Wavelet transform
for intra frame coding

feasibility study on DWT hardware
implementation and integration with H.263

M. Verhappen

Carried out at the Philips Research Laboratories
Eindhoven, the Netherlands

Coaches:  M.J.M. Bekooij M.Sc.  (Philips Research Laboratories)
          A.C. Verschueren Ph.D.  (Eindhoven University of Technology)
Professor: Prof. M.P.J. Stevens  (Digital Information Systems Group)
Date:     September 30, 1998

The Faculty of Electrical Engineering of the Eindhoven University of Technology does not accept any responsibility regarding the contents of traineeship reports
Public version

This is a public version of this traineeship report. The original report contains research results that may be subject to patenting. Therefore, certain parts have been left out of this version.

Abstract

This report discusses the use of the discrete wavelet transforms for video intra frame coding. In the context of future hardware implementation and integration with the H.263 low bitrate video standard, several bottlenecks are investigated, such as computational cost and memory access bandwidth. Different solutions are proposed that enable the discrete wavelet transform to compete with existing DCT intra frame coding schemes. A model is developed that combines the Two-Ten wavelet transform with a block-based H.263 run length and variable length encoder. Very acceptable results were achieved even though the combination can still be optimized. A memory size reduction method is discussed for the combination of this model with the proposed architectures.
Contents

1 Introduction 1

2 Wavelet transform 2
   2.1 Subband decomposition 2
   2.2 Wavelet filter properties 3
   2.3 Wavelet transform on pictures 3
   2.4 Wavelet transform example 4

3 Computational requirements and memory access bandwidth 9
   3.1 Operation cost calculation and comparison 9
   3.2 Conclusions 12

4 Memory access bandwidth reduction 14
   4.1 Bandwidth reduction using global memory 14
      4.1.1 Filter placement between sensor and memory 14
      4.1.2 Bandwidth reduction using pseudo-dual port memory 15
   4.2 Bandwidth reduction using local memory 17
   4.3 Conclusions 22

5 Integration of the TT-transform with H.263 24
   5.1 Block composition, scanning and quantization 25
   5.2 Telenor run length and variable length encoder 27
   5.3 CREW bypass procedure 27
      5.3.1 Bypass procedure program flow 27
      5.3.2 Command line syntax 28
      5.3.3 Verbosity 29
      5.3.4 Grayscale and color pictures 30
   5.4 Bypass procedure evaluation 31
      5.4.1 Results, Grayscale pictures 33
      5.4.2 Results, Color pictures 33
   5.5 Conclusions 34

6 Memory size reduction 36
   6.1 Storage of quantized wavelet coefficients 36
7 Conclusions and recommendations

7.1 Conclusions ........................................... 39
7.2 Recommendations and remarks .......................... 40
    7.2.1 Recommendations ................................ 40
    7.2.2 Remarks ........................................... 41

A Bypass procedure function description ................. 44
    A.1 Functions ........................................... 45
    A.2 Remarks ............................................ 49

B Results: Plots ........................................... 50
    B.1 Grayscale pictures .................................. 51
        B.1.1 The Lena 512×512 grayscale picture ........... 51
        B.1.2 The Lena QCIF grayscale picture ............... 54
        B.1.3 The Eref 512×512 grayscale picture ........... 55
        B.1.4 The Eref QCIF grayscale picture ............... 58
    B.2 Color pictures ...................................... 59
        B.2.1 The Lena 512×512 color picture ................ 59
        B.2.2 The Lena QCIF color picture .................... 62
        B.2.3 The Eref 512×512 color picture ................. 63
        B.2.4 The Eref QCIF color picture ..................... 66

C Results: Images .......................................... 67
    C.1 The Lena picture ..................................... 68
        C.1.1 The Lena 512×512 picture ....................... 68
        C.1.2 The Lena QCIF picture ............................ 69
    C.2 The Eref picture ..................................... 70
        C.2.1 The Eref 512×512 picture ....................... 70
        C.2.2 The Eref QCIF picture ............................ 71

iii
Chapter 1

Introduction

Presently, most intra frame coding schemes are implemented using the discrete cosine transform (DCT). Recently however, several publications have emerged that suggest the use of a different kind of transform: the wavelet transform. The DCT is usually performed on blocks of 8x8 pixels, whereas the discrete wavelet transform (DWT) generally operates on entire frames. The main advantage of frame coding over block coding is the prevention of disturbing block-artifacts in heavily quantized frames.

An application of DCT is video communication by the H.263 low bitrate standard [1]. Therefore, the reference frames (intra frames) of an H.263 video stream may suffer from block-artifacts. The research objective of this traineeship is the development of an efficient hardware implementation of a discrete wavelet transform and the integration of this transform with an existing H.263 codec. This report should be considered a feasibility study on these subjects.

Literature provides many different approaches to the compression of pictures using wavelet transforms. Most articles aim at increasingly better compression and better picture quality. These state-of-the-art algorithms are generally not the most efficient ones when low complexity and low-power are considered. Since a mobile video phone, for example, has these requirements, a different approach will be made.

In chapter 2, the wavelet transform will be discussed in more detail. Chapter 3 involves computational requirements and memory access bandwidth demands of a particular efficient wavelet transform and compares these to other transforms, such as the DCT. Chapter 4 provides solutions for the reduction of memory access bandwidth for this wavelet transform. Chapter 5 shows how wavelet transforms can be integrated with existing video codecs. In addition an implementation of this integration is discussed and results are presented. A suggestion for memory size reduction will be discussed in chapter 6. Finally, conclusions and recommendations will be presented in chapter 7.
Chapter 2

Wavelet transform

2.1 Subband decomposition

When the wavelet transform is used in image coding, it can be interpreted as the well-known subband decomposition scheme. In subband decomposition, the full-bandwidth source signal is first split into signals of smaller bandwidth by means of a filter bank. A two-channel analysis filter bank example is shown in figure 2.1.

![Figure 2.1: Two-channel analysis filter bank](image)

In the analysis filter bank, the input signal is split into a low-frequency signal, 'smooth signal' $s(n)$, and a high-frequency signal, 'detail signal' $d(n)$. For now, the filters are assumed to be ideal and to have the same cut-off frequency. This means that no information is lost in the filtering process. Low-pass filtering in $H_0$ can be interpreted as a moving average operation and high-pass filtering in $H_1$ as a moving difference operation. Subsampling of the outputs by a factor 2 removes induced redundancy. This also causes the number of samples per unit time to be maintained.

The output signals $s(n)$ and $d(n)$ tend to have very different statistical properties, which can be made use of by a compression system. For typical video communication for example, we know that the amplitude of high-frequency components in an image is generally quite small, although a few high values may appear. Applying a Huffman filter on separate subbands will therefore yield a better compression than applying it directly to
2.2 Wavelet filter properties

the original signal, especially after quantization of the smooth and detail signal.

Figure 2.2 shows how the original signal can be regained from the smooth and detail signal after reception of these signals.

![Diagram](image)

Figure 2.2: Two-channel synthesis filter bank

Both signals are upsampled and filtered again by ideal low-pass and high-pass filters ($G_0$ and $G_1$ respectively). Adding both filter outputs will yield the original signal.

2.2 Wavelet filter properties

Since practical filters are not ideal, filter slopes are finite and need to overlap to prevent loss of information. This overlap causes aliasing and therefore leads to errors in the reconstructed signal. It would be best to have the smallest overlap possible to minimize the aliasing and to achieve maximum coding gain. A small overlap, however, requires filters with very steep slopes. If the low-pass filter slope is too steep, a so-called 'ringing' effect will disturb the image.

Biorthogonal filter sets offer a solution for this problem. These filter sets cancel the aliasing caused by the overlap. The analysis filters of a biorthogonal filter bank are allowed to be non-symmetric and non-power-complementary. This enables the filter designer to develop a low-pass filter with a smooth characteristic and a high-pass filter with a steep slope. Above all, filter sets may be chosen that are fairly cheap to implement. For detailed information about these filter sets, see [2], chapter 4.

2.3 Wavelet transform on pictures

The one-dimensional subband decomposition shown in the previous section can be expanded to a multi-dimensional decomposition. Figure 2.3 shows a logarithmic tree of filter banks for a two-dimensional decomposition. This tree consists of three levels in which horizontal and vertical filtering are done successively. Subsampling after each filter ensures that we get exactly the same number of wavelet coefficients as the number of pixels of the original
2.4 Wavelet transform example

picture. The transform is applied recursively to the low-pass output of each level.

Figure 2.4 shows how a picture is processed through the logarithmic tree of filter banks. First the original picture is filtered along the picture rows. This horizontally transformed picture is then filtered along its columns in the second part of the first decomposition level. Every next decomposition level deals only with the low-pass components of the previous level and therefore takes roughly four times less operations than the previous level decomposition.

Subbands are named after the four outputs of each level:

- The HH-subband is a horizontally and vertically high-pass filtered picture.
- The HL-subband is a horizontally high-pass and vertically low-pass filtered picture.
- The LH-subband is a horizontally low-pass and vertically high-pass filtered picture.
- The LL-subband is a horizontally and vertically low-pass filtered picture.

The number appended to the subband is the decomposition level in which the subband was generated.

An example of different levels of the wavelet transform of a picture is shown in figure 2.5. As we go from top-left to bottom-right in the final decomposed frame, we encounter higher spatial frequencies and therefore expect an increasing number of zero-valued coefficients in typical pictures. Note that the locations of the subband coefficients are spatially correlated to the original picture, unlike discrete cosine transformed pictures. This property will be made use of in the coding step.

2.4 Wavelet transform example

An example of a wavelet transform is the Two-Ten transform (TT-Transform), which is used in CREW (Compression with Reversible Embedded Wavelets) [3]. Its name is Two-Ten transform because of the two taps in the low-pass analysis filter and ten taps in the high-pass analysis filter. The transform is a reversible wavelet from a family of wavelet transforms derived from the LeGall-Tabatabai polynomial of case 'p=3' [4]. The analysis filter bank of the TT-transform is defined as follows:
2.4 Wavelet transform example

\[ s(n) = \frac{x(2n) + x(2n + 1)}{2} \quad (2.1) \]

\[ d(n) = x(2n) - x(2n + 1) + \frac{3s(n - 2) - 22s(n - 1) + 22s(n + 1) - 3s(n + 2) + 32}{64} \quad (2.2) \]

where \( x(n) \) denotes the input signal, \( s(n) \) and \( d(n) \) denote the low-pass and high-pass filter outputs respectively. The synthesis filter bank is defined in equations 2.3, 2.4 and 2.5.

\[ x(2n) = s(n) + \left\lfloor \frac{d(n) - p(n) + 1}{2} \right\rfloor \quad (2.3) \]

\[ x(2n + 1) = s(n) - \left\lfloor \frac{d(n) - p(n)}{2} \right\rfloor \quad (2.4) \]

where

\[ p(n) = \left\lfloor \frac{3s(n - 2) - 22s(n - 1) + 22s(n + 1) - 3s(n + 2) + 32}{64} \right\rfloor \quad (2.5) \]

Because the coefficients are fixed integers, it is cheap to implement these filter banks. The fixed integer multiplications can be implemented using only shift and add operations. The divisions are convenient powers of two, so they can also be calculated with only shift operations. The \( s(n) \) of equation 2.1 and \( p(n) \) of equation 2.5 are used in the calculation of \( d(n) \) and the reconstructed signal \( x(n) \) respectively, leading to an even cheaper implementation in terms of computational cost. Because the TT-transform is such an efficient wavelet transform, it will be looked at in more detail in the remaining part of this report. The objective is to find out whether it is a good candidate for the implementation of wavelet transform in intra-frame coding.
Figure 2.3: Logarithmic tree of filter banks for a two-dimensional wavelet transform.
Figure 2.4: Three-level wavelet transform path
2.4 Wavelet transform example

Figure 2.5: Subband decomposition of a picture
Chapter 3

Computational requirements and memory access bandwidth

In this chapter a comparison is made between the TT-transform, a so-called DWT 9/7-transform and two different discrete cosine transforms. We will focus on two aspects: computational cost and memory access bandwidth. Main objective is to determine what measures need to be taken in order to make the TT-transform compete with conventional DCT transforms.

3.1 Operation cost calculation and comparison

In figure 2.4 it was shown how many read and write actions are required for each wavelet transform level. The total number of read actions and write actions for the transform is

\[
#RD = #WR = XY + XY + \frac{XY}{4} + \frac{XY}{4} + \ldots
\]

\[= \sum_{i=1}^{J} (\frac{XY}{16} + \frac{XY}{16} + \ldots) = 2XY \sum_{i=1}^{J} 4^{1-i} \tag{3.1}
\]

where \(J\) is the wavelet decomposition level, \(X\) is the original picture width and \(Y\) is the original picture height. The number of read and write actions converges to the value \(2\frac{3}{4}XY\) as the wavelet decomposition level increases.

The operation count for the TT-transform per two pixels is

- 1 Additions/subtractions for the low-pass filter. The division by 2 is performed by a shift to the right.
- 6 Additions/subtractions for the high-pass filter. The value \(x(2n + 1)\) is reused. Again, the division is performed by a shift to the right.
3.1 Operation cost calculation and comparison

- 4 Multiplications. These can be halved because we are dealing with symmetric coefficients. The multiplication $3s(n-2)$, for example, can be stored for four clock periods and be used to avoid the calculation of $-3s(n+2)$. The only operation there is a sign change. The number of multiplications will therefore be two per two pixels.

Since the coefficients of the TT-transform are fixed, they can be implemented using 4 shift operations and 3 add/subtract operations (1 operation and 2 operations for the coefficients 3 and 22 respectively). The TT-transform can therefore be implemented without the use of multipliers at the cost of substituting 3 add operations and 4 shift operations per multiplier. This results in the following two possibilities for the operation count per read pixel:

- 1 Multiplication
- 3.5 Additions/subtractions

or

- 6.5 Additions/subtractions

The absolute number of operations for a TT-transform is the total number of pixels of a color component multiplied by the factors shown above.

Tables 3.1 and 3.2 show the number of operations that are needed to transform one frame.

Besides the TT-transform, the costs of three other transforms have been added to the table for reasons of comparison. The first is a Discrete Cosine Transform algorithm proposed by Loeffler, Ligtenberg and Moschytz [5]. This DCT algorithm is one of the most efficient ones available since it requires only 11 multiplications and 29 additions per 8-pixel one-dimensional DCT transform. The assumption is made that this algorithm is implemented using array-multipliers of depth 11. In order to make a fair comparison, we assume the DCT to be implemented with fixed multipliers, i.e. they can be implemented with shift and add operations as well. A rough estimation of the number of shift and add operations per 11-bit multiplication is 5.5 shifts and 4.5 adds. In tables 3.1 and 3.2 this DCT is called DCT 11/29.

The DCT 11/29 algorithm, which is implemented with butterflies, is generally not one that is implemented in hardware. Usually, a very regular recursive structure is used instead of the DCT 11/29. The regular recursive structure requires 64 fixed multiplications of 3 or 4 shift/add operations and 64 additions per 8-pixel one-dimensional DCT. This DCT is denoted DCT 64/64.

The third transform for comparison is a DWT using a 9/7 symmetric biorthogonal filter bank with integer coefficients, as discussed by Strang and
### 3.1 Operation cost calculation and comparison

<table>
<thead>
<tr>
<th>Transform</th>
<th>Level</th>
<th>Reads</th>
<th>Writes</th>
<th>Multiplications</th>
<th>Adds/Subs</th>
</tr>
</thead>
<tbody>
<tr>
<td>TT</td>
<td>3</td>
<td>66 528</td>
<td>66 528</td>
<td>299 376 / 199 584</td>
<td>232 848</td>
</tr>
<tr>
<td>TT</td>
<td>4</td>
<td>67 320</td>
<td>67 320</td>
<td>302 940 / 201 960</td>
<td>235 620</td>
</tr>
<tr>
<td>DCT 11/29</td>
<td>-</td>
<td>25 344</td>
<td>25 344</td>
<td>383 328 / 313 632</td>
<td>183 744</td>
</tr>
<tr>
<td>DCT 64/64</td>
<td>-</td>
<td>25 344</td>
<td>25 344</td>
<td>1 622 016 / 1 216 512</td>
<td>405 504</td>
</tr>
<tr>
<td>DWT 9/7</td>
<td>3</td>
<td>66 528</td>
<td>66 528</td>
<td>665 280 / 266 112</td>
<td>665 280</td>
</tr>
<tr>
<td>DWT 9/7</td>
<td>4</td>
<td>67 320</td>
<td>67 320</td>
<td>673 200 / 269 280</td>
<td>673 200</td>
</tr>
</tbody>
</table>

Table 3.1: Number of operations per frame (one color component) for different transforms of a QCIF image (176×144).

<table>
<thead>
<tr>
<th>Transform</th>
<th>Level</th>
<th>Reads</th>
<th>Writes</th>
<th>Multiplications</th>
<th>Adds/Subs</th>
</tr>
</thead>
<tbody>
<tr>
<td>TT</td>
<td>4</td>
<td>266 112</td>
<td>266 112</td>
<td>1 197 504 / 798 336</td>
<td>931 392</td>
</tr>
<tr>
<td>TT</td>
<td>5</td>
<td>270 072</td>
<td>270 072</td>
<td>1 215 324 / 810 216</td>
<td>945 252</td>
</tr>
<tr>
<td>DCT 11/29</td>
<td>-</td>
<td>101 376</td>
<td>101 376</td>
<td>1 533 312 / 1 254 528</td>
<td>734 976</td>
</tr>
<tr>
<td>DCT 64/64</td>
<td>-</td>
<td>101 376</td>
<td>101 376</td>
<td>4 866 048 / 1 622 016</td>
<td>1 622 016</td>
</tr>
<tr>
<td>DWT 9/7</td>
<td>4</td>
<td>266 112</td>
<td>266 112</td>
<td>2 661 120 / 1 064 448</td>
<td>2 661 120</td>
</tr>
<tr>
<td>DWT 9/7</td>
<td>5</td>
<td>270 072</td>
<td>270 072</td>
<td>2 700 720 / 1 080 288</td>
<td>2 700 720</td>
</tr>
</tbody>
</table>

Table 3.2: Number of operations per frame (one color component) for different transforms on a CIF image (352×288).
3.2 Conclusions

Nguyen [2]. Again, the symmetry property is used to halve the number of multiplications. The filter coefficients are

\[
h_{9_{LPF}} = \left[1, 0, -8, 16, 46, 16, -8, 0, 1\right]/64 \quad (3.2)
\]
\[
h_{9_{HPF}} = \left[-1, 0, 9, -16, 9, 0, -1\right]/16 \quad (3.3)
\]

This means that 3 multiplications and 6 additions/subtractions are required per pixel for the low-pass filter. Only 2 multiplications and 4 additions/subtractions are needed for the high-pass filter. If we reuse the 16x(n) from the low-pass filter for the middle coefficient of the high-pass filter, we need only one multiplication \((9x(n \pm 1))\) in the high-pass filter.

The convenient construction of the \textit{TT-transform} allows the use of the low-pass filter output \(s(n)\) to be used in the high pass filter. This trick cannot be applied in the 9/7 filter bank and therefore we do need the number of operations as described above. However, we are still able to replace the multiplications by shift operations and additions/subtractions. Because the \(h_9\)-filter coefficients are so convenient, only 3 additions/subtractions and 9 shift operations are needed to replace all multiplications. \((46x(n) = (32 + 16 - 2)x(n))\)

The total number of operations per read pixel/coefficient for all transformations mentioned above is shown in table 3.3. For uncomplicated comparison, tables 3.4 and 3.5 express the total computational requirements in shift and add operations. Of course, the shift operations may be ignored in case of a hardware implementation.

3.2 Conclusions

Tables 3.4 and 3.5 show that the number of operations for the TT-transform and DCT are in the same order of magnitude. The DCT requires more operations per read pixel than the TT-transform. However, the number of read operations \textit{per frame} is far less. These two disadvantages of both transforms seem to cancel out and therefore cause the number of operations to be approximately the same. This implies that the TT-transform is a competitor of the DCT in terms of operation cost. However, the memory access bandwidth of the TT-transform is still fairly high. At the present, processing power is not the bottle-neck, but memory access bandwidth is. Therefore we need to decrease the memory access bandwidth of the TT-transform, to make it compete with the DCT in both areas. In the next chapter a few methods will be discussed to decrease this bandwidth.
3.2 Conclusions

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Shifts /</td>
<td>Adds</td>
</tr>
<tr>
<td>TT</td>
<td>$4\frac{1}{2}$ /</td>
<td>3</td>
</tr>
<tr>
<td>DCT 11/29</td>
<td>$15\frac{2}{5}$ /</td>
<td>12\frac{3}{5}</td>
</tr>
<tr>
<td>DCT 64/64</td>
<td>64 /</td>
<td>48</td>
</tr>
<tr>
<td>DWT 9/7</td>
<td>9 /</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 3.3: Operations per read pixel/coefficient for different transforms

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TT</td>
<td>3</td>
<td>66 528</td>
<td>66 528</td>
<td>299 376</td>
<td>432 432</td>
</tr>
<tr>
<td>TT</td>
<td>4</td>
<td>67 320</td>
<td>67 320</td>
<td>302 940</td>
<td>437 580</td>
</tr>
<tr>
<td>DCT 11/29</td>
<td>-</td>
<td>25 344</td>
<td>25 344</td>
<td>383 328</td>
<td>497 376</td>
</tr>
<tr>
<td>DCT 64/64</td>
<td>-</td>
<td>25 344</td>
<td>25 344</td>
<td>1 622 016</td>
<td>1 622 016</td>
</tr>
<tr>
<td>DWT 9/7</td>
<td>3</td>
<td>66 528</td>
<td>66 528</td>
<td>665 280</td>
<td>931 392</td>
</tr>
<tr>
<td>DWT 9/7</td>
<td>4</td>
<td>67 320</td>
<td>67 320</td>
<td>673 200</td>
<td>969 480</td>
</tr>
</tbody>
</table>

Table 3.4: Total number of operations per frame (one color component) for different transforms on a QCIF image (176×144), expressed in shift and add/sub operations.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TT</td>
<td>4</td>
<td>266 112</td>
<td>266 112</td>
<td>1 197 504</td>
<td>1 729 728</td>
</tr>
<tr>
<td>TT</td>
<td>5</td>
<td>270 072</td>
<td>270 072</td>
<td>1 215 324</td>
<td>1 755 468</td>
</tr>
<tr>
<td>DCT 11/29</td>
<td>-</td>
<td>101 376</td>
<td>101 376</td>
<td>1 533 312</td>
<td>1 989 504</td>
</tr>
<tr>
<td>DCT 64/64</td>
<td>-</td>
<td>101 376</td>
<td>101 376</td>
<td>6 488 064</td>
<td>6 488 064</td>
</tr>
<tr>
<td>DWT 9/7</td>
<td>4</td>
<td>266 112</td>
<td>266 112</td>
<td>2 661 120</td>
<td>3 725 568</td>
</tr>
<tr>
<td>DWT 9/7</td>
<td>5</td>
<td>270 072</td>
<td>270 072</td>
<td>2 700 720</td>
<td>3 781 008</td>
</tr>
</tbody>
</table>

Table 3.5: Total number of operations per frame (one color component) for different transforms on a CIF image (352×288), expressed in shift and add/sub operations.
Chapter 4

Memory access bandwidth reduction

4.1 Bandwidth reduction using global memory

Previously we have seen that a discrete wavelet transform requires approximately the same processing power as the DCT 11/29 and almost 4 times less than the DCT 64/64. During a discrete cosine transform, every pixel is only read once and every DCT-coefficient is only written once. The wavelet transform however, requires approximately $2^{\frac{2}{3}}$ as much bandwidth as a DCT ($BW_{DCT}$). Because a two-dimensional transform is calculated, half-level wavelet coefficients should be stored in global memory. These coefficients are the result of a horizontal transform and must be vertically transformed after being reread from global memory. Depending on the wavelet decomposition level, this process should be carried out multiple times. However, the amount of coefficients to be processed, decreases by a factor 4 for each new decomposition level. Hence, the total amount of memory operations converges to $2^{\frac{2}{3}}$ times the operations needed for a DCT.

For now, we assume that there is a global memory, which is accessible through one global bus for all processing units. This architecture forces us to reduce the DWT memory access bandwidth in order to compete with the DCT. First we will discuss two methods to decrease global memory size. Next, we will discuss how the TT-transform can be performed with a small local memory.

4.1.1 Filter placement between sensor and memory

Most memory access bandwidth is required in the first decomposition level. This is because this level handles an entire frame. Assuming that the sensor generates the original picture data on a line-by-line basis, there is a possibility to reduce memory access bandwidth substantially. Figure 4.1 shows
4.1 Bandwidth reduction using global memory

Figure 4.1: Horizontal filter placement between sensor and memory

Bandwidth reduction can be decreased by immediately applying a horizontal transform to the output data stream of the sensor. In this case, the original picture is never stored in global memory, saving \( XY \) write operations and \( XY \) read operations per color component, where \( X \) and \( Y \) are the frame width and height in pixels. The same filter bank can be used for all horizontal and vertical transforms of next levels.

The amount of bandwidth saved by this method is shown in equation 4.1,

\[
\lim_{J \to \infty} BW_{\text{saved}} = \lim_{J \to \infty} \frac{XY}{2XY \sum_{i=1}^{J} 4^{1-i}} BW_{\text{DCT}} = \frac{3}{8} BW_{\text{DCT}}
\]  

(4.1)

where \( J \) is the decomposition level. This means that the number of read operations as well as the number of write operations is decreased by approximately 37.5% at almost no extra cost. Tables 4.1 and 4.2 show the number of memory operations for QCIF and CIF frames when this method is used.

In the next subsection another bandwidth reduction method is presented that can be combined with this method to make the discrete wavelet transform even more efficient in terms of bandwidth.

4.1.2 Bandwidth reduction using pseudo-dual port memory

Pseudo-dual port memory is a memory type that is able to load and store data within one clock cycle. The only restriction is that the loading and storing should be done at the same memory address. Pseudo-dual port memory is a standard feature on Philips memories. This type of memory can be used to reduce the number of read and write operations of a discrete wavelet transform.

In order to find out how much we can save by using pseudo-dual port memory, we need to know in what order the Two-Ten analysis filter bank produces its output and whether there are any clock cycles in which there is no read or write action. Every possible absence of read or write ac-
4.1 Bandwidth reduction using global memory

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TT</td>
<td>3</td>
<td>41 580</td>
<td>41 580</td>
</tr>
<tr>
<td>TT</td>
<td>4</td>
<td>42 075</td>
<td>42 075</td>
</tr>
<tr>
<td>DCT 11/29</td>
<td>-</td>
<td>25 344</td>
<td>25 344</td>
</tr>
<tr>
<td>DCT 64/64</td>
<td>-</td>
<td>25 344</td>
<td>25 344</td>
</tr>
<tr>
<td>DWT 9/7</td>
<td>3</td>
<td>41 580</td>
<td>41 580</td>
</tr>
<tr>
<td>DWT 9/7</td>
<td>4</td>
<td>42 075</td>
<td>42 075</td>
</tr>
</tbody>
</table>

Table 4.1: Total number of memory operations per frame (one color component) for different transforms on a QCIF image (176×144), using a horizontal filter between sensor and memory for the wavelet transforms.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TT</td>
<td>4</td>
<td>166 320</td>
<td>166 320</td>
</tr>
<tr>
<td>TT</td>
<td>5</td>
<td>168 300</td>
<td>168 300</td>
</tr>
<tr>
<td>DCT 11/29</td>
<td>-</td>
<td>101 376</td>
<td>101 376</td>
</tr>
<tr>
<td>DCT 64/64</td>
<td>-</td>
<td>101 376</td>
<td>101 376</td>
</tr>
<tr>
<td>DWT 9/7</td>
<td>4</td>
<td>166 320</td>
<td>166 320</td>
</tr>
<tr>
<td>DWT 9/7</td>
<td>5</td>
<td>168 300</td>
<td>168 300</td>
</tr>
</tbody>
</table>

Table 4.2: Total number of memory operations per frame (one color component) for different transforms on a CIF image (352×288), using a horizontal filter between sensor and memory for the wavelet transforms.
4.2 Bandwidth reduction using local memory

In the previous solutions for bandwidth reduction, the global system bus is still burdened with intermediate (half-level) results of calculations performed by the DWT unit. This subsection deals with the concept of using a local memory to reduce bandwidth.

The package PHIDEO [6] was used as an analysis tool to generate an ASAP-schedule of a one-dimensional TT-transform. We assumed that the execution of all single calculations requires one clock cycle. Figure 4.2 shows the order of the \( s(n) \) and \( d(n) \) outputs of the TT-filter bank.

From this schedule we can see that the coefficients of an entire line are generated almost every clock cycle, except for the first three smooth values and last three detail values. The reason for this is that the detail outputs are delayed because they depend on four smooth outputs. Output \( d(n) \) cannot be calculated before \( s(n + 2) \) is known.

If more lines and columns are processed, the calculations can be pipelined. Another PHIDEO scheduling run was done to prove this. The scheduling of the processing of an entire frame proved to be very similar to that of a one-dimensional TT-transform. A result is generated every clock cycle, except for the first and last few values of the frame. Relatively, these few coefficients are negligible and therefore we conclude that almost every read operation can be combined with a write operation.

This implies that memory access bandwidth can be reduced by almost 50% by the use of pseudo-dual port memory. Tables 4.3 and 4.4 show the number of memory operations for QCIF and CIF frames when this method is used and combined with placement of a horizontal filter between sensor and memory.

A penalty for the use of this type of memory is the need for more complex addressing. How much more complex the addressing will be compared to the straightforward load/store solution can only be determined if pseudo-dual port memory is supported by PHIDEO in the future. We expect this disadvantage to be far smaller than the benefits of halving memory access bandwidth.

### Figure 4.2: Analysis filter bank output order

\[
\begin{array}{cccccccccccc}
\text{\( s(0) \)} & \text{\( s(1) \)} & \text{\( s(2) \)} & \text{\( d(0) \)} & \text{\( s(3) \)} & \text{\( d(1) \)} & \text{\( s(4) \)} & \text{\( d(2) \)} & \text{\( s(5) \)} & \text{\( d(3) \)} & \text{\( s(6) \)} & \text{...} \\
\text{\( s(n) \)} & \text{\( d(n) \)} & \text{\( s(n+1) \)} & \text{\( d(n+1) \)} & \text{\( s(n+2) \)} & \text{\( d(n+2) \)} & \text{\( s(n+3) \)} & \text{\( d(n+3) \)} & \text{\( s(n+4) \)} & \text{\( d(n+4) \)} & \text{\( s(n+5) \)} & \text{\( d(n+5) \)}
\end{array}
\]

Five, if a takes one clock cycle to perform a shift operation.
4.2 Bandwidth reduction using local memory

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TT</td>
<td>3</td>
<td>≈20 790</td>
<td>≈20 790</td>
</tr>
<tr>
<td>TT</td>
<td>4</td>
<td>≈21 038</td>
<td>≈21 038</td>
</tr>
<tr>
<td>DCT 11/29</td>
<td>-</td>
<td>25 344</td>
<td>25 344</td>
</tr>
<tr>
<td>DCT 64/64</td>
<td>-</td>
<td>25 344</td>
<td>25 344</td>
</tr>
<tr>
<td>DWT 9/7</td>
<td>3</td>
<td>≈20 790</td>
<td>≈20 790</td>
</tr>
<tr>
<td>DWT 9/7</td>
<td>4</td>
<td>≈21 038</td>
<td>≈21 038</td>
</tr>
</tbody>
</table>

Table 4.3: Total number of memory operations per frame (one color component) for different transforms on a QCIF image (176x144), using a horizontal filter between sensor and pseudo-dual port memory for the wavelet transforms.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TT</td>
<td>4</td>
<td>≈83 160</td>
<td>≈83 160</td>
</tr>
<tr>
<td>TT</td>
<td>5</td>
<td>≈84 150</td>
<td>≈84 150</td>
</tr>
<tr>
<td>DCT 11/29</td>
<td>-</td>
<td>101 376</td>
<td>101 376</td>
</tr>
<tr>
<td>DCT 64/64</td>
<td>-</td>
<td>101 376</td>
<td>101 376</td>
</tr>
<tr>
<td>DWT 9/7</td>
<td>4</td>
<td>≈83 160</td>
<td>≈83 160</td>
</tr>
<tr>
<td>DWT 9/7</td>
<td>5</td>
<td>≈84 150</td>
<td>≈84 150</td>
</tr>
</tbody>
</table>

Table 4.4: Total number of memory operations per frame (one color component) for different transforms on a CIF image (352x288), using a horizontal filter between sensor and pseudo-dual port memory for the wavelet transforms.
memory near the DWT unit, inspired by Chakrabarti et al. [7]. Since it is unacceptable to use a local memory that is as large as one frame, a minimum amount of memory should be determined for calculating parts of the frame, e.g. only a certain number of lines.

Equations 2.1 and 2.2 show that the first $d(n)$ value ($d(0)$) can only be generated after the calculation of the signals $s(-2), s(-1), s(1), s(2)$ and the input signal difference

$$\Delta(n) := x(2n) - x(2n + 1). \quad (4.2)$$

It is useful to calculate this signal difference immediately after the input signal has been retrieved from global memory. In this case only the $\Delta$-value should be stored, instead of the two input values. A $\Delta(n)$ value is needed for the calculation of $d(n)$, but should be stored for at least four clock cycles because the $s(n-2) \cdots s(n+2)$ are necessary for $d(n)$ and should therefore be calculated first. This is not a problem for the horizontal transform, since the pixels of the original picture are read along rows and there is a fixed period of time between the generation of $s(n)$ and $d(n)$. A detailed look at equations 2.1 and 2.2 reveals that only two $\Delta$-values need to be stored in the horizontal transform process. For vertical transforms however, we need to store two $\Delta$-values for every column.

Figure 4.3 shows the usage of a 7-line local memory for the TT-transform. The first picture shows the situation just after the horizontal transform of one picture line. The second picture line is read, vertical low-pass components LLI and HLI are computed simultaneously and stored in the same place as the horizontal transform coefficients Ll and H1. The $\Delta$-values are stored below the fifth line. After six picture lines have been read, the first vertical high-pass components can be generated for each column. The first $\Delta$-line is involved in this calculation before the third $\Delta$-line is computed. This allows us to reuse the memory space that was occupied by the first $\Delta$-line. The first two vertical high-pass components are computed differently from all other components because $s(-2)$ and $s(-1)$ do not exist. A mirroring scheme [3] is applied for the computation of these two components:

- $d(0)$ uses $s(1), s(0), s(1), s(2)$ and $\Delta(0)$
- $d(1)$ uses $s(0), s(0), s(2), s(3)$ and $\Delta(1)$
- $d(2)$ uses $s(0), s(1), s(3), s(4)$ and $\Delta(2)$
- $d(3)$ uses $s(1), s(2), s(4), s(5)$ and $\Delta(3), \ldots$

After computation of both horizontal and vertical transforms of the first 10 picture lines, there are two options for storage of the first level results. Of course, LH-, HL- and HH-components can be stored into global memory (if any) or passed on to other system units because they are no longer used in
next level decomposition. The LL-components however, should be available to the DWT unit for next stage processing. Two decisions can be made, namely:
4.2 Bandwidth reduction using local memory

- Option 1: Store the LL-components in global memory and reread them for horizontal transformation of the next stage. Of course, the values can only be reread after all picture lines have been decomposed in the first stage. This option does not require more local memory than the 7 lines that were previously used for the first decomposition.

- Option 2: Do not store the LL-components in global memory, but immediately apply a horizontal transform and store the results in extra local memory. This extra local memory is only half as large as the memory needed for the first transform. Every next stage uses half as much local memory as its previous stage.

The first option reduces bandwidth from $2\frac{2}{3}BW_{DCT}$ to $1\frac{1}{3}BW_{DCT}$. The second option keeps all coefficients for further processing in local memory and therefore has the same bandwidth demand as a DCT. The amount of local memory should now be determined for both options to make a useful trade-off.

Unfortunately, a 7-line memory for every stage may not be enough. As discussed before, symmetry of high-pass filter coefficients is used to reduce the number of multiplications (see equations 2.1 and 2.2). This can only be done if the multiplication results are stored for four clock cycles. This implies that we need two extra words to store $3s(n)$ and $22s(n)$ for every coefficient in the first 5 lines of local memory. This means that we need $5 \times 2$ extra lines of memory to halve the number of multiplications. The amount of local memory needed for both options mentioned above is shown in tables 4.5 and 4.6.

<table>
<thead>
<tr>
<th>Level</th>
<th>Storage option</th>
<th>Bandwidth</th>
<th>Memory size</th>
</tr>
</thead>
<tbody>
<tr>
<td>≥ 4</td>
<td>1</td>
<td>$1\frac{1}{3}BW_{DCT}$</td>
<td>2 992 words</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>$BW_{DCT}$</td>
<td>5 610 words</td>
</tr>
</tbody>
</table>

Table 4.5: Amount of local memory required for a single color component QCIF-frame for two storage options. Option 1: LL-coefficient storage in global memory. Option 2: LL-coefficient storage in local memory.

<table>
<thead>
<tr>
<th>Level</th>
<th>Storage option</th>
<th>Bandwidth</th>
<th>Memory size</th>
</tr>
</thead>
<tbody>
<tr>
<td>≥ 5</td>
<td>1</td>
<td>$1\frac{1}{4}BW_{DCT}$</td>
<td>5 984 words</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>$BW_{DCT}$</td>
<td>11 594 words</td>
</tr>
</tbody>
</table>

Table 4.6: Amount of local memory required for a single color component CIF-frame for two storage options. Option 1: LL-coefficient storage in global memory. Option 2: LL-coefficient storage in local memory.
4.3 Conclusions

The bandwidth needed for storage option 1 converges to $\frac{4}{3}BW_{DCT}$ as the wavelet decomposition level increases. The exact bandwidth needed for this option can be calculated as shown in equation 4.3,

$$BW_{option1} = BW_{DCT} \sum_{i=1}^{J} 4^{1-i}$$

(4.3)

where $J$ is the wavelet decomposition level. The exact amount of memory for any image format, using storage option 2, can be calculated with equation 4.4,

$$MEMSIZE_{option2} = (7 + 5 \times 2)X \sum_{i=1}^{J} 2^{1-i}$$

(4.4)

where $X$ is the picture width in pixels.

The amount of memory needed is unacceptably large due to the fact that all $s(n)$ coefficients should be stored as well as $3s(n)$ and $22s(n)$ during a vertical transform. This extra local memory may be traded back again for multiplications, since the algorithm does not require a lot of processing power. If $3s(n)$ and $22s(n)$ are recalculated instantly when needed, only 5 lines of $s(n)$ coefficients and 2 lines of $\Delta(n)$-coefficients have to be stored at the cost of doubling the number of multiplications. Table 4.7 and 4.8 show the hardware cost of this solution. The number of calculations needed for the DCT transform are repeated in this table for reasons of comparison. The calculation cost of the TT-transform has increased because of the extra in-place calculations. However, it is still in the same order of magnitude as the DCT transforms.

4.3 Conclusions

In this chapter, several architectures were shown which offer a solution to the memory access bandwidth problem of wavelet transforms. The first two solutions assumed that there is a global memory available that is large enough to store one frame. These architectures decrease the memory bandwidth dramatically without increasing overall complexity. A third solution was found that requires the use of a local memory near the wavelet transform unit. Future research should include the combination of all three architectures mentioned here. If the Two-Ten transform is implemented with a filter between the camera sensor and local pseudo-dual port memory, tools like PHIDEO will prove to be very rewarding.
4.3 Conclusions

Table 4.7: Amount of local memory and number of operations required for QCIF with two storage options. Values $3s(n)$ and $22s(n)$ are not stored for further calculation, but recalculated when needed. Option 1: LL storage in global memory. Option 2: LL storage in local memory.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TT</td>
<td>4</td>
<td>1</td>
<td>1 232</td>
<td>598 752</td>
<td>632 016</td>
</tr>
<tr>
<td>TT</td>
<td>4</td>
<td>2</td>
<td>2 310</td>
<td>605 880</td>
<td>639 540</td>
</tr>
<tr>
<td>DCT 11/29</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>383 328</td>
<td>497 376</td>
</tr>
<tr>
<td>DCT 64/64</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1 622 016</td>
<td>1 622 016</td>
</tr>
</tbody>
</table>

Table 4.8: Amount of local memory and number of operations required for CIF with two storage options. Values $3s(n)$ and $22s(n)$ are not stored for further calculation, but recalculated when needed. Option 1: LL storage in global memory. Option 2: LL storage in local memory.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TT</td>
<td>5</td>
<td>1</td>
<td>2 464</td>
<td>2 395 008</td>
<td>2 528 064</td>
</tr>
<tr>
<td>TT</td>
<td>5</td>
<td>2</td>
<td>4 774</td>
<td>2 424 648</td>
<td>2 565 684</td>
</tr>
<tr>
<td>DCT 11/29</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1 533 312</td>
<td>1 989 504</td>
</tr>
<tr>
<td>DCT 64/64</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>6 488 064</td>
<td>6 488 064</td>
</tr>
</tbody>
</table>
Chapter 5

Integration of the TT-transform with H.263

As mentioned before, H.263 uses a block-based DCT transform instead of a frame-based wavelet transform. In order to allow integration of the wavelet transform with H.263 intra-frame coding, we need to group the wavelet coefficients into manageable blocks of $8 \times 8$ coefficients. In the following sections we will discuss a method to compose, scan and encode these blocks. This method was added as an option to the crewcore program [3]. For estimation of the size of the encoded bit stream we used the Telenor H.263 source code [8]. Objective and subjective results of picture quality and compression ratio will be shown at the end of this chapter.

De Queiroz et al. [9] suggest replacement of the DCT by a DWT in a JPEG encoder, without actually changing the coefficient encoding procedure. They report that their method performs approximately 1 dB better than a baseline JPEG-encoder.

Of course, this method can be made use of in an H.263 encoder. Figure 5.1(a) depicts a standard H.263 intra frame coder. The zig-zag scan, which is a good scan for $8 \times 8$ DCT blocks, is not the best scan for all subbands of a wavelet transformed picture. Therefore, a modified scan (figure 5.1(b)) should be applied that is suitable for run length encoding of wavelet subbands. The quantizer and run length/variable length encoder can be adapted by designer-defined quantization and RLE/VLE tables. This is necessary because we are dealing with wavelet coefficients instead of DCT coefficients. Figure 5.1(c) shows the integration of the DWT with the H.263 coefficient coding of figure 5.1(a). The same designer-defined table specifications are used in this scheme. The rescan mentioned in figure 5.1(c) means that an inverse zig-zag scan is applied after the modified scan of figure 5.1(b). This method ensures that the DWT with rescan can be integrated with existing H.263 coefficient coding blocks.
5.1 Block composition, scanning and quantization

Figure 5.1: Replacement of the discrete cosine transform by a discrete wavelet transform in an H.263 intra frame coder

Note that the original idea by De Queiroz et al. was to replace the DCT in a JPEG encoder instead of a H.263 encoder. Therefore, we do not expect the combination of CREW and this method to be optimal. Similarly, the Telenor H.263 RLE/VLE was not specifically designed to encode the data stream that is generated by this method. However, both are low-complexity solutions that enable us to draw quick conclusions about this suboptimal case.

5.1 Block composition, scanning and quantization

Figure 5.2 shows a three-level decomposed frame. The arrows in this figure show the parent-children relationships between wavelet coefficients. Parent and children coefficients are highly correlated. Insignificant root pixels in the LL-subband, for example, are unlikely to have significant children or grandchildren. Furthermore, pixels in high-frequency subbands are mostly zero in sufficiently quantized pictures. This is the reason why the composed block is scanned as depicted in figure 5.2. In order to maximize the proba-
5.1 Block composition, scanning and quantization

bility of having long strings of zeroes, the diagonal subbands (HH-subbands) are scanned in a zig-zag manner. Horizontal subbands (HL-subbands) are scanned in a vertical manner and vertical subbands (LH-subbands) are scanned horizontally.

A block is constructed for every coefficient in the LL-subband of a frame. Each coefficient in the LL-subband is the root of a composed block. The output order of the blocks is fixed as the LL-subband is scanned in a line-wise manner. The output is passed to the quantizer and RLE/VLE respectively.

The quantization table is also constructed as proposed by De Queiroz et al. [9]. As in JPEG, this table contains higher quantization values (step sizes) for wavelet coefficients of higher spatial frequency. Note that the original H.263 uniform quantization is replaced here.
5.2 Telenor run length and variable length encoder

The run length and variable length encoder are a part of the entire H.263 encoder/decoder package, which is public domain and available through Telenor Research and Development on the Internet [8]. This part is used to make a comparison between a DCT-based image coder and our mix of CREW and the previously mentioned scan method. If successful, this method can be optimized and integrated with previously mentioned architectures in chapter 4.

Only a small part of the Telenor encoder is used: the procedures that calculate the bit length of a block after it has been encoded by the RLE/VLE encoder. Note that actual encoding is omitted as we are only interested in the amount of bits that is used to represent the stream of composed blocks.

5.3 CREW bypass procedure

This section describes the program structure of block composition, quantization and RLE/VLE bit count implementation in CREW. This implementation will be called the bypass procedure from now on, because several steps in the original CREW program flow are bypassed by it. Appendix A provides a function description.

5.3.1 Bypass procedure program flow

The bypass procedure is added to the crewcore program by means of a command line switch named -bypass. This switch is only an addition; the original CREW source code was not altered, except for some small additions to the command line parsing procedure.

Figure 5.3 depicts the program flow of CREW and the bypass switch. If the bypass switch is enabled, the CREW code is only used to load a picture file and apply the TT-transform to it. The transformed picture is offered to the bypass section. Blocks are composed, scanned and quantized and raw binary data are written to files. The CREW image saving procedures are used to write the picture data to a file. This quantized picture file can be compared to the original picture by means of the -psnr switch of the crewcore program. Finally, the Telenor code accepts the output stream of blocks and calculates the number of bits that would be produced by the RLE/VLE. Bit count is performed on a block-by-block basis. The compression factor of the picture is calculated by dividing the original picture size by the accumulation of all bit counts of all blocks.

The bypass code distinguishes between grayscale and color pictures. In case of a grayscale picture, only one frame, the luminance (Y) frame will be processed. In the latter case, all three frames Y, U and V will be processed. The chrominance frames (U and V frames) are processed in a different way.
than the luminance frame. More information on this subject is given in subsection 5.3.4.

Figure 5.3: Bypass switch in CREW

5.3.2 Command line syntax

The command line syntax of crewcore with the bypass switch is not very different from the original CREW command line syntax. This subsection clarifies the syntax by means of the following example:

```
crewcore -verbose 1 -bypass -level 3
          OriginalPictureName OutputPictureName
```

This most common command reads OriginalPictureName, transforms it into a 3-level decomposed frame, composes blocks of size $2^3 \times 2^3$, scans
and quantizes these blocks and calculates the bit count. The picture file is saved as OutputPictureName.

The bypass code is designed to generate *sequences of pictures* that are quantized at different quantization levels. This is the reason why the bypass code asks for a sequence length. This is the number of times the original picture is read and processed with a different quantization level. Next, the bypass code prompts for a scaling value. The scaling value determines how fine or coarse quantization should be. It is the same scaling value \( A \) as discussed by De Queiroz et al. [9]. A higher scaling value results in coarser quantization and therefore yields a compressed picture of smaller size. The user will be prompted for a scaling value at the start of every new processing step in the sequence. This process can be automated by a simple command line redirection statement:

```
crewcore -verbose 1 -bypass -level 3
OriginalPictureName OutputPictureName < InputList
```

in which *InputList* contains, for example:

```
5
4
10
30
55
120
```

The 5 on the first line is the sequence length and the following elements, 4, 10, 30, 55 and 120, are the scaling values for each quantization process. All pictures in the sequence are saved separately as quant\(_x\).pcx, where \( x \) is the scaling value.

Finally, a few *extra switches* were added for testing purposes. These are:

- *-noquant*, which forces the quantizer *not* to quantize.
- *-discardhighest*, which forces the three subbands with the highest spatial frequencies to zero. These are subbands HL-1, LH-1 and HH-1 for Y-frames and subbands HL-2, LH-2 and HH-2 for U and V frames.

### 5.3.3 Verbosity

The *-verbose* switch in CREW can take values ranging from -1 to 5. The bypass switch only uses values 0, 1 and \( \geq 2 \). In the following, these three levels will be explained.

- Verbosity level 0. Shown are
5.3 CREW bypass procedure

- the original picture size in bytes,
- the compressed file size in bytes and
- the compression factor.

• Verbosity level 1. Besides the information printed at verbosity level 0, the following data are printed for every picture component (Y, U and V if available):
  - the frame dimensions,
  - the component type (luminance or chrominance frame),
  - the number of composed blocks,
  - numbers and percentages of zeroes and non-zeroes in the transformed frame,
  - the average position of the last non-zero in each quantized block,
  - the number of bytes in the compressed frame and
  - the compression factor of this component.

• Verbosity level ≥2. All of the information above is printed plus
  - quantization step sizes for each subband and
  - the quantizer output for each composed block. These are denoted as block \((i,j)\), where \(i\) and \(j\) denote the location of the root coefficient of that block in the LL-subband.

5.3.4 Grayscale and color pictures

Color pictures and grayscale pictures are treated differently. Similar to JPEG coding techniques, the chrominance frames are subsampled, whereas luminance frames are not. However, there is no special chrominance frame subsampling algorithm in CREW. The reason for this is that sufficiently coarse embedded quantization of chrominance frames automatically yields a subsampled frame [3]. Figure 5.4 shows how luminance and chrominance frames are treated differently in the bypass procedure. Subsampling is done indirectly by omission of the subbands of the first decomposition level. Inverse transformation will then yield a nicely smoothed frame.

In CREW, the user will have to choose a quantization level for these subbands that is sufficiently high. In the bypass procedure, the chrominance frames are decomposed by one extra level and the subbands of the first decomposition level are always discarded, regardless of the quantization level. This way, it is not necessary to calculate the subbands of the first decomposition level, except for the LL-1 subband, which is used for further transformations. Besides, the block composition procedure (figure 5.2) generates equally large blocks for both frame types.
5.4 Bypass procedure evaluation

![Diagram](image)

(a) Luminance frame: Y, L-level
(b) Chrominance frame: U,V, (L+1)-level

Figure 5.4: Luminance and "subsampled" chrominance frames

Complete compatibility with 8×8 H.263 blocks requires a 3-level decomposition of the luminance frame and subsequently a 4-level decomposition of both chrominance frames. A constraint is that frame width and height are multiples of \(2^4 = 16\).

A second difference in the processing of color images is that the chrominance frames are quantized more coarsely than the luminance frame. In the current version of the bypass program, a multiple of the scaling value is used for quantization of chrominance frames. Furthermore, the quantization matrix is a \(2^4 \times 2^4\) matrix, unlike the original 8×8 matrix [9]. In this case, the same construction method described in the article is used for the generation of the 16×16 matrix. Of course, this straightforward method may not be optimal. Further research on this subject is necessary.

5.4 Bypass procedure evaluation

In this section, results will be shown of the application of the bypass procedure to pictures of different dimensions. Grayscale and color versions of the Lena picture were used for this purpose. In addition, a typical video phone picture (Eref) was made with a digital camera. The original picture sizes are 512×512 pixels. Since this is not one of the resolutions supported by H.263 and seems to be rather large for small LCD displays anyway, a resized picture of QCIF size has also been used for evaluating the bypass procedure. Resizing was done with the package xv. Other graphics processing packages offer different algorithms for resizing a picture, some of which apply a soft filtering to the picture after resizing. Of course, compression results would vary a lot if such a method is used. We chose to use the package xv because
5.4 Bypass procedure evaluation

It appears to use a rather straightforward resizing algorithm. The most accurate evaluation results can be obtained when using pictures that are made with a digital camera sensor that has a QCIF resolution (or multiples of QCIF).

In each evaluation, three different image coding techniques are compared:

- the CREW algorithm,
- the bypass procedure and
- non-baseline DCT-JPEG encoding provided by the ImageMagick package [10].

It is not fair to compare our compression method to baseline JPEG, such as generated by the package xv [11]. Therefore, the decision was made to use the package ImageMagick, which supports non-baseline JPEG.

Appendix B contains evaluation plots for both pictures, different decomposition levels and different dimensions. Each plot shows the compression factor on the x-axis and the PSNR (Peak Signal to Noise Ratio) on the y-axis. Subjectively, we conclude that the quality of pictures with a PSNR of 29...30dB or lower is "not sufficient". Therefore it is sensible to compare the plots in the range of 30dB or more. Note that the PSNR does not always tell everything about the quality of a picture. Blocking measurements can also be performed. However, PSNR measurements here enable comparison to other compression methods in literature.

Appendix B shows plots for grayscale pictures (subappendix B.1) and color pictures (subappendix B.2) respectively. Each subappendix contains four sections:

- the results for the 512×512 Lena picture, at three different decomposition levels (3, 4 and 5) and a comparison of the Bypass algorithm with itself (at levels 3, 4 and 5),
- the results for the QCIF Lena picture, at decomposition level 3,
- the results for the 512×512 Eref picture, at three different decomposition levels (3, 4 and 5) and a comparison of the Bypass algorithm to itself (at levels 3, 4 and 5) and
- the results for the QCIF Eref picture, at decomposition level 3.

Appendix C contains a few subjective results of grayscale images. The original Lena or Eref picture, a DCT-JPEG coded picture and a Bypass-coded picture are shown for both resolutions. Compression factors of DCT-JPEG and Bypass-compressed pictures are approximately the same.

1 Because dedicated JPEG coding software was not available we used xv and ImageMagick. Experiments have proved that images of lower JPEG-quality are smaller when generated by ImageMagick. We therefore conclude that ImageMagick supports non-baseline JPEG.
5.4 Bypass procedure evaluation

5.4.1 Results, Grayscale pictures

The four plots of the Lena picture in subappendix B.1 show that the Bypass algorithm performs approximately within 1.5dB of the DCT-JPEG algorithm (figures B.1 to B.4). Generally, the CREW algorithm outperforms both compression schemes due to its superior context model. Figure B.4 shows that the 3-level decomposition is better for low compression factors. This is due to the fact that the composed blocks contain 256 coefficients in the 4-level case. The probability of long strings of zeroes is much higher for such large blocks. The Telenor coder generates an exception code for strings that contain 26 zeroes or more [8]. The overhead of these exception codes cause the 4-level model to perform worse at fine quantization levels. When quantization becomes coarser, the end-of-block marker averagely appears sooner in the composed blocks, which decreased the probability of many exception codes. This is where the 4-level decomposition performs better than the 3-level variant. The same explanation is given for the inferiority of the 5-level decomposition compared to the 4-level. Eventually, the 5-level variant will perform better, but this is far below the PSNR level that is considered "acceptable".

Figure B.5 shows the same situation, albeit that the Bypass algorithm performs slightly better than in the case of 512x512 pictures. The explanation for this is that there is less correlation between adjacent pixels in the resized picture. As this may be disadvantageous for the CREW and DCT-JPEG algorithms, its effect is less on the Bypass algorithm. This is because the block composition in the Bypass algorithm is based on parent-children relations, which appear to maintain higher correlation after resizing.

Figures B.6 to B.8 show that a typical video phone image can be compressed much better than the Lena image. The three-level decomposed 512x512 picture shows slight inferiority of the Bypass algorithm. The 4-level variant however, performs much better than DCT-JPEG and is within 2dB of the CREW algorithm. Compression is 75% better than DCT-JPEG at a PSNR of 30dB. The same explanation as the one for the Lena plot (figure B.4) can be given for figure B.9. The 4-level variant is best for this resolution.

Lastly, from figure B.10 we may conclude that the Bypass algorithm approaches the performance of the CREW algorithm for typical low resolution video phone images.

5.4.2 Results, Color pictures

Color pictures are processed as described in section 5.3.4. The scaling value (SV) for chrominance frames is experimentally chosen as

\[ SV_{chrom} = 4 \cdot SV_{lum} \]  

(5.1)
5.5 Conclusions

As mentioned before, more research should be done on the coarseness of quantization for chrominance frames.

Figures B.11 to B.20 in subappendix B.2 show a comparison between the Bypass-algorithm with chrominance subsampling, DCT-JPEG with subsampling and CREW without\(^2\) subsampling.

The results in the grayscale section showed us that the PSNR of the Bypass algorithm was always lower than the PSNR of CREW. Therefore, we conclude from figures B.11 and B.12 that the gain due to subsampling is at least 3dB.

Figure B.14 shows that a 4-level decomposition is the best compromise for images of these dimensions. Results are slightly better for QCIF images, as shown in figure B.15. At a PSNR of 30dB, the compression factor is 120% higher than the DCT-JPEG algorithm.

The results of the Eref\(^3\) image in figures B.16 to B.20 show that for typical video phone intra frames, a compression factor of 100 (512×512 images) or 45 (QCIF images) should not be a problem.

5.5 Conclusions

The CREW program [3] provides an easy way to experiment with a different scan method (De Queiroz [9]) and run length/variable length coding (Telenor [8]) and their effect on Two-Ten transformed images.

The combination of these methods is suboptimal, but offers a very reasonable performance, especially with color images. Subsampling of chrominance frames is necessary in order to achieve high compression factors for intra frames. An additional advantage of the chrominance subsampling is that fewer calculations and memory access operations are required in the forward transform. Future research should aim on improving quantization and RL/VL coding.

When image resolution becomes smaller and adjacent pixel correlation decreases, the combination of methods mentioned above performs relatively better than existing image compression methods.

Up to now, only the RLE/VLE bit count has been used to calculate the compression factor. Therefore, it is very important that test results are verified by using the compression and decompression procedures of the Telenor source code [8]. Present results have been verified by a zero count, non-zero count and average last non-zero count for all composed 8×8 blocks. Nevertheless, further verification should be done by writing the compressed data to a file, and decompressing the reread data, so we can be absolutely

\(^2\)Rosenfeldt [3] mentions that chrominance subsampling should be possible by setting the correct discarding factors for certain chrominance subbands. We experienced that the command line switches \(-ycomp\) and \(-uvcomp\) do not seem to work properly in this case.
sure that the test results are correct. In that case, compressed file size can be used for compression factor calculations.

It is sensible to use results of the Lena picture for the dimensioning of a future DWT-unit. Typical images, such as the $E_{ref}$ image used here, only give an indication of a typical compression ratio for intra frames. If the unit is dimensioned in that way, the intra frame quality of typical video phone communication will be sufficient.

The applied scanning method (figure 5.2) requires a global memory of almost an entire frame if no special actions are taken. In the next chapter, a suggestion is made to reduce this memory size. In this way, we will still be able to use the convenient integration method described in this chapter.
Chapter 6

Memory size reduction

6.1 Storage of quantized wavelet coefficients

After each vertical transform that processes high-pass coefficients, quantization can be applied immediately because these coefficients are no source for further calculations. Only coefficients in the LL-subband cannot be quantized since these are recursively processed. The amount of memory saved depends on the coarseness of quantization.

Tables 6.1 to 6.4 give an indication of how much memory can be saved by this method. The two pictures, Lena and Eref, were processed by the bypass procedure. The scaling values were adjusted to obtain quantized pictures with PSNRs of approximately 31dB, i.e. pictures of "reasonable quality". Chrominance frames were subsampled and quantized more coarsely than the luminance frame.

From the tables we conclude that this method does not save very much memory, since reduction factors by quantization range from 3.26 to 5.03. The advantages of this method are its simplicity and the exact knowledge of the amount of memory needed for frame storage.

Memory size can be decreased by another approach to the problem, which would have been in the following section if you had not been reading the public version of this report.
6.1 Storage of quantized wavelet coefficients

<table>
<thead>
<tr>
<th>Picture</th>
<th>lena512.ppm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>512 × 512</td>
</tr>
<tr>
<td>Decomposition Level</td>
<td>4</td>
</tr>
<tr>
<td>Scaling Value</td>
<td>60</td>
</tr>
<tr>
<td>Scaling Values Y:U:V</td>
<td>1:4:4</td>
</tr>
<tr>
<td>PSNR</td>
<td>31.1 dB</td>
</tr>
<tr>
<td>Memory for quantized Y-frame</td>
<td>1 099 776 bits</td>
</tr>
<tr>
<td>Memory for quantized U-frame</td>
<td>246 784 bits</td>
</tr>
<tr>
<td>Memory for quantized V-frame</td>
<td>246 784 bits</td>
</tr>
<tr>
<td>Memory for all quantized frames</td>
<td>1 593 344 bits</td>
</tr>
<tr>
<td>Size of uncompressed frames</td>
<td>6 291 456 bits</td>
</tr>
<tr>
<td>Reduction by quantization</td>
<td><strong>3.95 x</strong></td>
</tr>
</tbody>
</table>

Table 6.1: Memory size reduction by storage of quantized coefficients of 24-bit lena512.ppm color picture

<table>
<thead>
<tr>
<th>Picture</th>
<th>lenaqcif.ppm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>176 × 144</td>
</tr>
<tr>
<td>Decomposition Level</td>
<td>3</td>
</tr>
<tr>
<td>Scaling Value</td>
<td>40</td>
</tr>
<tr>
<td>Scaling Values Y:U:V</td>
<td>1:4:4</td>
</tr>
<tr>
<td>PSNR</td>
<td>30.7 dB</td>
</tr>
<tr>
<td>Memory for quantized Y-frame</td>
<td>131 868 bits</td>
</tr>
<tr>
<td>Memory for quantized U-frame</td>
<td>27 225 bits</td>
</tr>
<tr>
<td>Memory for quantized V-frame</td>
<td>27 225 bits</td>
</tr>
<tr>
<td>Memory for all quantized frames</td>
<td>186 318 bits</td>
</tr>
<tr>
<td>Size of uncompressed frames</td>
<td>608 256 bits</td>
</tr>
<tr>
<td>Reduction by quantization</td>
<td><strong>3.26 x</strong></td>
</tr>
</tbody>
</table>

Table 6.2: Memory size reduction by storage of quantized coefficients of 24-bit lenaqcif.ppm color picture
### 6.1 Storage of quantized wavelet coefficients

<table>
<thead>
<tr>
<th>Picture</th>
<th>eref512.ppm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>512 x 512</td>
</tr>
<tr>
<td>Decomposition Level</td>
<td>4</td>
</tr>
<tr>
<td>Scaling Value</td>
<td>110</td>
</tr>
<tr>
<td>Scaling Values Y:U:V</td>
<td>1:4:4</td>
</tr>
<tr>
<td>PSNR</td>
<td>31.1 dB</td>
</tr>
<tr>
<td>Memory for quantized Y-frame</td>
<td>854 016 bits</td>
</tr>
<tr>
<td>Memory for quantized U-frame</td>
<td>197 888 bits</td>
</tr>
<tr>
<td>Memory for quantized V-frame</td>
<td>197 888 bits</td>
</tr>
<tr>
<td>Memory for all quantized frames</td>
<td>1 249 792 bits</td>
</tr>
<tr>
<td>Size of uncompressed frames</td>
<td>6 291 456 bits</td>
</tr>
<tr>
<td>Reduction by quantization</td>
<td>5.03x</td>
</tr>
</tbody>
</table>

Table 6.3: Memory size reduction by storage of quantized coefficients of 24-bit eref512.ppm color picture

<table>
<thead>
<tr>
<th>Picture</th>
<th>erefqcif.ppm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>176 x 144</td>
</tr>
<tr>
<td>Decomposition Level</td>
<td>3</td>
</tr>
<tr>
<td>Scaling Value</td>
<td>65</td>
</tr>
<tr>
<td>Scaling Values Y:U:V</td>
<td>1:4:4</td>
</tr>
<tr>
<td>PSNR</td>
<td>31.2 dB</td>
</tr>
<tr>
<td>Memory for quantized Y-frame</td>
<td>106 920 bits</td>
</tr>
<tr>
<td>Memory for quantized U-frame</td>
<td>24 156 bits</td>
</tr>
<tr>
<td>Memory for quantized V-frame</td>
<td>24 156 bits</td>
</tr>
<tr>
<td>Memory for all quantized frames</td>
<td>155 232 bits</td>
</tr>
<tr>
<td>Size of uncompressed frames</td>
<td>608 256 bits</td>
</tr>
<tr>
<td>Reduction by quantization</td>
<td>3.92x</td>
</tr>
</tbody>
</table>

Table 6.4: Memory size reduction by storage of quantized coefficients of 24-bit erefqcif.ppm color picture
Chapter 7

Conclusions and recommendations

7.1 Conclusions

Discrete wavelet transforms can be performed by computationally cheap binary symmetric filters. These filters contain small integer coefficients and may therefore be implemented with only shift operations and additions.

The Two-Ten wavelet transform was compared to the discrete cosine transform in terms of computational cost and memory access bandwidth. The number of operations required for the DCT and this DWT are in the same order of magnitude. The memory access bandwidth of the TT-transform is much higher, but can be decreased dramatically by the use of pseudo-dual port memory, different filter placement and the use of a local memory. These methods cause the memory access bandwidth of the TT-transform to be approximately equal to the bandwidth needed for the DCT. Therefore we may conclude that the TT-transform is a competitor of the DCT in intra frame transformation.

A special scanning method enables the combination of a frame-based DWT with existing H.263 coders, which operate on blocks of 8×8 pixels. A suboptimal model was implemented that yields good results, especially in the encoding of color images. Further verification and optimization of the integration model should be carried out.

The scanning method, as discussed above, needs almost an entire frame memory. That is why a suggestion for memory size reduction was presented. The scheme reduces the memory size by a factor 3 to 5 by storing quantized coefficients in memory, instead of unquantized coefficients.

The overall conclusion is that wavelet transforms may replace the DCT for intra frame coding and can be combined with existing techniques. There are no extra computational and memory requirements and image quality is such that we recommend the use of this transform for intra frame coding in
7.2 Recommendations and remarks

7.2.1 Recommendations

- performance of the bypass procedure, as discussed in chapter 5, should be verified more thoroughly. Zero count and non-zero count were compared to the size of the RL/VL encoded data. These comparisons give the indication that the results are correct. Nevertheless, absolute certainty cannot be guaranteed until images are reconstructed from compressed data.

- LL-subband components represent the DC-values of the image. These values tend to vary quite slowly across the picture. Therefore, it may be rewarding to apply lossless compression to LL-subband components by means of a simple differential coding scheme and run-length coding. We expect the gain of such a coding scheme to be relatively better for coarsely quantized images because most of the bit-budget is spent on LL-subband components.

- Future research should involve

  - susceptibility to noise. A small amount of redundancy in the LL-subband, may prevent total block-loss from being very disturbing.
  
  - the performance of (H.263) DCT-based motion estimation and compensation in combination with wavelet transformed intra frames.
  
  - improved quantization and RL/VL encoding models. Research emphasis should be in these two areas instead of the actual wavelet transform itself [13].

  - video sequence coding using 3-dimensional wavelet transforms. Frames are not only transformed horizontally and vertically, but the transform is also applied to a number of subsequent video frames (i.e. the time-axis). This technique is used in 3D-SPIHT [14]. This option becomes more attractive if the proposed memory reduction schemes (chapter 6) are used for video sequences. This combination might be a replacement for very complex motion estimation/compensation. SPIHT without ME/MC performs equally well as MPEG-2 [14]. This performance was achieved for MPEG-2 resolution and frame rate (25/30 fps). Smaller frames and lower frame rates induce less correlation between the coefficients of each wavelet transformed frame and also between consecutive frames. Therefore we expect this method to perform relatively worse for video sequences at QCIF resolution and a refresh rate of 10 fps.
7.2 Recommendations and remarks

7.2.2 Remarks

- The number of operations required for the TT-transform, as described in chapters 3 and 4, can be slightly reduced because of the simple fact that

\[
3s(n) = s(n) + 2s(n) \quad (7.1)
\]
\[
22s(n) = 16s(n) + 4s(n) + 2s(n) \quad (7.2)
\]
but also
\[
22s(n) = 8 \cdot 3s(n) - 2s(n) \quad (7.3)
\]

Previous considerations on computational complexity are based on representation 7.2 instead of 7.3. The latter representation saves one addition/subtraction.

- Most coding algorithms that involve wavelet transforms, like CREW, embedded zerotree wavelet (EZW) [12] and set partitioning in hierarchical trees (SPIHT) [14],[15], use embedded ordering of subbands or even bitplanes. This allows progressive image coding. The pipelined nature of our transformation process with a small local memory (chapter 4) and restrictions on memory size and access bandwidth disallow embedded ordering.
Bibliography


[6] Phideo: *VLSI architecture synthesis for bit-parallel, micro programmed digital signal processors*


Appendix A

Bypass procedure function description

This appendix consists of two parts: a function description of the bypass procedure and some additional remarks.
### A.1 Functions

#### Bypass

**Function:** Bypass  
**Returns:** Number of bytes (int) in compressed frame  
**Input:**  
- `img`  
- `ComponentType`  
- `Scaling Value`  
- `DecompLevel`  
**Changes:** Structure `img`  
**Remarks:** Two files are written:  
- `raw.out` (raw 32-bit binary data)  
- `vlc.out` (RLE/VLE logfile)

#### QuantMatrix

**Function:** QuantMatrix  
**Returns:** nothing  
**Input:**  
- `ComponentType`  
- `DecompLevel`  
- `StepSizes`  
- `StepSizeArray`  
**Changes:** `StepSizes`, `StepSizeArray`  
**Remarks:** Calculates \(2^{DecompLevel} \times 2^{DecompLevel}\) quantization matrix. `StepSizes`, suggested by De Queiroz et al. [9], and stores subband step sizes in array `StepSizeArray`. Only `StepSizeArray` is used in further calculations.

#### QuantModel

**Function:** QuantModel  
**Returns:** Step size (double) for position \((i,j)\) in quantization matrix `StepSizes`  
**Input:**  
- `i`  
- `j`  
- `m`  
- `Scaling Value`  
**Changes:** nothing  
**Remarks:** Returns \(\Delta_{ij}\), shown in formula (1) in [9].
Function: **AverageMatrix**  
**Returns:** nothing  
**Input:**  
- double* `StepSizes`  
- int `r`  
- int `DecompLevel`  
- int* `StepSizeArray`  
- double `ScalingValue`  
  
Changes: nothing  
Remarks: Recursively calculates averages of all subbands [9].

Function: **PrintMatrix**  
**Returns:** nothing  
**Input:**  
- double* `StepSizes`  
- int `DecompLevel`  
  
Changes: nothing  
Remarks: For testing purposes. Prints matrix of doubles in $2^{\text{DecompLevel}} \times 2^{\text{DecompLevel}}$ format.

Function: **ScanBlockQuant**  
**Returns:** nothing  
**Input:**  
- Img* `img`  
- Img* `quantimg`  
- int `ComponentType`  
- int `i`  
- int `j`  
- int `r`  
- int `DecompLevel`  
- int* `Block`  
- int* `StepSizeArray`  
  
Changes: `quantimg, Block`  
Remarks: Scans and quantizes a composed block of wavelet coefficients for position $(i,j)$ of the LL-subband of image `img`. Quantized results are written to the array `Block`, which should at least be of size $2^{\text{DecompLevel}} \times 2^{\text{DecompLevel}}$.  

<table>
<thead>
<tr>
<th>Function:</th>
<th>HoriScanQuant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns:</td>
<td>Pointer to next coefficient to be scanned</td>
</tr>
<tr>
<td>Input:</td>
<td>Img* <code>img</code> Pointer to transformed CREW image structure</td>
</tr>
<tr>
<td></td>
<td>Img* <code>quantimg</code> Pointer to quantized image structure</td>
</tr>
<tr>
<td></td>
<td>int <code>Address</code> Pixel address in <code>img</code></td>
</tr>
<tr>
<td></td>
<td>int <code>Level</code> Wavelet transform decomposition level</td>
</tr>
<tr>
<td></td>
<td>int* <code>Block</code> Pointer to array for composed block storage</td>
</tr>
<tr>
<td></td>
<td>int* <code>StepSizeArray</code> Pointer to short array of step sizes</td>
</tr>
<tr>
<td>Changes:</td>
<td><code>quantimg</code>, <code>Block</code></td>
</tr>
<tr>
<td>Remarks:</td>
<td>Performs horizontal scan and quantization of wavelet coefficients, starting at <code>Address</code> in <code>img</code>. Quantized results are stored at address <code>Block</code>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function:</th>
<th>VertScanQuant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns:</td>
<td>Pointer to next coefficient to be scanned</td>
</tr>
<tr>
<td>Input:</td>
<td>Img* <code>img</code> Pointer to transformed CREW image structure</td>
</tr>
<tr>
<td></td>
<td>Img* <code>quantimg</code> Pointer to quantized image structure</td>
</tr>
<tr>
<td></td>
<td>int <code>Address</code> Pixel address in <code>img</code></td>
</tr>
<tr>
<td></td>
<td>int <code>Level</code> Wavelet transform decomposition level</td>
</tr>
<tr>
<td></td>
<td>int* <code>Block</code> Pointer to array for composed block storage</td>
</tr>
<tr>
<td></td>
<td>int* <code>StepSizeArray</code> Pointer to short array of step sizes</td>
</tr>
<tr>
<td>Changes:</td>
<td><code>quantimg</code>, <code>Block</code></td>
</tr>
<tr>
<td>Remarks:</td>
<td>Performs vertical scan and quantization of wavelet coefficients, starting at <code>Address</code> in <code>img</code>. Quantized results are stored at address <code>Block</code>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function:</th>
<th>ZZScanQuant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns:</td>
<td>Pointer to next coefficient to be scanned</td>
</tr>
<tr>
<td>Input:</td>
<td>Img* <code>img</code> Pointer to transformed CREW image structure</td>
</tr>
<tr>
<td></td>
<td>Img* <code>quantimg</code> Pointer to quantized image structure</td>
</tr>
<tr>
<td></td>
<td>int <code>Address</code> Pixel address in <code>img</code></td>
</tr>
<tr>
<td></td>
<td>int <code>Level</code> Wavelet transform decomposition level</td>
</tr>
<tr>
<td></td>
<td>int* <code>Block</code> Pointer to array for composed block storage</td>
</tr>
<tr>
<td></td>
<td>int* <code>StepSizeArray</code> Pointer to short array of step sizes</td>
</tr>
<tr>
<td>Changes:</td>
<td><code>quantimg</code>, <code>Block</code></td>
</tr>
<tr>
<td>Remarks:</td>
<td>Performs ZZ-scan and quantization of wavelet coefficients, starting at <code>Address</code> in <code>img</code>. Quantized results are stored at address <code>Block</code>.</td>
</tr>
</tbody>
</table>
A.1 Functions

Function: PrintBlock
Returns: nothing
Input: int* Block  Pointer to Block
       int Size     Number of elements in Block
Changes: nothing
Remarks: Prints contents of Block if verbosity level $\geq 2$.

Function: BlockToFile
Returns: nothing
Input: int* Block  Pointer to Block
       int Size     Number of elements in Block
       FILE* fp     Pointer to file structure.
Changes: File fp.
Remarks: This procedure is used to append raw quantized image data to the file raw.out.

Function: ZeroCount
Returns: nothing
Input: int i       Horizontal coefficient position
       int j       Vertical coefficient position
Changes: global variables zeroes, nonzeroes and lastnonzero.
Remarks: For verification purposes. Counts number of zeroes, non-zeroes, and position of the last non-zero of composed block $(i,j)$.

Function: Erase3HighPass
Returns: nothing
Input: int* Block  Pointer to Block
       int Size     Number of elements in Block
Changes: Block
Remarks: Erases last 75% of the composed block: all high-pass coefficients. For testing purposes. Used with switch -noquant

Function: CodeCoeff
Returns: Number of bits (int) in compressed block
Input: int Mode     Frame mode. Use MODE_INTRA.
       int* qcoeff  Pointer to block to be encoded
       int block    Block number. Use 0 here.
       int ncoeffs  Number of coefficients in block
Changes: File tf (vlc.out)
Remarks: Function by Telenor Research and Development [8].
A.2 Remarks

- SS, SD, DS and DD are subband names in CREW for subbands LL, LH, HL and HH respectively.

- `#define qfactor 8`. Factor by which the quantization matrix coefficients are divided. Enables easier use of the scaling value. The (0,0)-coefficient of the quantization matrix is always set to zero, so DC-components are not quantized.

- StepSizeArray contains the average step sizes for all subbands [9].
  Order: [LL,HL(N),LH(N),HH(N),HL(N-1),LH(N-1),HH(N-1),...,
  ...,HL(1),LH(1),HH(1)].

- `Img* quantimg` is a local CREW image structure for storage of the quantized frame. This frame is copied to `Img* img` after quantization has finished.
Appendix B

Results: Plots
B.1 Grayscale pictures

B.1.1 The Lena $512 \times 512$ grayscale picture

Figure B.1: Comparison of PSNR to compression factor of 8-Bit $512 \times 512$ grayscale image $lena512.pgm$, for 3-level CREW, 3-level Bypass and DCT-JPEG transforms
Figure B.2: Comparison of PSNR to compression factor of 8-Bit 512x512 grayscale image "lena512.pgm", for 4-level CREW, 4-level Bypass and DCT-JPEG transforms

Figure B.3: Comparison of PSNR to compression factor of 8-Bit 512x512 grayscale image "lena512.pgm", for 5-level CREW, 5-level Bypass and DCT-JPEG transforms
Figure B.4: Comparison of PSNR to compression factor of 8-Bit 512×512 grayscale image *lena512.pgm*, for 3-level, 4-level and 5-level Bypass transforms.
B.1.2 The Lena QCIF grayscale picture

Figure B.5: Comparison of PSNR to compression factor of 8-Bit 176×144 (QCIF) grayscale image lenaqcif.pgm, for 3-level CREW, 3-level Bypass and DCT-JPEG transforms
B.1.3 The Eref 512×512 grayscale picture

![Graph showing PSNR to compression factor for different transforms.](image)

Figure B.6: Comparison of PSNR to compression factor of 8-Bit 512×512 grayscale image `eref512.pgm`, for 3-level CREW, 3-level Bypass and DCT-JPEG transforms
Figure B.7: Comparison of PSNR to compression factor of 8-Bit 512×512 grayscale image *eref512.pgm*, for 4-level CREW, 4-level Bypass and DCT-JPEG transforms.

Figure B.8: Comparison of PSNR to compression factor of 8-Bit 512×512 grayscale image *eref512.pgm*, for 5-level CREW, 5-level Bypass and DCT-JPEG transforms.
Figure B.9: Comparison of PSNR to compression factor of 8-Bit 512x512 grayscale image *eref512.pgm*, for 3-level, 4-level and 5-level Bypass transforms.
B.1.4 The Eref QCIF grayscale picture

Figure B.10: Comparison of PSNR to compression factor of 8-Bit 176x144 (QCIF) grayscale image *erefqcif.pgm*, for 3-level CREW, 3-level Bypass and DCT-JPEG transforms
B.2 Color pictures

B.2.1 The Lena 512×512 color picture

Figure B.11: Comparison of PSNR to compression factor of 24-Bit 512×512 color image lena512.ppm, for 3-level CREW, 3-level Bypass and DCT-JPEG transforms
B.2 Color pictures

Figure B.12: Comparison of PSNR to compression factor of 24-Bit 512x512 color image *lena512.ppm*, for 4-level CREW, 4-level Bypass and DCT-JPEG transforms

Figure B.13: Comparison of PSNR to compression factor of 24-Bit 512x512 color image *lena512.ppm*, for 5-level CREW, 5-level Bypass and DCT-JPEG transforms
Figure B.14: Comparison of PSNR to compression factor of 24-Bit 512x512 color image `lena512.ppm`, for 3-level, 4-level and 5-level Bypass transforms.
B.2 Color pictures

B.2.2 The Lena QCIF color picture

Figure B.15: Comparison of PSNR to compression factor of 24-Bit 176×144 (QCIF) color image lenaqcif.ppm, for 3-level CREW, 3-level Bypass and DCT-JPEG transforms
The Eref 512×512 color picture

B.16: Comparison of PSNR to compression factor of 24-Bit 512×512 image cref512.ppm, for 3-level CREW, 3-level Bypass and DCT-JPEG.
Figure B.17: Comparison of PSNR to compression factor of 24-Bit 512×512 color image *eref512.ppm*, for 4-level CREW, 4-level Bypass and DCT-JPEG transforms

Figure B.18: Comparison of PSNR to compression factor of 24-Bit 512×512 color image *eref512.ppm*, for 5-level CREW, 5-level Bypass and DCT-JPEG transforms
Figure B.19: Comparison of PSNR to compression factor of 24-Bit 512x512 color image `eref512.ppm`, for 3-level, 4-level and 5-level Bypass transforms.
B.2 Color pictures

B.2.4 The Eref QCIF color picture

Figure B.20: Comparison of PSNR to compression factor of 24-Bit 176×144 (QCIF) color image `erefqcif.ppm`, for 3-level CREW, 3-level Bypass and DCT-JPEG transforms
Appendix C

Results: Images
C.1 The Lena picture

C.1.1 The Lena 512×512 picture

Figure C.1: Comparison of three 8-Bit 512×512 grayscale images: the original \textit{lena512.pgm}, Bypass and DCT-JPEG compressed
C.1.2 The Lena QCIF picture

![Comparison of three 8-Bit 176x144 (QCIF) grayscale images: the original lenaqcif.pgm, Bypass and DCT-JPEG compressed](image)

(a) Original  
(b) 3-Level Bypass, 1:7.87, PSNR=30.18dB

(c) DCT-JPEG quality 40, 1:7.89, PSNR=28.93dB

Figure C.2: Comparison of three 8-Bit 176x144 (QCIF) grayscale images: the original lenaqcif.pgm, Bypass and DCT-JPEG compressed
C.2 The Eref picture

C.2.1 The Eref 512\times 512 picture

Figure C.3: Comparison of three 8-Bit 512\times 512 grayscale images: the original $eref512.pgm$, Bypass and DCT-JPEG compressed
C.2.2 The Eref QCIF picture

Figure C.4: Comparison of three 8-Bit 176×144 (QCIF) grayscale images: the original `erefqcif.pgm`, Bypass and DCT-JPEG compressed