Generic Algorithms for Motion Compensation and Transformation

Henryk Richter[a], Benno Stabernack[b] and Erika Müller[a]

[a]University of Rostock, Institute of Communications Engineering,
R.-Wagner-Str. 31, D-18119 Rostock, Germany
[b]Fraunhofer-Institute for Telecommunications – Heinrich-Hertz-Institut
Einsteinufer 37, D-10587 Berlin, Germany

ABSTRACT

In this paper, we propose algorithms that map the low-level motion compensation and transformation functions of MPEG-1/2, H.263/MPEG-4 ASP and H.264/MPEG-4 AVC video codecs onto common workflows. This way, a single discrete implementation of luma prediction, chroma prediction and residual transform stages is sufficient for all covered video coding standards.
The proposed luma prediction is based on 4x4 blocks to cover the H.264 specifications as well as the elder standards. The design consists of a singular four stage pipeline for two block interpolation and two block averaging stages. Targeted for hardware implementation, a strictly linear execution is provided, avoiding branch operations. The algorithmic behavior is entirely dictated by the contents of the parameter ROM.
Since chrominance prediction must cover blocks as small as 2x2 pixels, a distinct operation is proposed for chroma. The bilinear operation scheme in H.264 is able to carry out the operations for the elder standards with minor changes only.
In H.264, the classic 8x8 DCT transformation was replaced by a simplified 4x4 integer transform, based on a heavily quantized DCT scheme. By modifications of a well-known multiplier-adder-based scheme, a generalized transformation stage can be derived.

Keywords: Multistandard Decoder, Motion Compensation, MPEG-2, MPEG-4, H.264, iDCT

1. INTRODUCTION

Current multistandard video decompressor architectures are focused on software-driven handling of higher level management functions in order to support the bitstream semantics at the necessary flexibility level. Since the higher level management functions require only a limited amount of computations, processor-based approaches provide a good trade-off between chip-area, power consumption, performance and flexibility in this respect.[1] Most of the computational complexity in a video decoder can be attributed to the image processing operations.[2] Consequently, optimization is focused on improving the behavior of the motion compensation, residual transform and post processing stages.

In a multistandard capable video decompressor, all operations have to be carried out with the distinct accuracy requirements of the respective standard. ISO MPEG-1 / MPEG-2 Video and ITU-T H.263 require an accuracy of $\frac{1}{2}$ pel, while for the later standards MPEG-4 Part 2 Visual and H.264/MPEG-4 Part 10 AVC an accuracy of $\frac{1}{4}$ pel is mandatory. Another improvement in the evolution of the coding standards was the interpolation filter response using multi-tap Wiener filters in the $\frac{1}{4}$ pel motion compensation instead of the simpler bi-linear interpolation scheme. The tradeoff for higher order interpolation is an increased complexity in the image processing operations.

The lower level motion compensated prediction and residual transform algorithms are preferably implemented in hardware to reduce system clock and power consumption, respectively. One solution in traditional hardware assisted approaches is to provide specialized functional units for each algorithm. That approach guarantees optimal power consumption and runtime performance for all individually optimized algorithms. On the other hand, distinct hardware units require an additional amount of gates for functionality that possibly can be shared among the supported algorithms.

Further author information: (Send correspondence to H.R.)
H.R.: E-mail: henryk.richter@comlab.uni-rostock.de, Telephone: +49 (0)381 498 7303
B.S.: E-mail: benno.stabernack@hhi.fraunhofer.de, Telephone: +49 (0)30 31002 661
E.M.: E-mail: erika.mueller@uni-rostock.de, Telephone: +49 (0)381 4987300

Another common approach consists of software driven workflows with appropriate instruction set extensions for run-time acceleration.[3–6] The software driven workflows are very attractive in terms of flexibility. Each required algorithm can be provided as individually optimized software sniplet and extensions for future application areas are easily feasible.

The method proposed in the following sections consists of a generic algorithm set that allows a single hardware implementation to accurately execute all required algorithms.

2. GENERIC LUMA MOTION INTERPOLATION ALGORITHM

2.1 Brief review of motion interpolation algorithms in current standards

This section describes the discrete operations to be carried out in the respective standards to perform the interpolation within the fractional pel motion compensation. The basic operation within motion compensation is to fetch a rectangular block of pels from the active reference frame. The block is addressed by the integer part of the motion vector plus a number of extra pixels left and above the initial position, depending on the requirements of the following interpolation. The fractional part of the motion vector (usually in $\frac{1}{2}$ or $\frac{1}{4}$ pel resolution) dictates the interpolation algorithm leading to the desired predictor block.

In the elder standards MPEG-1, MPEG-2, H.263 and MPEG-4 part 2 a bi-linear interpolation in half pel accuracy takes place.[7] To obtain fractional pel motion compensated blocks, a simple averaging between pels adjacent to the desired position is performed. With H.263, the rounding control bit was introduced in order to reduce the bias of the rounding operation.[8–10]

Optionally, MPEG-4 part 2 Advanced Simple Profile (ASP) provides an interpolation method with quarter pel accuracy.[11,12] In that quarter pel interpolation mode, the half pel positions are interpolated using a symmetric 8 tap FIR filter with the coefficients $Co1[0,\ldots,7] = [-8, 24, -48, 160, 160, -48, 24, -8]$ to a total denominator of 256. The interpolation is performed in discrete steps. First, the horizontal quarter pel position is computed and rounded/clipped to 8 bit range. Afterwards, these horizontally interpolated pels are used for vertical interpolation. In order to maintain the same memory bandwidth as with the half pel interpolation mode, the extra predictors required for the FIR filter are obtained by block boundary mirroring. On the left and upper side of the predictor block*, three pels are mirrored. One the lower and right block boundary, two pels are extended by mirroring.[13]

The standard H.264 / MPEG-4 part 10 Advanced Video Coding (AVC) uses a quarter pel interpolation scheme by default.[14] Compared to MPEG-4 ASP the impulse response length of the FIR filter was reduced to 6 taps with the coefficients $Co2[0,\ldots,5] = [1, -5, 20, 20, -5, 1]$. Since no block boundary mirroring is performed in H.264, the interpolation requires up to 81 source pels for interpolation of a single 4x4 block. Another difference to MPEG-4 ASP can be attributed to the interpolation accuracy. In case of bi-directional half pel interpolation, rounding and clipping operations are carried out after both directions are computed, while maintaining the full accuracy. In the last step, quarter pel positions are obtained by averaging half pel positions.

2.2 General approach

As outlined in the previous section, the algorithms relevant for the scope of this paper feature a number of different properties. A generic data flow covering all relevant data paths is depicted in fig. 1. To ensure bit-accurate results, the rounding and clipping operations have to be carried out in the mandatory order. The direct approach to a generic motion interpolation scheme as shown in fig. 1 would introduce a lot of conditional execution paths. Since the input blocks are required in multiple stages within the hypothetical structure, a general pipeline structure with in-place operations is not feasible using the depicted approach. It is clearly conceivable that this hypothetical structure would not be appropriate for an efficient implementation.

The algorithm proposed in this paper is tailored to avoid conditional execution. Instead, a pipeline structure approach is taken which dictates the behavior solely by parameter ROM tables. Most of the table addresses can be set up once for each decoded frame. Only the current fractional pel position and the MPEG-4 block boundary mirroring condition are used for the parameterization within the scope of a single block. For this algorithm it is assumed that all motion vectors are being scaled to quarter pel resolution.

*either 9x9 or 17x17 pels in 8x8 or 16x16 motion compensation, respectively

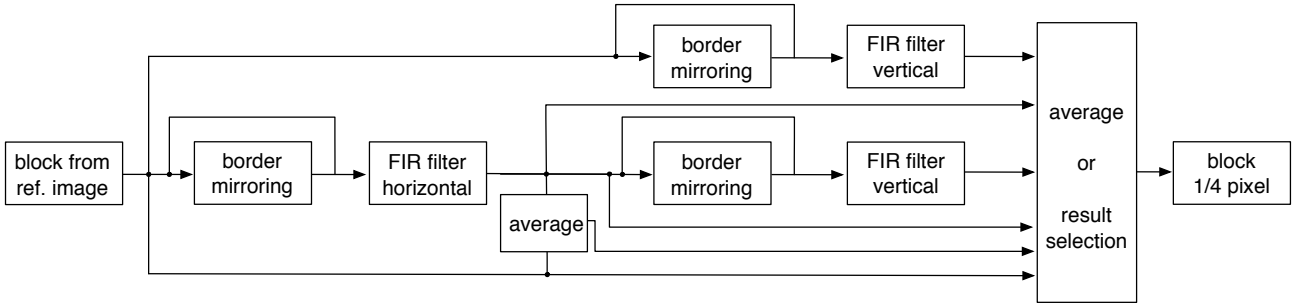

Figure 1. *Generic data flow graph of quarter pel interpolation respecting the algorithms of MPEG-1/-2, MPEG-4 ASP und H.264, active path depends on current algorithm and desired sub-pel*

Starting with the requirements of H.264 in terms of minimal allowed block size, the algorithm is tailored for block sizes of 4x4. In MPEG-4 quarter pel mode, an 8 tap FIR filter needs to be executed. Therefore the input block for interpolating a 4x4 block needs 7 extra pixels in each dimension, hence $\mathbf{X_{r11,11}}$. The basic components of the bi-directional interpolation in all considered standards consist of the following building blocks:

- horizontal interpolation FIR filter
- vertical interpolation FIR filter
- clipping and rounding of filter outputs
- position dependent averaging

The proposed strategy for a unified solution to the interpolation workflow is a four stage pipelined structure. The main property of the proposed algorithm is the calculation of all half pel positions so that the desired full-, half- or quarter pel positions are available in the final stage. That concept is outlined in fig. 2.

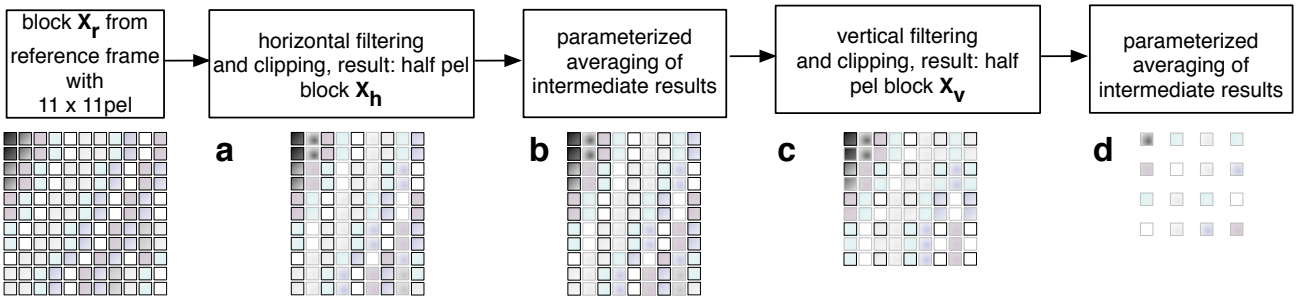

Figure 2. *Outline of proposed generic quarter pel motion interpolation algorithm as a four stage structure for MPEG-1/-2, MPEG-4 ASP und H.264*

The input block $\mathbf{X_{r11,11}}$ is (**a**) horizontally filtered and optionally clipped to obtain block $\mathbf{X_{h9,11}}$, consisting of four filtered columns and five non-filtered columns. After interpolation, an optional averaging of intermediate results (**b**) takes place. In stage (**c**), the vertical interpolation is performed with the resulting block $\mathbf{X_{v9,9}}$. The final stage (**d**) averages a pair of input pixels into the final predicted block $\hat{\mathbf{X}}_{\mathbf{4,4}}$. All four stages feature a static operation flow. The desired algorithm behavior according to the respective compression standard is solely achieved by ROM parameters.

2.3 Horizontal interpolation stage (a)

The horizontal filtering uses 121 input pels to interpolate half pel positions from horizontally neighboring raster pels. The result consists of four half pel positions per line. For each output pel, 8 coefficients are applied from the coefficient matrix. The coefficient memory $\mathbf{Co3}\,[\tau, \sigma_h, \xi, k]$ contains a distinct set of entries for each pixel ξ to cover the border coefficient mirroring (instead of border pel mirroring) in order to support MPEG-4 ASP in quarter pel mode. This step ensures linear

input data fetching for every supported standard. Besides the position dependency ξ of each filtered pixel and the coefficient index k, the interpolation operation τ with the special condition σ_h is included in the definition.

Since MPEG-4 ASP demands rounding of interpolation results directly after the summation, an optional rounding r_h with clipping on parametric limits γ_{hmin} and γ_{hmax} was integrated into the scheme. The pel $\mathbf{X_r}[0,0]$ is defined as the position addressed by the integer part of the motion vector within the current reference frame.

$$\begin{aligned} \mathbf{X_h}[2i+1,j] &= Clip3\Big(\gamma_{hmin},\gamma_{hmax},\big(\sum_{k=0}^{7}\mathbf{Co3}[\tau,\sigma_h,i,k]\cdot\mathbf{X_r}[i-3+k,j]\big)+r_h[\tau]\Big)\&\ V_h[\tau] \\ &\quad \text{with } i=0,1,2,3\ ;\ j=-3,-2,\ldots,7 \qquad (1) \\ \mathbf{X_h}[2i,j] &= 256\cdot\mathbf{X_r}[i,j] \\ &\quad \text{with } i=0,1,2,3,4\ ;\ j=-3,-2,\ldots,7 \qquad (2) \end{aligned}$$

After the operation outlined above, the output block contains four half pel positions and five full pel positions per line. The whole resulting block is 9 columns and 11 lines big. The entries are integer valued with 32 bit resolution. The additional parameter V_h for bit-wise AND operation allows the removal of lower-valued bits in case of rounding. This way, a rounding operation can be performed while maintaining the same dynamic range for all output pels.

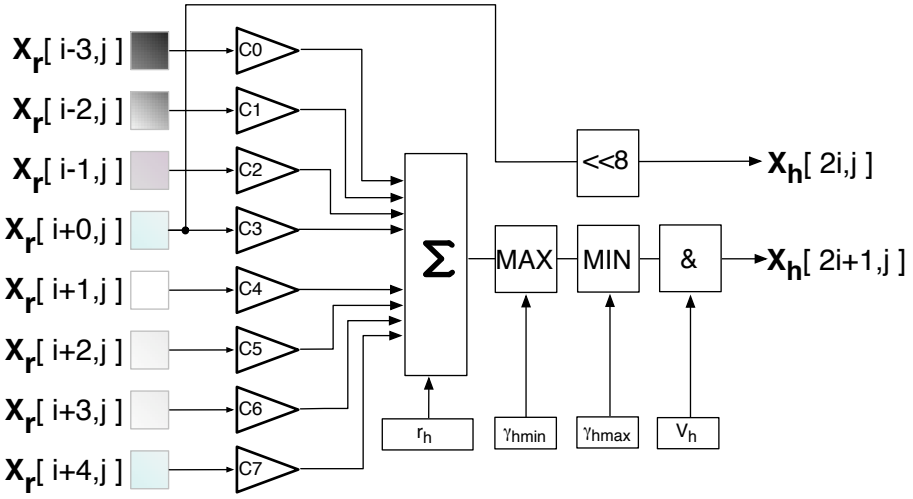

Figure 3. *Per pixel scheme for the generic horizontal interpolation; half pel positions are gained by an 8 tap FIR filter, full pel positions are normalized to factor 256; introduction of parameters $r_h[\tau],\gamma_{hmin},\gamma_{hmax},V_h[\tau]$ and coefficients C0. . . C7 from the current set in $\mathbf{Co3}[\tau,\sigma_h,i,k]$*

Figure 3 graphically demonstrates the algorithm for a single pixel position. All intermediate values are normalized to eight binary digits of fractional accuracy. This is accomplished by an 8 bit shift for the full pel values $\mathbf{X_h}[2i,j]$ and by scaled coefficients in $\mathbf{Co3}\,[\tau,\sigma_h,\xi,k]$. The advantage of this normalization is a uniform backward scaling and rounding in the vertical filtering stage, regardless whether the data was uni- or bi-directionally filtered. That step is especially important for H.264, where different rounding is applied in uni- and bi-directional cases, normatively.

The parameters $\tau, V_h[\tau], r_h[\tau], \gamma_{hmin}, \gamma_{hmax}$ are set up globally for each reconstructed frame. The only per-block local variable besides the reference block address is the coefficient set σ_h.

2.4 Intermediate averaging stage (b)

The stage **b** was mainly introduced to cover the requirements of MPEG-4 in quarter pel mode. Since the input values for the vertical filtering in MPEG-4 are already averaged horizontally filtered pels, this stage ensures correct pixel values. Dependent on the fractional part of the motion vector d_x, four cases of averaging per filter method τ need to be covered. The paired index field $\mathbf{idx_{b\ 3,4,2}}$ is per picture statically dependent on the filter method τ. The per block dynamic dependency is the fractional part d_x of the motion vector.

The indices pair from $\mathbf{idx_b}$ specifies tho horizontal offsets between 0 and 2 to address entries from $\mathbf{X_h}[n,j]$ to $\mathbf{X_h}[n+2,j]$ for averaging. The averaging follows the scheme $c=(a+b+r_b)>>1$. Including all parameters, the averaging of the horizontal filtered block $\mathbf{X_h}$ is as follows:

$$\begin{aligned}\mathbf{X_h}[i,j] &= \Big(\big(\mathbf{X_h}\big[i+\mathbf{idx_b}[\tau,d_x,0],j\big]+\mathbf{X_h}\big[i+\mathbf{idx_b}[\tau,d_x,1]j\big]+r_b[\tau]\big)>>1\Big)\&\ V_b[\tau]\\ &\quad \text{with } i=0,1,\ldots,7\ ;\ j=-3,-2,\ldots,7 \end{aligned} \tag{3}$$

The averaging stage can be effectively disabled by setting $\mathbf{idx_b}[\tau,d_x,0]=\mathbf{idx_b}[\tau,d_x,1]=0$. This way, the averaging is performed on the same input pels, so that for every $r_b[\tau]\leq 1$ the original value in $\mathbf{X_h}$ is retained. Another application for this stage is the half pel interpolation of MPEG-1 and MPEG-2, since two neighboring full pel positions can be addressed with the offset range 0..2. In analogy to stage **a**, an accuracy limiter $V_b[\tau]$ is available.

2.5 Vertical interpolation stage (c)

The vertical interpolation is performed in analogy to stage **a** using an 8 tap FIR filter. The inputs to the vertical stage are the contents of the block $\mathbf{X_h}$. All nine columns of the input matrix serve as input to an individual filter run. After summation, the final data word size of unsigned 8 bit is obtained by appropriate downscaling. Since every standard covered by our proposal uses a downscaling to 8 bit after the vertical stage and the data ranges in $\mathbf{Co3}[\tau,\sigma_v,i,k]$ are matched to the multiplier 256, special clipping constants are not required in stage **c**. The rounding control, mandatory to H.263 and MPEG-4 ASP is applied to the rounding parameter $r_v[\tau]$. The coefficient buffer $\mathbf{Co3}\,[\tau,\sigma_v,\xi,k]$ is the same as for the horizontal stage. Only the special case σ_v can differ. The filtering algorithm can be expressed as follows:

$$\begin{aligned}\mathbf{X_v}[i,2j+1] &= Clip\Big(\big(\sum_{k=0}^{7}\mathbf{Co3}[\tau,\sigma_v,j,k]\cdot\mathbf{X_h}[i,j-3+k]\big)+r_v[\tau]\Big)>>16\\ &\quad \text{mit } i=0,1,\ldots,8\ ;\ j=0,1,2,3 \end{aligned} \tag{4}$$

$$\begin{aligned}\mathbf{X_v}[i,2j] &= Clip\big((\mathbf{X_h}[i,j]+128)>>8\big)\\ &\quad \text{mit } i=0,1,\ldots 8\ ;\ j=0,1,2,3,4 \end{aligned} \tag{5}$$

By normalizing all entries in $\mathbf{X_h}$ to the factor 256, all four half pel variants[†] can be downscaled by the same factor without negative impact of the rounding constants 128 and $r_v[\tau]$, respectively. For all non filtered values with a data range extension by factor 256, a rounding constant of <256 vanishes by the integer shift operation.

The same principle applies for rounding of exclusively vertically filtered signal parts. By scaling the original raster pels by factor 256, the rounding parameter $r_v[\tau]$ and the down scaling range of 16 is the same for uni- and bi-directional filtered pels (fig. 4). Additional parameters are not needed. The result of the vertical filtering stage is the matrix $\mathbf{X_v}$ with 9x9 entries in unsigned 8 bit resolution.

2.6 Final averaging stage (d)

Starting with the requirements of H.264, the last step in the processing chain is the averaging of two half-pel resolution predictors. These predictors can consist of interpolated sub-pels, raster pels or a combination of both. In H.264, there are a number of sub-pel positions with diagonal averaging. Hence, the index selector in stage **d** uses two-dimensional coordinates within the source block $\mathbf{X_v}$. In similar fashion to stage **b**, the offset pair from the coordinate table is applicable to all 16 output pels of the motion interpolated block. The coordinate index field $\mathbf{idx_d}[\tau,d_x,d_y,\rho]$ depends on both fractional components d_x,d_y of the motion vector on a per-block base and is dimensioned as $\mathbf{idx_{d\ 3,4,4,4}}$. For each motion interpolation algorithm τ with a motion vector resolution of $\frac{1}{4}$ pel there are sixteen distinct cases with 2 two-dimensional indices ρ on the input matrix $\mathbf{X_v}$ have to be covered. In analogy to stage **b**, the averaging rule is of the type $c=(a+b+r_d)>>1$, executed on all 16 output pels.

$$\begin{aligned}\hat{\mathbf{s}}[i,j] &= \Big(\big(\ \mathbf{X_v}\big[i+\mathbf{idx_d}[\tau,d_x,d_y,0],j+\mathbf{idx_d}[\tau,d_x,d_y,1]\big]\\ &\quad +\ \mathbf{X_v}\big[i+\mathbf{idx_d}[\tau,d_x,d_y,2],j+\mathbf{idx_d}[\tau,d_x,d_y,3]\big]+r_d[\tau]\big)>>1\Big)\\ &\quad \text{mit } i=0,1,2,3\ ;\ j=0,1,2,3 \end{aligned} \tag{6}$$

[†] raster pel, horizontal, vertical, diagonal

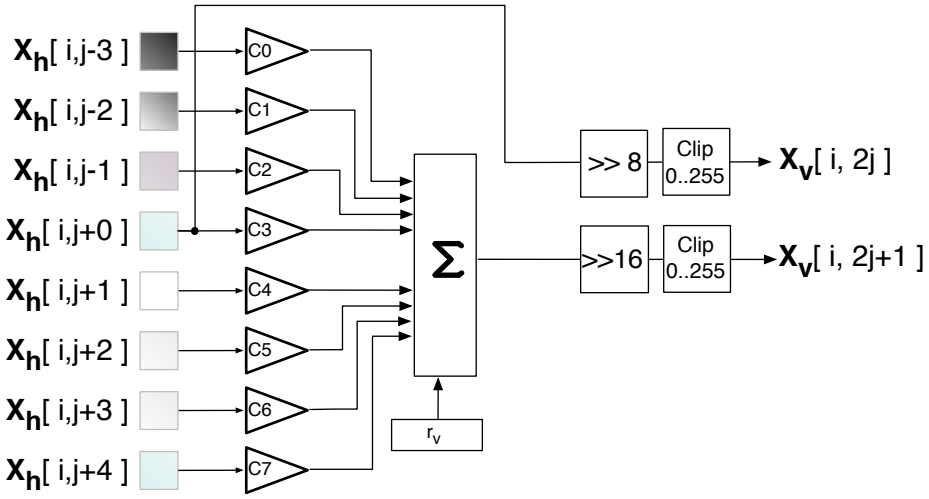

Figure 4. *Per pixel scheme for the generic vertical interpolation; half pel positions* $\mathbf{X_v}[\mathbf{i}, \mathbf{2j}]$ *are gained by an 8 tap FIR filter, scaled back to 8 bit range by 16 bit and clipped to range* $[0 \ldots 255]$*; full and uni-directional filtered pels* $\mathbf{X_v}[\mathbf{i}, \mathbf{2j}+\mathbf{1}]$ *are obtained by scaling 8 bit back and clipping; introduction of parameters* $r_v[\tau]$ *and coefficients C0. . . C7 from the current set in* $\mathbf{Co3}[\tau, \sigma_v, j, k]$

The result $\hat{\mathbf{s}}[i, j]$ of stage **d** is the final motion compensated block $\hat{\mathbf{S}}$. In order to support the H.263 and MPEG-4 rounding control, the constant $r_d[\tau]$ was specified which is 1 for usual cases and 0 for active rounding control, respectively.

For sub-pels where averaging in stage **d** is not necessary, that step can be disabled by appropriate addressing. By setting $\mathbf{idx_d}[\tau, d_x, d_y, 0] = \mathbf{idx_d}[\tau, d_x, d_y, 2]$ and $\mathbf{idx_d}[\tau, d_x, d_y, 1] = \mathbf{idx_d}[\tau, d_x, d_y, 3]$ onto the same indices on matrix $\mathbf{X_v}$, the original contents are retained after the averaging operation as long as $r_d[\tau] \leq 1$.

2.7 Discussion

The generic motion interpolation algorithm requires a considerable amount of input pixel data. For interpolating to a 4x4 pel output block, a 11x11 input block is used. Most practical cases of sub-pel positions actually require only a fraction of that data. In a system concept, where the pixel data from the reference frame buffer is directly passed to the interpolation stage, that overhead is undesirable. However, modern architectures include internal cache memory to efficiently utilize the characteristics of DRAM in terms of burst transfers. Data transfers between the internal cache and the interpolation stage confine the overhead of our proposed algorithm to the internal bus while the DRAM accesses can be limited to the minimum required bandwidth. Hence, there is no additional load on off-chip circuitry.

The exemplary workflow mentioned the use of 32 bit integer variables. The actual dynamic range in stage **a**is 17 bit plus sign. Stage **b**increases the dynamic range by an additional bit to 18 bit plus sign. In stage **c**, the intermediate dynamic range before downscaling and clipping is 26 bit plus sign. Multiplier coefficients in $\mathbf{Co3}$ $[\tau, \sigma, \xi, k]$ are discrete 8 bit values plus sign. Since only a limited number of significant bits in the multiplier coefficients are present, the multipliers in stages **a** and **c** can be implemented as 5 bit units.

The appropriate parameters and sample testbed source code are available in.[15]

3. GENERIC CHROMA MOTION INTERPOLATION ALGORITHM

Traditionally, the chroma interpolation scheme is carried out with lower complexity than the luma counterpart. Since in H.264, the smallest block size in interpolation is 2x2 chroma pels, a distinct implementation is desirable. All standards covered in this paper use a bi-linear interpolation scheme for chroma. In H.264, the accuracy of the bi-linear interpolation is $\frac{1}{8}$ pel, while the elder standards use $\frac{1}{2}$ pel accuracy. Assuming that the chroma motion vector adjustment was carried out according to the respective standards, the interpolation algorithm in H.264 is able to carry out the operations for MPEG-1, MPEG-2, MPEG-4 ASP and H.263 based streams. The only additional conditions to be respected are the rounding control bit and the scaling of the half-pel motion vectors to $\frac{1}{8}$ pel.

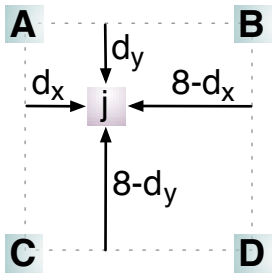

Figure 5. *Chroma interpolation with $\frac{1}{8}$ pel accuracy: estimate the sub-pel **j** from four surrounding raster pels*

The bi-linear interpolation with $\frac{1}{8}$ pel accuracy is illustrated in fig. 5. The pel A is addressed by the integer part of the motion vector. A weight factor w is calculated for each raster pel surrounding the desired sub pel j. These weights are constant across the predicted block $\hat{\mathbf{S}}$:

$$\begin{array}{rcccc} w_A & = & (8-d_x) & \cdot & (8-d_y) \\ w_B & = & d_x & \cdot & (8-d_y) \\ w_C & = & (8-d_x) & \cdot & d_y \\ w_D & = & d_x & \cdot & d_y \end{array} \tag{7}$$

The sub-pel j within the predicted block $\hat{\mathbf{S}}$ is obtained by position dependent weighting of the raster pels A, B, C and D as follows:

$$\hat{s}\,[\,x,y\,] = (w_A \cdot \mathsf{A} + w_B \cdot \mathsf{B} + w_C \cdot \mathsf{C} + w_D \cdot \mathsf{D} + r_c[\tau]) >> 6 \tag{8}$$

For H.264, the usual rounding constant of $r_c[0] = 32$ is applied. Since all weights are calculated to a total sum of 64 instead of providing distinct algorithms for uni- and bi-directional interpolation, the rounding control bit from H.263 and MPEG-4 ASP can be applied as a constant to r_c. Hence, the rounding constant in presence of the RC bit can be calculated as $r_c[1] = r_c[2] = 16 \cdot (2 - RC)$ in H.263 and MPEG-4 ASP quarter pel modes, respectively.

4. UNIFIED 8X8 DCT AND 4X4 H.264 TRANSFORM

The default de-correlation method in the classic image and video compression standards like JPEG and MPEG is based on variants of the discrete cosine transform. In the standards up to MPEG-4 part 2 it is carried out by a 8x8 point 2D DCT.

Fast algorithms For the computation of inverse cosine transform at a block size 8x8, 64 multiplications and additions are required when applying the direct method. In real-time applications this complexity is undesirable. As consequence, a number of fast methods have been proposed. These fast methods can be categorized as direct and indirect approaches.

The indirect methods are based on integration of the DCT in existing algorithms of computing the Fast Fourier Transform (FFT) or Hartley Transform.[16–23] Direct methods include the factorization of the DCT[24–28] or recursive workflows.[29–32] Most of the well-known algorithms rely on the separable nature of the DCT and consequently perform the row and column transform steps in distinct computation passes.

Duhamel[33] predicted the minimal number of multiplications for computing a 8 point inverse DCT with 11. The fastest known algorithm by Loeffler et al.[34] requires 11 multiplications and 29 additions.

In analogy to the graphic interpretation of the classic FFT[35,36] , most of the optimized DCT algorithms are visualized in form of butterfly diagrams. It is notable that in contrast to the regular structure of the classic FFT, fast DCT variants often utilize irregular data paths. In hardware applications, these introduce additional complexity in computation units and data paths.[37]

Chen et al.[24] proposed a factorization of an N point DCT into two N/2 transformation matrices with an attached butterfly. By application of the first factorization step, the Chen algorithm reduces the complexity of a single dimension 8 point DCT to $N^2/2$=32 operations. Consecutive decompositions of the 4 point DCTs allow the reduction to 16 multiplications using the Chen algorithm. In fig. 6, a single decomposition is shown.

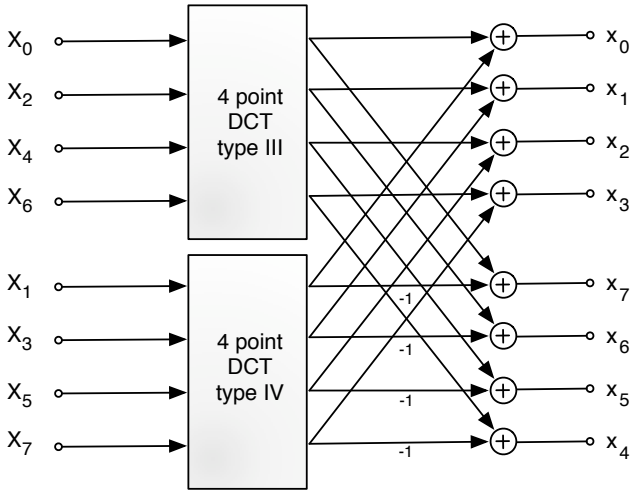

Figure 6. *Factorization of the inverse 8 point DCT after Chen into two 4 point transformations plus a butterfly operation in the output stage*[38]

Extension of the Chen algorithm for H.264 A major advantage of the Chen algorithm[24] is its regular structure. In matrix-vector notation the algorithm can be expressed as follows:[39,40]

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} A & B & A & C \\ A & C & -A & -B \\ A & -C & -A & B \\ A & -B & A & -C \end{bmatrix}\begin{bmatrix} X_0 \\ X_2 \\ X_4 \\ X_6 \end{bmatrix} + \frac{1}{2}\begin{bmatrix} D & E & F & G \\ E & -G & -D & -F \\ F & -D & G & E \\ G & -F & E & -D \end{bmatrix}\begin{bmatrix} X_1 \\ X_3 \\ X_5 \\ X_7 \end{bmatrix}$$

$$\begin{bmatrix} x_7 \\ x_6 \\ x_5 \\ x_4 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} A & B & A & C \\ A & C & -A & -B \\ A & -C & -A & B \\ A & -B & A & -C \end{bmatrix}\begin{bmatrix} X_0 \\ X_2 \\ X_4 \\ X_6 \end{bmatrix} - \frac{1}{2}\begin{bmatrix} D & E & F & G \\ E & -G & -D & -F \\ F & -D & G & E \\ G & -F & E & -D \end{bmatrix}\begin{bmatrix} X_1 \\ X_3 \\ X_5 \\ X_7 \end{bmatrix}$$

$$\text{with} \quad A = cos\frac{\pi}{4},\ B = cos\frac{\pi}{8},\ C = sin\frac{\pi}{8},\ D = cos\frac{\pi}{16},$$
$$E = cos\frac{3\pi}{16},\ F = sin\frac{3\pi}{16},\ G = sin\frac{\pi}{16}, \tag{9}$$

The core operation of this scheme consists of four parallel or successive multiplications per path, followed by an accumulation of the results. The compact operation logic of the Chen algorithm is very attractive for hardware applications,[37,40–42] even up to HDTV resolution.[43]

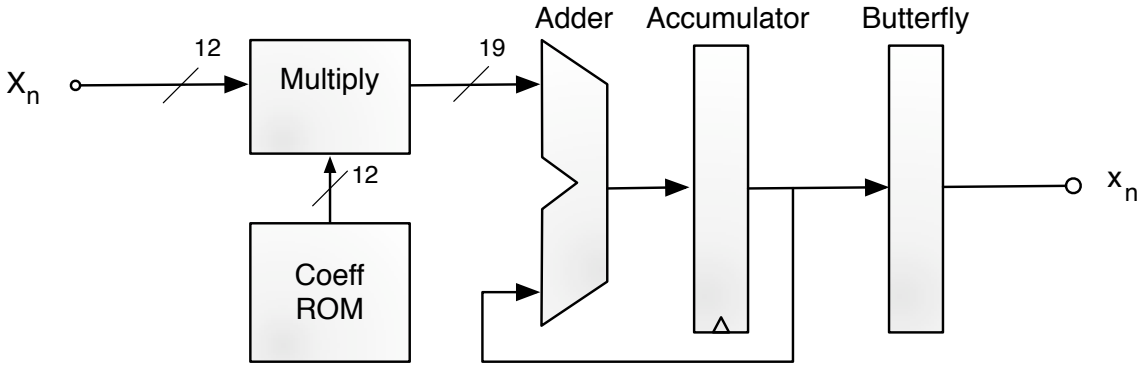

Figure 7. *Multiplier-adder structure of the inverse DCT core operation by using the Chen algorithm*[41]

By visualizing the inverse DCT in form of a multiplier-adder structure (fig. 7), it is plausible that a linear workflow can be applied on the input data[‡].

This property of the multiplier-adder scheme is very useful for the integration of the H.264 transform in existing inverse DCT designs. Since the integer transform in H.264 is a highly quantized form of a DCT-III,[44] the pattern of an inverse H.264 transform already matches the form of the left side in eq. 9. Respecting the scaling factor $\frac{1}{2}$ in front of the matrix, the four point inverse H.264 transformation can be expressed as follows:

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} A_0 & B_0 & A_0 & C_0 \\ A_0 & C_0 & -A_0 & -B_0 \\ A_0 & -C_0 & -A_0 & B_0 \\ A_0 & -B_0 & A_0 & -C_0 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix}$$
$$\text{with} \qquad A_0 = 2,\ B_0 = 2,\ C_0 = 1 \tag{10}$$

Based on the initial condition of a linear access to the coefficients in conjunction with the Chen DCT algorithm, the second matrix can be modified as well. One modification is the doubling of the coefficient memory to cover the demands of H.264. The other modification consists of a conditional butterfly stage. By introducing the factor η_τ, the butterfly operation can be applied for the DCT with $\eta_1 = \frac{1}{2}$. In the case of H.264, $\eta_0 = 0$ disables the butterfly in order to process two 4x4 blocks in one run.

To keep the output order of results $x_4 \dots x_7$ for DCT-based transformation steps intact, the order of coefficients in the right matrix is sorted accordingly for the H.264 transform. Therefore the H.264 coefficients are vertically mirrored. The whole computation algorithm with substituted generic coefficients $H_\tau, \dots, W_\tau$ in the right matrix was derived as follows:

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} A_\tau & B_\tau & A_\tau & C_\tau \\ A_\tau & C_\tau & -A_\tau & -B_\tau \\ A_\tau & -C_\tau & -A_\tau & B_\tau \\ A_\tau & -B_\tau & A_\tau & -C_\tau \end{bmatrix} \begin{bmatrix} X_0 \\ X_2 \\ X_4 \\ X_6 \end{bmatrix} + \eta_\tau \begin{bmatrix} H_\tau & I_\tau & J_\tau & K_\tau \\ L_\tau & M_\tau & N_\tau & O_\tau \\ P_\tau & Q_\tau & R_\tau & S_\tau \\ T_\tau & U_\tau & V_\tau & W_\tau \end{bmatrix} \begin{bmatrix} X_1 \\ X_3 \\ X_5 \\ X_7 \end{bmatrix}$$
$$\begin{bmatrix} x_7 \\ x_6 \\ x_5 \\ x_4 \end{bmatrix} = \eta_\tau \begin{bmatrix} A_\tau & B_\tau & A_\tau & C_\tau \\ A_\tau & C_\tau & -A_\tau & -B_\tau \\ A_\tau & -C_\tau & -A_\tau & B_\tau \\ A_\tau & -B_\tau & A_\tau & -C_\tau \end{bmatrix} \begin{bmatrix} X_0 \\ X_2 \\ X_4 \\ X_6 \end{bmatrix} - \frac{1}{2} \begin{bmatrix} H_\tau & I_\tau & J_\tau & K_\tau \\ L_\tau & M_\tau & N_\tau & O_\tau \\ P_\tau & Q_\tau & R_\tau & S_\tau \\ T_\tau & U_\tau & V_\tau & W_\tau \end{bmatrix} \begin{bmatrix} X_1 \\ X_3 \\ X_5 \\ X_7 \end{bmatrix} \tag{11}$$

The substitution of variables $D, \dots, G$ in eq. 9 for the 4 point DCT-IV results in the following coefficients:

$$\begin{array}{llllllll} H_1 = & cos\frac{\pi}{16}, & I_1 = & cos\frac{3\pi}{16}, & J_1 = & sin\frac{3\pi}{16}, & K_1 = & sin\frac{\pi}{16}, \\ L_1 = & cos\frac{3\pi}{16}, & M_1 = & -sin\frac{\pi}{16}, & N_1 = & -cos\frac{\pi}{16}, & O_1 = & -sin\frac{3\pi}{16}, \\ P_1 = & sin\frac{3\pi}{16}, & Q_1 = & -cos\frac{\pi}{16}, & R_1 = & sin\frac{\pi}{16}, & S_1 = & cos\frac{3\pi}{16}, \\ T_1 = & sin\frac{\pi}{16}, & U_1 = & -sin\frac{3\pi}{16}, & V_1 = & cos\frac{3\pi}{16}, & W_1 = & -cos\frac{\pi}{16}, \end{array} \tag{12}$$

In analogy, the H.264 coefficients are given by mapping of the variables A_0, B_0, C_0 to the substituted coefficients:

$$\begin{array}{llllllll} H_0 = & 2, & I_0 = & -2, & J_0 = & 2, & K_0 = & -1, \\ L_0 = & 2, & M_0 = & -1, & N_0 = & -2, & O_0 = & 2, \\ P_0 = & 2, & Q_0 = & 1, & R_0 = & -2, & S_0 = & -2, \\ T_0 = & 2, & U_0 = & 2, & V_0 = & 2, & W_0 = & 1, \end{array} \tag{13}$$

This way, the 8 point inverse DCT transformation can be re-used for transforming two blocks of four coefficients. In order to comply with the rules in IEEE1180, pure integer arithmetic requires 12 bit plus sign for the coefficients. Consequently, the multiplier of integer valued coefficients $A_1, \dots, W_1$ is 4096. The intermediate values in the first transformation

[‡]after properly dividing between pair and impair indices of x_n

step require 3 bit fractional accuracy. Therefore the input values X_n in the vertical transformation step require 15 bit total accuracy. After accumulation, the rounding, back scaling and clipping operations are applied. As scaling constant, a 9 bit shift operation is used.

In H.264, no fractional accuracy is required. Taking the 9 bit shift operation into account, a factor of 2048 for the coefficients $A_0, \ldots, W_0$ is appropriate. By this factor, all multiplicators stay in the 12 bit plus sign range used for the DCT. The complete structure of the proposed system for generic inverse transform is depicted in fig. 7. Sample test code and IEEE1180 verification results are available in.[15]

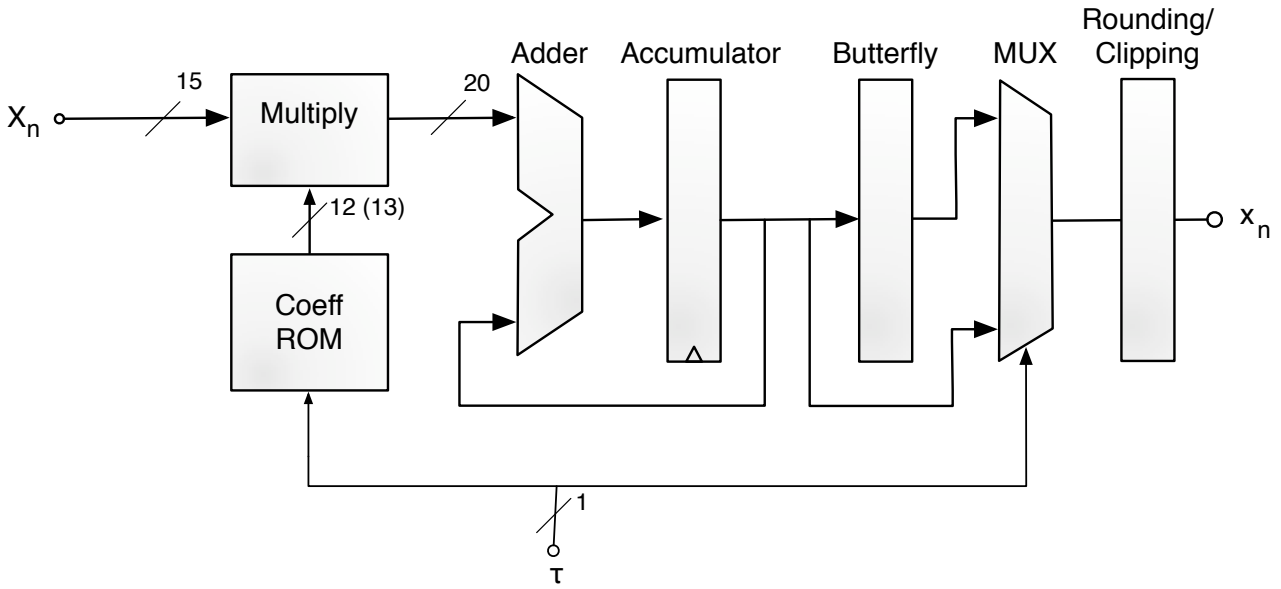

Figure 8. *Operation core of combined inverse DCT and inverse H.264 transform by extending the addressing of transform coefficient by one bit and type dependent deactivation of butterfly stage*

5. CONCLUSION

In this paper, algorithms were presented that map motion compensation and residual transform on common work-flows. It has been shown that – despite a significant number of differences – luma motion compensation can be unified on a generic processing model. Algorithm-specific branching was avoided. The algorithmic behavior of the common processing core is modified only by selecting the contents of fixed coefficient, clipping and addressing tables. Furthermore it has been shown that for chroma motion compensation, the H.264 algorithm can be slightly modified to accompany the needs of elder standards.

In system designs where the chip area of individual components plays a significant role, the proposed inverse transform structure might be a viable choice. The additional complexity of the generic transform compared to a single DCT stage can be kept minimal.

REFERENCES

1. B. Stabernack and H. Richter, "Media Processor Architectures for Mobile DVB-H Terminals," *Proceedings of GSPx 2005 Pervasive Signal Processing Conference, Santa Clara, U.S.A.* , Oct. 2005.
2. H. Richter and E. Müller, "Multistandard video decompression based on a uniform meta format stream," *Proceedings of Picture Coding Symposium (PCS' 2007), Lisbon, Portugal* , Nov. 2007.
3. N. Penisoara, "Optimizing inter-processor communication on freescale i.300-30 multi-core platform," *Proc. of GSPX 2005, Santa Clara, U.S.A.* , Oct. 2005.
4. Texas Instruments, "TMS320DM642 Technical Overview," 2002. http://www.ti.com,order #SPRU615.

5. ARC International, "How To Build Optimized Products With the ARCtangent-A4 User-Customizable Processor," 2002. http://www.arc.com/documentation/whitepapers/.
6. R. E. Gonzalez, "Xtensa — A configurable and extensible processor," *IEEE Micro* **20**(2), pp. 60–70, 2000.
7. J. L. Mitchell, W. B. Pennebaker, C. E. Fogg, and D. J. LeGall, *MPEG Video Compression Standard*, Chapman and Hall, New York, USA, 1 ed., 1997.
8. Y. Nakaya, "Avoindance of Rounding Error Accumulation in Motion Compensation with Half Pel Accuracy," *Proceedings of the 1998 IEICE General Conference* , pp. D–11–44, Mar. 1998. Japanese /w engl. translation.
9. Y. Nakaya, "Image decoding method." US Pat. 6,574,371, June 2001.
10. Y. Nakaya, "Avoindance of Rounding Error Accumulation in Motion Compensation with Half Pel Accuracy." US Pat. 6,868,185, Jul 2003.
11. U. Benzler and O. Werner, "Improving multiresolution motion compensating hybrid coding by drift reduction," *Picture Coding Symposium (PCS' 96), Melbourne, Australia* , Mar. 1996.
12. U. Benzler, "Performance evaluation of a reduced complexity implementation for quarter pel motion compensation," *ISO/IEC JTC1/SC29/WG11 MPEG-4: N3146* , Jan 1998.
13. ISO/IEC JTC1/SC29/WG11, "MPEG-4 Video VM 16.0." MPEG00/N3312, Noordwijkerhout, Netherlands, 2000.
14. Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, "ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC)." JVT-G050, 7th Meeting Pattaya, Thailand, Mar. 2003.
15. H. Richter, *Standardübergreifende Konzepte für blockbasierte Videodecodierung*. PhD thesis, Universität Rostock, Nov. 2006.
16. R. M. Haralick, "A storage efficient way to implement the discrete cosine transform," *IEEE Transactions on Computing* **C-25**, pp. 764–5, July 1976.
17. B. D. Tseng and W. C. Miller, "On computing the discrete cosine transform," *IEEE Transactions on Computers* **C-27**, pp. 966–8, Oct. 1978.
18. M. J. Narashimha and A. M. Peterson, "On the computation of the discrete cosine transform," *IEEE Transactions on Communications* **COM-26**, pp. 934–6, June 1978.
19. D. Hein and N. Ahmed, "On a real-time Walsh-Hadamard cosine transform image processor," *IEEE Transactions on Electromagnetic Compatibility* **EMC-20**, pp. 453–7, Aug. 1978.
20. H. W. Jones, D. N. Hein, and S. C. Knauer, "The Karhunen-Loeve, discrete cosine and related transforms via the Hadamard transform," *Proc. Int. Telemeter. Conf.* , pp. 87–98, Nov. 1978.
21. M. Vetterli and H. Nussbaumer, "Simple FFT and DCT algorithms with reduced number of operations," *Signal Processing* **6**, pp. 267–78, Aug. 1984.
22. H. S. Malvar, "Fast computation of discrete cosine transform through fast Hartley transform," *Electronic Letters* **22**, pp. 352–353, Mar. 1986.
23. H. S. Hou, "The fast Hartley transform algorithm," *IEEE Transactions on Computers* **C-36**, pp. 147–56, Feb. 1987.
24. W.-H. Chen, C. H. Smith, and S. C. Fralick, "A Fast Computational Algorithm for the Discrete Cosine Transform," *IEEE Transactions on Communications* **COM-25**, pp. 1004–9, Sept. 1977.
25. M. Ghanbari and D. E. Pearson, "Fast cosine transform implementation for television signals," *IEEE Proc* **129**, pp. 59–68, Feb. 1982.
26. Z. Wang, "Fast algorithms for the discrete W transform and for the discrete Fourier transform," *IEEE Trans. Acoust., Speech, Signal Processing* **ASSP-32**, pp. 803–16, Aug. 1984.
27. B. G. Lee, "A new algorithm to compute the discrete Cosine transform," *IEEE Trans. Acoust., Speech, Signal Processing* **ASSP-32**, pp. 1243–5, Dec. 1984.
28. N. Suehiro and M. Hatori, "Fast algorithms for the DFT and other sinusoidal transforms," *IEEE Trans. Acoust., Speech, Signal Processing* **ASSP-34**, pp. 642–4, June 1986.
29. H. S. Hou, "A Fast Recursive Algorithm for Computing the Discrete Cosine Transform," *IEEE Trans. Acoust., Speech, Signal Processing* **ASSP-35**, pp. 1455–61, Oct. 1987.
30. Z. Cvetković and M. V. Popović, "New Fast Recursive Algorithms for the Computation of Discrete Cosine and Sine Transforms," *IEEE Transactions on Signal Processing* **40**, pp. 2083–6, Aug. 1992.
31. P. Lee and F.-Y. Huang, "Restructured Recursive DCT and DST Algorithms," *IEEE Transactions on Signal Processing* **42**, pp. 1600–9, July 1994.

32. C. W. Kok, "Fast Algorithm for Computing the Discrete Cosine Transform," *IEEE Transactions on Signal Processing* **45**, pp. 757–60, Mar. 1997.
33. P. Duhamel and H. HMida, "New 2^n DCT Algorithms suitable for VLSI Implementation," *Proc. IEEE International conference on Acoustics, Speech and Signal Processing* **ICASSP-87**, pp. 1805–8, Apr. 1987.
34. C. Loeffler, A. Ligtenberg, and G. S. Moschytz, "Practical Fast 1D-DCT Algorithms with 11 multiplications," *Proc. IEEE International conference on Acoustics, Speech and Signal Processing* **ICASSP-89**, pp. 988–91, Apr. 1989.
35. J. Cooley and J. Tukey, "An Algorithm for the Machine Computation of Complex Fourier Series," *Math. Comp* **19**, pp. 291–301, 1965.
36. E. O. Brigham, *FFT, Schnelle Fourier-Transformation*, Oldenbourg, Mnchen, 6 ed., 1995.
37. C. Schneider, M. Kayss, T. Hollstein, and J. Deicke, "From Algorithms to Hardware Architectures: A Comparison of Regular and Irregular Structured IDCT Algorithms," *Design Automation and Test in Europe (DATE '98)* , pp. 186–90, 1998.
38. C.-M. Liu and W.-C. Lee, "A Unified Fast Algorithm for Cosine Modulated Filter Banks in Current Audio Coding Standards," *Journal of the Audio Engineering Society* **12**, pp. 1061–75, Dec. 1999.
39. S. Kim and W. Sung, "Fixed-Point Optimization Utility for C and C++ Based Digital Signal Processing Programs," *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing* **45**, pp. 1455–64, Nov 1998.
40. T. Xanthopoulos and A. P. Chandrakasan, "A Low-Power DCT Core Using Adaptive Bitwidth and Arithmetic Activity Exploiting Signal Correlations and Quantization," *IEEE Journal of Solid-State Circuits* **35**, pp. 740–50, May 2000.
41. S. Kim and W. Sung, "Fixed-Point Error Analysis and Word Length Optimization of 8×8 IDCT Architectures," *IEEE Transactions on Circuits and Systems for Video Technology* **8**, pp. 935–40, Dec 1998.
42. B. Scott, D. Johnson, and V. Akella, "Asynchronous 2D Discrete Cosine Transform Core Processor," *IEEE International Conference on Computer Design* , pp. 380–5, Oct. 1995.
43. J.-I. Guo and J.-C. Yen, "An Efficient IDCT Processor Design for HDTV Applications," *Journal of VLSI Signal Processing* **33**, pp. 147–55, 2003.
44. H. S. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, "Low-Complexity Transform and Quantization in H.264/AVC," *IEEE Transactions on Circuits and Systems for Video Technology* **13**, pp. 598–603, July 2003.