Multiplication and Error Free Implementation of H.264 like 4x4 DCT/Quan_IQuan/IDCT using Algebraic Integer Encoding

Mohammad Norouzi *Karim Mohammadi* *Mohammad Mahdy Azadfar*

Mechatronics Research Lab (MRL), Qazvin, Iran
Iran University of Science & Tech (IUST), Tehran, Iran
Iran Telecom Research Center(ITRC), Tehran, Iran

Summary

This paper presents a novel error-free (infinite-precision) algorithm for the fast implementation of H.264 like 4×4 DCT/Quan_IQuan/IDCT based on algebraic integer-encoding scheme.
This encoding technique eliminates the requirement to approximate the transform matrix elements by obtaining their exact representation. The proposed algorithm has regular structure and simulation results shows reducing computing complexity with enhanced quality simultaneously. Furthermore it is multiplication free and suitable for the high speed implementation with a fully pipelined systolic architecture.

Key words:
H.264, Video Coding, DCT, Algebraic Integer

1. Introduction

The recently approved digital video standard known as H.264 promises to be an excellent video format for use with a large range of applications. Real-time encoding/decoding is a main requirement for adoption of the standard to rich its goal. In the initial H.264 standard, which was completed in May 2003, to simplify the implementation it use an approximated integer 4×4 transform which helps reduce blocking and ringing artifacts. Also a scaling multiplication (part of transform) is integrated into the quantizer, reducing the total number of multiplication. But for each 4x4 transform 16 number of 16 bit multiplication is needed. And the transform isn't a real 4x4 DCT. [1]
This paper presents a novel error-free (infinite-perception) algorithm for the fast implementation of H.264 like 4x4 Transform/Quantization and their reverse action based on algebraic integer-encoding scheme which is suitable for the implementation with a fully pipelined systolic architecture. Software solutions for the DCT are widely used, but for high data rate application, VLSI hardware implementations are still preferred.
For completeness, the 4x4 Discrete Cosine Transform (DCT) operates on X, a block of 4x4 samples and creates Y, a 4x4 block of coefficients. The action of DCT (and its reverse, the IDCT) can be described in terms of a transform matrix A. The forward DCT of a 4x4 sample block is given by:

$$Y = AXA^T \quad (1)$$

And the reverse DCT (IDCT) by:

$$X = A^T YA \quad (2)$$

Where X is a 4x4 matrix of samples, Y is a matrix of coefficients and A is a 4x4 transform matrix.
The elements of A are:

$$A = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & c \end{bmatrix} \quad (3)$$

$$a = \frac{1}{2},\ b = \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right),\ c = \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right)$$

The matrix multiplication of (1) can be factorized to the following equivalent form:

$$Y = \left(CXC^T\right) \otimes E_f \quad (4)$$

Where:

$$C = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & d & -d & -1 \\ 1 & -1 & -1 & 1 \\ d & -1 & 1 & -d \end{bmatrix} \quad E_f = \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix}$$

$W = CXC^T$ is a 'core' 2D transform. E is a matrix of scaling factors and the symbol $\otimes$ indicates that each element of W is multiplied by the scaling factor in the same position in matrix E (scalar multiplication rather than matrix multiplication). The constants a and b are as before and d is c/b. In the H.264 to simplify the implementation of the transform, d is approximated by

0.5. In order to ensure that the transform reminds orthogonal, b also need to be modified so that: [1]

$$a_h = \frac{1}{2},\ b_h = \sqrt{\frac{2}{5}},\ d_h = \frac{1}{2}$$

Thus this transform is an approximated to the 4x4 DCT because of the changes to factors d and b, so the result of the H.264 transform will not be identical to the 4x4 DCT.

The post-scaling operation $\otimes E_f$ is absorbed into the quantization process but still 16 number 16-bit multiplications for each 4x4 block is needed. The post-scaling and quantization formulas are shown in Equations 5-7.

$$qbits = 15 + floor(QP/6) \quad (5)$$

$$|Z_{ij}| = (|W_{ij}|.MF + f) >> qbits \quad (6)$$

$$sign(Z_{ij}) = sign(W_{ij}) \quad (7)$$

Where QP is a quantization parameter that enables the encoder to accurately and flexibly control the trade-off between bit rate and quality. It can take any integer value from 0 up to 51. Z_{ij} is an element in the quantized transform coefficients matrix. MF is a multiplication factor that depends on $(m = QP \bmod 6)$ and the position (i, j) of the element in the matrix. The $>>$ indicates a binary shift right. In the reference model [13] software, f is $2^{qbits}/3$ for Intra blocks or $2^{qbits}/6$ for Inter blocks. [1]

2. Algebraic Integer Encoding

Algebraic integers are defined by real numbers that are roots of monic polynomials with integer coefficients [9]. As an example, let $\omega = e^{\frac{2\pi}{16}j}$ denote a primitive 16th root of unity over the ring of complex numbers. Then ω is a root of the equation $x^8 + 1 = 0$. If ω is adjoined to the rational numbers, then the associated ring of algebraic integers is denoted by $Z[\omega]$. The ring $Z[\omega]$ can be regarded as consisting of polynomials in ω of degree 7 with integer coefficients. The elements of $Z[\omega]$ are added and multiplied as polynomials, except that the rule $\omega^8 = -1$ is used in the product to reduce the degree of powers of ω to below 8. For an integer, M, $Z[\omega]_M$ is used to denote the elements of with coefficients between $-\frac{M}{2}$ and $\frac{M}{2}$. In summary, algebraic integers of an extension of degree n can be assumed to be of the form:

$$a_0\omega_0 + a_1\omega_1 + ... + a_{n-1}\omega_{n-1} \quad (8)$$

Where, $\{\omega_0, \omega_1, ..., \omega_{n-1}\}$ is called the algebraic integer basis and the coefficients a_i are integers.

The idea of using algebraic integers in DSP applications was first explored by Cozzens and Finkelstein [5]. The V. S. Dimitrov and G. A. Jullien group was the first which introduced algebraic integer coding to provide low complexity error-free computation of the DCT and IDCT. [11]

The elements of the transform matrix for the DCT/IDCT are real numbers of the form $\cos\frac{n\pi}{2N}$, where n is an integer. Rather than applying the classical procedure of using approximation to these elements, the algebraic integer encoding scheme processes number of this form using exact representation. There have been several previous publications on algebraic integer encoding schemes [3,4,9]. If we denote

$$z = 2\cos\left(\frac{\pi}{8}\right) = \sqrt{2+\sqrt{2}} \quad (9)$$

Then z is root of the polynomial $F(x) = x^4 - 4x^2 + 2$ and the elements of $F(x)$ have a polynomial form. Consider the polynomial expansion:

$$f(z) = \sum_{i=0}^{3} a_i z^i \quad (10)$$

Where a_i are integers. Definitely, z, that is $\sqrt{2+\sqrt{2}}$ corresponds to the following particular choice $a_i: (0,1,0,0)$

Therefore, we have an exact code for z. For the other elements consider these intermediate

$$a' = \sqrt{\frac{1}{2}}, b' = 2\cos(\frac{\pi}{8}), c' = 2\cos(\frac{3\pi}{8})$$

Then:

$$b = 2^{-1}a'b',\ c = 2^{-1}a'c' \quad (11)$$

With these intermediate definitions we can represent rest of elements exactly as combinations of four small integers a_i too. Table 1 provides the corresponding coefficients.

Table 1: the exact representation of the cosines participating in the 4 x 4 DCT

	a_0	a_1	a_2	a_3
a'	-1	0	0.5	0
b'	0	1	0	0
c'	0	-3	0	1
d	-3	0	1	0

3. Algorithm Derivation

It has been pointed out previously, the most critical problem for algebraic integer DCT and IDCT implementation is the correct choice of 1D and 2D algorithms. [3] The 2D algorithm proposed by Cho-Lee is good candidate for algebraic integer implementation. [4] The main reason is that it uses only one portion of 1D DCTs with remainder of the algorithm a traditional butterfly structure only. The proposed algorithm has butterfly structure too which fully uses the symmetry present in the transform matrix. It is error-free up until the final reconstruction step and needs no multiplication which is very suitable from VSLI view point.

Remind the core transform of equation (4):

$$W = CXC^T \tag{12}$$

Consider:

$$C = C_1 + dC_2 \tag{13}$$

Where:

$$C_1 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & -1 \\ 1 & -1 & -1 & 1 \\ 0 & -1 & 1 & 0 \end{bmatrix} \quad C_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 \end{bmatrix} \tag{14}$$

Now assume:

$$W_a = (C_1 + dC_2) \times (C_1^T + dC_2^T) \tag{15}$$

$$E = C_1 X C_1^T$$

The matrix multiplication of (15) can be factorized to the following equivalent form:

$$W_a = E + dk + d^2 h = E + K + H \tag{16}$$

Where E, k and h are 4x4 matrixes with integer elements:

$$h = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & E_{44} & 0 & -E_{42} \\ 0 & 0 & 0 & 0 \\ 0 & -E_{24} & 0 & E_{22} \end{bmatrix} \tag{17}$$

$$k = \begin{bmatrix} 0 & -E_{14} & 0 & E_{12} \\ -E_{41} & -E_{24} - E_{42} & -E_{43} & E_{22} - E_{44} \\ 0 & -E_{34} & 0 & E_{32} \\ E_{21} & -E_{44} + E_{22} & E_{23} & E_{42} + E_{24} \end{bmatrix}$$

Assume:

$$B = C_1 X \tag{18}$$

Let us now consider a 4 input 4 output butterfly graph (BFg_a). Figure 1 illustrates the (BFg_a) structure and shows how we can compute elements $(B_{21}, B_{31}, B_{11}, B_{41})$ of matrix B from the elements $(X_{11}, X_{41}, X_{21}, X_{31})$ of matrix X as input samples.

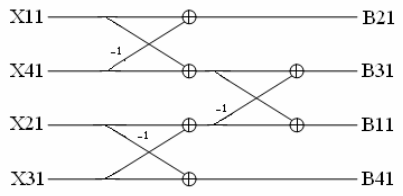

Fig. 1 Butterfly Graph (BFg_a).

If we define BF_a as a function which is determined with BFg_a graph, then we can compute B and E as bellow:

$(B_{21}, B_{31}, B_{11}, B_{41}) = BFa\ (X_{11}, X_{41}, X_{21}, X_{31})$
$(B_{23}, B_{33}, B_{13}, B_{43}) = BFa\ (X_{13}, X_{43}, X_{23}, X_{33})$
$(B_{22}, B_{32}, B_{12}, B_{42}) = BFa\ (X_{12}, X_{42}, X_{22}, X_{32})$
$(B_{24}, B_{34}, B_{14}, B_{44}) = BFa\ (X_{14}, X_{44}, X_{24}, X_{34})$

$(E_{12}, E_{13}, E_{11}, E_{14}) = BFa\ (B_{11}, B_{14}, B_{12}, B_{13})$
$(E_{32}, E_{33}, E_{31}, E_{34}) = BFa\ (B_{31}, B_{34}, B_{32}, B_{33})$
$(E_{22}, E_{23}, E_{21}, E_{24}) = BFa\ (B_{21}, B_{24}, B_{22}, B_{23})$
$(E_{42}, E_{43}, E_{41}, E_{44}) = BFa\ (B_{41}, B_{44}, B_{42}, B_{43})$ (19)

After computing E, the next step is applying d to network. So suppose BFd_g signal flow graph that has been shown in Fig2 and assume like BFg_a it determine BFd function.

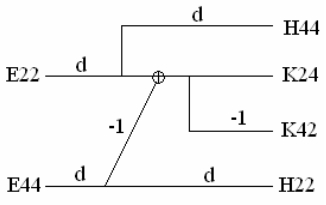

Fig. 2 BFd_g Signal Flow Graph.

With this definition, mixed components of K and H are computed as bellow:

$(H_{44}, K_{24}, K_{42}, H_{22}) = BFd(E_{22}, E_{44})$
$(-H_{42}, K_{44}, K_{22}, H_{24}) = BFd(E_{24}, -E_{42})$ (20)

The rest components of matrixes K and H computed with direct applying of d to correspond output in graph E.

4. Final Reconstruction Step (SFG)

Up to this point, we have computed the matrixes E, h, and k without error and multiplication operation. In order to computing the core transform (W) we need applying 8 coefficients $d, (d = z^2 - 3)$ to the network. The final reconstruction depends on the precision used to represent $z = \sqrt{2+\sqrt{2}}$. Since the final result is in an error free format, we can easily estimate the precision we need to guarantee sufficient accuracy. If the input and output data are 8-bits maximum, then the representation of z as $z = 2 - 2^{-3} - 2^{-5} + 2^{-9}$ is sufficient. [3] Here we can use Horner's Rule to further simplify the FRS.

$$f(z) = a_0 + a_1 z^1 + a_2 z^2 + a_3 z^3 = ((a_3 z + a_2)z + a_1)z + a_0 \quad (21)$$

But since for computing d only z^2 is needed, we can represent d with good precision as:

$$d = 1 - 2^{-1} - 2^{-4} - 2^{-6} - 2^{-7} \quad (22)$$

Here the post-scaling and quantization operation $(\otimes E_f)$ is absorbed into the quantization process too (Multiplication Factor Matrix, MF). Since operations are scalar multiplication rather than matrix multiplication, tables for multiplication and scaling factors $(MF \,\&\, V)$[1] can be replaced with correspond tables 2 and 3 with good precision.
(Error < 0.2%)

5. Simulation and Results

In order to fair compression, we implemented the traditional algorithm with similar butterfly structure too. But since anyone components of scaling matrixes E_f and E_i isn't zero [1], so the correspond graph BF_h has 8 adder that has been shown in Fig 3.

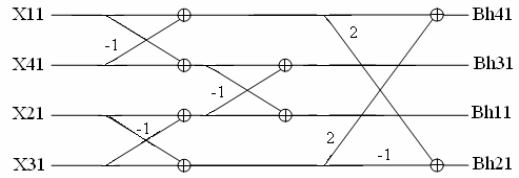

Fig. 3 BF_h graph for traditional algorithm.

Note that it has 8 adder, while BFg_a has 6 one.

The complete processes of both algorithms have brought in table 4. Due to this table, in step 3 of new approach, f is cancelled and computational complexity comparison is presented in table 5.

Table 5: Comparisons for Computing Complexity

Method	***Multiplication (16 bit)***	***Additions***
H.264	32	144
Algebraic	0	*333*

Also table 6 shows a measure of image reconstruction quality, Peak Signal-Noise Ratio (PSNR) for 4 images shown in Fig 4.

Table 6: Quality Measurement, PSNR (dB) with QP=10

Picture	***H.264***	***Algebraic***
Girl	*49.0578*	*52.5594*
Nature	*48.8580*	*51.8868*
Bird	*49.3277*	*52.7856*
Horse	*48.8693*	52.2872

6. Conclusion

In this paper, we have introduced a novel approach for Multiplication and Error Free Implementation of Transform and Quantization blocks like H.264 using an error free algebraic integer representation of basis functions.

Simulation results show less computational complexity with better quality performance. The main advantages of the method proposed are:
1) It is error-free up until the final reconstruction; 2) It needs no multiplication, which is very suitable from VLSI view point;
3) It is like Chen's 1D algorithm and suitable for the implementation with a fully pipelined systolic architecture and can be combined with many already existing algorithms for DCT and IDCT

Table 2: Multiplication Factors (MF_a) in Algebraic method

MFa QP=5	***MFa QP=4***	***MFa QP=3***	***MFa QP=2***	***MFa QP=1***	***MFa QP=0***	***Position***
$2^{-2}-2^{-4}+2^{-6}-2^{-8}$	2^{-2}	$2^{-2}+2^{-5}+2^{-8}$	$2^{-2}+2^{-4}-2^{-8}$	$2^{-1}-2^{-3}-2^{-7}-2^{-8}$	$2^{-1}-2^{-3}+2^{-6}+2^{-7}$	(0,0), (2,0), (0,2), (2,2)
$2^{-2}+2^{-6}-2^{-8}$	$2^{-2}+2^{-4}+2^{-6}-2^{-9}$	$2^{-2}+2^{-3}-2^{-9}$	$2^{-1}-2^{-3}+2^{-5}-2^{-8}$	$2^{-1}-2^{-5}+2^{-7}-2^{-9}$	$2^{-1}+2^{-5}-2^{-7}$	(1,1), (1,3), (3,1), (3,3)
$2^{-2}+2^{-4}+2^{-5}-2^{-9}$	$2^{-1}-2^{-4}-2^{-7}-2^{-9}$	$2^{-1}-2^{-6}+2^{-8}$	$2^{-1}+2^{-5}-2^{-7}+2^{-9}$	$2^{-1}+2^{-3}-2^{-8}$	$1-2^{-1}-2^{-4}-2^{-8}$	Other

Table 3: Scaling Factors (V_a) in Algebraic method

Va QP=5	***Va QP=4***	***Va QP=3***	***Va QP=2***	***Va QP=1***	***Va QP=0***	***Position***
$2^{-2}+2^{-4}$	2^{-1}	$2^{-4}+2^{-5}+2^{-3}$	$2^{-3}+2^{-4}-2^{-6}$	$2^{-3}-2^{-5}-2^{-6}$	$2^{-3}+2^{-5}$	(0,0), (2,0), (0,2), (2,2)
$2^{-1}+2^{-3}-2^{-5}+2^{-4}$	$2^{-2}+2^{-4}+2^{-6}-2^{-9}$	$2^{-2}+2^{-8}-2^{-5}$	$2^{-2}-2^{-6}$	$2^{-2}-2^{-5}+2^{-7}-2^{-9}$	$2^{-2}+2^{-5}-2^{-6}$	(1,1), (1,3), (3,1), (3,3)
$2^{-1}+2^{-5}+2^{-9}$	$2^{-1}-2^{-4}-2^{-7}-2^{-9}$	$2^{-1}-2^{-3}+2^{-9}$	$2^{-1}+2^{-3}-2^{-5}+2^{-8}$	$2^{-2}+2^{-4}-2^{-6}-2^{-8}$	$2^{-2}-2^{-6}$	Other

Table 4: Complete Processes for H.264 & Algebraic Integer
Note f is cancelled in step 3 of new approach

step	***H.264***	***Algebraic scheme***
1	$W=C_fXC_f^T$	$W=C_aXC_a^T$
2	qbits=15+floor(QP/6)	qbits=floor(QP/6)
3	az = [abs (W)⊗MF+f] >> qbits	az_a = [abs (W)⊗MF_a] >> qbits
4	Z=round(az)⊗sign(W)	Z_a=round(az_a)⊗sign(W_a)
5	$W'=Z\otimes V\times 2^{floor(QP/6)}$	$W'_a=Z\otimes V_a\times 2^{floor(QP/6)}$
6	$Xr=round(C_i^T W' C_i) / 64$	$Xr=round(C_a^T W'_a C_a)$

Fig 4. Experimented Figures

7. References

[1] I. E. G. Richardson, H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia, *John Wiley & Sons Ltd., Sussex, England, December 2003.*

[2] "ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC," Draft Text of Final Draft International Standard for Advanced Video Coding, [Online]. Available: http://www.chiariglione.org/mpeg/working_documents.htm , March 2003.

[3] V. S. Dimitrov, G. A. Jullien and W. C. Miller, "A New DCT Algorithm Based on Encoding Algebraic Integers", *IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 1377-1380, 1998.*

[4] Minyi Fu; V.S. Dimitrov, V.S. and G.A. Jullien, "An Efficient Technique for Error-free Algebraic-integer Encoding for High Performance Implementation of the DCT and IDCT", *in Proc. IEEE International Symposium on Circuits and Systems, Sydney Australia, May 2001, pp. 906-909.*

[5] J. H. Cozzens and L. A. Finkelstein, "Computing the Discrete Fourier Transform using Residue Number Systems in a Ring of Algebraic Integers", *IEEE Transactions on Information Theory, vol. 31, pp. 580-588, 1985.*

[6] V. Dimitrov and G. A. Jullien, "Multidimensional Algebraic Integer Encoding for High Performance Implementation of the DCT and IDCT", *IEE Electronics Letters, vol. 29, no. 7, pp. 602-603, 2003.*

[7] Khan Wahid, Vassil Dimitrov and Graham Jullien, "Error-Free Computation of 8x8 2-D DCT and IDCT using Two-Dimensional Algebraic Integer Quantization", *Proceedings of the 17th IEEE Symposium on Computer Arithmetic (ARITH'05), 1063-6889/05 © 2005 IEEE*

[8] V. Dimitrov and R. Baghaie, "Computing discrete Hartley transform using algebraic integers," in Proceedings of the 33rd Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, USA, vol. 2, pp. 1351-1355, October 1999.

[9] K. A. Wahid, V. S. Dimitrov and G. A. Jullien, "Error-Free Arithmetic for Discrete Wavelet Transforms using Algebraic Integers" *Proceedings of the 16th IEEE Symposium on Computer Arithmetic (ARITH'03) 1063-6889/03 (C) 2003*

[10] Yeonsik Jeong, Imgeun Lee, Taekhyun Yun, Gooman Park and Kyu Tae Park, "A Fast Algorithm Suitable for DCT Implementation with Integer Multiplication", *IEEE TENCON - Digital Signal Processing Applications, pp. 784 – 787, 1996*

[11] Richard A. Games, Daniel Moulin, Sean D. O'Neil, and Joseph J. Rushanan, "ALGEBRAIC-INTEGER QUANTIZATION AND RESIDUE NUMBER SYSTEM PROCESSING" *CH267S2/89/ 0000-0948, pp. 948-951, 1989 IEEE*

[12] R. Baghaie and V. Dimitrov, "Systolic Implementation of Real-valued Discrete transforms via Algebraic Integer Quantization", *An International Journal on Computers and Mathematics with Applications, vol. 41, pp. 1403-1416, 2001.*

[13] H.264 Reference Software Version JM6.1d, http://bs.hhi.de/~suehring/tml/, March 2003.