International Journal of Computer Applications (0975 – 8887)
Volume 131 – No.7, December2015

Implementing Lossy Compression Technique for Video Codecs

Islam Mohammad Saif
Department of Electronics and Communications, Ain Shams University

Abdelhalim Zekry
Department of Electronics and Communications, Ain Shams University

ABSTRACT

This paper provides an overview of the transform and quantization operations in H.264 lossy coding techniques. It declares the detailed simplification process for arithmetic operations included in the implementation for the 4x4 AC and the 2x2 & 4x4 DC luma and chroma blocks applying fast DCT Butterfly implementation method for the AC component and the effective Hadamard Transform implementation for the DC components, in addition to the quantization process procedure. However, this paper main aim is to provide a complete software design and implementation for the decoder process as defined in the ITU-T standard release 2011, besides, it defines a proper way for implementing the encoder process according to the defined decoder procedure defined in the ITU-T Standard.

General Terms

H.264/AVC transform and quantization, video lossy compression

Keywords

H.264, AVC, DCT, Hadamard, Butterfly, Quantization, AC, DC

1. INTRODUCTION

Images and video coding techniques have been updated over the last years very widely due to the dramatically updates in the image quality, so the need for updating coding techniques became a must. It has been updated from MPEG-1, MPEG-2, H.261, and H.263 until the H.264 rose up to be the most usable technique for the high quality imagery data transmission. According to the image nature, as the image and video quality increases, the frame storage area increases. So, an urgent need for a new technique for image and video compression came up in order to reduce the storage consumption, also, the reduction in the transmission bandwidth needed to transmit the image or video data. H.264/AVC was developed by Joint Video Team (JVT), which was formed by the ISO/IEC Moving Picture Expert Group (MPEG) and the ITU-T Video Coding Expert Group (VCEG) [1][2]. Since this time, H.264 came up to be the most used technique in high quality imagery data. This is due to the great reduce in the imagery processing time as it depends on a 4x4 blocks instead of 8x8 blocks in previous models. Also, it depends on the fast DCT implementation using the Butterfly method. Most of the papers done before declared either a way to reduce the PSNR without implementation, declaring the encoder idea without actual simulation for the process overall. Also, most of the previous papers aim is to declare the ITU-T release 2003 version.

The 4x4 H.264 forward matrix transformation in the Encoding process is called the core integer transformation operation. This operation depends on the core transform equation "CXC^T", where X is the input frame residual matrix, C is the core transform matrix and C^T is its transpose. [7]

$$C = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix}$$

This operation consumes 128 multiplications & 96 additions, comparing to those of the 8x8 H.263 implementation which has 1024 multiplications & 896 additions, so it dramatically reduces the computing and so the processing time. The processing time is further reduced using the fast 4x4 DCT Butterfly implementation technique which eliminates the use for multiplication and replaces it by only addition, subtraction and logical shifting operations only (This applies on the FPGA implementation which uses the binary coding technique).

In the decoding process, the 4x4 H.264 inverse matrix transformation is called the core integer inverse transformation operation. This operation depends on the core inverse transform equation "$C_iYC_i^T$", where Y is the received matrix, C_i is the core inverse transform matrix and C_i^T is its transpose.[7]

$$C_i = \begin{bmatrix} 1 & 1 & 1 & 1/2 \\ 1 & 1/2 & -1 & -1 \\ 1 & -1/2 & -1 & 1 \\ 1 & -1 & 1 & -1/2 \end{bmatrix}$$

Each 4x4 input block contains the high frequency data called "AC" portion of the block, and the low frequency data called "DC" portion of the block. According to the human visual system (HVS), the human eye is more sensitive to the low frequency data that the high frequency data [10]. So, in order to conserve most of the information included in the block, DC portion must be drawn from the frame and coded separately. However, this DC data must have the least processing time and resource consumption to minimize the overall operation latency. The DC content is the first element in the input 4x4 matrix (element (0, 0)).

There are three types of image coding in the prediction and motion estimation stage (before this compression stage), 4:4:4, 4:2:2, 4:2:0. All defines the size of the chroma macro block which defines the whole frame specifications. In this paper, the used technique is 4:2:0, in which the luma block is 16x16 elements, while chroma block is 8x8 elements. So, DC data will form 4x4 matrix for the luma macro Block and 2x2 matrix for the chroma macro Block.[2][7]

As the DC block contains most of the information in the frame, so it is coded separately using the Hadamard transform technique which applies the following equation "$HX_{DC}H^T$", where H is the Hadamard Core Transform matrix[1][2]

$$H_1 = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} \quad H_2 = \frac{1}{2}\begin{bmatrix}1 & 1 & 1 & 1\\ 1 & 1 & -1 & -1\\ 1 & -1 & -1 & 1\\ 1 & -1 & 1 & -1\end{bmatrix}$$

where, H_1 is the Hadamard transform matrix for the 2x2 chroma macro block and H_2 is the Hadamard transform matrix for the 4x4 luma macro block. The DCT and Hadamard transformation stage main purpose is to minimize the dependency for the compression process on the quantization step. The quantization process introduces the idea of possible signal loss for better compression efficiency. There are different quantization techniques for AC component other than the DC component.

All of the previously discussed issues was implemented using matlab and simulink software. The main goal in implementing the software step is to test the design operation and the idea on how the processing time is minimized. The core idea is to reduce the transform matrix intense multiplication steps into just element by element simple multiplication, and even converting multiplication into addition stages in order to save resources. The software implementation using matlab and simulink codes will be declared in details in the following sections.

2. TRANSFORM CODING

The main purpose for the transform coding stage is to reduce the overhead for the quantization step. Transform coding portion is the most resources and time consuming portion of the whole lossy encoding operation, this is why several process reduction techniques have been applied to reduce the calculations processing time with maintaining the transformation accuracy. The whole 4x4 input matrix is transformed using the DCT technique, then the DC portion of the block is separated. AC component is quantized in order to improve the compression technique efficiency, however, the DC component is further transformed and quantized separately in order to maintain its information as it contains most of the frame information. The following blocks illustrates this operation:

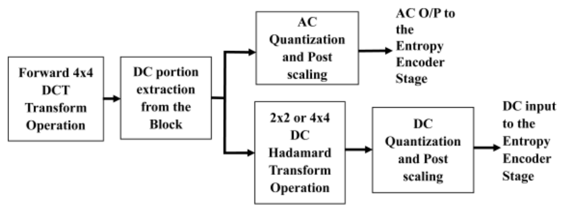
Forward 4x4 DCT Transform Operation
DC portion extraction from the Block
AC Quantization and Post scaling
AC O/P to the Entropy Encoder Stage
2x2 or 4x4 DC Hadamard Transform Operation
DC Quantization and Post scaling
DC input to the Entropy Encoder Stage

Fig 1: Lossy encoder block diagram

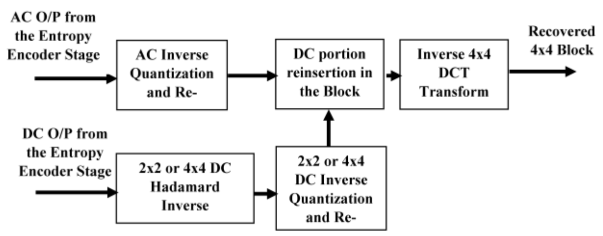
AC O/P from the Entropy Encoder Stage
AC Inverse Quantization and Re-
DC portion reinsertion in the Block
Inverse 4x4 DCT Transform
Recovered 4x4 Block
DC O/P from the Entropy Encoder Stage
2x2 or 4x4 DC Hadamard Inverse
2x2 or 4x4 DC Inverse Quantization and Re-

Fig 2: Lossy decoder block diagram

The ITU-T standard reference only illustrates the decoding technique, this paper clarifies a method to encode input signal.

2.1 4x4 DCT block forward transform

This procedure takes place in the encoder side. The transform operation idea is to eliminate the dependency of the compression operation on the quantization only by decomposing the signal using the discrete cosine transform (DCT) technique. The transformation process is performed according to the equation CXC^T, where C is the core forward transformation matrix and X is the input signal matrix. So the equation implementation will be as follows: [2] [7]

$$CXC^T = \begin{bmatrix}1 & 1 & 1 & 1\\ 2 & 1 & -1 & -2\\ 1 & -1 & -1 & 1\\ 1 & -2 & 2 & -1\end{bmatrix}\begin{bmatrix} & & \\ & X & \\ & & \end{bmatrix}\begin{bmatrix}1 & 2 & 1 & 1\\ 1 & 1 & -1 & -2\\ 1 & -1 & -1 & 2\\ 1 & -2 & 1 & -1\end{bmatrix} \tag{2.1}$$

This is a 3 4x4 matrix multiplication of about 128 multiplications & 96 additions in the whole process which is a great time and resources consumption [11]. However, the DCT core operation could be simplified using the butterfly method. The butterfly method converts the 3 stages 4x4 matrices multiplication operations into just addition, subtraction and shifting (in FPGA implementation, in integer coding it is multiplication by 2), it also converts the 2-D matrix multiplication into two stages 1-D matrix operation. Its idea of operation is as follows: [5] [6]

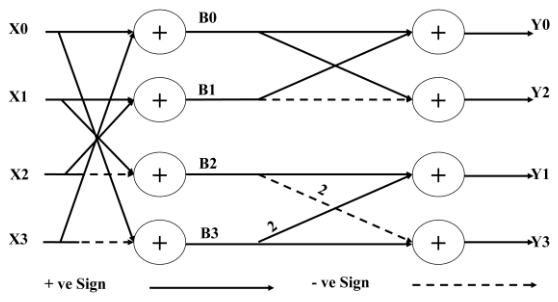

Fig 3: Forward 1-D Butterfly implementation

where, X0 to X3 are the 4x4 matrix 4-row elements and Y0 to Y3 are the corresponding transformed 4-elements respectively. B0 to B3 and Y0 to Y3 are defined as:

B0=X0+X3	Y0=B0+B1
B1=X1+X2	Y2=B0-B1
B2=X1-X2	Y1=B2+ (B3<<1)
B3=X0-X3	Y3=B3- (B2<<1)

Note that the "<<" sign refers to the shifting process which compensate the multiplication process (e.g. 3 is represented in binary as 0011, 3*2=6 which is represented in binary as 0110, so by shifting 0011 to the left by one bit the number is multiplied by 2). This operation is done row by row, so the operation is no more a complex 2-D matrix multiplication any more, it is now a simple 1-D matrix operation. The overall row by row process is illustrated as follows:

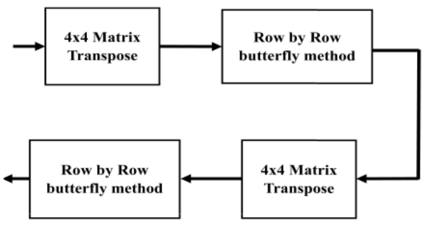

Fig 4: Fast DCT implementation block diagram

So, the input matrix is transposed, then the row by row transformation takes place. At last the resulting matrix is further transposed, then again transformed.

2.2 4x4 DCT block inverse transform

This procedure takes place in the decoder side, which is the portion that is declared in the ITU-T standard and from which the encoding technique was estimated. Accordingly, the transform operation also relies on the core inverse transform equation “$C_iYC_i^T$ “. So the equation implementation will be as follows: [9]

$$C_iYC_i^T = \begin{bmatrix} 1 & 1 & 1 & 1/2 \\ 1 & 1/2 & -1 & -1 \\ 1 & -1/2 & -1 & 1 \\ 1 & -1 & 1 & -1/2 \end{bmatrix} \begin{bmatrix} & & & \\ & Y & & \\ & & & \\ & & & \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1/2 & -1/2 & -1 \\ 1 & -1 & -1 & 1 \\ 1/2 & -1 & 1 & -1/2 \end{bmatrix} \quad (2.2)$$

Again, the Butterfly method could be applied on the decoder side as an inverse butterfly method as follows:

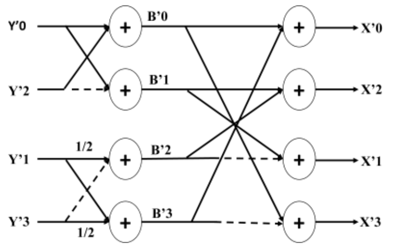

Fig 5: Inverse 1-D Butterfly implementation

where, Y’0 to Y’3 are the received 4-elements row of the 4x4 matrix after forward quantization process, and X’0 to X’3 are the corresponding recovered elements. B’0 to B’3 and X’0 to X’3 are defined as follows:

B’0=Y’0+Y’2	X’0=B’0+B’3
B’1=Y’0-Y’2	X’1=B’1+B’2
B’2= (Y’1>>1) –Y’3	X’2=B’1-B’2
B’3=Y’1+ (Y’3>>1)	X’3=B’0-B’3

Note that the “>>” sign refers to the shifting process which compensate the division process (e.g. 6 is represented in binary as 0110, 6/2=3 which is represented in binary as 0011, so by shifting 0110 to the right by one bit the number is divided by 2). This will also be applied to the same operation as Fig.4.

2.3 2x2 and 4x4 DC block Hadamard transform

DC component is the low frequency component of the block, the human visual system (HVS) is more sensitive to the low frequency frame component than the high frequency component as mentioned before. So, the DC component is further transformed using the Hadamard transform technique, also it is separately quantized and transmitted. The Hadamard transform forward and inverse operations are identical, this is because the core Hadamard transform matrix and its transpose are identical ($H=H_i=H^T$). The Hadamard transformation operation follows the equation “$HX_{DC}H$”.

The equation implementation for 4x4 DC component and 2x2 DC component respectively is: [1]

$$H_2X_{DC}H_2 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} & & & \\ & X_{DC} & & \\ & & & \\ & & & \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} / 4 \quad (2.4)$$

$$H_1X_{DC}H_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} X_{DC} & \\ & \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} / 2 \quad (2.5)$$

The butterfly method can also be applied to Hadamard transform equations. However this time there is no values in the Hadamard transform matrix other than 1 or -1, so the butterfly method contains only additions and subtractions only.

The 2x2 DC Hadamard transform method could be applied (without the division by 2 step) using the following technique:

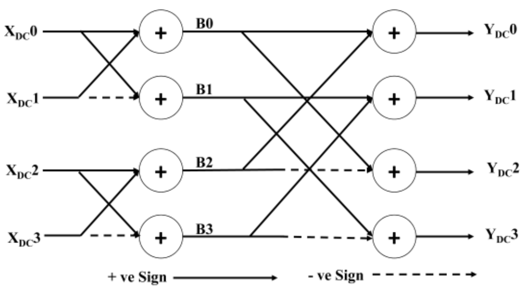

Fig 6: 2x2 Fast Hadamard Transform Butterfly Method

The 4x4 DC Hadamard transform method could be applied (without division by 4 step) using the following technique:

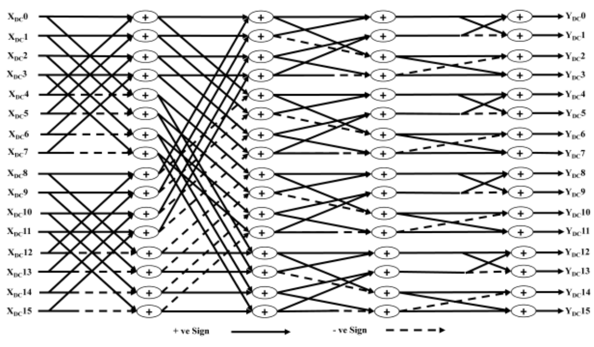

Fig 7: 4x4 Fast Hadamard Transform Butterfly Method

As noticed, there is no operations other than addition and subtraction only. So the operation is widely simplified.

3. QUANTIZATION

The Integer AC & DC frame components are quantized in order to eliminate the least important data to achieve the highest compression efficiency. H.264 quantizer is defined to be a scalar quantizer which maps every input sample into a corresponding output one. The big challenges facing the forward and inverse quantizers are:

a. Eliminating the division processes
b. Avoiding floating point arithmetic operations

c. Applying the post and rescaling matrices
d. Determining the proper quantization parameter (QP)

The AC component is quantized in a special technique, while the DC component is quantized by another technique. This is due to the difference in the sensitivity between AC and DC components. The ITU-T also provides only inverse quantization technique for the decoder portion of the operation. The quantization operation is very dependent on the QP value which varies from 0 to 51. Its value is determined variably according to the type and the amount of information contained in the frame. It also varies between the luma and chroma frame components, it even varies between the AC and DC components within the same block.

3.1 4x4 AC inverse quantization

The main inverse quantization equation is: [2] [7]

$$Y' = Z'.Qstep \qquad (3.1)$$

where, Y' is the inverse quantized data and Z' is the received data. What should be done in this stage is to re-scale the received data in order to prepare it to the transformation stage to sustain the most proper recovered data in the received signal, also the equation should be simplified in order to minimize the processing consumption portions such as multiplication and division. So, the previous equation is simplified according to the following graph

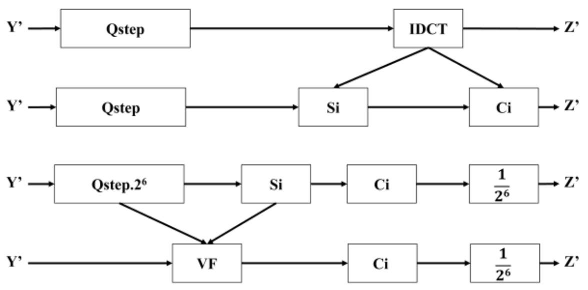

Fig 8: Inverse quantization simplification procedure

where, Y' is the received data and Z' is the recovered data, IDCT is the full transformation operation without simplification, Ci is the core inverse transformation operation defined in equation 2.2, Si is the rescaling matrix, Qstep is the quantization step size and VF is defined according to fig 8 as

$$VF = Qstep.Si.2^6 \qquad (3.2)$$

also Si is defined as

$$Si = \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix}$$

Qstep is very dependent on QP value. There is a different Qstep value for every QP value. Qstep value doubles every increase in the QP value by 6 which is illustrated in the following table

Table 1. Qstep values vs. QP

QP	Qstep	QP	Qstep
0	0.625	…	…
1	0.6875	18	5
2	0.8125	…	…
3	0.875	24	10
4	1	…	…
5	1.125	30	20
6	1.25	…	…
7	1.375	36	40
8	1.625	…	…
9	1.75	42	80
10	2	…	…
11	2.25	48	160
12	2.5	…	…
…	…	51	224

Respectively, VF is derived according to fig 8 from the following equation

$$VF \sim Si.Qstep.2^6 \qquad (3.3)$$

and is defined in the ITU-T standard referring to QP values as

Table 2. VF values according to the QP Values

QP	Positions (0,0), (0,2), (2,0), (2,2)	Positions (1,1),(1,3),(3,1), (3,3)	Other Positions
0	10	16	13
1	11	18	14
2	13	20	16
3	14	23	18
4	16	25	20
5	18	29	23

In order to provide a complete simplified form for the inverse quantization equation, the ITU-T standard provides the following procedure [1]

dij=(cij.LevelScale(QP%6,i,j)) <<(QP/6 - 4), for QP>=24
dij=(cij.LevelScale(QP%6,i,j)+(1<<(3-QP/6)))>>(4 – QP/6)
,for QP<24 (3.4)

where, dij is the inverse quantized data which is defined previously as Y', cij is the received data which is defined previously as Z', the "<<" sign refers to the shifting process which compensate the multiplication process (e.g. 3 is represented in binary as 0011, 3*2=6 which is represented in binary as 0110, so by shifting 0011 to the left by one bit the number is multiplied by 2) and the ">>" sign refers to the shifting process which compensate the division process (e.g. 6 is represented in binary as 0110, 6/2=3 which is represented in binary as 0011, so by shifting 0110 to the right by one bit the number is divided by 2). The LevelScale is defined as

LevelScale(QP%6,i,j)=weightScale(i.j).v(QP%6,n) (3.5)

where the weightScale is=2^4=16 and v(QP%6,n) is the VF factor defined in table 2. The weightScale value importance is mainly in high profile imagery data, however, in this paper the main goal is to implement the ordinary imagery types, as the weightScale step is compensated in the inverse quantization equation by the shifting by 4 step (<<4 and >>4), so both could be eliminated from the equations. So the final inverse quantization equation becomes [7]

$$Y' = Z'.VF/2^{floor(QP/6)} \qquad (3.6)$$

The Y' output is inverse transformed by equation 2.2 and rounded by the following equation according to the standard

$$Rij = (hij + 2^5) >> 6 \qquad (3.7)$$

where, rij is the recovered data and hij is the transformed data, this operation is equivalent to dividing by 2^6 and flooring the resulted data.

3.2 4x4 AC forward quantization

The core quantization operation follows the following equation [2] [7]

$$Z=\text{round}(Y/Qstep) \tag{3.8}$$

where, Z is the quantizer output, Y is the transformed 4x4 AC integer block matrix and Qstep is the quantization step size which is very dependent on QP value as discussed before. In order to simplify the operation and eliminate the Qstep fraction calculations, the core equation is transformed into a different form which follows the following procedure:

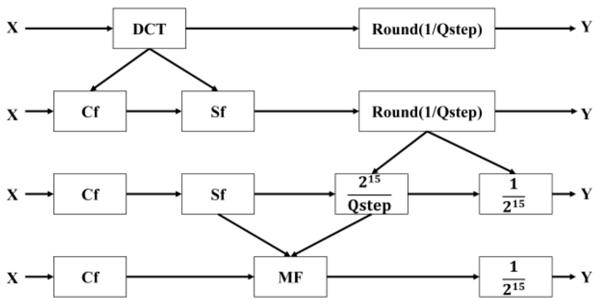

Fig 9: Quantization simplification procedure

where, DCT is the full discrete cosine transform operation without simplification, Cf is the core transform operation declared in equation 2.1, Sf is the post scaling factor which is the result of the transformation simplification operation and has the value of

$$Sf=\begin{bmatrix} a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \\ a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \end{bmatrix}$$

where, $a=\frac{1}{2}$, $b=\sqrt{\frac{2}{5}}$. MF is defined according to table 3 as follows

Table 3. MF values according to the QP Values

QP	Positions (0,0), (0,2), (2,0), (2,2)	Positions (1,1),(1,3),(3,1), (3,3)	Other Positions
0	13107	5243	8066
1	11916	4660	7490
2	10082	4194	6554
3	9362	3647	5825
4	8192	3355	5243
5	7282	2893	4559

The data in the table was derived from the fact that

$$MF\sim Sf.(2^{15}/Qstep) \tag{3.9}$$

Referring to the VF and combining with equation 3.3, MF will be defined as

$$MF\sim Sf.Si.\ (2^{15}/VF) \tag{3.10}$$

which is the actual equation from where the MF value was derived from. Referring to the fact that Qstep is doubled for every increase in QP value by 6, so the final forward quantization equation becomes [7]

$$Z=\text{round}((Cf.MF)>>(15+\text{floor}(QP/6))) \tag{3.11}$$

3.3 4x4 Luma DC forward quantization

According to the forward AC quantization model, all of the implemented procedure remains the same, but there will be a slight change in the equation. Besides, the DC component is always allocated in the (0, 0) place in the frame, so the values of the MF factor will be only MF (0, 0). All of the rest factors remains the same. So, the 4x4 DC forward quantization core equation becomes [7]

Z_{DC}=round $((Y_{DC}>>((QP/6)-6)).(MF_{DC(0,0)}>>10))$, for QP>=36

Z_{DC}=round $(((Y_{DC}<<(6-(QP/6)))$ -

$(1<<(5-(QP/6))))).(MF_{DC(0,0)}>>10))$, for QP<36 (3.12)

All the previous variables is defined according to the 4x4 AC forward quantization model, and MF_{DC} is defined according to the following table

Table 4. MF_{DC} values according to the QP Values

QP	Positions (0,0), (0,2), (2,0), (2,2)	Positions (1,1),(1,3),(3,1), (3,3)	Other Positions
0	100	63	77
1	91	56	71
2	77	50	63
3	71	44	56
4	63	40	50
5	56	35	44

The MF_{DC} value was derived from the following equation

$$MF_{DC}=\text{round}((1/VF).1000) \tag{3.13}$$

and the multiplication by 1000 is to eliminate the fractional part of the operation and eliminate the need for the division operation, also this part is compensated by the shifting operation (>>10) which is equivalent to dividing by 1024.

3.4 4x4 Luma DC inverse quantization

According to the inverse AC quantization model, again all the implemented procedures are the same, but also with the changes in the equation. Also the VF factor will be only for VF (0, 0). Everything else remains the same. So, the 4x4 DC inverse quantization core equation becomes: [1]

Y_{DC}=round $(Z_{DC}.VF_{(0,0)})<<((QP/6)-6)$, for QP>=36

Y_{DC}=round $(((Z_{DC}.VF_{(0,0)})+(1<<(5-QP/6))))>>(6-(QP/6))$

, for QP<36 (3.14)

All the previous variables is defined according to the 4x4 AC Inverse quantization model.

3.5 2x2 Chroma DC forward quantization

The forward quantization model for the 2x2 chroma DC quantizer model is exactly the same as the 4x4 luma DC model which is

$$Z_{DC}=\text{round}((Y_{DC}.MF_{DC(0,0)})>>(5+(QP/6))) \tag{3.15}$$

There is no difference at all in the values.

3.6 2x2 Chroma DC inverse quantization

The inverse quantization model for the 2x2 chroma DC have a slight differences than the 4x4 luma DC model. The core 2x2 chroma DC inverse quantization model equation becomes [1]

$$Y_{DC}=\text{round}\ (((Z_{DC}.VF_{(0,0)})<<(QP/6))>>5) \qquad (3.16)$$

All the variables and values are defined as before.

4. IMPLEMENTATION

The whole encoder and decoder operation was implemented using the matlab software as a code and simulink software as a functional blocks. The following sections will discuss the full operation in details.

4.1 Encoder implementation

4.1.1 Matlab implementation

The encoder matlab implementation is a full implementation for the encoder operation including the forward block transform, DC component extraction, AC component quantization and DC component transformation and quantization prior to forwarding it to the next stage (entropy lossless encoding). The matlab implementation could be summarized according to the following flow chart:

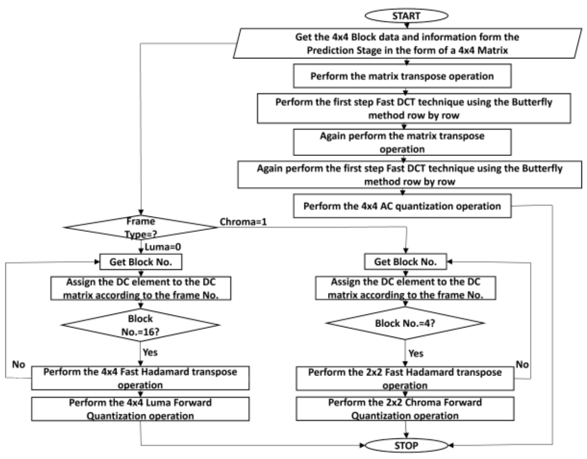
START
Get the 4x4 Block data and information form the Prediction Stage in the form of a 4x4 Matrix
Perform the matrix transpose operation
Perform the first step Fast DCT technique using the Butterfly method row by row
Again perform the matrix transpose operation
Again perform the first step Fast DCT technique using the Butterfly method row by row
Perform the 4x4 AC quantization operation
Frame Type=?
Chroma=1
Luma=0
Get Block No.
Assign the DC element to the DC matrix according to the frame No.
Block No.=16?
Block No.=4?
Yes
No
Perform the 4x4 Fast Hadamard transpose operation
Perform the 2x2 Fast Hadamard transpose operation
Perform the 4x4 Luma Forward Quantization operation
Perform the 2x2 Chroma Forward Quantization operation
STOP

Fig 10: Full Transform – Quantization Encoder Block Diagram

According to the previous chart, the procedure goes as follows: First of all, the 4x4 predicted block is gathered from the prediction stage, then reformed in the form of matrix, then the transformation operation takes place. According to the previously discussed method of operation, the full 4x4 block is transformed using the simplified butterfly fast DCT transform declared in equation 2.1 and fig 3 and fig 4. According to the frame formation, the low frequency data is formed in the first element of the 4x4 residual block, so each DC element from each macro block is gathered according to the frame type. If the frame was a luma frame, then there will be 16 DC elements forming a 4x4 DC matrix, however, if the frame was a chroma frame, so only 4 elements is needed to form a 2x2 DC matrix. Those formed DC matrices is further transformed using the Hadamard transform technique, each one according to its frame type as declared in equations 2.4 and 2.5. All the matrix multiplication procedure is converted into just addition, subtraction and element by element multiplication by only the value 2. After the DC frame is formed, it is quantized using the specified quantization technique declared in equations 3.12 and 3.14 each of its type and then forwarded separately to the entropy encoder step, then the AC portion of the macro block is also quantized according to equation 3.11 and forwarded separately to the entropy encoder step.

4.1.2 Simulink implementation

The simulink implementation makes the idea of operation clearer, as it is formed of functional blocks, each one has its own purpose. It is implemented as follows:

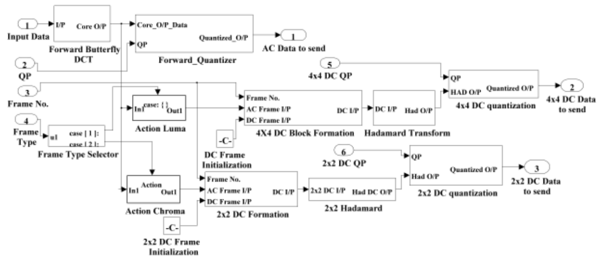

Fig 11: Full Transform – Quantization Encoder Simulink Implementation

This block formation was done according to the matlab coding implementation technique. The “forward butterfly DCT” block contains the 4x4 fast butterfly DCT implementation for the input block, “forward quantizer” block contains the quantization procedure for the AC portion, the “frame type selector” block defines whether the block is luma or chroma, and accordingly passes the signal to the proper stage (2x2 DC transform stage or 4x4 DC transform stage). The “2x2 DC formation” and “4x4 DC block formation” blocks gathers the DC data from each received block in order to form the 2x2 chroma DC block or the 4x4 luma DC block. After that, the data is passed to the proper Hadamard transformation block, then to the proper quantization block then transmitted with the quantized AC 4x4 block to the decoder side.

4.2 Decoder implementation

4.2.1 Matlab implementation

The decoder matlab implementation is a full implementation for the decoder operation including the AC block rescaling and inverse quantization and DC component rescaling and inverse transformation and re-insertion into the 4x4 block, then the final block inverse transformation process takes place using the inverse fast DCT butterfly implementation method. The matlab implementation could be summarized according to the following flow chart:

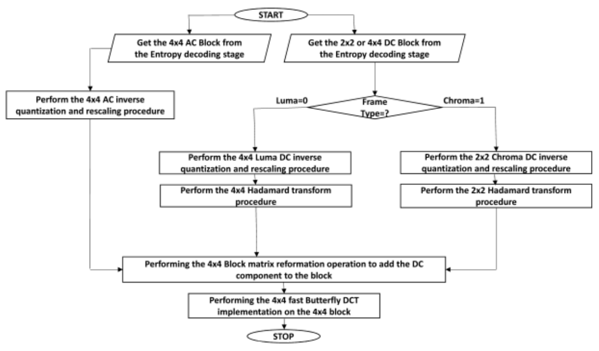
START
Get the 4x4 AC Block from the Entropy decoding stage
Get the 2x2 or 4x4 DC Block from the Entropy decoding stage
Perform the 4x4 AC inverse quantization and rescaling procedure
Luma=0
Frame Type=?
Chroma=1
Perform the 4x4 Luma DC inverse quantization and rescaling procedure
Perform the 2x2 Chroma DC inverse quantization and rescaling procedure
Perform the 4x4 Hadamard transform procedure
Perform the 2x2 Hadamard transform procedure
Performing the 4x4 Block matrix reformation operation to add the DC component to the block
Performing the 4x4 fast Butterfly DCT implementation on the 4x4 block
STOP

Fig 12: Full Transform – Quantization Decoder Block Diagram

According to the previous chart, the procedure goes as follows: The received signal is actually of two types, AC and DC, each signal has its rescaling and inverse transformation technique. For the AC block, it is always 4x4 block, so it is reverse quantized and re-scaled according to equation 3.6. However, for the DC signal it has two types, 2x2 chroma block and 4x4 luma block, each signal type has its own technique, so, the type of signal is detected at first, then rescaled according to equations 3.14 or 3.16 and inverse transformed according to equations 2.4 or 2.5 according to its type. At last, the DC value is re-inserted again in the block, and then the whole block is further inverse transformed using the fast 4x4 butterfly DCT transform technique in equation 2.2.

4.2.2 *Simulink implementation*

The simulink implementation makes the idea of operation clearer, as it is formed of functional blocks, each one has its own purpose. It is implemented as follows:

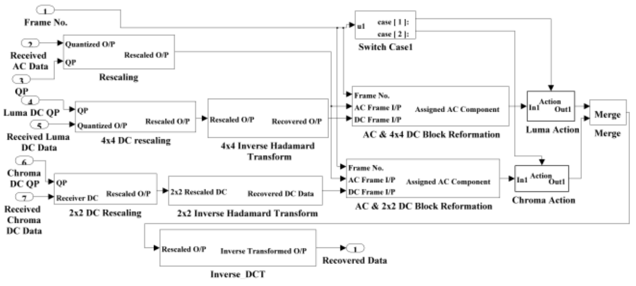

Fig 13: Full Transform – Quantization Decoder Simulink Implementation

This block formation was done according to the matlab coding implementation technique. As the quantization and the transformation in the encoder is reversed, the operation sequence is also reversed. The “rescaling” block contains the AC portion rescaling process, the “4x4 DC rescaling” and “2x2 DC rescaling” blocks contains the luma and chroma DC rescaling processes respectively, the “4x4 inverse hadamard transform” and “2x2 inverse hadamard transform” blocks are the inverse DC transformation techniques for luma and chroma DC data respectively, the “AC & 4x4 DC block reformation” and “AC & 2x2 DC block reformation” blocks contains the reinsertion for the DC data into the block prior to the last transformation step in the “inverse DCT” block.

5. VERIFICATION

5.1.1 *AC Results verification*

In order to verify the correctness of the proposed model “The H.264 advanced video compression standard” provides a full calculated example for the operation. Suppose the input at the encoder is the 4x4 matrix

$$X = \begin{bmatrix} 58 & 64 & 51 & 58 \\ 52 & 64 & 56 & 66 \\ 62 & 63 & 61 & 64 \\ 59 & 51 & 63 & 69 \end{bmatrix}$$

and suppose QP = 6.

Then the output of the core transform W = CXC^T after calculation is

$$W = \begin{bmatrix} 961 & -41 & 15 & -48 \\ -34 & 72 & -30 & -104 \\ -15 & 3 & 15 & 24 \\ 13 & 81 & -5 & 8 \end{bmatrix}$$

And the output of the forward quantizer is

$$Z = \begin{bmatrix} 192 & -5 & 3 & -6 \\ -4 & 5 & -3 & -8 \\ -3 & 0 & 3 & 3 \\ 1 & 6 & 0 & 0 \end{bmatrix}$$

Making loop back such that the output of the encoder is input to the decoder, the final recovered data will be

$$X' = \begin{bmatrix} 58 & 63 & 51 & 59 \\ 53 & 64 & 57 & 66 \\ 62 & 63 & 60 & 64 \\ 59 & 52 & 63 & 68 \end{bmatrix}$$

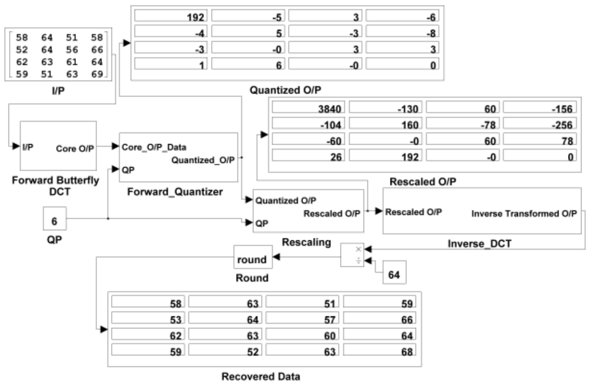

Fig 14: Full AC transform – quantization encoder - decoder implementation verification

Figure 14 shows the input and different stages output results in the built simulink model of the codec. By comparing these results with the manual computation above, a full agreement is found which proves the correctness of the developed simulink model. Since the codec is lossy because of the quantization errors there will be errors between the input and output of the codec.

5.1.2 *4x4 DC Results verification*

The luma DC model has no verification in the reference, however its tests were done manually for the input DC block

$$X_{DC} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

In this case, the error is minimized in order to maintain most of the low frequency information data, in this example the error is a zero matrix.

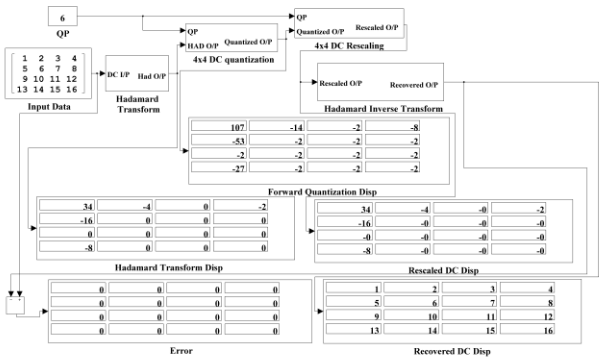

Fig 15: Full luma DC transform – quantization encoder - decoder implementation verification

5.1.3 2x2 DC Results verification

The chroma DC model also has no verification in the reference, however its tests were done manually for the input DC block

$$X_{DC} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

In this example, the error does not exceed "1" value.

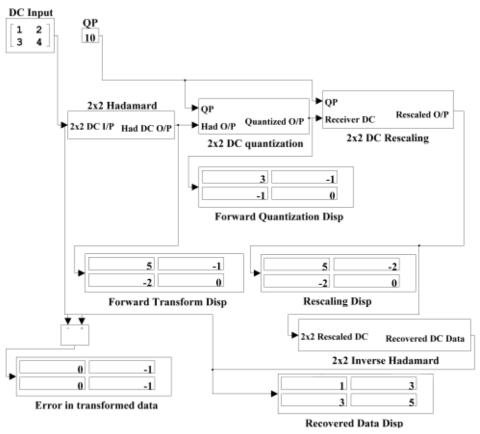

Fig 16: Full chroma DC transform – quantization encoder - decoder implementation verification

6. CONCLUSION

This paper defines in details the theoretical basis and the software implementation of the decoder for the lossy part in H264 source coding. It also defines a way to implement the encoder portion as the ITU-T standard reference only defines the decoder portion. The software implementation includes the matlab programming code and the simulink functional blocks model for both encoder and decoder portions.

The future part of the previously discussed technique is accurately estimating a proper quantization parameter (QP) in order to achieve the best quantization efficiency and also best PSNR. Also, there may be a functional hardware implementation using FPGA modules.

7. REFERENCES

[1] "ITU-T H.264 Advanced Video Coding for Generic Audio Visual Services", Standard Reference Book

[2] "H.264 and MPEG-4 Video Compression", by Ian E. G. Richardson

[3] "Adaptive Initial Quantization Parameter Determination for H.264/AVC Video Transcoding", by Zhenyu Wu, Hong Yang Yu, Bin Tang and Chang Wen Chen, Fellow IEEE

[4] "Implementation and Analysis of Architecture for the 4x4 2-D Forward Hadamard Transform of H.264/AVC", by Daniel Palomino, Guilherme Correa, Robson Dornelles, Felipe Sampaio, Diego Nobel, Luciano Agostini

[5] "Optimization of 4x4 Integer DCT in H.264/AVC Encoder", by Charles S. Lubobya, Mqele M. Dlodlo, Gerhard De. Jager and Keith L. Ferguson

[6] "Architecture for Area Efficient 2-D Transform in H.264/AVC", by Yu-Ting Kuo, Tay-Jyi Lin, Chih-Wei Liu and Chein-Wei Jen

[7] "The H.264 Advanced Video Compression Standard", by Ian E. Richardson

[8] "Source Coding and Compression Transform Coding", by Dr. Eng. Khaled Shawky

[9] "Low Complexity Transform and Quantization in H.264/AVC", by Henrique S. Malvar, Fellow IEEE, Antti Hallapuro, Marta Karcz Ewicz and Louis Kerofsky, Member IEEE

[10] "The VC-1 and H.264 Video Compression Standards for Broadband Video Services", by Lee, Jae-Beom, Kalva, Hari

[11] "Low complexity DCT engine for image and video compression", by Maher Jridi, Yousri Ouerhani, Ayman Alfalou

[12] "Reference Design Software".