Design of a FPGA based MPEG post-processing unit

Massimo Petricca

Kongens Lyngby 2008
IMM Master Thesis-2008-95
<table>
<thead>
<tr>
<th>CHAPTER 1</th>
<th>Introduction</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.1 ASSUMPTION</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.2 TECHNOLOGICAL ISSUE</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.3 THE NEED FOR POST-PROCESSING</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1.4 THESIS OBJECTIVE</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>1.5 THESIS OVERVIEW</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHAPTER 2</th>
<th>Deblocking algorithm</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.1 FILTERING METHODS</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>2.1.1 Selective low-pass filters on block edges</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>2.1.2 Deblocking algorithms</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>2.1.3 Adaptive deblocking algorithms</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>2.2 THE IMPLEMENTED DEBLOCKING FILTER</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>2.2.1 QS control scheme</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>2.2.2 Adaptive deblocking filter</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>2.2.3 Filtering behavior tests</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHAPTER 3</th>
<th>Design</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.1 PPU ARCHITECTURE OVERVIEW</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>3.2 2D DISCRETE COSINE TRANSFORM ENGINE</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>3.2.1 Classes and comparisons of fast DCT algorithms</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>3.3 MPEG QUANTIZATION PARAMETER ESTIMATION ENGINE</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>3.3.1 Multipliers Stage</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>3.3.2 N2 histogram estimator</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>3.3.3 DCTs non-zero counter</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>3.3.4 N3 histogram estimator</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>3.3.5 QS estimation for the MB</td>
<td>57</td>
</tr>
</tbody>
</table>
Aknowledgments

There are many people who directly or indirectly contributed to the achievement of the present work. So, with these few lines, I wish to express my sincere thanks to their own.

First, many thanks to my parents and my brother, who have supported the relevant economic weight of my Danish experience. I wish to thank Valentina for her maturity and patience demonstrated in supporting the unpleasant situation of isolation during the long period of time I spent on the project instead of her.

Special thanks are due to Prof. Marco Re, because, without him, the Danish experience would not have been possible, and the Prof. Alberto Nannarelli which, with his own experience, has helped to guide me through a technical preparation, from a side, and through a maturity in everyday life, from another side. I have appreciate the help during the different steps of the project made by Prof. Søren Forchhammer and Huiying Lee, therefore, heartfelt thanks also to them. I wish to thank Prof. Flemming Stassen which has made Denmark a more hospitable country.

I would finally like to thank the Italian guys I met during the last seven months away from Italy for their hospitality and especially because they gave me the feeling to be less far from home than I was: Paolo, Mattia, Matteo, Nicolò, Francesco, Davide. Special thanks go to Davide because he was always ready to support me about the repeated technical difficulties encountered during the project.

Massimo Petricca
CHAPTER 1

Introduction

1.1 Assumption

Nowadays, home digital television on high resolution screens is becoming more attractive as demonstrated by the growing of the wide flat-panel market [1].

The applications oriented to the digital television (SDTV or HDTV) like broadcasting, for example, are based on MPEG standard compression [2].

1.2 Technological issue

This type of standard represents the ideal solution to allow the diffusion of the video information on a narrow bandwidth transmission channels (e. g. cable or satellite).

From another side, we have to take into account that compression, due to its nature, adds a certain level of degradations to the original transmitted images so, if we also consider that a wide screen magnify these degradations (because of larger pixels), the final result will be an overall reduction of the image quality.

From some considerations about the standard MPEG, we can observe that most of the codec, video encoder and decoder, are designed as a block-based motion-compensated hybrid transform coder, where the transformation is done by a Discrete Cosine Transform (DCT) on 8x8 pixels blocks. Further-
more, the DCT coefficients are quantized through a quantization parameter called QS (also called QP). This last step is the main responsible in the deterioration of the video image quality which expresses itself into two different types of visible artifacts: blocking and ringing. The blocking artifacts are seen as unnatural discontinuity between pixel values of neighboring blocks due to independence of the compression process which is performed on different blocks, while, the ringing artifacts are seen as high frequency irregularities around the image edges due to the quantization step for the high frequency components.

1.3 The need for POST-PROCESSING

Considering such an outline, it is easy to think that there is the need for a better image quality, today. Since it is not possible to simply increase the transmission channel bandwidth to satisfy this request, POST-PROCESSING appears to be the most feasible solution because it does not require any existing standards to be changed and because it is also capable to maintain the computational effort under a proper level suitable for real-time applications. In Figure 1.3.1 is shown the MPEG block diagram scheme. By the use of this type of processing, the designer is able to interface the filter architecture directly at the output of a MPEG decoder with all the required flexibility.

Figure 1.3.1  MPEG-2 block diagram. A “post-processing filter” block is shown starting from MPEG decoded output.
1.4 Thesis objective

Furthermore, it is important to underline that the post-processing is aim at real-time applications so, to ensure this task, the improvement of the image quality has to be escorted by a low computational complexity. By this view of the topic, the thesis object is to design a FPGA based MPEG post-processing unit in which the filter is selectively applied through a DCT based scheme which use an estimation of the Quantization Parameter to control an adaptive deblocking filter.

1.5 Thesis overview

The rest of the report is structured in the following way:

− **Chapter 2** introduces and analyzes the behavior of the implemented algorithm used to control the adaptive deblocking filter. The behavior of the deblocking filtering algorithm will be presented and analyzed, in this chapter, as well.

− **Chapter 3** describes and analyzes the architecture of the designed post-processing filter. Some Matlab simulations have been performed in order to investigate the performance of the implemented architectures.

− **Chapter 4** presents the obtained results and conclusions of this thesis regarding the implemented design on the FPGA.
A deblocking algorithm is a filter applied to blocks of 8x8 pixels in decoded video to improve visual quality and prediction performance by smoothing the sharp edges which can form between blocks when block coding techniques are used. The filter aims to improve the appearance of decoded pictures.

In the present chapter will be introduced the deblocking algorithm implemented for the project.

2.1 Filtering methods

In frame pictures compressed using techniques based on the quantization of the DCT coefficients and, above all, in cases of high ratio of compression, it is possible to see the blocking artifacts of the images. As already said, the blocking artifacts are seen as unnatural discontinuity between pixel values of neighboring blocks due to independence of the quantized coefficients originate by the processing on different blocks. Figure 2.1.1 shows an example of blocking artifacts (Figure 2.1.2 display its enlargements).

There are two different approaches used to improve the image quality:

− using optimized techniques for the compression;
− Applying a post-processing on the images affect by artifacts.

Even if in some cases the first methods is able to guarantee excellent results, it is considered few efficient due to the increased complexity behind the
code/encode used algorithms. Furthermore, the optimized techniques imply to modify the coder and decoder stages.

![Figure 2.1.1: Blocking artifact in a picture compressed by the MPEG coder.](image)

The second method, instead, has considered a better choice to improve the image quality because of mainly two reasons: it is able to guarantee a less computational complexity and it does not need the coder or decoder stages to be modify. However, in the second methods, the post-processed image quality results improved as much as necessary.

Figure 2.1.2 shows the enlargements of the details of the picture in Figure 2.1.1 which are bordered with a red rectangle. In the crossing from a block to its neighbor, it is possible to observe that the blocking artifacts appears like rough steps of different luminance values resulting in a high frequencies contribution introduced into the image after the code and decode stages. Therefore, in a first analysis, a low-pass filter might be the right choice to correctly improve the image quality in the post-processing stage.

Indeed, in the design of a deblocking post-processing technique, three objects must be guarantee, that is:
2.1 Filtering methods – Selective low-pass filters on block edges

- nice reduction of the undesired effects;

- preservation of the information contained in the image and its high frequencies components;

- simplified implementation (that is, reduced complexity) for real-time post-processing.

In order to accomplish the just listed objects, a simple low-pass filter cannot be taken into account. Instead using a simple low pass filter, for example, it is possible to selectively use some proper filtering algorithm in order to obtain a reduction in the computation effort and an improvement on the image quality.

2.1.1 Selective low-pass filters on block edges.

The idea in using a low pass filter is not completely wrong. Indeed, a low-pass filter might help to reduce the blocking artifacts but also degrades the image quality introducing some blur and therefore reducing the image’s details.
Deblocking algorithm

To preserve that nice feature, the low-pass filtering could be organized to be applied selectively to the image processing only the block edges, in order to reduce the blur it introduces. On the other side, such an idea to apply the low-pass filter can grantee a less computational complexity. Therefore, Figure 2.1.1.1 shows an example of low-pass filter applied on the edges of an 8x8 block.

![Figure 2.1.1.1: Example of a selective low-pass filtering on block edges.](image)

In other words, the filtered image is obtained performing the following formula, where “h” is the filter matrix, “x” the pixels to filter and “y” the filtered pixels.

\[
y(n_1, n_2) = \begin{cases} 
\sum_{m_1}^{M_1} \sum_{m_2}^{M_2} h(m_1, m_2) \cdot x(n_1 - m_1, n_2 - m_2) & \text{on edges} \\
x(n_1, n_2) & \text{otherwise}
\end{cases} \quad (2.1.1.1)
\]

Therefore, the low-pass filter applied only on the block edges in the picture, can improve the quality of a compressed image and also reduce the blur which is typical of the low-pass filtering. However, the computational complexity can still remain quite high.
2.1 Filtering methods – Deblocking algorithms

2.1.2 Deblocking algorithms

In order to reduce the computational complexity behind the image processing, it is necessary to employ different algorithms specifically designed for such a purpose: that is the case of the deblocking algorithms family. There are many different methods adopted to improve the image quality and also to leave the computational complexity under a certain low level. Making an example, one algorithm could be referred to Figure 2.1.2.1.

\[
x = C - B \\
A' = A + x/8 \\
B' = B + x/8 \\
C' = C - x/8 \\
D' = D - x/8
\]

Figure 2.1.2.1: Example of a deblocking filter on block edges.

In Figure 2.1.2.1, the pixels A, B, C and D are replaced with the values A', B', C' and D' evaluated starting from the difference between the neighbor pixels B and C. In this way, the original pixel value is modified to an intermediate value in order to mitigate the rough luminance step between a block and its neighbor. The main feature of such algorithms consists in the smoothing behavior it can guarantee and, at the same time, the reduced computational complexity it involves.

2.1.3 Adaptive deblocking algorithms

The algorithm presented in the previous section allows the image filtering on only 4 pixels of the image per time. The technique it uses is defined soft due to its characteristic to preserve the image’s details. However, in some cases may be necessary use a strong filter to reduce the hard luminance steps. A method used to filter the image, satisfying both the purposes to perform a strong filtering and to preserve the image details, is the adaptive deblocking
algorithm. Figure 2.1.2.2 shows an example of the processing performed by such a method.

In the way to perform a deblocking filtering shown in Figure 2.1.2.2, it is possible to chose if perform a strong filtering (in the cases it is necessary) or a default filtering (in case it is not necessary) an all the block edges. The difference values used to apply the choice to performing the filter, is related to more than only two pixels. In cases the differences \((x_1\) and \(x_2\) in Figure 2.1.2.2) are quite big, it means that the image contains a lot of details to preserve in that edge and thus the soft filter is applied, meanwhile, in other cases the image need to have a more performed filter to be applied. Concluding, the final resulting image will have more details preserved and more strong filtered zones which had the need to be strong filtered.

### 2.2 The implemented deblocking filter

After a briefly introduction on the different types of post processing filter, let now to consider the filter algorithm has been implemented in the presented work.

Referring to Section 2.1.3, it is possible to include the implemented filtering algorithm into the adaptive deblocking filter family. The following list summarizes the main features of the implemented algorithm:
2.2 The implemented deblocking filter – QS control scheme

- adaptive algorithm controlled by the estimated MPEG Quantization Parameter (QP or QS, as well);
- two mode of filtering: default mode (weak) and DC_offset mode (strong);
- deblocking module performed on both horizontal and vertical block boundaries.

The implemented deblocking filtering algorithm proposed by Forchhammer and Li [3] is focused on an efficient post-processing design which is based on a new DCT control scheme which allows to significantly reduce the load of computation. Due to that feature, the algorithm is also suitable for the HD (High Definition) television MPEG stream.

In the following section will be presented the detailed on the behavior of the algorithm.

2.2.1 QS control scheme

As already said, the adaptive filtering algorithm implemented in the present work is based on a control scheme which is able to estimate the Quantization Parameter (QP or also QS) at the macroblock level. Using an estimation of the QS parameter is possible to control the application of the adaptive filter which, thus, results more selective. In this section, it will be presented the behavior of the QS estimation algorithm.

The QS estimator works on the MPEG-2 decoded video stream. Figure 2.2.1.1 shows an MPEG decoder block diagram. It is composed of: a Variable Length Decoder block (VLD), a Inverse Quantization block (IQ), a Rounding block (ROUND), a Inverse Discrete Cosine Transform block (IDCT), a Motion Compensation block (MC), a single frame Memory buffer block (MEM).

Starting from the leftmost side of the above block diagram, the output of the MPEG-2 coder (the MPEG-2 encoded data stream) enters in the VLD block in which a variable length decoding is performed to obtain the QI(\(u, v\)) integer coefficients. The so obtained QI(\(u, v\)) coefficients are then inversely quantized by the IQ block to obtain the DCT coefficients, that is, the QF(\(u, v\)) coefficients, in the above figure. The inverse quantization is differently performed for DC or AC coefficients. Once the QF(\(u, v\)) values are reconstructed, the IDCT is then applied. The Inverse Discrete Cosine Transformation converts the so obtained values from the coefficients space to the pixels space, that is, into the \(r(u, v)\) values shown in figure. The \(r(u, v)\) usually are fractional values and thus a rounding is performed to bring those
values into the integer range $[0, 255]$ to obtain the $r'(u, v)$ values displayed in Figure 2.2.1.1.

The QS estimator focuses its work on the reconstructed AC coefficients $r'(u, v)$. In Equation 2.2.1.1 it is possible to see the formula applied in performing such a reconstruction. In such an equation “$w$” corresponds to the label of different quantization matrix QM ($w=1$ is the index relatives to the intra block quantization matrix). The quantization matrix (QM) is written in Equation (2.2.1.2).

$$QF(u, v) = \frac{[2QI(u, v) + k] \cdot QM(w, u, v) \cdot QS}{32} \quad (2.2.1.1)$$

where

$$k = \begin{cases} 
0 & \text{intra block} \\
\text{sign}[QI(u, v)] & \text{non – intra block} 
\end{cases}$$
2.2 The implemented deblocking filter – QS control scheme

Furthermore, QI, QM and also QS are all integer values. The QM’s object is to assign different quantization weight to the DCT coefficients which are in different position in the \( QI(u, v) \) matrix block.

\[
\text{QM} = \begin{bmatrix}
8 & 16 & 19 & 22 & 26 & 27 & 29 & 34 \\
16 & 16 & 22 & 24 & 27 & 29 & 34 & 37 \\
19 & 22 & 26 & 27 & 29 & 34 & 34 & 38 \\
22 & 26 & 27 & 29 & 32 & 35 & 40 & 48 \\
26 & 27 & 29 & 32 & 40 & 48 & 58 & 58 \\
27 & 29 & 35 & 46 & 56 & 69 & 83 & 83 \\
\end{bmatrix}
\] (2.2.1.2)

The purpose is to use smaller values to quantize lower frequencies components and bigger values to quantize higher frequencies components (to obtain the compression). Typically, a unique QM is used to quantize the whole MPEG-2 sequence. The default quantization matrix taken into account is then shown in Equation (2.2.1.2).

The QS value shown in Equation (2.2.1.1) is the quantization scale which can be different from MB to MB. Table (2.2.1.1) shows the allowed QS values which the MPEG-2 coder model takes into account in the present work can use to codify the data.

<table>
<thead>
<tr>
<th>QS value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>

Table 2.1.2.1: Quantization Scale values.

The goal of the QS estimation algorithm is to estimate the QS values for every intra MB in the decoded stream. Therefore, the DCT is performed on the MBs of the decoded data stream in both Progressive mode (P-mode) and Interlaced mode (I-mode) as shown in Figure 2.2.1.2.

As will be explained in Chapter 3, the two type of DCT so computed are used to decide whether the decoded MB has been coded using a field or a frame coding. Therefore, the 64 coefficients of each DCT 8x8 block are compared with a threshold (which could be 0.2 or 0.25, as well) and is then performed a count about how many coefficient are greater than the chosen
Deblocking algorithm

threshold. The count is computed for both P-mode and I-mode DCT MB to obtain two different counts of non-zero coefficients which come from different DCT mode, as already said.

Comparing once again the resulting counts, it is possible to assert if the MB was coded by P-mode or I-mode DCT: the minimum count value will give the DCT mode label to the MB. After the non-zero coefficients count has been performed, it is possible to choose one of the two DCT MB coefficients to perform the following operations which will bring to obtain an estimation on the QS for the MB.

Once the DCT coefficients are chosen, the QF(\(u, v\)) values are obtained and, using the known QM, it is possible to reconstruct the QI(\(u, v\)) \cdot QS products performing Equation (2.2.1.3) which is the inverse of Equation (2.2.1.1).
2.2 The implemented deblocking filter – QS control scheme

\[ QI(u, v) \cdot QS = \frac{16 \cdot QF(u, v)}{QM(1, u, v)} \]  \hspace{1cm} (2.2.1.3)

Once used the algorithm designed by Huiying for the QS parameter estimation, the following steps summarize the operations which must be performed on the QF(u, v) coefficients to obtain the histogram analysis and then the QS value:

- computing \( QI(u, v) \cdot QS \) products using Equation (2.2.1.3);
- rounding \( QI(u, v) \cdot QS \) to the closest even integer values, labeled \( K(u, v) \);
- setting all the DC coefficient value, and also all the AC coefficients less than 4, to zero;
- performing an histogram analysis of \( K(u, v) \) and storing the results into a 3x30 array labeled \( Nq_p \), where \( q \in [1,3] \) and \( p \in [1,30] \);
- evaluating \( QS = \max_{N1.p} [N2.p + N3.p] \).

Therefore, the first step involves the computations of Equation (2.2.1.3). Once the \( QI(u, v) \cdot QS \) are obtained, a rounding operation to the nearest even integer value is evaluated on all the coefficients. The DC coefficient and the AC coefficients lower than 4 are, then, discarded. Indeed, in the presented implementation of the algorithm the value 4 has been also taken into account (see Section 3.2.1, 3.2.2, 3.2.4). The \( K(u, v) \) values so obtained, are then subjected to a histogram analysis. The following example will show in which way the histogram estimation is performed.

In Table 2.1.2.2 is shown an \( Nq_p \) array which is possible to obtain performing the above steps (the true QS value of the example correspond to 12). In the column \( N1 \) there is the list of all the \( QI(u, v) \cdot QS \) products observed in the MB under evaluation once the first three steps are computed according with QS values shown in Table 2.1.2.1. The \( N2 \) column element, represents the number of repetitions of the \( N1 \) value (in its same table’s row) for the MB. The \( N3 \) column is formed computing an addition between those \( N2 \) elements referred to \( N1 \) elements which can be divisors for other \( N1 \) element. For example, \( N3 \) element in the first row is formed computing an addition between \( N2_6 \), \( N2_{12} \), \( N2_{24} \) and \( N2_{48} \) in which the relative \( N1 \) element, that is 6, is a common divisors for each \( N1 \) element relative to the listed \( N2_p \) element which take part in the addition.
The estimation algorithm is based on the following assumption and properties:

- \( N_{3.6} \geq N_{3.12} \geq N_{3.24} \geq N_{3.48} \);
- \( N_{2,p} \) where \( p \) is lower than the true QS has a very high potential to be zero. The exceptions are due to the error;
- \( N_{2,p} \) where \( p \) is equal to the true QS is a very high potential to be the maximum among \( N_{2} \) elements. The exceptions are due to the error and also to the maximum \( N_{2} \) element can be located into the multiple of the true QS value.

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>( N_{2,6} )</td>
<td>( N_{2,6} + N_{2,12} + N_{2,24} + N_{2,48} )</td>
</tr>
<tr>
<td>12</td>
<td>( N_{2,12} )</td>
<td>( N_{2,12} + N_{2,24} + N_{2,48} )</td>
</tr>
<tr>
<td>24</td>
<td>( N_{2,24} )</td>
<td>( N_{2,24} + N_{2,48} )</td>
</tr>
<tr>
<td>48</td>
<td>( N_{2,48} )</td>
<td>( N_{2,48} )</td>
</tr>
</tbody>
</table>

Table 2.1.2.2: The relation among \( N_{1}, N_{2} \) and \( N_{3} \) histograms for the QS estimation in a MB.

Combining the above properties and assumption, the algorithm estimates the QS value solving Equation (2.2.1.4), that is, it takes the \( N_{1} \) values which correspond to the maximum among \( N_{2} \) or \( N_{3} \) values.

\[
QS = \max_{N_{1,p}} [ N_{2,p} + N_{3,p} ] \tag{2.2.1.4}
\]

Therefore, if the true QS value is 12 the following expression has the higher probability to be satisfied:

\[
N_{2,12} + ( N_{2,12} + N_{2,24} + N_{2,48} ) > N_{2,6} + ( N_{2,6} + N_{2,12} + N_{2,24} + N_{2,48} )
\]

that is, \( N_{2,12} > (N_{2,6} \cdot 2) \);

\[
N_{2,12} + ( N_{2,12} + N_{2,24} + N_{2,48} ) > N_{2,24} + ( N_{2,24} + N_{2,48} )
\]

that is, \( 2 \cdot (N_{2,12}) > N_{2,24} \);

\[
N_{2,12} + ( N_{2,12} + N_{2,24} + N_{2,48} ) > N_{2,48} + ( N_{2,48} )
\]
2.2 The implemented deblocking filter – QS control scheme

That is, \(2 \cdot (N2_{12}) + N2_{24} > N2_{48}\).

2.2.2 Adaptive deblocking filter

The estimated QS value obtained by the estimation algorithm is used as the control of the adaptive deblocking filter. In this section will be analyzed the behavior of such a deblocking algorithm making the use of some detailed Matlab code which will help its explanation.

The algorithm is composed of two different type of filtering: one is called default filter mode and the other is called DC_offset mode. In the filter operations the two modes are used alternatively depending on the pixel conditions around the boundaries.

The procedure shown in Figure 2.2.2.1 is used to locate very smooth regions in which the blocking artifacts are due to small DC offset and assign them a DC offset mode filtering operation. In other cases the procedure assigns a default filtering operation. The Matlab code shown in figure analyzes ten pixels placed on the boundary of the 8x8 pixels block (dbrow in the code) and discriminates which type of filtering must be applied. The vector “eq_cnt” is formed by the absolute differences computed between a pixel and its neighbor for all the elements of the vector “dbrow”. If the sum of all the so obtained differences less than threshold LG_THR1 is lower than threshold LG_THR2, the default mode filtering is then applied, otherwise, the DC_offset mode filtering is applied.

```
LG_THR1 = 2;
LG_THR2 = 6;
dbrow = [v0,v1,v2,v3,v4,v5,v6,v7,v8,v9];
eq_cnt = abs(dbrow(1:end-1)-dbrow(2:end));
if(sum(eq_cnt<LG_THR1)< LG_THR2)
    %%default mode
    d = defaultdb(dbrow,QP)
v4 = v4 - d;
v5 = v5 + d;
else
    %%DC_offset mode
    dbrow = Dcmodedb(dbrow,QP)
end
```

![Figure 2.2.2.1: Procedure used to select the filtering mode to apply on block boundaries.](image-url)
The goal of the filtering modes consists in updating the values of the pixels in the vectors dbrow with the newer values evaluated performing the filtering.

Figure 2.2.2.3 shows the Matlab code relative to the default filtering function. In the default filtering, a signal adaptive smoothing scheme is applied by differentiating image details at the block discontinuities using the information of the neighbor pixel array, that is, dbrow in the shown Matlab code. Therefore, the default filtering function accepts the vector dbrow and the quantization parameter QP as control inputs and computes the value “d” as output. The “d” value will be used to update only the pixels v4 and v5 shown in Figure 2.2.2.1.

function [r] = DCmodedb(dbrow, QP)
maxv = max(dbrow(2:end-1));
minv = min(dbrow(2:end-1));
r = dbrow(2:end-1);
if(maxv-minv < floor(QP/2))
    longrow = zeros(1,16);
    longrow(5:12) = dbrow(2:end-1);
    if (abs(dbrow(1) - dbrow(2))< max(floor(QP/2),3))
        longrow(1:4)= dbrow(1);
    else
        longrow(1:4)= dbrow(2);
    end
    if (abs(dbrow(end) - dbrow(end-1))< max(floor(QP/2),3))
        longrow(13:16)= dbrow(end);
    else
        longrow(13:16)= dbrow(end-1);
    end
    filter = [1 1 2 2 4 2 2 1 1]/16;
    r(1) = floor(sum(longrow(1:9).*filter));
    r(2) = floor(sum(longrow(2:10).*filter));
    r(3) = floor(sum(longrow(3:11).*filter));
    r(4) = floor(sum(longrow(4:12).*filter));
    r(5) = floor(sum(longrow(5:13).*filter));
    r(6) = floor(sum(longrow(6:14).*filter));
    r(7) = floor(sum(longrow(7:15).*filter));
    r(8) = floor(sum(longrow(8:16).*filter));
else
    d = defaultdb(dbrow, QP);
    r(4) = r(4)-d;
    r(5) = r(5)+d;
end

Figure 2.2.2.2: Matlab code of the DC_offset filtering mode.

In very smooth regions, the default filtering is not good enough to reduce the blocking artifacts due to DC offset. In this case, the stronger DC_offset filtering is applied replacing vector dbrow with a new updated vector. If the condition in the “if statement”, labeled with the letter “I”, is not satisfied,
then the default mode is applied instead the DC_offset mode, even if it is the selected mode by the procedure in Figure 2.2.2.1.

```matlab
function [d] = defaultdb(dbrow, QP)
a30= (dbrow(6)-dbrow(5))*5 + (dbrow(4)-dbrow(7))*2;
d=0;
if (abs(a30) < min(4*QP,80))
a31=( dbrow(4)-dbrow(3) )*5 + (dbrow(2)-dbrow(5))*2;
a32=( dbrow(8)-dbrow(7) )*5 + (dbrow(6)-dbrow(9))*2;
a30=abs(a30)-min(abs(a31),abs(a32));
end
A30 = A30*5;
A30 = floor((A30 + 32)/64);
d = dbrow(5)-dbrow(6);
if(d>=0)
   d = floor(d/2);
   if A30>=d
      d = d;
   else
      d = max(A30,d);
   end
else
   d = floor(d/2);
end
end
```

Figure 2.2.2.3: Matlab code of the default filtering mode.

In the Matlab code shown in Figure 2.2.2.2 and in Figure 2.2.2.3, the if statements are labeled with some upper case letters. In Chapter 3, the filter design will be analyzed making some references to the so labeled if statement.

The complete filtering operation is applied for all block boundaries first filtering the vertical edges and then the horizontal edges. In Chapter 3 will be clarify the reasons behind the chosen order of application.
2.2.3 Filtering behavior tests

In the present section, the effectiveness of the filtering algorithm has been considered. Therefore, some results, obtained performing a Matlab analysis, will be presented.

The pictures shown in the present section are the result of some Matlab simulations related to the entire Post-Processing Filter (PPF) implemented in the present work.

Figure 2.2.3.1: SOCCER2M testbed.
2.1 The implemented deblocking filter – Filtering behavior tests

As will be described in Chapter 3, the PPF (also called PPU in the Design chapter) is composed of a DCT stage, a QS estimation stage, a Deblocking filter stage. The used Matlab models, relative to the three stages, are all in fixed point. This means that the Matlab behavior is the behavior which the implemented design attempts to reproduce in hardware.

Figure 2.2.3.2: ICE1M testbed.

Figure 2.2.3.1 and Figure 2.2.3.2 shows the two luminance picture used like testbed for the deblocking filter. It is possible to observe that the details of the images are preserved and a certain amount of blocking has been filtered. Indeed, the algorithm demonstrates to be effective. Its features are
also analyzed using the PSNR (Peak Signal to Noise Ratio) using Formula 2.2.3.1:

\[
\text{PSNR} = 10 \log_{10} \left( \frac{255^2 \cdot N_1 \cdot N_2}{\text{mse}^2} \right)
\]

\[
\text{mse} = \sqrt{\frac{\sum \sum (\text{or}(i,j) - \text{pr}(i,j))^2}{N_1 \cdot N_2}}
\] (2.2.3.1)

where \(\text{or}(i,j)\) and \(\text{pr}(i,j)\) are the elements of the original picture and the processed picture, \(N_1\) and \(N_2\) are the image size (\(N_1=704\), \(N_2=576\)). The results are shown in Table 2.2.3.1.

<table>
<thead>
<tr>
<th></th>
<th>PSNR1</th>
<th>PSNR2</th>
<th>PSNR3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICE1M</td>
<td>36,320</td>
<td>36,493</td>
<td>36,663</td>
</tr>
<tr>
<td>SOCCER2M</td>
<td>31,372</td>
<td>31,488</td>
<td>31,520</td>
</tr>
</tbody>
</table>

Table 2.2.3.1: Evaluated PSNR for the testbed pictures.

PSNR1 in table 2.2.3.1 is related to the original image and its MPEG-2 compressed image, while, the PSNR2 is related to the original image and its processed image only on the vertical edges and the PSNR3 is related to the original image and its filtered image with both vertical and horizontal edges filtered.
CHAPTER 3

Design

The previous chapter presented an overview on the common post-processing technique used to aim the image quality of a decoded video. Furthermore, it has been presented the algorithmic behavior of the implemented adaptive deblocking algorithm.

This chapter will explain in which manner the algorithm has been translated into the hardware. In the first section the main architecture of the Post-Processing Unit (PPU) will be presented starting from a high-level overview and going down into a more detailed hardware level discussion. The following sections will show the features and details inside every single diagram forming the PPU reaching an exhaustive analysis. Every section includes also the error analysis of the relative designed architecture.

3.1 PPU architecture overview

This Section will present a Top-Level overview of the Post-Processing Filter implemented in hardware. In Figure 3.1.1, it is possible to have a basic idea about the main tasks involved by the blocks forming the PPU architecture. The presented block diagram shows a PPU composed of four main blocks: a Memory Management Unit (MMU), a two-dimensional Discrete Cosine Transform engine (DCT), a MPEG Quantization parameter estimation block (QS) and a Deblocking Filter engine (DF). The Post-Processing Filter is interfaced to a standard MPEG decoded data stream by the use of a memory management unit (MMU) which has the task to control all the read and write accesses to each memory RAM block.
As arrows indicate, there is only one unit, among the four, which is also able to write the memory banks (excluding the external data input), that is, the deblocking filter. Therefore, following the arrows directions, the input MPEG data stream is acquired in the architecture by continuously storing its data values in the on-device memory. Once a proper amount of data is acquired, the MMU starts to pass the pixels values to the DCT block in order to perform the two-dimensional discrete cosine transform. Then, the DCT output results (8x8 matrix coefficients) are passed to the QS estimation block where the MPEG quantization parameter will be estimated for each MB of the frames. The so estimated QS values represent the control for the subsequent step of processing, that is, the deblocking filtering operation. Therefore, the filter block loads a certain amount of pixels values from the previous stored data in the memory and operates a QS controlled adaptive deblocking filtering on the edges of each 8x8 pixels block of the frames.

The following sections will describe in detail the unit’s ways to operate.

### 3.2 2D Discrete Cosine Transform engine

In the previous chapter has been introduced and explained the behavior of the implemented post-processing algorithm for filtering the pictures of the MPEG stream decoded video. As already said in that section and also shown
in Figure 3.1.1, the data, which has been previously read from the memory banks, enters into the two-dimensional Discrete Cosine Transform engine block. Therefore, in the following pages, the most commonly used DCT algorithms will be discussed. Furthermore, the block diagram architecture of the chosen and implemented method will be discussed and then the results of an error analysis on the relative fixed point (FXP) model will be presented.

3.2.1 Classes and comparisons of fast DCT algorithms

There are a wide range of orthogonal and reversible transformations which are suitable for compression and coding purposes. Although, the best, for the images, is the Karhunen-Loewe discrete transformation [4], the most popular, and also MPEG standardize, is the DCT in Equation (3.2.1.1).

\[
Z(i,j) = \frac{K(i)K(j)}{2} \sum_{x=0}^{7} \sum_{y=0}^{7} z(x,y) \cos\left(\frac{(2x+1)i\pi}{16}\right) \cos\left(\frac{(2y+1)j\pi}{16}\right)
\]

\[K(i),K(j) = \begin{cases} 
1/\sqrt{2} & \text{if } i,j = 0 \\
1 & \text{if } i,j > 0
\end{cases}
\]

(3.2.1.1)

For the purpose of the compression and the coding, Formula (3.2.1.1) is applied to the 8x8 blocks of pixels, which forms the frame picture in a sequence, and produces 8x8 blocks of coefficients. The discrete cosine transformation is a method to decompose a matrix (8x8, for example), of an image, into a weighted sum of spatial frequencies (like in the Discrete Fourier Transformation but with different basic function: the cosine function). Therefore, the resulting coefficients represent the weight of the frequencies in an image.

In an 8x8 DCT coefficients block, the values which occupy the upper-left positions of the 8x8 matrix are referred to the low frequencies in the image, while, the values occupy the bottom-right positions are referred to high-frequencies in the image. Once the DCT transformation has been performed, many of the obtained coefficients, relative to the high frequencies, have a small absolute value. However, the other coefficients have a greater absolute value, specially, the upper-left one which is referred to the DC frequency component. Such a feature, which characterizes the energy transformation like the DCT, allows the compression of data. Therefore, the MPEG coder
applies a quantization step on those coefficients to compact the information inside the image. The quantization step is performed according to the perception of edge and color of the human eye. The visible graphics artifacts on the image (like the blocking artifact, for example), are then caused by the quantization step.

Nowadays, the DCT operation [5] has became the standard energy transformation which has been proposed not only in the 1D and 2D cases but also in the 3D case. Formula (3.2.1.1) demonstrates that the DCT is a computationally intensive problem in many different signal processing applications. Therefore, numerous algorithms have been developed for its fast computation.

It is possible to divide 1D DCT algorithms into three distinct groups [6]:

- **DIRECT**: algorithms based on the factorization of the DCT matrix [7], [5], [8];
- **INDIRECT**: algorithms which use the DFT, permutation [9] and auxiliary operation [10].
- **OPTIMAL**: algorithms (in term of number of multiplication) [11], [12]. The optimal algorithms are either indirect (N odd) or direct (N power of 2) and rely on results for the complexity of the DFT and of polynomial multiplications.

In the 2D DCT, the INDIRECT algorithms are usually used, that is, permutations followed by 2D DFT and some output rotation [13], [14], [15], [16]. However, the best performances are achieved when the true 2D FFT algorithms are used for the DFT parts (like vector-radix or polynomial algorithms)[17], [18].

Table 3.2.1.1 shows an interesting comparison between different common algorithms suitable to perform not only the mono-dimensional DCT but also the two-dimensional DCT.

Some other researchers, indeed, have developed methods for computing only the 2D DCT concentrating the effort to obtain fast performances starting from any well known 1D DCT. Table 3.2.1.2 shows a little comparison between them. In both the tables, the features of the fast methods are evaluated for a vector of 8 elements, in the case of 1D DCT, or on a matrix of 8x8 elements in the case of 2D DCT.

Looking at the performances in term of multiplications and additions, it is possible to analyze the behavior of the different fast DCT algorithms. For the mono-dimensional case, the best results are obtained with the method proposed by Loeffler, Litgengerb and Moschytz [23] in its fast version. For the two-dimensional case, indeed, the best method is proposed by Feig and Winograd [18].
3.2 2D DCT engine – classes and comparisons of fast DCT algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Year</th>
<th>Mult 1D*</th>
<th>Add/Sub 1D*</th>
<th>Mult 2D **</th>
<th>Add/Sub 2D**</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference 1D-DCT</td>
<td>1974</td>
<td>64</td>
<td>64</td>
<td>1024</td>
<td>1024</td>
</tr>
<tr>
<td>Ahmed, Natarjan, Rao [19]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chen, Smith, Fralick (Fast)</td>
<td>1977</td>
<td>13</td>
<td>29</td>
<td>176</td>
<td>464</td>
</tr>
<tr>
<td>Arai, Agui, Nakajima [22]</td>
<td>1988</td>
<td>13</td>
<td>29</td>
<td>144</td>
<td>464</td>
</tr>
<tr>
<td>Loeffler, Ligtengerb, Moschytz [23]</td>
<td>1988</td>
<td>12</td>
<td>32</td>
<td>224</td>
<td>416</td>
</tr>
<tr>
<td>Loeffler, Ligtengerb, Moschytz Fast</td>
<td>1989</td>
<td>11</td>
<td>29</td>
<td>176</td>
<td>464</td>
</tr>
</tbody>
</table>

*Operation on 8 elements vector (1D DCT)

Table 3.2.1.1: Fast 1D and 2D-DCT comparison.

The Feig and Winograd’s algorithm use a sophisticated mathematical approach (polynomial algorithm) which could be not proper suitable for a hardware implementation. Considering all the aspects behind the problem of choosing and implementing one of the fast 2D DCT algorithms, the method performances and also its “implementability” have been evaluated. Therefore, the “winner” of this “special challenge” has been the algorithm proposed by Lee [5] for the 1D DCT used to extend such mono-dimensional performances to the 2D case in the research made by Huang and Wu [25].
**Operations on 8x8 matrix (2D DCT)**

Table 3.2.1.2: Fast 2D-DCT comparison.

Table 3.2.1.3 shows a comparison between the chosen and implemented DCT method with the reference 2D DCT approach. From the table is it possible to understand the performances of the chosen method: it involves only 96 multiplications and 466 among additions and subtractions. The canonical 2D DCT performed by the row-column approach, indeed, uses at least 256 multiplications even if it takes a less quantitative of additions and subtractions.

**Operations on 8x8 matrix (2D DCT)**

Table 3.2.1.3: Comparison on the chosen and implemented 2D-DCT algorithm.

The row-column approach implies to perform a 1D DCT on the rows of the 8x8 matrix and then a 1D DCT on the column of the so obtained 8x8 resulting matrix coefficients in order to complete a 2D DCT. Indeed, the chosen
3.2 2D DCT engine – classes and comparisons of fast DCT algorithms 29

method proposes to perform the 2D DCT computing the 1D DCT on the rows of the 8x8 matrix’s elements after a proper permutation of the input elements and passing the so obtained intermediate results to a stage of only adders blocks. Therefore, Figure 3.2.1.1 shows the architecture of the 1D DCT engine [5] used to perform a 2D DCT.

According with Lee’s research, the block diagram shows 12 multipliers (the triangles in the figure) and 29 adders/subtractors (the square in the figure). A maximum parallelization for the proposed 2D DCT method assumes to use 8 1D DCT engines to perform the mono-dimensional DCT on all matrix’s rows at the same time and then to pass the so obtained intermediate results to the adders stage to perform the correct 2D DCT. Figure 3.2.1.2 shows the adders stage proposed by Huang and Wu. Following this way to highly improve the parallelism, the physical implementation of such a proposed method would require 96 multipliers and 466 adders/subtractors to perform the 2D DCT in a clock cycle. Considering the large quantitative of FPGA area resources occupy by a so implemented architecture, so much parallelism could be, also, not really necessary for the purpose of real-time filtering. To verify what has been just asserted, some investigations have been performed. It has been studied, for example, if the adopted method had been able to guarantee the proper performances with a reduced parallelism.
Figure 3.2.1.2: Adders stage FXP model architecture.
Therefore, Figure 3.2.1.3 shows the implemented 2D DCT architecture which represents a compromise between occupied area resources and speed. In the block diagram of Figure 3.2.1.3 are shown: a 1D DCT engine, an adders stage, a controller (called CTRL) and two register (one of those is a shift register). In the leftmost part of the diagram, the input image’s pixels, which take part in the 8x8 matrix block, are passed to the 1D DCT engine where is performed the mono-dimensional DCT according with Lee’ scheme shown in Figure 3.2.1.2. The intermediates results so obtained are then stored into the 8→64 shift register. Since every clock cycle 8 of those pixels are passed to the architecture, in 8 clock cycles all the 64 necessary intermediates coefficients will fill the shift register’s locations and will be ready to pass into the adders stage to perform the complete 2D DCT.

Therefore, after 8 clock cycles, the 64 DCT coefficients are ready to be used in the following process.

The following estimation will suggest an idea concerning the required performances for a 2D DCT engine to be suitable for both SDTV and HDTV. To perform the real-time 2D DCT for the SDTV (30 frame/sec) is necessary to compute the 2D DCT on a single block in less then 5.14μs. For a hypothetical HDTV implementation the requirement is 5 times more restrictive (1.02μs).
In the chapters relative to the work’s results, it will be shown that the chosen design is well suitable to perform the 2D DCT also for the real time HD applications.

Once the method has been carefully chosen, before to start to develop the VHDL implementation of the 2D DCT, an error analysis has been executed. Therefore, to accomplish such an object two different model under MATLAB environment have been realized: one in FLP and the other in FXP. To design the FXP model has been used the Simulink tool (one of the tool furnished whit MATLAB). Figure 3.2.1.1 and Figure 3.2.1.2 are the FXP model designed by Simulink.

The FXP model has required a quite large design effort to be correctly designed. The final result has been a model which has the feature to be completely adjustable into any arithmetic operations block it has.

To show the evaluated error results of the chosen method’s performance, it has been adopted to compute the 2D DCT in the progressive (P) and in the interlaced (I) mode for four different luminance matrices of four different images (that is, the testbed) for both FLP and FXP model. Once all the 2D DCT has been computed, the absolute maximum error and its mean value have been performed. Therefore, Figure 3.2.1.5 and Figure 3.2.1.6 shows the error analysis results. The simulation are performed whit an output precision of 26 BITs which grantees to have a suitable error to execute the following arithmetical operations.
All the displayed diagrams are referred to the four images chosen like testbed and already shown in the previous chapter: city, crew, ice, soccer. In Figure 3.2.1.5 it is possible to analyze the maximum computed error. Therefore, the absolute error between the 2D DCT FLP matrix and the 2D DCT FXP matrix (both of 704x576 pixels) has been performed. After that, the maximum error value in every 8x8 blocks and, then, the obtained maximum absolute values error has been histogram evaluated to form the shown blue diagrams.

![Histograms showing max absolute error values](image)

**Figure 3.2.1.5** Max absolute error values: the maximum absolute error has been computed for all the 8x8 blocks of the 2D DCT resulting matrix. The histogram is then evaluated.

Therefore, the diagrams shows that the behavior of the 2D DCT engine is always the same in all the four different circumstances in which has been analyzed (that is, the FXP model is not image dependent). Besides, the maximum absolute error never crosses the upper limit of 0.215 and has a mean value which is around 0.07.
The red diagrams in Figure 3.2.1.6, instead, shows the histograms of the mean error values computed for each 8x8 blocks in the absolute error matrix previously evaluated. The behavior of the error is always the same in the four performed test.

![Mean error values histograms](image)

**Figure 3.2.1.6** Mean error values: the mean error has been computed for all the 8x8 blocks of the 2D DCT resulting matrix. The histogram is then evaluated.

The mean error values computed never crosses 0.017 and the mean error value computed on all the matrix absolute error elements is around 0.01.
3.3 MPEG Quantization Parameter estimation engine

The QS estimation engine represents the most innovative part of the entire PPU architecture. The algorithm developed by Forchhammer and Li [3] has been implemented in VHDL. The algorithm’s objective concerns the estimation of the macroblock QS values used by the MPEG coder during the compression steps and, therefore, not included with the stream data output (that is the pixels values). Based on the DCT coefficients computed after the MPEG decoding steps, as already asserted, the estimated QS parameters represent the control for the filter to be applied.

Figure 3.3.1 shows a high level overview of the QS estimation architecture. There are five different blocks which form the engine: a multipliers stage, a non-zero DCT coefficients counter and three histogram estimators.

From the above blocks diagram, it is possible to analyze some details: the units work starting from the DCT coefficients previously computed (which are 26 BIT word-length), computes a 6 BITs QS estimation according to the algorithm’s behavior. The DCT coefficients, indeed, behaves like a ‘control’ for the multipliers stage and, also, for the non-zero counter.

The following section will present each of those blocks.
3.3.1 Multipliers Stage

The multipliers stage implements Equation 3.3.1.1. As already asserted in the previous chapter, the object of such a mathematical expression is to compute the \( QI(u, v) \cdot QS \) products. By a careful examination of the formula, it is possible to extract some important details. The most significant feature is, without doubts, the presence of a division which, as every designer knows, is the slowest and also “area hungry” among the four arithmetic operations.

\[
\text{round} \left[ \frac{\text{abs} \left( 8 \cdot \text{DCTs} \left( u, v \right) \right)}{\text{QM} \left( u, v \right)} \right] \cdot 2
\]  

(3.3.1.1)

Furthermore, there are some arithmetic shifts (related to multiplications by a power of 2), an abs operand which estimate the absolute value and, lastly, a round operation which compute the round to the nearest integer value.

Once a \((u, v)\) pair is chosen, the so selected DCT coefficient must be divided by a constant coefficient coming from the Quantization Matrix (QM). Since the QM elements are all constant quantities, indeed, divisions are not necessarily required. Such a constant division could easily become a constant multiplication by the reciprocal of the dividend. There are at least two reasons for this design method to be chosen: the first is, as already said, the divider’s slowness (and area hungry) and the second is related to the availability of 48 full-custom dedicated multipliers (DSPs) in the Virtex 4 target which are very fast compared with any other divider designed to use Configurable Logic Blocks (CLBs). Furthermore, a DSP does not involve any CLBs in the FPGA because it is placed on a dedicated area on the Virtex4 core.

Thinking to the different design possibilities to choose for a suitable implementation of the formula, it is important to keep in mind some different aspects of the design. It is basic to understand the hardware available resources (which represent the spatial constraints) and the clock available cycles (which represent the temporal constraints).

By this point of view, the hardware resources are the limited amount of slices and full custom multipliers. As already presented in the device overview’s section, the Virtex4 target available slices are 10,752, while, the available dedicated fast multipliers are 48. Once the two-dimensional DCT engine is been implemented the resources decrease around 56% and the multipliers to 36.
Considering that to perform a two-dimensional DCT on an 8x8 matrix block the DCT engine involves 8 clock cycles, the subsequent computation to perform the $QI(u,v) \cdot QS$ product must be performed within this “time budget”. Figure 3.3.1.1 shows a top view of the architecture implementing the formula (3.3.1.1) in which it is possible to recognize some fundamentals blocks: the multipliers (labeled by a $\times$), some memory elements (the registers REG), some counters, a control (labeled CTRL), a cycling controlled multiplexer, a demultiplexer, a memory (labeled ROM) and last but not least some arithmetic blocks contained in an “oval rectangle”.

It is possible, in principle, to design a high parallel architecture to perform the results of the formula, but, bearing in mind the spatial and temporal constraints just mentioned, such an approach in solving the design problem would be area expensive and also impracticable because of the fixed resources of slices and dedicated multipliers, as well. Therefore, it has been chosen to take full advantage by the “use” of the available time budget, that is, the way of the maximum serialization has been preferred to design the ar-

![QS estimation architecture – the Multipliers Stage (II).](image-url)
Architecture. For that reason, the engine involves a reuse of only 8 dedicated multipliers per 8 clock cycles to perform all the 64 necessary multiplication and, in fact, thank to the time to wait for the new DCT coefficients it has been possible to develop an architecture which is not only area cheap but is also fast as required for the subsequent computation end, hence, it matches the real-time filtering requirements.

Starting from the left side of Figure 3.3.2.1, the 64 DCT coefficients are stored into a register (64→64) which is activated by an enable signal, generated by the controller, every 8 clock cycles. The controller is only a counter which has been designed and carefully synchronized to signal the eighth clock cycle. The 8→1 multiplexer allows to pass at the following multipliers 8 DCT coefficients per clock cycle thanks to an automatically selection made by the synchronized 3 bit counter. The others factors at the input of the multipliers are selected from a ROM where there are stored 64 factors grouped by eight; at its output is placed a block able to route the eight factors to the respective multiplier (the multiplexer shown in the figure). The ROM is addressed by the same counter used for the multiplexer. Immediately after the multipliers, the remaining arithmetic operations are performed. It has been possible to split the architecture following an arithmetic scheme according to the operand’s priority, in the formula, which wants to start with the division (that is, a multiplication) of the DCT coefficients by its respectively constant factors, following with the left shift of 3 bits, the absolute value, the rounding operation and, again, with another left shift of 1 bit, this time. Regarding this last consideration, Figure 3.3.1.2 shows an enlargement of the oval rectangle in which it is possible to figure out what append to the data word-length. Once the arithmetic steps have been performed, the results are stored into two register which have the tasks to “put in a column” all the computed 64 results (8 by 8 per clock cycle into the shift register) and to store the so computed 64 \( QI(u, v) \cdot QS \) products for 8 clock cycles to allow the subsequent processing.

Once all the necessary arithmetic operations have been carefully analyzed, the attentions has to be focused on the input range of values to be sure about the right numbers of bits required to represent the data information during the processing. The inputs, of the architecture block which will perform the formula, are, evidently, the DCT coefficients coming from the previous DCT 2D engine. These values are formed by 26 bits and, as already said, 11 of those are dedicated to represent the fractional part. At this point, a question is pretty needed: are those 11 fractional bits really necessary to have the right error precision to accomplish the QS estimation object? Replying to the previous question, in a satisfactory way, some MATLAB simulations have been performed. The simulations have involved the analysis of the behavior of the QS estimation algorithm with different truncation (round toward minus infinity) on the bits forming the fractional part of DCT coefficients. The obtained results are shown in Table 3.3.1.1 and in Table 3.3.1.2.
Before analyzing the error table results, Figure 3.3.1.2 has to be carefully explained. The block diagram shows the chosen word-length precision which has been implemented in VHDL. Starting from the leftmost side of the diagram, the DCT coefficients are subject to a two’s complement truncation (round toward minus infinity) which reduces the coefficient’s word-length from 26 BITs to 18 BITs (most significant). Entering in the multiplier, the just truncated value is multiplied by a constant value which is 11 BITs word-length, but, since that constant is 3 digits smaller than 1.0 (indeed, all the constants), it is preferred to implement a 18x8 BITs multiplication leaving the right shift of 3 BITs implied (in this way, only 8 BITs are stored in the ROM memory instead of 11 BITs in which the 3 MSB are all zero). The multiplied output so obtained, then, is truncated in the same manner just mentioned above taking only the necessary 16 MSB (Most Significant Bit). At this point, a shift by 3 BITs is computed (multiplication by 8 decimal value) and the shifted value enters in the ABS block in which the absolute values is performed.

\[ \text{DCT}(u,v) \times \text{QI}(u,v) = \text{QI}(u,v) \times \text{QS} \]

\[ \text{DCT}'(u,v) \times \text{QI}'(u,v) = \text{QI}'(u,v) \times \text{QM}'(u,v) \]

\[ \text{SHIFT } R, 4 \text{ (IMPLICIT)} \]

\[ \text{SHIFT } L, 3 \]

Figure 3.3.1.2: Multipliers Stage (Π). Enlargement of the “oval rectangle”. The number of bits are shown and, also, the integer and the fractional parts of the word-length.
Figure 3.3.1.3 shows how the absolute value architecture has been realized. In the ABS block the 16 BITs of the input value are XOR evaluated with the input’s MSB in BIT. This means that the one’s complement is performed. The “xored” result is then passed in a 16 BITs adder which performs an addition between the xored result and the input’s MSB which is added like carry-in. The final result needs to have one BIT more to represent the exact absolute value in that case the input is the smallest negative values, that is, $-2^{N-1}$ (with N number of BITs word-length). In this way, if X is a positive value (MSB = 0) nothing append in the XOR stage and the addition is not computed. Vice versa, if X is a negative number the two’s complement necessary to perform its absolute value is obtained by performing a one’s complement by the XOR stage, at first, and, then, a two’s complement by adding the X’s MSB (which is 1, in this case).

Once the ABS block has been computed, the absolute value enters in the ROUND&CLIP block which performs a rounding to the nearest integer number and another subsequent truncation of 10 BITs. Indeed, the truncation is performed according to the idea that the $QI(u, v) \cdot QS$ products cannot be greater than 62, as already explained. Therefore, the 17 inputs BITs are upon clipped of 9 BITs, before to be rounded, to form an 8 BITs word-length.
3.3 MPEG QS estimation engine – multipliers stage

The rounding is simply performed by the addition of the X(2) BIT of the beginning 17 BITs input to the so clipped value and the last BIT is then discarded. Figure 3.3.1.4 (b) shows the rounding, clipping and left-1 shifting operated on the 17 BITs word-length at the input.

Following this design method, has been introduced an additional error. As a matter of fact, discarding the 9 MSB of the word-length it is possible to include some bigger values in the $QI(u, v) \cdot QS$ products which, otherwise, would have been discarded because of their starting value (bigger than the decimal value 62), according to the algorithm’s behavior. Fortunately, the error so introduced can be easily neglected. To confirm and also to proof what just asserted, the Matlab simulations have included the behavior of a so modified algorithm (the error results are shown in the following tables).

In the table 3.3.1.1 and in the table 3.3.1.2, as well, it has been taken into account an absolute difference between the original QS used by MPEG coder and its estimated value using the estimation algorithm. One table shows the error results for the FLP version of the algorithm, while, the other gives the results for the FXP implemented algorithm.

In table 3.3.1.1, columns show the frame pictures chosen like testbed, while, rows show the error computed. The errors displayed are grouped in four bands of different colors to mean the different weights among different errors (from green to red). The last two rows indicate the percentage of total error computed with and without the green error (which could be neglected, in principle). The numbers of error computed each time represent the repetitions of the same absolute error appear in the picture. Considering that the algorithm intends to compute an estimation, it does not evaluate the exact QS value per each MB, and, because to the nature of the MPEG coding, it will always introduce a small error.

Therefore, table 3.3.1.1, summarizes the error percentage computed using the MATLAB floating point version (FLP) of the QS estimation algorithm.
Design

Table 3.3.1.1: Number of repetitions of the absolute errors between the MPEG QS matrix and the QS estimated matrix using the FLP algorithm version.

<table>
<thead>
<tr>
<th></th>
<th>CITY</th>
<th>CREW</th>
<th>ICE</th>
<th>SOCCER</th>
</tr>
</thead>
<tbody>
<tr>
<td># ERRORS = 2</td>
<td>17</td>
<td>2</td>
<td>8</td>
<td>149</td>
</tr>
<tr>
<td># ERRORS = 4</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td># ERRORS = 6</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td># ERRORS = 8</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td># ERRORS = 10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># ERRORS = 12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># ERRORS = 14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># ERRORS = 16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># ERRORS = 18</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL ERRORS</td>
<td>18</td>
<td>2</td>
<td>29</td>
<td>15</td>
</tr>
</tbody>
</table>

With the term FLP is intended that the algorithm has been implemented in MATLAB without to take into account any hardware reference, that is, only the behavior has been carefully analyzed using the maximum allowed precision.

Considering the four pictures used like testbed and their respective QS matrix (the actual QS matrix used by the coder in the coding steps), it is possible to assert that the algorithm works well enough in general for all the tested images. The error percentage differences, which one can notice, are due to the differences in term of frequencies components which clearly characterize the different pictures. In those pictures in which there are many total errors there is, also, a larger weight of the high frequencies components: this is the case of ICE frame, for example.

Table 3.3.1.2 shows the behavior of the FXP algorithm. Comparing the error results between the two tables, it can be noticed that the FXP model’s performance, which include the above discussed hardware implementations, can be reasonably accepted, taking into account just a further small error introduced.
### 3.3 MPEG QS estimation engine – multipliers stage

Summarizing, the adjustments introduced into the algorithm’s behavior have involved:

- the truncated 11 BITs on the DCT coefficients;
- the truncated 10 BITs on the multiplier output;
- the clipped 9 BITs in the ROUND&CLIP block.

In this way, it has been possible to precede whit an implementation which has produced (in VHDL simulation environment) the same results shown in the FXP table’s error.

<table>
<thead>
<tr>
<th>Number of different absolute error for the FXP modified algorithm behavior</th>
<th>CITY</th>
<th>CREW</th>
<th>ICE</th>
<th>SOCCER</th>
</tr>
</thead>
<tbody>
<tr>
<td># ERRORS = 2</td>
<td>19</td>
<td>4</td>
<td>16</td>
<td>5</td>
</tr>
<tr>
<td># ERRORS = 4</td>
<td>1</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td># ERRORS = 6</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td># ERRORS = 8</td>
<td>1</td>
<td>9</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td># ERRORS = 10</td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td># ERRORS = 12</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td># ERRORS = 14</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># ERRORS = 16</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td># ERRORS = 18</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># ERRORS = 20</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># ERRORS = 24</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># ERRORS = 26</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># ERRORS = 28</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># ERRORS = 30</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># ERRORS = 32</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># ERRORS = 34</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># ERRORS = 36</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># ERRORS = 38</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># ERRORS = 40</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL ERRORS</td>
<td>26</td>
<td>5</td>
<td>40</td>
<td>13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ERROR PERCENTAGE</th>
<th>CITY</th>
<th>CREW</th>
<th>ICE</th>
<th>SOCCER</th>
</tr>
</thead>
<tbody>
<tr>
<td>WITHOUT #ERROR 2&amp;4</td>
<td>1,64%</td>
<td>0,32%</td>
<td>2,53%</td>
<td>0,82%</td>
</tr>
<tr>
<td>ERROR PERCENTAGE</td>
<td>0,44%</td>
<td>0,06%</td>
<td>1,45%</td>
<td>0,51%</td>
</tr>
</tbody>
</table>

Table 3.3.1.1: Number of repetitions of the absolute errors between the MPEG QS matrix and the QS estimated matrix using the FXP algorithm version.
3.3.2 N2 histogram estimator

Once the architecture that computes the $QI(u, v) \cdot QS$ products has been carefully designed, the 7 BITs word-length values must be pass through the N2 histogram estimator block. Figure 3.3.2.1 shows the implemented architecture.

The N2 estimator block’s object consists in performing a count which includes the number of repetitions of the same QS value present in a MB among the all thirty probable QS values (taken from the QS standard table). To accomplish the task, the block performs some different subsequent steps: to detect the QS value at its inputs, to sum all the repetitions of the same QS values so recognised and to store the final result for the entire MB.

Because of the Qs estimation is obtained starting from a progressive or an interlaced DCT on the four blocks of the MB, the $QI(u, v) \cdot QS$ products will be referred to those two different ways to compute the pixels transformation, as well. Therefore, every 8 clock cycles the $QI(u, v) \cdot QS$ products are related to a P-mode DCT or to an I-mode DCT alternatively (according with the DCT explanation section). The N2 estimator block must compute a result which is related to a MB in the frame, as already said. This means that to correctly evaluate the N2 histogram, the block must performs two potential valid results: one is related to those inputs which come from a P-mode DCT, and, vice versa, the other is related to those inputs which come from an I-mode DCT.

Let now consider the figure 3.3.2.1 in which the high-level architecture for the N2 estimation is shown.

Starting from the leftmost side, the $QI(u, v) \cdot QS$ products, which were stored in the register (64 → 64) of the previous block (the multipliers stage), are now connected to a 64→8 multiplexer which is controlled by a cycling 3 BITs counter. Those middle results are available for 8 clock cycles. Following the already adopted design strategy (that is, maximizing the use of all available clock cycles, whereas available), the $QI(u, v) \cdot QS$ are passed to a CODER stage 8 by 8 every clock cycle (hence, all the 64 values in the block are passed in 64 clock cycles). In the coder stage the QS values are detected. The detection is performed using a 30 BITs vector in which every BIT represents a flag for a specific revealed value. In this way, if the vector’s LSB is 1 it means that $QI \cdot QS = 4$ and, similarly, if the MSB is 1 it means that, this time, $QI \cdot QS = 62$ and so on for all the QS standard values taken into account.

It must be noted that only one BIT at time in the vector flag can be turned on (this is because there is a vector of flag for every value, as the block diagram
3.3 MPEG QS estimation engine – N2 histogram estimator

explain). In the figure 3.3.2.1, the labels X1 and X8 indicate two different vectors of flags: there are 8 of those in the implemented architecture.

Once the specific QI \cdot QS value has been detected, all the same flag BITs are column summed using a particular adder (so called column adder) which allows to save some FPGA slices. The figure 3.3.2.2 shows the designed 8 BITs column adder [26].

![Diagram of N2 histogram estimator architecture](image)

Figure 3.3.2.1: N2 histogram estimator architecture.

The presented column adder has been designed to perform an addition on only 8 BITs every clock cycle according with the serialization allowed. This means that, to perform the N2 final result, the partial computation, obtained by the column adder, has to be stored and accumulated each clock cycle with the previous results for 8 clock cycles (and to get the N2 result for a 8x8 block in a MB). Therefore, in the figure 3.3.2.4 is shown the developed accumulator which guarantees to add and to store the correct result. The block diagram shows a column adder, a 6 BITs adder, a 6 BITs register and a stage of 6 AND gates.
The architecture, presented in the block diagram, receives 8 BITs like input every clock cycle (X0…X7). Those inputs pass through the column adder just mentioned and are summed to form an intermediate result of 4 BITs word-length. In the canonic 6 BITs adder another addition is performed summing the so obtained intermediate result with another intermediate result which is the stored previous computation which comes from the 6 BITs register.

![Column Adder Diagram](image.png)

**Figure 3.3.2.2**: Column adder principles. HA=half adder; FA=full adder.

Thanks to the AND logic gates is allowed to reset the accumulation after 8 clock cycle when a different count, related to a different 8x8 matrix block, must be performed. The logic has been carefully designed to use only the minimum necessary word-length to correctly represent all the values considering that the output can’t be major than the decimal value 63 (so that only 6 BITs are pretty necessary).

Every 8 clock cycles, the results SUM_4…SUM_62 in the figure 3.3.2.1, related to a progressive/interlaced 8x8 DCT, as already said, must be stored in order to perform another subsequent addition which allow to take into account the main N2 result for the entire MB.

In this way, after 8 clock cycles, the N2_x (where x denotes one of the even number between 4 and 62 allowed values) relative to the P-mode (progressive DCT mode) of the top-left 8x8 block in the MB are computed and stored; after the subsequent 8 clock cycles the N2_x relative to I-mode (interlaced DCT mode) on even rows of the two left 8x8 blocks in the MB will
be computed and stored; then, 8 clock cycle to perform and to store the valid results which come from P-mode of the bottom-left 8x8 block in the MB; then, 8 clock cycle for the N2_x result come from I-mode for the odd rows of the two left 8x8 blocks. At this point, the first left half-part of the MB has been correctly performed, so, the same process is repeated for the second right half-part of the MB. The entire cycle involves 8 clock cycles per each of the four blocks in a MB repeated twice for both P/I-mode DCT inputs which get to obtain 8 valid MB results every $8 \cdot 4 \cdot 2 = 64$ clock cycles.

![Column adder and Accumulator architecture.](image)

Starting from the 8 valid MB results so obtained (in which 4 are related to P-mode DCT and the other four to the I-mode DCT), only one result will be passed to the following blocks.
Figure 3.3.2.4 N2_x selection architecture.

Figure 3.3.2.4, therefore, shows the architecture which allows to pass the right result to the following arithmetic block to complete the QS estimation. The block diagram of the figure 3.3.2.4 shows a shift register, a multiplexer, and three 6 BITs adder. The purpose of the scheme is, as already said, to allow the selection among the 8 different potential valid results. Once all the sums of the thirty N2 elements are performed for each of the four blocks in the MB (for both P and I mode), the multiplexer, driven by the QS_SEL(I/P) signal, operates the selection. The four selected right results are, then, passed to the adders step in order to compute the valid result for the entire MB. The QS_SEL(I/P) signal is the result given by the architecture which has the task to count how many non-zero DCT coefficients there are in the MB for both P and I DCT (Figure 3.3.2.5 will show its block diagram).

3.3.3 DCTs non-zero counter

The QS estimation algorithm must perform a choice between two potential valid results, as already asserted in the relative section. The choice, which it will perform, consist in a selection between the QS value obtained by the P-mode DCT and the QS value estimated using the I-mode DCT, both of those related to the four 8x8 blocks forming the MB. Section 3.3.2 has explained that such a selection is made using a 2→1 multiplexer in the N2 histogram estimator architecture. Until now, no word has been spent to explain in which manner the choice is arithmetically performed.

Figure 3.3.3.1 shows the architecture block diagram related to the operations performed to derive the choice. Starting from the computed DCT coefficients, the non-zero counter performs a count related to the number of DCT coefficients which are greater than a fixed threshold at the MB level. It
means that the architecture computes two different counts: one is related to the P-mode DCT coefficients and the other is related to the I-mode DCT coefficients.

As already seen in the previous architecture, using a multiplexer with a cycling selection activated by a 3 BITs counter, it is possible to pass 8 DCT coefficients every clock cycle to the subsequent evaluation blocks. In Figure 3.3.3.1, the first blocks met by the DCT coefficients are a sort of comparator and so called threshold comparator which has the task to perform a comparison between the threshold and the DCT coefficients (Figure 3.3.3.2 shows the details concerning those blocks).

The threshold comparator (illustrated by a white triangle in Figure 3.3.3.1) gives an output flag: if the flag is 1 the absolute value of the DCT coefficient is greater than the threshold, while, if the flag is 0 the DCT coefficient is smaller than the threshold. In the case that the flag is 1 the correspondent DCT coefficient is called a non-zero coefficient, otherwise, (flag is 0) the DCT coefficient is called a zero coefficient.

![Figure 3.3.3.1 DCT coefficients non-zero counter architecture.](image)

The so obtained 8 flags are then passed into the column adder which is the same entity already seen in the previous section. By the use of its integrated accumulator and waiting for 8 clock cycles, the column adder is able to per-
form a sum of all 64 flags relative to the 64 DCT coefficients in an 8x8 block of the MB. Therefore, once the 6 BITs result is valid at the column adder’s output, the shift register acquires the data (that is, every 8 clock cycles). The register is cycling controlled by a 3 BITs counter which flags the eighth clock cycle (that is, the enable signal which allows the register to acquire the data). Once all the 8 partial results have been computed and stored (reminding that four are related to the P-mode DCT coefficients and the other four to the I-mode DCT coefficients), the so obtained values are summed to obtain the two different non-zero DCT coefficients count: # nz_MB P-mode and # nz_MB I-mode (that is, the MB count for the P and for the I DCT mode). In Figure 3.3.3.1 is also shown a comparator which allow to compute the minimum value between the two non-zero coefficients values. The result produced by the comparator is a flag which is then stored for 64 clock cycles using a register enabled by a controller which signals the 64th clock cycle. In case the flag’s value is 0 then the QS estimated value will correspond to the P-mode DCT coefficient previously computed, otherwise, in case of 0, the Qs estimated value will be related to the I-mode DCT coefficients.

Let now consider the figure 3.3.3.2 which shows the internal logic used into the threshold comparator to signal the non-zero DCT coefficients.

![Figure 3.3.3.2 Threshold comparator.](image)

To accomplish this task, the architecture is composed of a 26 BITs adder, a stage of XOR gates and a XNOR gate. The table shown in Figure 3.3.3.2 summarizes the behavior of the block diagram. Thanks to the stage of XOR
gates, if the DCT coefficient is greater than or equal to zero, the adder will compute the addition between X and the negative one’s complement number which corresponds to the threshold value, that is, -(THR+1) which will become the right two’s complement including the carry-in into the addition performed by the adder (THR is a positive value chosen to be the 13 BITs binary representation of the decimal value 0.2 according whit the algorithm’s behavior); in case X is negative, then, the adder will compute an addition between X and THR (left positive this time). The meaning of such a behavior is explained in the fact that the architecture must compute a comparison between the threshold and the absolute value of the DCT coefficients, therefore, instead to perform the ABS operation on the 26 BITs word-length DCT coefficient, the threshold is chosen positive or negative according to the cases. Once the adder has performed the addition, the XNOR gate will correctly signal those values greater than or equal to the threshold (in absolute value, as just mentioned).

The architecture just explained allows to perform the threshold comparison using less logic than a standard approach could involve. In principle, the architecture could have been chosen following a “direct transposition” of the high-level behavior of the algorithm (reproducing the statements in the VHDL environment). However, in this way, the amount of occupied slices in the FPGA could have been twice or even more. Therefore, Figure 3.3.3.2 represents a design oriented to optimize the percentage of occupied slices.

In the section 2.1 has been presented the implemented DCT engine. In that section, has been introduced the error analysis related to the precision of the resulting DCT coefficients. The so obtained error propagates itself into the subsequent arithmetic operations. Figures 3.3.3.3 - 3.3.3.6, thus, shows the error analysis in the non-zero DCT coefficients count evaluated in the comparison between the FLP and FXP models.

Figures 3.3.3.3 (and figure 3.3.3.5, as well) shows four pictures which describes the computed absolute error between the FLP non-zero count and the FXP non-zero count. The pictures are related to the four testbed pictures already taken into account to evaluate the DCT coefficients error in the above section. Every square point represents the absolute difference between two 8x8 block partial counts of the just mentioned matrix (the FLP and the FXP non-zero counts). Using a colored scale, it has been possible to see the weight of each absolute difference (that is the weight of the error). All the pictures are 88 points width and 72 point height. In Figure 3.3.3.3 a threshold value of THR=0.2 has been chosen to perform the simulation while in the Figure 3.3.3.4 a THR=0.25 has been chosen.

Because of the too many two’s complement truncation performed in the DCT paths (starting from the 8 BITs pixels input and going down to the 26 BITs coefficients output), the so accumulated error produces a certain amount of false evaluations which manifests itself into the shown Figures.
Analyzing and comparing the Figures, it is possible to assert that the error is quite uniformly distributed and is not image dependent. The Figures 3.3.3.5 and 3.3.3.6 give the histograms of the testbed pictures.

![Images of figures showing histograms](image)

**Figure 3.3.3.3** Absolute error evaluated for the FXP non-zero counter model: the colored square points measure the error according to the colored scale shown in the right side. The threshold is THR=0.2.

Using the histograms it is possible to compare the error performed among the four images with THR=0.2 and THR=0.25. In the first case, the maximum absolute error is 2 units, while, in the second case, even if the threshold is greater, the number of absolute error increase till 5. The number of repetitions of the same absolute error increase as well, passing from the use of a THR=0.2 to a THR=0.25.

The table 3.3.3.1 shows the percentage error so estimated. It is possible to notice the differences and to assert that with a THR=0.2 the behavior of the FXP implemented algorithm is quite better.

From the error analysis explained in the section 3.3, has been observed that the absolute error produced by the FXP DCT model engine is under a certain
level about 0.23. Even if the absolute error is so small, Figures from 3.3.3.3 to 3.3.3.5 demonstrate that it is not possible to completely avoid the error in the DCT coefficients counting.

Figure 3.3.3.4 Absolute error evaluated for the FXP non-zero counter model: the colored square points measure the error according to the colored scale shown in the right side. The threshold is THR=0.25.
Figure 3.3.3.5 Histogram of the absolute error evaluated for the FXP non-zero counter model. The threshold is THR=0.2.

Figure 3.3.3.6 shows a line which allows to figure out the meaning of a so obtained result.

The FXP computed DCT coefficient and the correspondent FLP value can be across the THR value and in this case, even if the computed error is less than the maximum estimate error, there is wrong count which could be overpriced or underpriced (depending on the relative position of the FLP DCT value).
Figure 3.3.3.5 Histogram of the absolute error evaluated for the FXP non-zero counter model. The threshold is $\text{THR}=0.25$.

Figure 3.3.3.6 Meaning of the error.
Table 3.3.3.1 Error percentage for the testbed pictures.

<table>
<thead>
<tr>
<th>CITY5</th>
<th>CREWS5</th>
<th>ICE5</th>
<th>SOCCER</th>
</tr>
</thead>
<tbody>
<tr>
<td>17,12</td>
<td>15,41</td>
<td>11,34</td>
<td>17,18</td>
</tr>
<tr>
<td>29,40</td>
<td>31,96</td>
<td>23,86</td>
<td>30,74</td>
</tr>
</tbody>
</table>

3.3.4 N3 histogram estimator

Once the histogram N2 has been estimated, the potential QS values (the thirty N2_x values) must enter into the N3 histogram estimator block (in Figure 3.3.1) to perform the N3 estimation according to the behavior of the algorithm explained in the previous section. Figure 3.3.4.1 shows the N3 histogram estimator block diagram.

The N3_x resulting values represent the sum of the N2_x repetitions. Making an example, as already explained in the relative section, if three repetitions of the value Q1 \cdot QS = 12 appear in the MB, then will be three repetitions of its factors 4 and 6. At the same time, one repetitions of the value
3.3 MPEG QS estimation engine – N3 histogram estimator

Q1 · QS = 8 could appear in that MB with its factor 4 one time, as well. To derive the N3 estimation, the N3 estimator block must count all the repetitions of the same factor come from different Q1 · QS values. In this way, the N3_4 value is the result of the addition among those Q1 · QS values which can have the same factors to be counted. Therefore, the architecture in Figure 3.3.4.1 reproduces, by a dedicated digital logic, the behavior just mentioned for all the thirty potential QS values. Obviously, the shown adders in the diagram are not of the same size and, considering that there are some Q1 · QS repetitions (that is the case of the value 62, for example) which do not need be summed, only 14 of those adders are enough to perform all N3_x estimations.

3.3.5 QS estimation for the MB

The last step which brings to the QS parameter estimation for the MB, is, as already seen in the relative section, the result which is obtained performing Formula (3.3.5.1).

\[
QS = \max_{N1_p}[N2_p, N3_p]
\] (3.3.5.1)

Formula (3.3.5.1) requires the N2_p and N3_p to be added. For example, the value N2_4 must be summed with the value N3_4 and so on for all the thirty values. Once all the necessary additions are performed, the estimated QS value will be the value N1_p which correspond to the maximum among all the values obtained from the previous evaluated additions.

Since it would had been very difficult to draw a block diagram with all the thirty N2_p and the thirty N3_p values shown (the readability would had been compromised), Figure 3.3.5.1 shows the implemented architecture which reproduces the behavior of Formula (3.3.5.1) for only a restricted subset among the thirty QS potential values (eight of those). Therefore, in Figure 3.3.5.1, eight N2_p values (represented by the ‘X’) and eight N3_values (represented by the ‘Y’) are shown. Accordingly, the shown stages of comparators are inferior than the amount really implemented (in the shown block diagram there are three levels of comparators instead of five). In the bottom side of the Figure 3.3.5.1, there is a table which should help to understand the meaning of the designed block diagram.
Once values $N_2_p$ and values $N_3_p$ are evaluated, a stage of adders operates the sums on those $N_2$ and $N_3$ values has the same final number on the label (as already said, $N_2.4$ and $N_3.4$ etc…). The so computed results are then ready to cross the levels of comparators. The comparators are necessary to discriminate whether $X$ is greater than $Y$ or vice versa. Therefore, the first level of comparators is composed of a kind of block which computes the difference between its two inputs and, once the MSB of such a result has been
analyzed, it returns the maximum among its input and, also, a 1 BIT flag which signals who is the maximum for that level of comparison. At the same manner, the comparators which form the others levels have the same properties except for the number of BITs which forms the flag.

Looking the diagram in Figure 3.3.5.1, it is possible to understand in which manner the comparisons are performed. The table under the block diagram describes the operations which bring to have a flag correspondent to the QS estimated value. Starting from the X and Y values (which, respectively, correspond to N2_p and N3_p values) a first level of comparisons are then performed on X1-Y1, X2-Y2, and so on. In red upper case are shown those values which are greater than its correspondent. In this way, X1 is greater than Y1 etc.... The 2 BITs flag, then, will signal the greater using a BIT equal to 1 and the smaller using a BIT equal to 0. Therefore, if X1 is greater than Y1 the flag produced by the comparator will be “10”; otherwise, if X1 is smaller the Y1 the flag will be “01”. In the second level of comparisons, the comparators are not properly the same comparators seen for the first level. The comparators of all the levels except the first level are of the same type. They receive four inputs: two of those are 6 BITs word-length, while, the other two are the flags inputs come from the previous level and have different word-length (depending on the level in which the comparator is). The output consists in a 6 BITs value and a variable word-length value which correspond to the flag. In the second level of comparison, two results which come from the first level are compared and the resulting flag will be a 4 BITs vector. In this way, if the flags input are “10” and “10”, and if the flag which is related to the greater is the second one, the resulting second level flag will be “0010”, where the previous flag has been rewritten and also two 0 BITs have been concatenated in the superior part of the vector. Using such a method to signal the greater it is possible to arrive at the final stage with a flag which has 30 BITs word-length. A so obtained flag vector has the feature to have only 1 of those 30 BITs equal to 1. Therefore, thanks to this property, has been possible to make in correspondence the position of the BITs of the word with one of the allowed QS values starting with the value 4 from the leftmost side and arriving with the value 62 to the rightmost side. The final decoder will convert the flag vector into the relative QS estimated.

3.4 Memory Management Unit

Before to continue in the analysis of the computational blocks of the PPU architecture, that is, before to show the Deblocking Filter Unit, it has been preferred to analyze the Memory Management Unit. In this way, it should result more easy to follow the temporal consideration concerning the memory accesses performed by the Filter Unit described in the following section.
3.4.1 Details on the memory design

The memory of the PPU is composed of 32 RAM blocks in which the pixels data are stored in 720 locations of 8 BITs. Therefore, the RAM blocks is capable to contain 32 rows of pixel values: 16 of those are related to a single MBs slice of the frame picture to filter, while, the other 16 are related to a single MBs slice of the filtered frame picture (a MB is formed of 16x16 pixels). The motivations which have moved to adopt such a number of memory units are based on the idea that the PPU must perform two distinct operations which cannot be stopped. The mentioned operations are the acquisition of the data from the MPEG data stream and the process of filtering which cannot be computed on the same RAM block. Therefore, the 32 blocks are shared into 16 blocks to form two distinct groups of memory blocks. While the first group is evaluated for the deblinking filtering, in the second group there are blocks of memory either set to acquire the pixel data or set to output the filtered data. Once the cycle of acquisition and the processing have been performed, the two groups switch the parts.

The memory’s behavior just mentioned, is detailed shown in the timing diagram of Figure 3.4.1.1 in which its colored left side represents the tasks which the PPU must perform and in the horizontal axis is shown the clock cycle scale.

As it is possible to see by the timing diagram, the same operations are colored using the same color. Therefore, there are acquisition operations (labeled INPUT), operations of reading filtered data (labeled OUTPUT), DCT and QS estimation operations (labeled DCT) and filtering operations labeled (V, H1, H2, H3 filter). In the diagram, the 32 RAM blocks are divided into four 4 group of 8 RAM blocks labeled A, B, C, D. The diagram in figure shows that the filtering operation is of 4 different types. Indeed, there are only two filtering operations and what differs among H1, H2 and H3 filters is the application on different RAM blocks.

In the diagram, every little square represents a state and it involves 720 clock cycles. A cycle is composed of 30 states. Therefore, at the same time there are at least two different operations simultaneously performed on the four groups of RAM blocks. The diagram shows an entire cycle which the PPU must perform in order to complete the filtering process.
3.4 Memory Management Unit – details on the memory design

The design of the high level timing diagram has been necessary in order to realize the Finite State Machine (FSM) closely related to the diagram. The architecture of the FSM is shown in Figure 3.4.2.1.

![Figure 3.4.1.1 Memory Management Timing Diagram](image)

3.4.2 Finite State Machine architecture

In the following will briefly describe the FSM. Figure 3.4.2.1 and 3.4.2.2 shows the entire PPU in which the architecture of the MMU has been exploded to be briefly analyzed. Basically, the FSM is connected with the
RAM block elements represented by the pink diagrams and with all other processing block except the QS estimator block.

Thanks to the write enable signals (labeled WE), the FSM connect the RAM blocks with the proper address generator and data input. In fact, the pink blocks shown in Figure 3.4.2.3 are related to the interface of the RAM element with the other block elements.
Figure 3.4.2.2 Memory Management Unit architecture (2/2).

Thanks to the MUX 4→1 and MUX 2→1, the FSM uses the respectively CTRL to enable the proper data input and address input to the RAM element (EAB, in figure). The RAM blocks are always in read mode (EN to Vcc) and the WE is enabled by the connected address generator which bring with itself also the WE signal. Therefore the addresser blocks shown in Figure 3.4.2.1 are cycling address generator which are connected to the RAM blocks only in the right state shown in the timing diagram. The same append for the data.
The other signals produced by the FSM are some flag signals which are used to enable the other architecture to perform the correct operation in each different state of the timing diagram.

3.5 Debloking Filter unit

The goal of the DCT stage and the QS estimator stage is to evaluate a parameter to control the application of an adaptive filter. As already described in Section 2.1.3, the implemented filter is a debloking filter controlled by that parameter, the QS, which is performed for each MB of the picture to process.

In this section will be described the implemented filter architecture which correspond to the filtering algorithm presented in the previous chapter. The section will present the interface which allows the filter to load pixels data.
from the memory and it will show the architecture of the default filtering mode and the $DC_{offset}$ filtering mode, as well.

### 3.5.1 Filter unit interface

As already seen in the previous section, the memory is composed of 32 RAM banks (720 locations of 8 BITs each one). A single row of the frame is stored in one RAM bank. Only one location of a bank can be accessed in a clock cycle, therefore, 32 elements located in different RAM bank can be read or written (exclusively) each clock cycle. Figure 3.5.1.2 shows in which way the filter performs the memory accesses to load and update the stored pixels. In the figure, all the squares represent the pixels in the memory while the gray squares represent those pixels on which the filtering must be applied. The leftmost side of the figure shows the loaded data to perform a deblocking filtering on the vertical edges of the 8x8 block, while, the rightmost side of the figure shows the loaded data to perform a horizontal deblocking filtering. Thanks to the independent accesses among the RAM banks, it is possible, as already said, to load more than one pixel per clock cycle (one pixel per RAM bank). Making the use of such a property, it has been preferred design a filter unit interface able to apply the filter on the vertical boundaries, before on the horizontal boundaries. Therefore, a Matlab simulation has been design with the purpose to investigate the behavior of the entire deblocking filtering unit applied on vertical and then on horizontal boundaries rather on horizontal and then on vertical boundaries.

<table>
<thead>
<tr>
<th></th>
<th>PSNR1</th>
<th>PSNR2</th>
<th>PSNR3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICE1M</td>
<td>36,3201</td>
<td>36,4964</td>
<td>36,658</td>
</tr>
<tr>
<td>SOCCER2M</td>
<td>31,3722</td>
<td>31,4085</td>
<td>31,5199</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>PSNR1</th>
<th>PSNR2</th>
<th>PSNR3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICE1M</td>
<td>36,3201</td>
<td>36,4937</td>
<td>36,6628</td>
</tr>
<tr>
<td>SOCCER2M</td>
<td>31,3722</td>
<td>31,4887</td>
<td>31,5201</td>
</tr>
</tbody>
</table>

Table 3.5.1.1 Comparison between the application of the deblcoking filtering on vertical and then on horizontal boundaries and the in the opposite order.
Table 3.5.1.1 shows the resulting PSNR values relative to the testbed pictures ICE1 and SOCCER2. It is possible to notice that there are no relevant differences between the different order to perform the deblocking filtering, thus, the second order of processing (vertical & horizontal) has been preferred on the first one, aiming to reduce the memory accesses to the minimum number.

Figure 3.5.1.2 shows the filter interface architecture. Before to continue with the analysis of the memory accesses operated by the deblocking filter, it is necessary to present the implemented filter unit architecture. The filter interface is composed of: two distinct filter units, two 2D registers, two demultiplexers, a multiplexer and a simple controller (CTRL). Thanks to the use of two filter units, which will be shown in the following section, the parallelism of the entire architecture increases. Let now consider in which way the vertical deblocking filtering is performed by the filter interface architecture. In Figure 3.5.1.1, the CTRL set the MUX and the two DEMUX to operate with the 2D registers in order to enable both the two filter units to perform the processing. The vertical filtering loads 16 pixels (vertically) every clock cycle (as represented in Figure 3.5.1.2 by the gray squares). After 10 clock cycles, all the gray squares in the leftmost diagram of the figure are stored in the 2D SHIFT REG 16→20. The register acquires the data in 10 clock cycles and in 8 clock cycles shares its content between the two filter units in order to increase the parallelism. The register 2D SHIFT REG 16→16 acquires the data in 8 clock cycles (that is, the results of the filtering) and in other 8 clock cycles sends the filtered data to the memory through the DEMUX 24→16, in order to store the results of the filtering process.

In the horizontal filtering, the CTRL sets the MUX and the two DEMUX to be unconnected from the two 2D SHIFT REG. In this way, all the loaded pixels are processed only by one filter unit (FILTER UNIT 2).
Therefore, without storing any intermediate result, that is, without using any register element, a column of 10 pixels (in the right diagram of Figure 3.5.1.2) is loaded and filtered in a clock cycle and the resulting pixels are updated in the subsequent clock cycle. In this way, after 16 clock cycle the horizontal boundary is then correctly filtered.

Table 3.5.1.2 summarizes the memory accesses performed by the filter interface to operate the deblocking filtering for both vertical and horizontal edges.

Figure 3.5.1.2 and Table 3.5.1.2 are referred to each other.
In Table 3.5.1.2, the performance of the filter unit, in term of clock cycles, is compared. In case of vertical filtering, the designed filter must perform a processing on 16 rows of 10 pixels, while, in case of horizontal pixels it must perform a processing on 8 columns of 10 pixels. Such a different approach, employed to evaluate the vertical and the horizontal filtering, requires the use of two filter units to satisfy the timing constraint shown in the timing diagram of the previous section.

In Section 3.5.2 and 3.5.3 will be shown the implementation of the two filtering modality described in Section 2.2.2, that is, the default filtering and the \textit{DC\_offset} filtering. Let now consider the architecture regarding one of the filter unit shown in Figure 3.5.1.1. Figure 3.5.1.3 shows the high-level architecture of the filter unit.
In the above diagram, it is possible to recognize the architectures which implement the default filter and the DC_offset filter. Thanks to a multiplexer, the selection between the two types of filtering is performed. The control signal for the selection in the multiplexer block is evaluated by the architecture diagram shown in Figure 3.5.1.4 and commented in the following. As already seen in the deblocking filter behavior, the DC_offset block performs the DC_offset deblocking filtering but, however, in its statements it is evaluated a further selection which may allowed the default filter to be applied. Therefore, the result of the default filter block, that is the “d” value, is passed to the DC_offset filter block. Figure 3.5.1.4 shows the architecture of “filtering mode selection” which allows to perform one of the two filtering modes controlling the multiplexer in Figure 3.5.1.3. Following the Matlab code shown in the rightmost part of the figure, the architecture is composed of 9 THR block, a column adder and a comparator.
Figure 3.5.1.4 Filtering mode selection architecture.

Figure 3.5.1.5 THR architecture.
3.5 Deblocking Filter unit – filter unit interface

In the THR blocks, the entering pixels values are subtracted each other and the absolute value is then taken according with the vector eq_cnt shown in the Matlab code.

The so obtained intermediate values are then threshold evaluated, that is, those values which result lower then LG_THR1 are flagged with a BIT flag. In the comparison with the threshold LG_THR1, the use of an adder resource has been avoided employing the AND gate followed by the OR8 gate which provide to signal (with a flag BIT) if the absolute value, just computed, is greater than LG_THR1 = 2 or not. The inverter provides to signal if the value is lower than threshold.

Once the eq_cnt vector has been computed, the column adder (its architecture is shown in Figure 3.5.2.2) provides to sum all the flagged values and the comparator, placed at the adder’s output, evaluates the filtering_mode signal comparing the adder’s output with LG_THR2.

3.5.2 Default filtering architecture

In this section, some references to the figures appeared in the section 2.1.3 will be recalled, therefore, the diagrams shown in that part of the present work should be taken into account proceeding in the following. Figure 3.5.2.1 shows the high-level architecture of the default filtering mode of the deblocking filter. The shown block diagram is clearly referred to Section 2.1.3 and, in particular, to the “if statement” labeled with an “F” in Figure 2.2.2.2 (in which the Matlab code is shown) and in Figure 2.2.2.1 (in which an 8x8 block of pixels is displayed).

The pixels from v2 to v9 and the QS value are the inputs to the architecture, while, the values r4 and r5 are the output. As already seen in Section 2.1.3, the default filter updates only the values which are situated at the boundary in the block, that is, v4 and v5. In Figure 3.5.2.1, values a30, a31 and a32 are evaluated using the architecture shown in the black rectangle in the rightmost part of the figure (which is related to the Matlab code in the bottom part of the figure). It has been chosen to not perform the onerous multiplication by 5 shown in the code, while, it has been preferred to shift-left by 2 digit and then to perform an addition as displayed in the black rectangle in the figure.

Once the differentiations on the pixel values are performed, that is, computing the a3x values (x is 0,1,2), the ABS of each a3x value is evaluated making the use of the architecture shown in Figure 3.5.2.2. The absolute value architecture is the already seen ABS scheme in the previous section which is able to execute the ABS operation using only an adder.
Design

Figure 3.5.2.1  Architecture of the module F of the default filtering mode.

Figure 3.5.2.2  Absolute value architecture (ABS) for a3x computations.
3.5 Deblocking Filter unit – default filtering architecture

Thanks to the MUX and the two comparator (which compare the inputs to either determine the minimum or to signal it by the use of a flag BIT), it is possible to operate the selection to enter in the “if” process or to not enter. In case the condition of the “if statement” is satisfied, the result of the elaboration performed by the module E will be passed to the output by the MUX. The use of a MUX to perform the if statements will be presented again in the following diagrams.

In Figure 3.5.2.3 is shown the architecture of the module E. The “if statement” condition is driven by the A30’s most significant bit, using a MUX also in this time. The word-length range have been carefully analyzed to save as much adder resources as possible. Therefore, evaluating the shown Matlab code, the diagram in the figure operates some truncations on the input A30.

![Diagram of module E](image)

Figure 3.5.2.3 Architecture of the module E of the default filtering mode.

It results quite easy, also for this diagram, to understand the implemented architecture design following the shown code which is an extracted of the main default algorithm code. If the condition A30>0 is satisfied, the MSB of
A30 will be 0 and, therefore, the value \( d_D \) and \( d_C \), produced by the respectively module D and module C, are then passed to the following computations.

Figure 3.5.2.4 and Figure 3.5.2.5 shows, respectively, the module D and the module C architectures.

**Figure 3.5.2.5** Architecture of the module C (on the left) and module D (on the right) of the default filtering mode.

**Figure 3.5.2.6** Architecture of the module A (left) and module B (right).
3.5 Deblocking Filter unit – default filtering architecture

In the module D architecture, it is possible to see the same design strategy has been used for the previous module. In the module C, indeed, the one’s complement, performed by XOR evaluating the input, and the two’s complement are obtained using the MSB of \( a_{30} \) value.

If the condition for the statement of module D is true, the result of module A is passed to the output otherwise the result of module B is passed. In Figure 3.5.2.6 are shown the module A and B architectures.

3.5.3 DC_offset filtering architecture

Figure 3.5.3.1 shows the high-level architecture for the \( DC_{\text{offset}} \) filter.
As already seen in the previous architecture, the if statement, in the Matlab code shown in figure, is performed by a multiplexer. The multiplexer is driven by a selection signal evaluated by the logic in the upper right corner of the figure. The values “minv” and “maxv” are shown in Figure 3.5.3.2.

![Diagram of a multiplexer](image)

Figure 3.5.3.2  Maxv, minv architecture.

The “maxv” and the “minv” values are evaluated using a stage of simple comparators. However, in the first stage (upper part of the figure) using only one subtractor resource (instead of two) and analyzing the MSB of the resulting subtraction value, it has been possible to discriminate the maximum and the minimum value among the entering pixels values.

In Figure 3.5.3.1, the pink blocks labeled RESULT_A and RESULT_B are referred, respectively, to the default and the DC_offset filtering modes. The diagram of the RESULT_A architecture is shown in Figure 3.5.3.3, while the architecture of RESULT_B block is shown in Figure 3.5.3.4. Although the architecture shown in figure 3.5.3.3 has been very simple to design, the architecture in Figure 3.5.3.4 has taken more time to be designed. In the diagram relative to RESULT_B block, the inputs are the pixels values and the maximum value between QS/2 and 3 computed by the architecture shown in Figure 3.5.3.5. Since the value 3 is always a constant value, such a maximum operation has been performed without the use of any comparator resources but using only an OR3 gate. RESULT_B block diagram shows how to form the longrow vector, which appears in the code of the algorithm shown in Figure 2.2.2.2, and gives the filtered pixels values (from r1 to r2). In Figure 3.5.3.4, the data proceed through MODULE_H (shown in Figure...
3.5 Deblocking Filter unit – DC offset filtering architecture

3.5.3.6 and inherent to the leftmost MODULE_H of diagram RESULT_B) and MODULE_G (shown in Figure 3.5.3.7).

\[ r(4) = r(4) + d; \]
\[ r(5) = r(5) + d; \]

Figure 3.5.3.3 RESULT_A architecture.

Therefore, in MODULE_H the data is evaluated to form the first four and the last four component of longrow vector. In Figure 3.5.3.6, the ABS block is the absolute value architecture already seen in the previous sections.

Figure 3.5.3.4 RESULT_B architecture.
The multiplexer, which performs the selection, is driven by the flag which is the result of the if statement in the extracted code shown in figure.

![Diagram of multiplexer with flag and max(QS/2, 3) as control signal.](image)

**Figure 3.5.3.5** Architecture designed for the maximum between QS/2 and 3.

```c
if (abs(dbrow(end) - dbrow(end-1)) < max(floor(QP/2),3))
  longrow(13:16)= dbrow(end);
else
  longrow(13:16)= dbrow(end-1);
end
```

**Figure 3.5.3.6** MODULE_H architecture.

MODULE_G represents the “core” of the DC offset filtering, in the sense that it is the architecture which computes the operations on the entering pixels values while the other shown architectural diagrams represent the control for the filtering to be applied. Therefore, in Figure 3.5.2.7, the integer and the fractional part of the processed values are shown (and its word-length, as well). The architecture has been carefully designed to employ the minimum number of adders resources with smaller dynamic as possible.
3.5 Deblocking Filter unit – DC_offset filtering architecture

The divisions for two powers are evaluated with the SHIFT (L,R) stage. The last one adder is used to perform a rounding on its entering fractional 9 BITs value to obtain an 8 BITs number which represents the pixel value.
This chapter will present the results obtained from the synthesis of the PPU architecture in the Xilinx Virtex4 FPGA and it will discuss some important observations. In Section 4.1 will be presented some details on the FPGA used to implement the PPU. In Section 4.1 some figure about the obtained results will be presented. Section 4.2 will present the conclusive words and the further developments which could be included to the presented project.

The Chapter three has presented the block diagrams of the Post-Processing Unit architecture. All the block diagrams, from the MMU to the Filter Unit have been described in VHDL using the Active-HDL 7.2 tool provided by Aldec. Therefore, the functionality test have been carried out comparing the VHDL simulation results of each blocks with the corresponding Matlab simulations of the FXP developed models. The so obtained VHDL has been synthesized using the software provides by Xilinx, that is, the ISE 10.1 to obtain area and timing results.

4.1 FPGA Virtex4 xc4vlx25 features

The target used in to implement the designed PPU has been the Virtex4 xc4vlx25 which is the second last proposed FPGA in term of available performances among the Xilinx Virtex4 family devices. In Table 4.1.1, the device’s features are summarized.

The FPGA is composed of configurable logic resources (CLB), memory elements (EAB), digital signal processors (DSP) and digital clock manager (DCM) [27].
In Figure 4.1.1, is shown the architecture of a CLB element (on the left side of the figure) and a slice element (on the right side of the figure).

A Configurable Logic Block (CLB) is composed of 4 slice elements: two of those (the pink cubes in figure) contains only combinational logic, that is, only LUTs, while the others two (the blue cubes in figure) contain the combinational logic and also two memory element.
4.2 Evaluation of the obtained area and timing results

The slice architecture shown in the above figure is relative to those slices in which memory elements are also present. Thanks to a 4-input lookup table it is possible to implement any combinational logic and, also with the memory elements, any sequential logic.

The DSP is composed of a dedicate 18x18 BITs multipliers and an accumulator.

4.2 Evaluation of the obtained area and timing results

Using the Xilinx synthesizer, the PPU has been synthesize and the obtained area results (in term of occupied slices) are shown in the pie graph in Figure 4.2.1.

Figure 4.2.1 Evaluation of the occupied area of the device.
Looking at the Figure 3.1.1 which shows the architecture of the PPU (in Chapter 3), the pie graph of Figure 4.2.2 represents the distribution of area resources among the four blocks forming the PPU architecture. The diagram is shared into five different parts making the use of five different color gradient. In green colors is the result relative to the 2D DCT engine, in blue colors is the result relative to the QS estimator engine, in red colors is the obtained result for the Filtering Unit, in gray color is the results relative to the Memory Management Unit and in the white color is the amount of free slices (not used). The five colored parts of the pie graph are then further divided into other parts using different gradient of the same color. The goal of the pie graph is to show the percentage of slices occupied by the architecture blocks forming the post-processing implemented filter.

In a first analysis, it emerges that almost all available slices resources have been employed to implement the PPU architecture. It remains only a 1% of those available. However, further then the used slices resources, a certain amount of DSP and EAB have been employed into the synthesize process. In Table 4.1.1, the total available DSP are 48, while the presented architecture uses only 20 of those DSP and in particular 20 dedicated 18x18 multipliers. Speaking about the EAB resources, only 33 RAM banks of those 72 available have been enough to store the necessary intermediate results of the processing.

Once the occupied area of the device has been investigated, the timing analysis has been performed to evaluate the critical path of the design. The PPU has an operating frequency of around 12Mhz. This means that the pixels are acquired in the device one by the other at this frequency speed.

In the following subsections will be analyzed in details the meaning of each segment of the pie graph and the timing details as well.

### 4.2.1 2D DCT synthesis considerations

From the pie graph, it is possible to see that the device area occupied by the 2D DCT engine. The total amount corresponds to the 34%. The 2D DCT engine, as already seen in the third chapter is composed of two block architecture: the 1D DCT engine and the Adders Stage. Therefore, the greater contribution to the amount of 34% is given by the Adders Stage which employs something like 234 adders occupying the 24% of the total available slices resources. On other hand, the 1D DCT engine uses only 29 adders (of lower dynamics) and therefore it occupies only the 4% of the total amount. However, the mono-dimensional discrete cosine transform uses 12 dedicate multipliers. The remaining 6% is related to the logic designed to interface the 1D_DCT engine with the Adder Stage and the 2D_DCT engine with the oth-
4.2 Evaluation of the obtained area and timing results

Eer architectures (above all, logic used to realize register, counters, controller, etc…).

From the timing analysis performed on the 2D DCT engine has been discovered that the critical path is inside the 1D DCT engine. Their value is around 47.207ns. The obtained result is something predictable because of the multipliers used in the 1d DCT architecture (Figure 3.1.1.1).

4.2.2 QS synthesis considerations

The architecture relative to the QS estimation algorithm has been optimized in some of its low-level blocks. After those many optimizations which, for example, have involved the reuse of the same architecture like the column adders, it has been possible to obtain the 38% of occupied area. Therefore, the QS estimator results to be the most area expensive part in the project (but also the 2D DCT engine result to be quite expensive). The N2 histogram estimation block is the architecture which occupies more slices resources, that is, the 18%. Indeed, the Multiplier Stage uses less slices resources but it use 8 dedicated 18x18 multipliers and one EAB element to store the QM values.

4.2.3 Filtering Unit synthesis considerations

In the red color, scale it is possible to see the synthesis results for the deblocking filtering unit. A single Filter Unit uses the 7% of the available slices resources and, therefore, two of those uses twice the slices. In the 9% occupied area of the device (labeled “**”) is included the two 2D registers already seen in the filter unit interface. Indeed, the architecture of the greater two-dimensional shift register is composed of 16*10=160 registers of 8 BITs.

4.2.4 MMU synthesis considerations

The last one architecture, the Memory Management Unit, employs less slices of any other block. Obviously, such a result is related to the absence of any computational logic in the MMU architecture.
4.3 Conclusions and future work

4.3.1 Conclusions

The objective of this thesis was to implement a Post-Processing adaptive filter controlled by the post-estimated MPEG quantization parameter (QS). Furthermore, a Virtex4 family FPGA was considered and an implementation analysis was derived to determine the implementation capability of the filtering algorithm and, in particular, of the QS estimation algorithm proposed by Forchhammer and Li.

The work started with the design of the 2D DCT engine. To accomplish this task, an exhaustive research on the Discrete Cosine Transform techniques has been realized. A comparison on the performances of the different techniques has been carefully evaluated. The comparison evaluation allowed to choose the 2D DCT algorithm to implement. Once the algorithm was chosen, its relative FXP model has been designed in order to analyze the error due to the FXP representation of the algorithm. The design of the model required a long time (about two months), but the final result was a variable word-length FXP model (developed in the Simulink tool of Matlab). Furthermore, an exhaustive simulation on the model has been performed and some considerations have been carried out on the evaluated error analysis. Based on the performed error analysis, it has been possible to proceed in the VHDL description of the FXP model using the proper B1Ts precision. The 2D DCT hardware description has been exhaustively simulated using Active-HDL 7.2 and comparing the result with the Simulink FXP model simulation. The 2D DCT engine has been synthesized into the FPGA (Virtex 4 xc4vlx25) to evaluate the amount of occupied area resources in the device. The result has been 34% of the available slices plus 12 DSP (used for the multipliers).

Once the 2D DCT has been designed, the following step was the design of the algorithm to estimate the control parameter. The behavior of the QS estimation algorithm has been evaluated first by making the use of different Matlab simulations, then the FLP version of the algorithm has been converted in the FXP version. Because of the introduced error of the FXP model some further Matlab simulations have been performed to understand the error gap with its FLP model. Carefully analyzing the differences, simulations have shown that the algo-
4.3 Conclusion and future work – conclusions

The algorithm behavior remained substantially unchanged even in its FXP representation and, therefore, it was possible to continue with its hardware implementation. The QS estimation architecture is composed of four parts: a multipliers stage, a non-zero DCT coefficients counter, a N2 histogram estimator, a N3 histogram estimator and a final QS value estimator. A VHDL description of all the four main parts have been designed and simulated in the same way just mentioned for the 2D DCT engine. After the simulations, the QS estimation VHDL has been synthesized into the FPGA to evaluate the amount of occupied area resources. The initial result was that about 60% of the slices available in the device plus 8 DSP were used. The result was clearly out of the constrains. Therefore, some optimization on the four main parts of the architecture has been performed. A different approach was considered and some of those parts forming the main QS architecture have been carefully redesigned in new versions. The effort employed to those hardware optimizations took about less than a month, but thanks to this work it has been possible to reduce the amount of occupied area resources from 60% to 38%.

Making the use of the experience gained in the design of the previous main parts of the project, the following step was the deblocking filtering implementation. The filtering algorithm is formed by two filter types: the default mode and the \textit{DC\_offset} mode. Also in this case a Matlab simulation has been used to compare the algorithm behavior in both FLP and FXP behavior. The FXP model of the filter, which left unchanged the characteristics of the filtering algorithm, was translated into a VHDL description. The high level architecture of the deblocking filtering uses two filter units to increase the parallelism. Because of this choice, the synthesis report has shown a 23% of occupied slices. Each filter unit employs only 7% of the slices that, for two units, adds up to 14%. The rest of the slices are used to implement the 2D register used to interface the filter units with the memory.

The last design step was the development of a memory management unit (MMU) which, thanks to its FSM, is capable to synchronize all the other computational blocks. In this part of the project, it has been designed a timing diagram to correctly understand the temporal constrains required by the designed architectures in order to correctly synchronize the whole machine. By the use of a timing diagram, it has been chosen to implement a 32 RAM blocks dual architecture which means that the RAM blocks is divided into two groups, and that, while the first group is busy in the processing operations, the other is busy
in input/output operations, and vice versa. The FSM allows to cyclically switch among the two groups of RAM blocks.

Once the whole architecture has been described in VHDL, the synthesis of the entire design has been performed verifying that the entire post-processing filter is suitable to be implemented into the chosen FPGA.

### 4.3.2 Future work

Carefully analyzing the pie graph obtained after the synthesis process, it is possible to consider some ideas and observations. The 2D DCT engine employs many hardware resources due to the chosen algorithm. As already motivated, the chosen 2D DCT engine has a high parallelism to perform an 8x8 two-dimensional DCT in only 8 clock cycles. Such a high parallelism might not be strictly necessary (it depends on the application target). A different implementation of the 2D DCT engine could employ two 1D DCT engines implementing the Lee’s algorithm, the same used by the actual design [5]. The 1D DCT engines would work in series (like in the row-column approach) making the use of an interposed 2D shift register to allow the second 1D DCT unit to process the correct permuted data originated by the first engine. Proceeding in such a way, it is possible to save the slices used to implement the Adders Stage (no more needed, now). Therefore, saving a 24% of slices, and employing no more than 7-8% of slices to implement the second 1D DCT engine, which results in having a wider dynamic, it becomes possible to save about the 15% of area resources. However, the second 1D DCT engine would use 12 DSP to perform the computation but the necessary DSP resources are available. Therefore, the saved amount of slices could be employed to implement the deringing filter presented in [3], for example, which is a filter suitable to filter the ring artifacts in the pictures compressed using the MPEG coding. Furthermore, the critical path in the 1D DCT engine can be reduced by pipelining it.
References


3. K. Virk, H. Li, S. Forchhammer. "Reduced complexity MPEG2 video post-processing for HD display".


**List of abbreviations:**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCT</td>
<td>Discrete Cosine Transform</td>
</tr>
<tr>
<td>PPU</td>
<td>Post-Processing Unit</td>
</tr>
<tr>
<td>MB</td>
<td>Macroblock 16x16 pixels</td>
</tr>
<tr>
<td>MSB</td>
<td>Most Significant Bit</td>
</tr>
<tr>
<td>LSB</td>
<td>Last Significant Bit</td>
</tr>
<tr>
<td>I-mode, I</td>
<td>Interlaced mode DCT</td>
</tr>
<tr>
<td>P-mode P</td>
<td>Progressive mode DCT</td>
</tr>
<tr>
<td># nz_MB P-mode</td>
<td>Number of non-zero coefficient for the MB DCT P-mode</td>
</tr>
<tr>
<td># nz_MB I-mode</td>
<td>Number of non-zero coefficient for the MB DCT I-mode</td>
</tr>
<tr>
<td>THR</td>
<td>Threshold</td>
</tr>
<tr>
<td>FLP</td>
<td>Floating point</td>
</tr>
<tr>
<td>FXP</td>
<td>Fixed Point</td>
</tr>
<tr>
<td>N2_x, N3_x</td>
<td>X indicates the generic element of the N2/N3 estimation</td>
</tr>
<tr>
<td>PPF</td>
<td>Post-Processing Filter</td>
</tr>
</tbody>
</table>