address offset adjustment is shown in the bottom left area.

It is still possible to perform optimizations for each of the sub-memories:

M_{1bit} This sub-unit contains only the coefficients having a value of 1 and 0. Applying the same strategy, we can reduce the occupied area until we have just two cells, one per value. A new register is required in order to save the address of the first occurrence of a zero. But this register is already present because it is used to pre-calculate which is the maximum ΔI address to be used before having just zero coefficients.

M_{2bit} The same technique can be easily applied, since this memory also contains only values 2 and 3. With a simple register containing the address of the first occurrence of 2, this sub-unit can be reduced to just two cells.

M_{3bit} The observation for optimizing this memory lies in the number range represented. In fact, it goes from 4 to 7, being numbers that can be represented with

3 bits. It may be a good idea to extract the offset of this interval and add it just after the memory. In this way it is reduced to two bits of parallelism, ranging from 3 to 0 and needing a plus 4 offset.

M_{4bit} **e** M_{5bit} The paradigm is the same as M_{3bit}. For M_{4bit} can reduced to 3 bit parallelism by adding an offset of 8. With M_{5bit} we change from 5 to 4 bits and the offset is equal to 16.

This further optimization, doing some math, leads to a theoretical area reduction equal to 43.9%, since only 9729 bits are needed (1217 B). We can think of having achieved a good result, but trying to bring the technique presented to the extreme for all the individual values of the coefficients can be achieved much more important reductions. This solution has been named *Index-Based*.

4.3.3 Index-Based

Given any row of the original coefficients memory, it is not known a priori what the exact content of each cell is, but it is certain that the values are ordered in decreasing way and that they are piecewise constant.

Assuming to consider a register that contains the address of each first occurrence of a new coefficient in a specific memory row. Consequently, for example, the first cell has the first address in which 31 appears, 30 in the second cell, and so on. Note that the addresses of the new register are in ascending order. In this way 32 memory cells, one for different coefficient, for each QP are needed. It is also possible to reduce the amount by one unit since the first occurrence of the value 31 is always at the zero address of a row.

Summarizing, a memory of 34 rows, equal to the possible values of QP, and 31 columns is needed. In this way the resources used are equal to $34 \cdot 31 = 1054$ cells. Experimentally it is shown that 8 bits are enough for the content representation, which brings the effective area to 8432 bits (1054 B). The saving compared to the original value-base memory is 51.4%.

Just to give an example, we consider the coefficient vector of the original memory for QP equal to 25:

$$31, 31, 30, 30, 29, 29, 28, 27, 26, ...$$

The new row related to the same quantization would have a content equal to:

$$2, 4, 6, 7, 8, ...$$

In addition, the last column of the new table, which contains the address of the first occurrence of 0, is actually a vector already used in the general architecture of the filter, as it serves to evaluate which values of ΔI are certainly equal to 0. It can be said that a horizontal compression of the data has been applied. In figure 4.12 the organization of the index-based memory is shown.

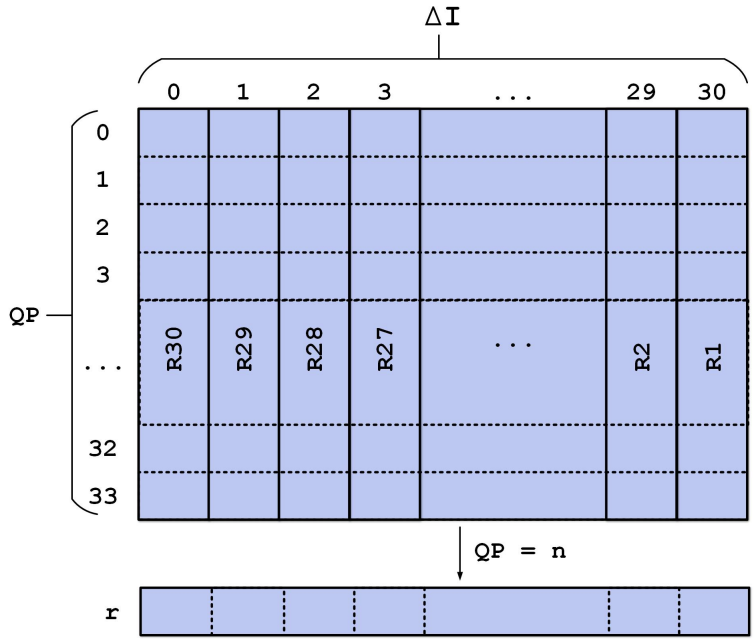

Figure 4.12. Organization of the index-based memory. There are 34 rows because QP can assume value from 18 to 51. The amount of columns is 31 due to the fact that coefficients reach a maximum value of 31 but the addresses of 31 itself are ignored because always zero. Looking the table by columns.

In order to actually use this new table, however, it is necessary to have comparators that determine which adjacent values include ΔI. The idea is that the new table is addressed by QP, and a vector of 31 elements is extracted. The values contained in the read line are used as the second argument of an array of 31 minority comparators, where the first argument is ΔI. The output of the 31 comparators is a 31-bit bus, in which the first occurrence of a 1 indicates exactly which is the value of ω.

Having for example the bus of the comparison results equal to

$$0, 0, 0, 0, 1, 1, ...$$

the first occurrence of 1 is at the address equal to 4, this implies that ω must be $31 - 4 = 27$. If every comparators output is negative, it means that ΔI is greater than the first zero occurrence address and ω has a null value.

But more simply we can think of connecting the bus to a 32-bit priority encoder (taking care to add a last signal always asserted indicating 0). The encoder output is exactly the value of ω encoded on 5 bits. Figure 4.13 illustrates the architecture with comparators and the encoder.

But what happens if every single value between 31 and 0 is not contained in a specific row? For example, with QP equal to 18, we have just six non null

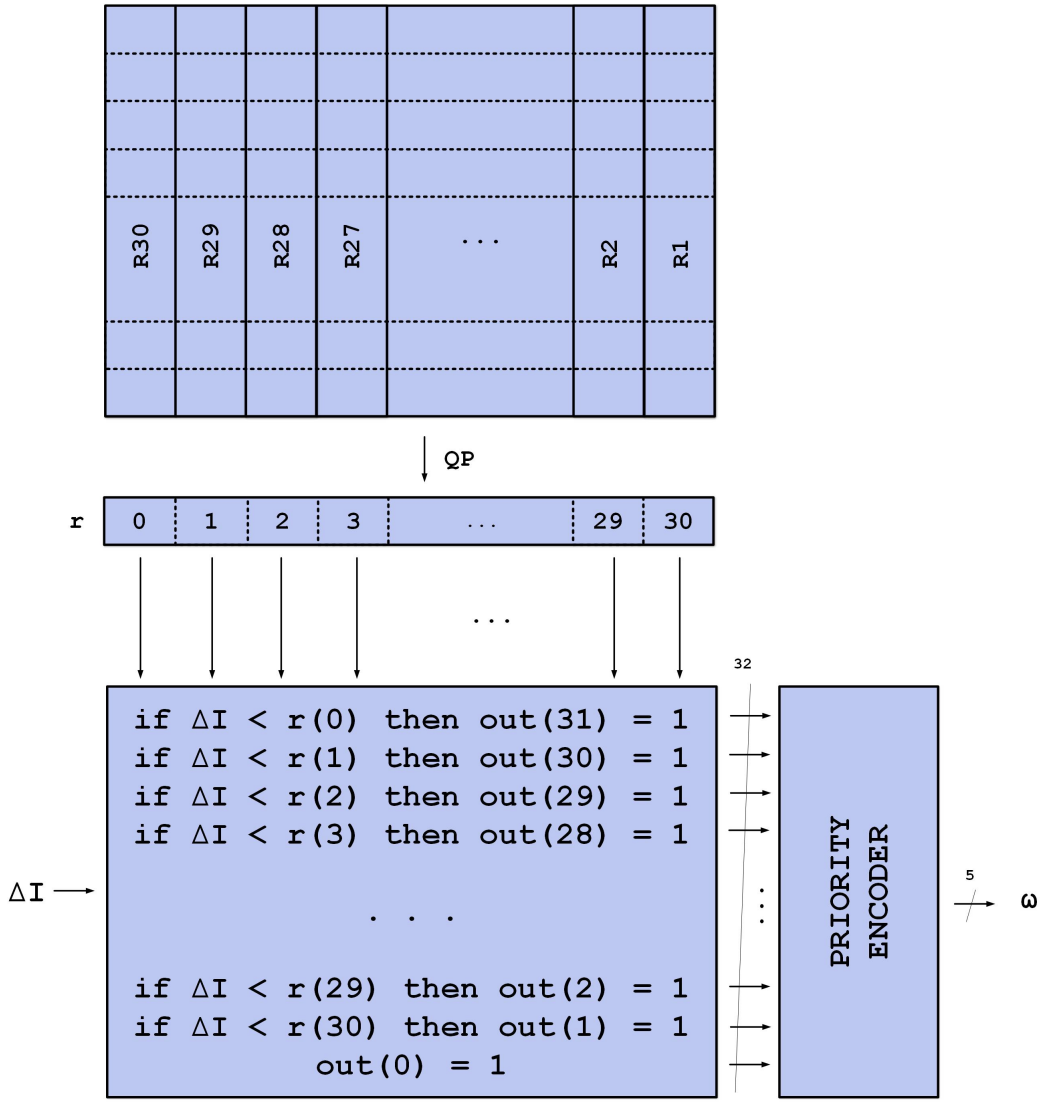

Figure 4.13. Index-based memory architecture with a 32-comparators array and a priority encoder.

coefficients. So a solution need to be found in order to fill the new table with right indexes. The answer is easy: the address of a coefficient first occurrence is always placed in the column related to that specific weight value, any vacuum interval caused by a discontinuity in the integer range 31 to 0, is filled with the index of the nearest smaller coefficient that exists, or in other word, with the first existing address to the right. In order to be more clear an example is shown. If a row in the original value-based memory is

$$31, 31, 30, 29, 27, 25, 23...$$

the result in the new table is

$$2, 3, 4, 4, 5, 5, 6, 6...$$

In the example we have that 28 is absent, so its position in the new memory row is filled with the index of 27, that is the nearest smaller coefficient. Full algorithm to derive *index-based* memory from the *value-based* one is shown in Alg.2.

Algorithm 2 Index-based memory derivation algorithm

```
value_based_Table[34][ ]                          ▷ full value-based coefficients LUT
w[32] ← {30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, ...
...11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0}                  ▷ coefficients possible values
max_pos[34] ← {6, 12, 18, 23, 29, 35, 41, 46, 52, 58, 64, 69, 75, 81, 87, 92, 98, ...
...104, 110, 115, 121, 127, 133, 138, 144, 150, 156, 161, 167, 173, 179, 184, 190, ...
...196}                          ▷ last value_based_Table address for each row
mem ← 0
for i ← [0, length(w) − 1] do
    mem ← value_based_Table[i][0]
    for j ← [0, max_pos[i]] do
        if value_based_Table[i][j] < mem then
            for n ← [0, mem − value_based_Table[i][j]] − 2 do
                index_based_Table[i].push(j)      ▷ a.push(b): add b to end of a
            end for
            index_based_Table[i].push(j)
            mem ← value_based_Table[i][0]
        end if
    end for
end for
```

Less comparators

The solution presented requires at least 32 comparators. Their number can be reduced through a subdivision of the row extracted from the index-based memory. For example, comparing only the first or second half of the vector 16 comparators are sufficient, but then the parallelism of the encoder must be restored according to which of the two parts has been chosen. In order to do this, however, it is necessary to know a priori in which of the two sub-rows ΔI is included. This knowledge can be obtained with a direct comparison between ΔI and the last cell of the first half vector, if the former is less than the latter one, the first 16 row values are used, otherwise the second ones. Since the second half has 15 cells, repetition of the LSC (*Least Significant Cell*) is needed to full the array. However, in this way there is a 4-bits bus output from the priority encoder. To restore the parallelism and have the correct coefficient ω it is necessary to add an offset equal to 16 if the first half has

been used, otherwise the result is already correct and it is sufficient to extend the value to 5 bits. It can be concluded that the offset can be calculated dynamically as a four times right shift of the direct comparison result. Figure 4.14 shows the complete architecture of the memory with a reduction to sixteen comparators.

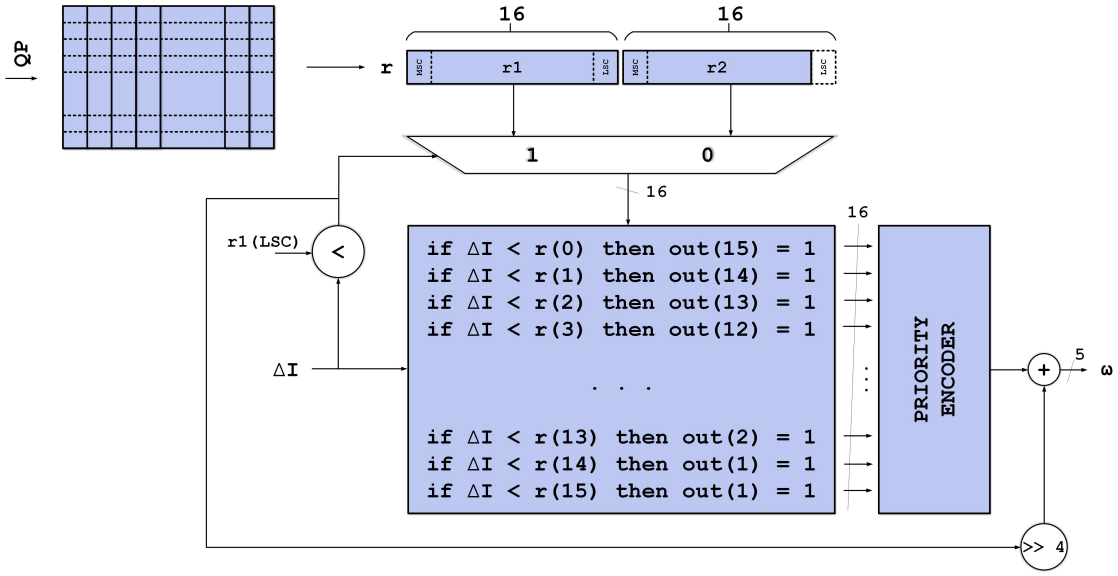

Figure 4.14. Index-based memory architecture with a 16-comparators array and a priority encoder. With a multiplexer signalled by a comparison, which half of the row is chosen. A last adder restores the parallelism adding an offset depending on the sub-row.

Obviously the same mechanism can be applied again to reduce the comparators to only 8 elements. In this case 4 sub-vectors are obtained from the addressed memory row and the selection of which quarter to send is defined by three multiplexers signalled by three controls result of a comparison among the LSCs of the three slices (0 to 7), (8 to 15), (16 to 23). Again basing on of the same controls the offset to add is chosen among 0, 8, 16 or 24. In figure 4.15 the complete architecture of the memory with a reduction to eight comparators is shown.

This approach has a logarithmic reduction of the number of comparators in the array, but at the cost of adding complexity in the sub-vector and offset to be added to restore parallelism selections. More than any other aspect, the reduction of area implies longer critical path and therefore worse timings. However, pipelining can be applied to improve the situation.

Sub-Tables

Considering the divisions applied to the row extracted from the memory in case of a comparators reduction, we can think of directly dividing the main memory.

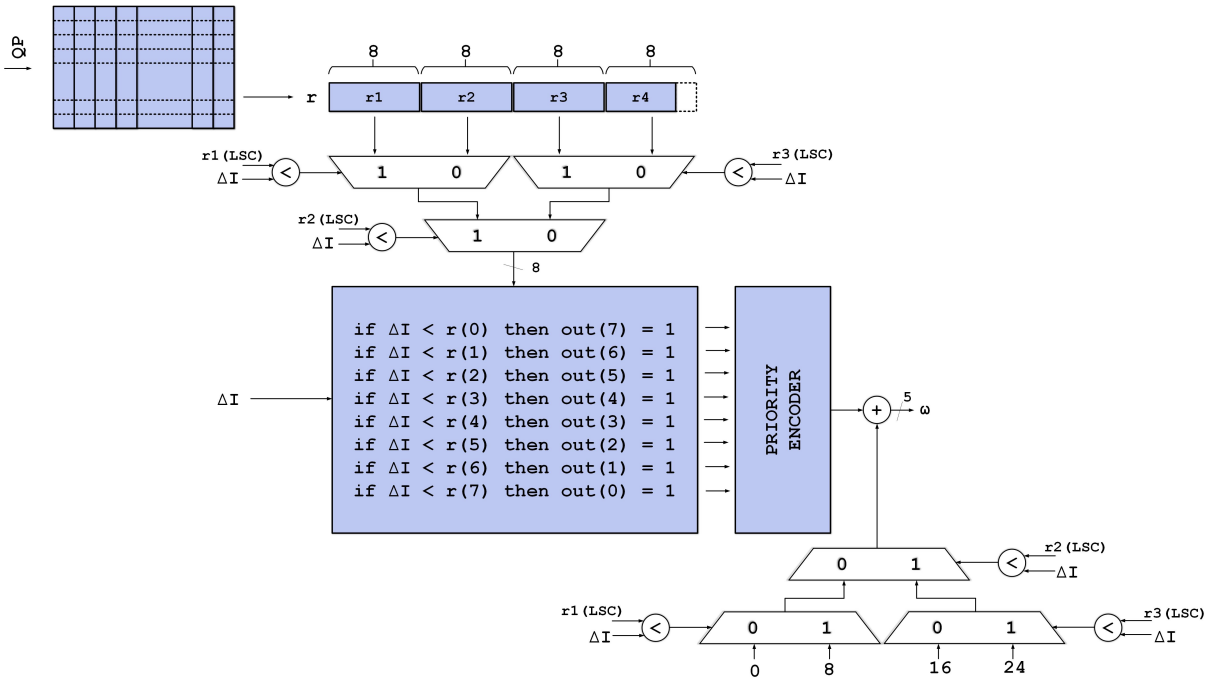

Figure 4.15. Index-based memory architecture with a 8-comparators array. The quarter used is chosen comparing ΔI to the LSCs of the first three slices, that in this picture are named `r1`, `r2` and `r3`.

An increasing values distribution for rows and columns is globally present, so the parallelism can be reduced by isolating some sections. The concept is that smaller numbers need fewer bits to be represented. The vertical division can therefore be realized in 2 or 4 parts according to the degree of optimization to be achieved. It is not necessary to introduce any additional complexity if the defined sub-vectors intervals used to reduce comparators is followed.

We can also think of operating horizontal subdivisions of the same memory: dividing it into 4 specific rows sets (0 to 4), (5 to 10), (11 to 21), (22 to 33) the occupied area is further reduced. However, this change is not free and requires three valuations of QP and multiplexers.

The hypothesis of subdivision into eight sub-tables, as shown in figure 4.16, determines a total of 6710 bits (839 B) used, equal to a reduction of 61.3% compared to the original memory.

Considering to subdivide in sixteen sub-tables, as shown in figure 4.17, a total of 6268 bits (784 B) are used, equal to a reduction of 63.9%.

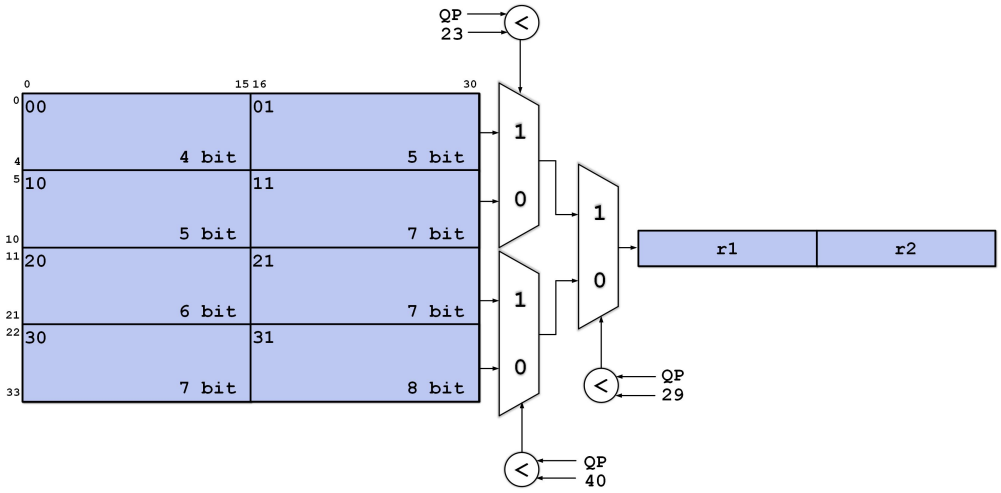

Figure 4.16. Architecture example of the index-based memory subdivided in 8 sub-tables, each one with its own parallelism. Three multiplexer and QP valuations are needed.

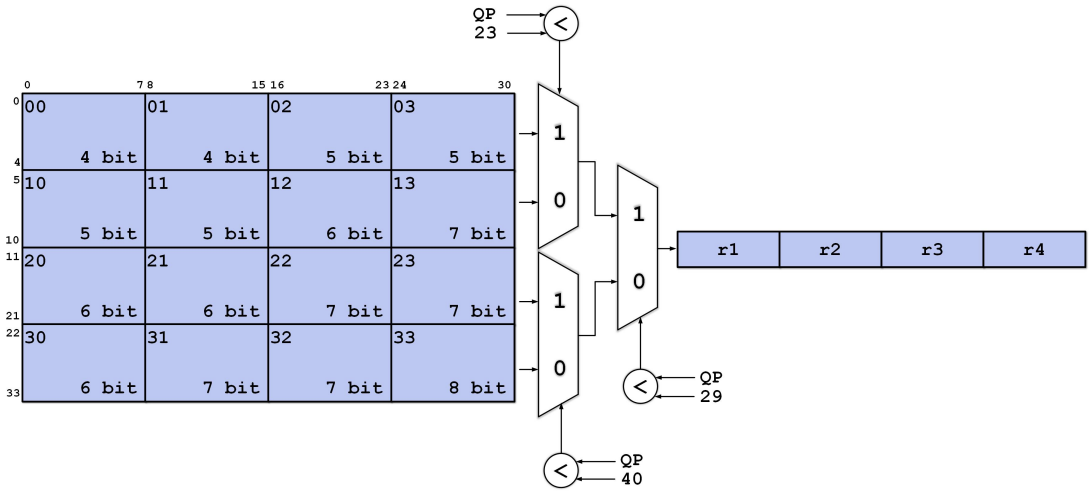

Figure 4.17. Architecture example of the index-based memory subdivided in 16 sub-tables, each one with its own parallelism. Three multiplexer and QP valuations are needed.

4.3.4 Division-Free LUT

In the architecture of the bilateral filter there are two other memories as already discussed and illustrated in figure 4.6. These two units, referred to hereinafter as Multiplier factor LUT and Scaling factor LUT, contain respectively the denominator reciprocal values of the fraction in (3.18) which avoids the division arithmetic

operation, and the scale factor dynamic offsets that allow to keep the approximation error due to the reciprocal calculation quite constant and reduced as possible.

Multiplier factor

The Multiplier factor LUT contains the reciprocal of the values included in the numerical range from 65 to 320. Since the representation must be integer and these values are fractional, they must be suitably rescaled. In figure 4.18 all the values contained in the memory are represented.

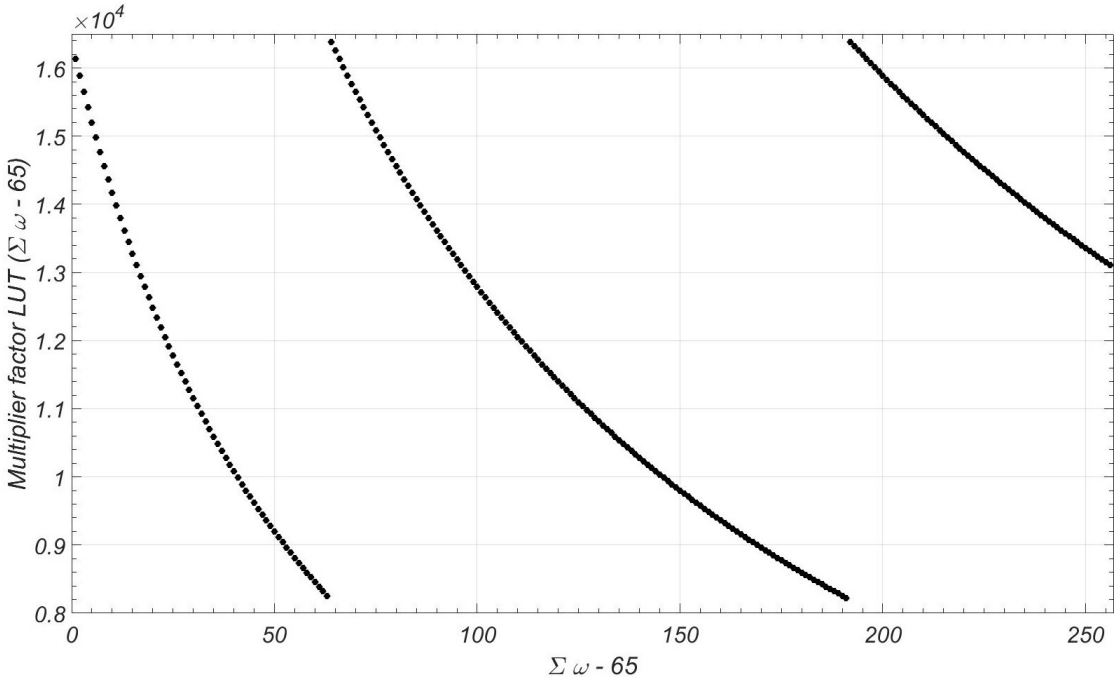

Figure 4.18. Diagram of values contained in Multiplier factor LUT. They represent the denominator reciprocals of the fraction in (3.18) which avoids the arithmetical division operation.

As can be seen, the graph shows a discontinuous trend, which can be divided into three intervals. The first one goes from 65 to 127, a second from 128 to 255, and finally the last one up to 320. At first sight it might seem difficult to find a correlation among them, but just making some transformations, the question is immediately resolved. In fact, if for example the three sections are translated so that the curves have overlapping origins, as illustrated in figure 4.19, it can be seen that the they appear qualitatively similar, but with different compression factors.

By trying to compare the three ranges numerically, the correlation is soon discovered: we have that the first one, that is the curve that goes from 65 to 127, is made up of the elements with even co-ordinates of the second interval, excluding

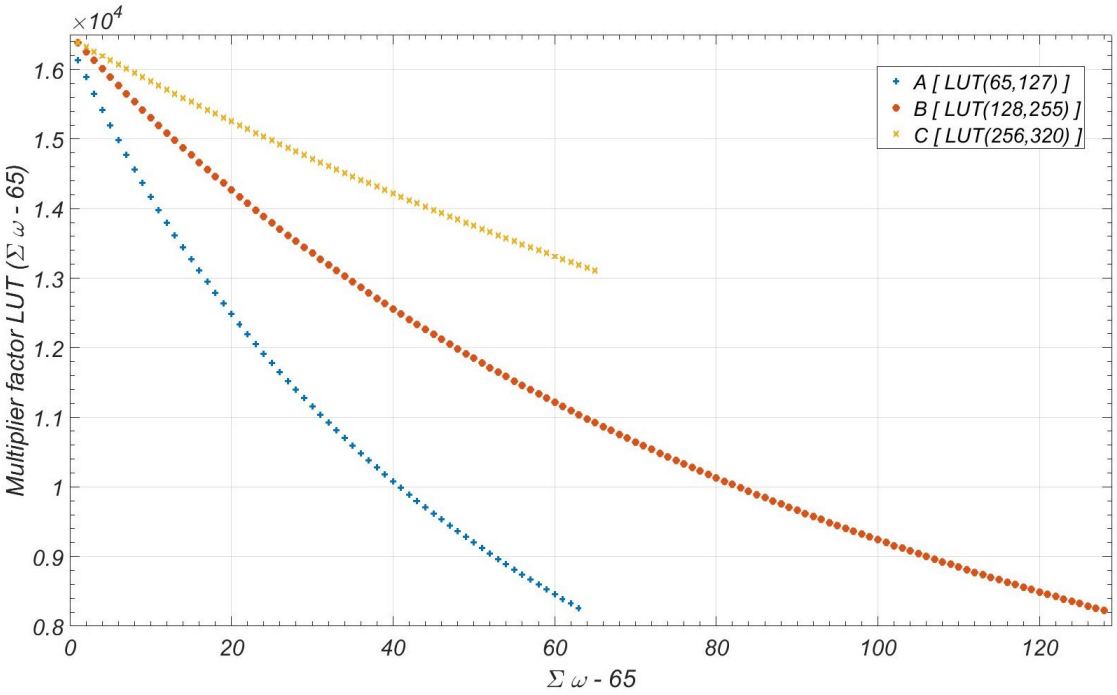

Figure 4.19. Diagram of values contained in the three sections in which Multiplier factor LUT has been divided. They have been translated so that the curves have overlapping origins.

zero. Likewise, it can be shown that this second curve can also be represented by the even points of the third and last interval from 256 to 320. Wanting to better formulate what is written, it can be argued that

$$A(i) = MultiplierfactorLUT(i) \quad \forall i \in [0, 62]$$
$$B(i) = MultiplierfactorLUT(i + 63) \quad \forall i \in [0, 127]$$
$$C(i) = MultiplierfactorLUT(i + 191) \quad \forall i \in [0, 64]$$

and then

$$A1(i) = MultiplierfactorLUT(i) \quad \forall i \in [0, 15]$$
$$A2(i) = MultiplierfactorLUT(i) \quad \forall i \in [16, 62]$$
$$B1(i) = MultiplierfactorLUT(i + 63) \quad \forall i \in [0, 32]$$
$$B2(i) = MultiplierfactorLUT(i + 63) \quad \forall i \in [33, 127]$$
$$A = A1 \cup A2$$
$$B = B1 \cup B2,$$

finally getting

$$A1(j) = B1(2j+2) = C(4j+4) \quad \forall j \in [0, 15]$$
$$A2(j) = B2(2j+2) \quad \forall j \in [16, 62]$$
$$B1(j) = C(2j) \quad \forall j \in [0, 32].$$

In other words it is possible to derive the entire interval from one subsections equal to about two thirds of the original, that is, to the last 160 elements (Multiplier factor LUT(97 to 256)). Thus there is a reduction of 37.5%.

Scaling factor

The Scaling factor LUT also contains 256 2-bit cells as shown in (3.27), for a total of 512 bits (64 B). But we must consider that the values actually are only three, more precisely 0, 1 and 2. They are also ordered in an increasing way, making the memory piecewise constant.

It is immediate to think of applying the same technique used for the coefficients memory. If the address is previously compared to the positions of the first occurrences of a new value, the exact scale factor offset can be obtained without further calculation. It is indeed that

$$k = \begin{cases} 0, & \text{if } D < 127 \\ 1, & \text{if } D < 255 \\ 2, & otherwise \end{cases}$$

with D from (3.27).

The area occupied by the new memory is now equal to 4 bits, with a saving of 99.2%. Figure 4.20 shows the optimized architecture of the Scaling factor LUT.

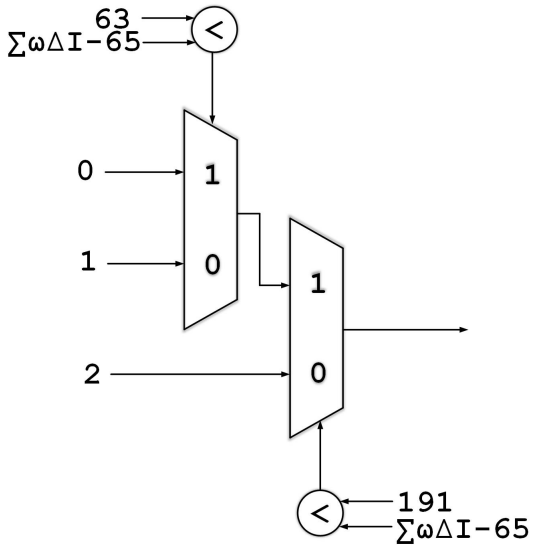

Figure 4.20. Optimized architecture of the Scaling factor LUT. There are just three cells for a total of 4 bit. Two comparators signal a pair of multiplexers in order to address the right value.

Chapter 5

Results

Finally, the results obtained from the architectural synthesis of the solutions described in the chapter 4 are presented in this chapter. The Synopsys Design Compiler software has been used for analysis, with the Ultra Compiler setting enabled. Through it, information concerning area and timings of each proposed architecture has been extracted. For the synthesis the educational library Nangate Open Cell 45nm [30] and the commercial one UMC 65nm [31] Low-K Multi-Voltage Low Leakage have been used. Last one should give more realistic and reliable results, but there are no direct comparisons between the two libraries because would not make sense to compare educational items to commercial ones. The values of both are shown only for academic purposes. Initially, results of the Coefficients LUT optimization for first and second order bilateral filter cases are discussed, followed by the Multiplier factor and Scale factor LUTs compression. To obtain combinatorial delays correct evaluations, an X4 buffer of the respective libraries has been connected to the filter outputs.

5.1 Coefficients LUT

Concerning the Coefficients LUT, results related to the following optimizing architectural solutions were collected and shown in table 5.1:

- *value-based*: Original solution with value-based memory.
- *index-based 32*: Solution with index-based memory and array of 32 comparators, as shown in the figure 4.13.
- *index-based 16*: Index-based memory solution with subdivision in 8 sub-units lower parallelism and 16 comparators array, as shown in figures 4.14 and 4.16.
- *index-based 8*: Index-based memory solution with subdivision in 16 sub-units lower parallelism and 8 comparators array, as shown in figures 4.15 e 4.17.

coefficients LUT	***NanGate 45 nm***		***UMC 65 nm***	
architecture	*area [µm]*	*max delay [ns]*	*area [µm]*	*max delay [ns]*
value-based	3377.67	2.23	6161.04	2.43
index-based 32	1435.34 (-57.5%)	2.20 (-13.5%)	2570.76 (-58.3%)	2.15 (-11.5%)
index-based 16	1500.51 (-55.6%)	3.73 (+67.3%)	2624.40 (-57.4%)	4.43 (+82.3%)
index-based 8	1293.29 (-61.7%)	3.59 (+61.0%)	2356.92 (-61.7%)	3.78 (+55.6%)

Table 5.1. Area and timings results of the LUT Coefficients optimizing architectural solutions synthesis. Nangate Open Cell 45nm and UMC 65nm Low-K Multi-Voltage Low Leakage have been used. The percentage difference between each solution and the reference *value-based* is indicated between brackets.

The architecture for the Coefficients LUT *index-based 8* is certainly the best among the solutions presented in terms of saving resources and reducing the occupied area. The decrease is in fact equal to 61.7% compared to the original *value-based* memory. The cost to be paid is a greater combinatorial delay due to the additional control logic inserted, with an increase of 61% (55.6%) with Nangate (UMC) library compared to the same reference. However, the problem can be solved through the pipelining technique, which is not actually considered because the device does not have a clock, and consequently the synthesis software is not able to insert intermediate registers.

The *index-based 32* solution has a peculiarity: given an area gain of 57.5% (58.3%), there is a reduction of the maximum delay equal to 13.5% (11.5%). This implies higher reachable frequencies and shows how the compression of the memory has advantages also from the point of view of performance, in fact shorter bit and word-lines have a lower parasitic capacity, and therefore a reduced intrinsic delays.

Finally, it can be argued that the results of *index-based 16* are the most disappointing, showing no quantitative advantage over *index-based 8*.

The table 5.2 shows instead the synthesis results of first and second order bilateral filter architectures, for all the above mentioned memory solutions. On the other hand, it does not include the optimization applied to the LUTs related to the division yet.

We have that *index-based 8* solution still reaches best performance and area reduction for both filter order. The total area occupation decrease from 9344.58

(17522.28) μm to 7085.27 (13569.12) μm in the first order case and using Nangate (UMC) library, bringing to a compression equal to 24.2% (22.6%). Since now pipelining can be applied due to sequential units physical presence, timings results are basically unaffected by added complexity. It is clear that considering the entire bilateral architecture, the effective gains are quantitatively pulled down. Anyway, coefficients memory optimization outcomes are definitely positive. On the contrary, even though *index-based 32* and *index-based 16* architectures achieve good area reduction however lower than the 8 comparators version, on the other hand performance is worse and leads to not considering as valid these solutions.

Second order filter results are qualitatively similar to the first case. Area performance reaches a compression of 31.7% (30.4%) for the *index-based 8* architecture. It is worthy of attention the maximum delay decrease of 6.3% using the UMC library.

bilateral filter architecture	***order***	***NanGate 45 nm***		***UMC 65 nm***	
		area [μm]	*max delay [ns]*	*area [μm]*	*max delay [ns]*
value-based	*I*	9344.58	0.98	17522.28	0.76
index-based 32	*I*	7283.35 (-22.1%)	1.00 (+2.0%)	13741.56 (-21.6%)	0.80 (+5.3%)
index-based 16	*I*	7413.42 (-20.7%)	0.97 (-0.1%)	14018.04 (-20.0%)	0.78 (+2.6%)
index-based 8	*I*	7085.97 (-24.2%)	0.97 (-0.1%)	13569.12 (-22.6%)	0.76 (+0%)
value-based	*II*	15502.75	0.99	30091.68	0.80
index-based 32	*II*	11019.58 (-28.9%)	1.02 (+3.0%)	20964.96 (-30.3%)	0.80 (+0%)
index-based 16	*II*	11195.14 (-27.8%)	1.04 (+5.1%)	21060.72 (-30.0%)	0.81 (+1.3%)
index-based 8	*II*	10591.06 (-31.7%)	1.00 (+1.0%)	20955.60 (-30.4%)	0.75 (-6.3%)

Table 5.2. Area and timings synthesis results of first and second order bilateral filter architectures. All the solutions in table 5.1 have been considered. On the other hand, the optimization applied to the LUTs related to the division is not included. The percentage difference between each solution and the reference *value-based* is indicated between brackets.

5.2 Division LUTs

Eventually, the Multiplier factor and Scale factor LUTs optimization results applied to the bilateral filter are shown in table 5.3. The values are related once again to all solutions for the first and second order cases. The figure 5.1 graphically illustrates the distribution of the results of table 5.3 on an area vs delay diagram.

optimized BF architecture	***order***	***NanGate 45 nm***		***UMC 65 nm***	
		area [μm]	*max delay [ns]*	*area [μm]*	*max delay [ns]*
value-based	*I*	8558.55 (-9.2%)	1.05 (+6.7%)	15405.12 (-13.7%)	0.75 (-1.3%)
index-based 32	*I*	6546.53 (-42.7%)	0.96 (-2.1%)	12601.80 (-39.0%)	0.70 (-8.6%)
index-based 16	*I*	6474.17 (-44.3%)	0.95 (-3.2%)	12638.88 (-38.6%)	0.72 (-5.6%)
index-based 8	*I*	6388.26 (-46.3%)	0.98 (+0%)	12265.92 (-42.9%)	0.68 (-11.8%)
value-based	*II*	14695.17 (-5.5%)	1.00 (+1%)	28438.56 (-5.8%)	0.75 (-6.7%)
index-based 32	*II*	10160.93 (-52.6%)	0.98 (-1%)	19161.72 (-57.0%)	0.75 (-6.7%)
index-based 16	*II*	10408.85 (-48.9%)	1.03 (+3.9%)	20409.48 (-47.4%)	0.73 (-9.6%)
index-based 8	*II*	10119.97 (-53.2%)	1.01 (+2.0%)	18959.76 (-58.7%)	0.73 (-9.6%)

Table 5.3. Area and timings of the architectural synthesis results for first and second order bilateral filter, including the optimization of the Multiplier factor and Scale factor LUTs. All solutions in table 5.1 have been considered.The percentage difference between each solution and the reference *value-based* in table 5.2 is indicated between brackets.

In this last optimized configuration we can observe that the Multiplier factor and Scale factor LUTs has a proper impact in area compression between 9.2% and 13.7% for first order, and between 5.5% and 5.8% for the second one, respect to the table 5.2. This percentages should be quite constant for any solution. Comparing the original *value-based* architecture to the *index-based 8* one in table 5.3, the compression reaches total values from 42.9% up to 58.7%.

Timings are quite similar for the NanGate library, they instead turn to be way better with the UMC one, due to the probably lower intrinsic delay of the new smaller memories. The *index-based 8* architecture confirms to be the best one among the others. The area reduction is the greater one for each case of order and library. Maximum delay reaches really good results for the first order filter, with a value less than 11.8% respect to the *value-based* solution, and less than 9.6% for the second order.

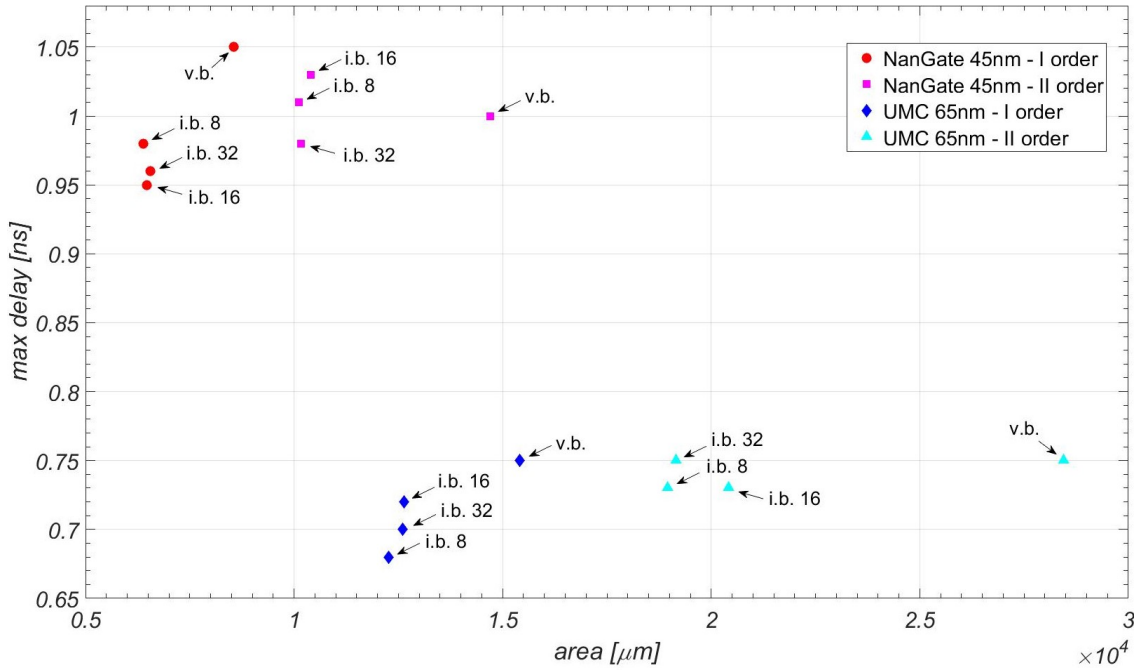

Figure 5.1. Area vs delay diagram of the architectural synthesis results in 5.3 for first and second order bilateral filter, including the optimization of the Multiplier factor and Scale factor LUTs. Each dot represent a specific solution. The acronyms *i.b.* and *v.b.* mean respectively *index-based* and *value-based*.

Chapter 6

Conclusions

In this thesis we presented an optimized architecture for the bilateral filter stage inside the JEM decoding loop. The original idea was to find a way to reduce area without sacrificing performance, but if possible, to increase it. The target throughput for desired resolutions and frame-rate, as already discussed in chapter 4, are:

- *FHD* 186.624 MPixel/s
- *UHD4K* 746.496 MPixel/s
- *UHD8K* 2985.984 MPixel/s.

The UHD8K is not exactly part of the purpose of this work, since it is a emergent world but the moment when it will become standard is still far away, and the related technology is actually on an early stage. Anyway, some considerations can still be done.

In the results chapter 5, we illustrated as the optimization operations on the involved LUTs have led to significant outcomes. In the best case, Coefficients LUTs has been compressed by a factor of 61.7%, allowing a global reduction between 24.2% (22.6%) and 31.7% (30.4%) for NanGate (UMC), depending on the filter order. The work on Multiplier factor and Scale factor LUTs has given a proper average area gain between 5.5% (5.8%) and 9.2% (13.7%), pulling down the total value up to 53.2% (58.7%). Furthermore, even timings got better, with NanGate there are some fluctuations around the reference value, but with UMC library they decrease up to 11.8% for first order and 9.6% for second one.

It has been show that the proposed *index-based 8* approach for the bilateral filter architecture, using the UMC 65 nm library, is promising since it allows an area reduction of 42.9% respect to the original one, reaching a total value equal to 12265.92 μm. Moreover, the critical path has a delay equal to 0.68 ns, enabling a

maximum frequency of 1.47 GHz. And since the unit can process a pixel for clock cycle, its throughput is 1.47 GPixel/s. So, FHD and UHD4K requirements have been widely satisfied and the filter is able to handle these high resolutions.

Anyway, in this work a second order filter it has also been presented. With just an area 54.6% larger than the first order case, we reach a double throughput. The related maximum delay is 0.73 ns, so the achievable frequency is 1.37 GHz. It is clearly slower, but the number of processed pixels for clock cycle is equal to 2.74 GPixel/s. Thus the maximum resolution allowed is 7357×4138, still insufficient for UHD8K. This means that better technology nodes are requested, or, in another way, a fourth order bilateral filter attempt can be pursued.

However, higher frequencies are not useless at all. Just for example, with the second order filter a throughput of 2.74 GPixel/s can be reached with 1.37 GHz, so, since for UHD4K just 746.496 MPixel/s are necessary, the clock can be slowed of 3.65 times. This has a huge impact in power consumption since it decrease of the same factor. Therefore, power estimation can be the next step for this architectural proposal, and low consumption techniques applied to it are eligible as future work.

Bibliography

[1] Tim Bemers-Lee, *Information Management: A Proposal* CERN DD/OC, March 1989. [Online] *http://cdsweb.cern.ch/record/1405411/files/ARCH-WWW-4-010.pdf*

[2] Cisco, *The Zettabyte Era: Trends and Analysis* June 2017. [Online] *https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.pdf*

[3] [Online] *https://jvet.hhi.fraunhofer.de/svn/svn_HMJEMSoftware/branches/HM-16.6-JEM-7.1-dev/*

[4] G. Sullivan, J. Ohm, W. Han and T. Wiegand,*Overview of the High Efficiency Video Coding (HEVC) Standard*, IEEE Transaction on Circuits and Systems for Video Technology, vol. 22, no. 12, pp 1649-1668, December 2012.

[5] [Online] *https://www.itu.int/en/ITU-T/studygroups/2017-2020/16/Pages/video/vceg.aspx*

[6] [Online] *https://mpeg.chiariglione.org/*

[7] *Video codec for audiovisual services at p x 384 kbit/s - Recommendation H.261 (11/88)*, ITU-T Std., 1988.

[8] *Video codec for audiovisual services at p x 64 kbit/s - Recommendation H.261 (30/93)*, ITU-T Std., 1993.

[9] *Video coding for low bit rate communication - Recommendation H.263 (01/05)*, ITU-T Std., 2005.

[10] *Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s - Part 2: Video - ISO/IEC 11172-2*, ISO/IEC, 1993.

[11] *MPEG-4 Overview - (V.21 – Jeju Version) - ISO/IEC JTC1/SC29/WG11 N4668*, ISO/IEC, March 2002.

[12] *Information technology – Generic coding of moving pictures and associated audio information: Video - Recommendation H.262 (02/00)*, ITU-T Std., 2000.

[13] D. Marpe, T. Wiegand and G. Sullivan, *The H.264/MPEG4 Advanced Video Coding Standard and its Applications*, IEEE Communications Magazine, 2006.

[14] [Online] *http://phenix.int-evry.fr/jct/*

[15] G. Sullivan, M. Bugadavi and V. Sze, *High Efficiency Video Coding (HEVC) Algorithms and Architectures*, Springer, 2014.

[16] *High efficiency video coding - Recommendation H.265 (02/18)*, ITU-T Std., 2018.
[17] [Online] *http://phenix.it-sudparis.eu/jvet/*
[18] W. Gao and S. Ma, *Advanced Video Coding Systems*, Springer, 2014.
[19] J. Chen, E. Alshina, G. Sullivan, J. Ohm, J. Boyce, *JVET-G1001: Algorithm description of Joint Exploration Test Model 7 (JEM7)*, ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, July 2017.
[20] K. Ramamohan Rao and P. Yip, *Discrete cosine transform: algorithms, advantages, applications*, Accademic Press, 2014.
[21] L. Merrit and R. Vanam, *x264: A high performance H.264/AVC encoder*, 2006.
[22] A. Norkin, G. Bjøntegaard, A. Fuldseth, M. Narroschke, M. Ikeda, K. Andersson, M. Zhou and G. Van der Auwera, *HEVC deblocking filter*, IEEE transaction on Circuits and systems for Video Technology, vol. 22, no. 12, pp. 1746-1754, 2012.
[23] C. Fu, E. Alshina, A. Alshin, Y. Huang, C. Chen, C. Tsai, C. Hsu, S. Lei, J. Park and W. Han , *Sample adaptive offset in the HEVC standard* IEEE transaction on Circuits and systems for Video Technology, vol. 22, no. 12, pp. 1755-1764, 2012.
[24] C. Tomasi and R. Manduchi, *Bilateral Filtering for Gray and Color Images*, Proceedings of the 1998 IEEE International Conference on Computer Vision, Bombay, India, 1998.
[25] G. Sapiro and D.L. Ringach, *Anisotropic diffusion of color images*, IEEE Transactions on Pattern Analysis and Machine Intelligence (Volume: 12 , Issue: 7 , Jul 1990).
[26] A. Nosratinia, *Enhancement of JPEG-Compressed Images by Re-application of JPEG*, Journal of VLSI signal processing systems for signal,image and video technology, vol. 27, pp. 69–79, Feb. 2001.
[27] F. Durand and J. Dorsey, *Fast bilateral filtering for the display of high-dynamic-range images*, ACM Transactions on Graphics (TOG), Volume 21, Issue 3, Pages 257-266, July 2002.
[28] J. Strom, K. Andersson, P. Wennersten, M. Pettersson, J. Enhorn and R. Sjoberg, *JVET-F0096, EE2-JVET related: Division-free bilateral filter*, ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, April 2017.
[29] P. Wennersten, J. Strom, Y. Wang, K. Andersson, R. Sjoberg and J. Enhorn, *Bilateral Filtering for Video Coding*, 2017 IEEE Visual Communications and Image Processing (VCIP), December 2017.
[30] [Online] *http://projects.si2.org/openeda.si2.org/projects/nangatelib*
[31] UMC, *UMK65LSCLLMVBBR_B UMC 65nm Low-K Multi-Voltage Low Leakage RVT Tapless Standard Cell Library Databook*, 2014.