VLSI Implementation of H.264 Transform and Quantization Algorithms

G. Dileepvamshi[1], P. Ramakrishna[2]

[1]M.Tech Student
CVSR College of Engineering, Hyderabad, India
dileep.vamshi@gmail.com

[2]Associate Prof, ECE Department
CVSR College of Engineering, Hyderabad, India
rkkittu111@gmail.com

Abstract: In this paper, we present a high performance and low cost hardware architecture for real-time implementation of forward transform and quantization and inverse transform and inverse quantization used in H.264 / MPEG4 Part 10 video coding standard. The hardware architecture is based on a reconfigurable data path with only one multiplier. This hardware is designed to be used as part of a complete low power H.264 video coding system for portable applications. The proposed architecture is implemented in Verilog HDL. The Verilog RTL code is verified to work at 81 MHz in a Xilinx Vertex II FPGA and it is verified to work at 210 MHz in a 0.18µ ASIC implementation. The FPGA and ASIC implementations can code 27 and 70 VGA frames (640x480) per second respectively.

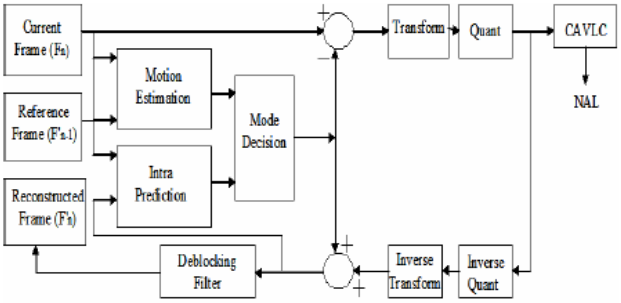
Keywords: H.264, forward transform and quantization**.**

1. Introduction

The video compression efficiency achieved in H.264 standard is not a result of any single feature but rather a combination of a number of encoding tools. As it is shown in the top-level block diagram of an H.264 Encoder in Figure 1, two of these tools are the transform and quantization algorithms. Even though most of the previous video coding standards, e.g. MPEG-1, H.261, MPEG-2, H.263 and MPEG-4, use the 8x8 Discrete Cosine Transform (DCT) to transform the residual data, H.264 uses a 4x4 integer transform for transforming residual data. The integer transform achieves very similar results to 8x8 DCT without any floating point operations. In addition, all the multiplication operations in the forward and inverse transform algorithms can be implemented in hardware with low cost binary shifters. Since the inverse transform in H.264 is defined by exact integer operations, inverse transform mismatches are avoided. Since a scaling factor is used in the quantization algorithm, a multiplier is needed for its implementation

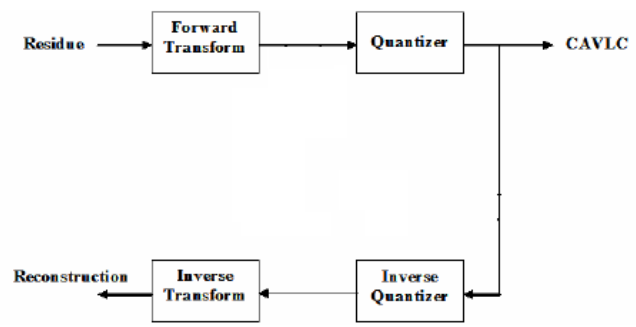
2. Proposed Algorithm

Video compression systems are used in many commercial products, from consumer electronic devices such as digital camcorders, cellular phones to video teleconferencing systems. These applications make the video compression hardware devices an inevitable part of many commercial products. To improve the performance of the existing applications and to enable the applicability of video compression to new real-time applications, recently, a new international standard for video compression is developed. This new standard, offering significantly better video compression efficiency than previous International standards, is developed with the collaboration of ITU and ISO standardization organizations. Hence it is called with two different names, H.264 and MPEG4 Part 10.



In this paper, we present a high performance and low cost hardware architecture for real-time implementation of H.264 forward transform and quantization and inverse transform and quantization algorithms. The hardware architecture is based on a reconfigurable data path with only one multiplier. This hardware is designed to be used as part of a complete low power H.264 video coding system for portable applications. The proposed architecture is implemented in Verilog HDL. The Verilog RTL code is verified to work at 81 MHz in a Xilinx Virtex II FPGA and it is verified to work at 210 MHz in a 0.18µ ASIC implementation. The FPGA and ASIC implementations can code 27 and 70 VGA frames (640x480) per second respectively.

Hardware architecture only for real-time implementation of H.264 forward and inverse transform algorithms is presented in [6]. This hardware achieves higher performance than our hardware design at the expense of a much higher hardware cost. Our hardware design is a more cost-effective solution for portable applications. They use 16 adders and 16 internal register files in their data path as opposed to 3 adders and 6 internal register files in the transform part of our data path. Their data path has an area of 6538 gates in TSMC 0.35µ technology. Our data path, on the other hand, has an area of 2904 gates in AMS 0.35µ technology.



3. Overview of H.264 Transform and Quantization Algorithms

The basic transform coding process in H.264, shown in Figure 1, is similar to that of previous standards. The process includes a forward transform and quantization followed by zig-zag ordering and entropy coding. The transform coded residual data is also reconstructed. The reconstruction process includes an inverse quantization and inverse transform followed by motion compensation. The reconstructed data before de blocking filter is used for intra prediction in current frame, and the reconstructed data after de blocking filter is used for motion estimation in future frames

A more detailed flow of the transform and quantization algorithms is presented in Figure 2. The input to the forward transform algorithm is a 4x4 block of residual data obtained by subtracting the prediction from the original image data. The transform and quantization algorithms process the blocks in a macro block as explained in the following sections, and send the resulting data to entropy coding and reconstruction as shown in the figure.

A.Transformation algorithm oveview

The purpose of the transform stage is to convert residual data into another domain (the transform domain). The choice of transform depends on a number of criteria:

a. Data in the transform domain should be de-correlated (separated into components with minimal inter-dependence) and compact (most of the energy in the transformed data should be concentrated in to a small number of values).
b. The transform should be reversible.
c. The transform should be computationally tractable (low memory requirement, achievable using limited-precision arithmetic, low number of arithmetic operations, etc.).

Discrete Cosine Transform which is a block-based transform requires low memory but tend to suffer from arte-facts at edges (blockiness) and also complex floating point multiplications are involved in its computation which decreases the computational speed. Discrete Wavelet Transform an image-based transform considers entire image as a single tile and operates on it, in this process it takes high memory as the whole image is being processed as a single unit. Integer transformation which we are going to use shares the common properties of DCT as it is developed from it. The main advantage of this is only integer values will be involved in computation process so finally we go transformation without the need of a multiplier we just fulfill our requirement with just adders and some shifters.

i. Development of Integer Transform from DCT

Discrete Cosine Transform:

$$Y = A \times X \times A^T$$

Where X is the residual matrix, A is the DCT kernel given as

Aij = Ci Cos (2j+1) iπ /2N where Ci= sqrt(1/N) for i=0; Ci=sqrt (2/N) for i>0 N-1 N-1

$Y_{xy} = CxCy \sum \sum X_{ij}$ (cos ((2j+1)y*π/2N) (cos((2i+1)x*π/2N) i=0 j=0 N-1 N-1

$X_{ij} = \sum \sum C_x C_y Y_{xy}$ (cos ((2j+1)y*π/2N) (cos((2i+1)x*π/2N) i=0 j=0

B. Quantization Algorithm Overview

1. Consider a block of pixel data that is processed by a two-dimensional Discrete Cosine Transform (DCT) followed by quantization, i.e. rounded division by a quantization step size, Qstep.
2. Rearrange the DCT process into a core transform ($\mathbf{C_{f4}}$) and a scaling matrix ($\mathbf{S_{f4}}$).
3. Scale the quantization process by a constant (2^{15}) and compensate by dividing and rounding the final result. The constant factor 2^{15} is chosen as a compromise between higher accuracy and limited arithmetic precision. Combine $\mathbf{S_{f4}}$ and the quantization process into $\mathbf{M_{f4}}$, where:

$$\mathbf{M_f} \approx \frac{\mathbf{Sf} \cdot 2^{\mathbf{qb}}}{Qstep} \qquad \text{---------- (1)}$$

M_f is actually derived from V_i (from eq: 2)

Development of Rescaling and Inverse Transform Process:

- Consider a re-scaling or 'inverse quantization' operation followed by a two-dimensional inverse DCT (IDCT).
- Rearrange the IDCT process into a core transform (**Ci**) and a scaling matrix ($\mathbf{S_i}$)
- Scale the re-scaling process by a constant (2^6) and compensate by dividing and rounding the final result Note that rounding need not be to the nearest integer.
- Combine the re-scaling process and $\mathbf{S_i}$ into $\mathbf{V_i}$ where:

$$\mathbf{V_i} \approx \mathbf{S_i} \cdot 2^6 \cdot Q_{step} \qquad \text{---------- (2)}$$

Developing C_{f4} and S_{f4} (for 4 X 4 blocks):

Consider a 4 X 4 two dimensional DCT of a block X:

$$\mathbf{Y = A \cdot X \cdot A^T} \qquad \text{---------- (3)}$$

Where · indicates matrix multiplication and

$$A = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix},$$

$a = 1/2$

$b=\sqrt{1/2}\cos\pi/8=0.6532\ldots.$

$c=\sqrt{1/2}\cos 3\pi/8 =0.2706\ldots$

The rows of **A** are orthogonal and have unit norms (i.e. the rows are orthonormal).Calculation of Eq (3) on a practical processor requires approximation of the irrational numbers b and c. A fixed-point approximation is equivalent to scaling each row of **A** and rounding to the nearest integer. Choosing a particular approximation (multiply by 2.5 and round) gives $\mathbf{C_{f4}}$:

$$\mathbf{C_{f4}} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix}$$

This approximation is chosen to minimize the complexity of implementing the transform (multiplication by $\mathbf{C_{f4}}$ requires only additions and binary shifts) whilst maintaining good compression performance.

The rows of $\mathbf{C_{f4}}$ have different norms. To restore the orthonormal property of the original matrix **A**, multiply all the values cij in row r by $\frac{1}{\sqrt{\sum_j c_{ij}^2}}$

Then the matrix computation in eq(3) can be factorized to the following equivalent form

$$\mathbf{Y} = [\mathbf{C_{f4}} \cdot \mathbf{X} \cdot \mathbf{C_{f4}^T}] \bullet \mathbf{S_{f4}}$$

Where • denotes element by element multiplication where **P=Q•R means** that each element pij = qij·rij and

$$S_{f4} = \begin{bmatrix} 1/4 & 1/2\sqrt{10} & 1/4 & 1/2\sqrt{10} \\ 1/2\sqrt{10} & 1/10 & 1/2\sqrt{10} & 1/10 \\ 1/4 & 1/2\sqrt{10} & 1/4 & 1/2\sqrt{10} \\ 1/2\sqrt{10} & 1/10 & 1/2\sqrt{10} & 1/10 \end{bmatrix}$$

Since the scaling matrix S_{f4} could be merged in to quantization process, then the core transform ($\mathbf{C_{f4}} \cdot \mathbf{X} \cdot \mathbf{C_{f4}^T}$) become 4X4 forward integer transform without multiplications.

Deriving $\mathbf{M_{f4}}$:

The core inverse transform $\mathbf{C_i}$ and the rescaling matrix $\mathbf{V_i}$ are defined in the H.264 standard. Hence now we will derive $\mathbf{M_f}$ from V_i. H.264 supports a range of quantization step sizes Q_{step} . The precise step sizes are not defined in the standard, rather the scaling matrix V_i is specified. Q_{step} values corresponding to the entries in $\mathbf{V_i}$ are shown in the following Table:

QP	0	1	2	3	4	5	6	..	12	..	18	..	48	..	51
Q_{step}	0.625	.702	.787	.884	.992	1.114	1.25	..	2.5	..	5	..	160	..	224

Table 1: QP vs Q_{step}

The ratio between successive Q_{step} values is chosen to be $\sqrt[6]{2} = 1.2246$ so that Q_{step} doubles in size when QP increases by 6. Any value of Q_{step} can be derived from the first 6 values in the table (QP0 – QP5) as follows:

$Q_{step}(QP) = Q_{step}(QP\%6) \cdot 2^{floor(QQP/6)}$ The values in the matrix $\mathbf{V_{i4}}$ depend on Q_{step} and hence QP and on the scaling factor matrix $\mathbf{S_{i4}}$. These are shown for QP 0 to 5 in Table below:

QP	v (r, 0): V_{i4} positions (0,0), (0,2), (2,0), (2,2)	v (r, 1): Vi4 positions (1,1), (1,3), (3,1), (3,3)	v (r, 2): Remaining Vi4 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

Table 2: V_i Defined in h.264 standard

Combining equations 1 and 2 we get Mf4 $\approx \frac{S_{i4} \cdot S_{f4} \cdot 2^{2n}}{V_{i4}}$

S_{i4}, S_{f4} are known and V_i is defined in the standard and we get M_{f4} as:

$$\mathbf{M_{f4}} = \begin{bmatrix} m(QP,0) & m(QP,2) & m(QP,0) & m(QP,2) \\ m(QP,2) & m(QP,1) & m(QP,2) & m(QP,1) \\ m(QP,0) & m(QP,2) & m(QP,0) & m(QP,2) \\ m(QP,2) & m(QP,1) & m(QP,2) & m(QP,1) \end{bmatrix}$$

Table 3: M_i Defined in h.264 standard

Denoting this as $\mathbf{M_{f4}} = m(QP,n)$ Where $m(r,n)$ is row r, column n of v.

For values of QP>5, index the row of array v by QP%6 and then multiply by 2floor (QP/6).
In general:

$M_{f4} = m(QP\%6, n) / 2^{floor(QP/6)}$

Where $m(r,n)$ is row r, column n of v.

The complete forward transform, scaling and quantization process for 4 × 4 blocks

$$\mathbf{Y} = \text{round } (\mathbf{[C_{f4}] \cdot [X] \cdot [C^T_{f4}] \bullet m(QP\%6, n)} \cdot (1/2^{15+floor(QP/6)}))$$

4. Conclusion

Hence the design and implementation of Forward Integer Transformation and Inverse Integer Transformation are done and the entire modules are simulated using Xilinx 12.2 version. There was no change in the output of the forward

integer transformation results compared to Mat lab results. In case of Inverse Integer Transformation as we have taken approximations, we got a PSNR of 40.44dB. Development of Integer transformation from Discrete Cosine Transformation has been studied and later development & implementation of Integer transformation architecture is done in Xilinx ISE using SPARTAN 3E family, XC3S1200E device, FG400 package. We got the maximum frequency of 138.22 MHz for this architecture.

References

[1] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra "Overview of the H.264/AVC Video Coding Standard", IEEE Trans. on Circuits and Systems for Video Technology vol. 13, no. 7, pp. 560–576, July 2003

[2] I. Richardson, H.264 and MPEG-4 Video Compression, Wiley,

[3] 2003

[4] Joint Video Team (JVT) of ITU-T VCEG and ISO/IEC MPEG, Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification, ITU-T Rec. H.264 and ISO/IEC 14496-10 AVC, M

[5] Joint Video Team (JVT) of ITU-T VCEG and ISO/IEC MPEG, Joint Model (JM) Reference Software Version 9.2,

[6] http://bs.hhi.de/suehring/

[7] H. Malvar, A. Hallapuro, M. Karczewicz. and L. Kerofsky, "Low-Complexity Transform and Quantization in H.264 / AVC",

[8] IEEE Trans. on Circuits and Systems for Video Technology, vol. 13, no. 7, pp. 598–603, July 2003.

[9] T. C. Wang, Y. W. Huang, H. C. Fang, and L. G. Chen, "Parallel 4x4 2D Transform and Inverse Transform Architecture for MPEG-4 AVC / H.264", Proc. of IEEE ISCAS, 20

[10] Xilinx Inc., Virtex-II™ Platform FPGAs: Complete Data Sheet

[11] DS031, http://www.xilinx.com, March 2004

Authors Profile

G. Dileepvamshi received his B. Tech degree in Electronics and Communication Engineering from Swami Ramananda Thirtha Inst of Science and Technology, JNTU University, Hyderabad in 2009, and currently pursuing the M. Tech degree in VLSI System Design from CVSR College of Engineering, JNTU University, Hyderabad respectively. His research interests include Image Processing and H.264 video compression.

P. Ramakrishna received the B. Tech degree in Electronics and Communication Engineering from NIT, Warangal in 2006, and the M. Tech degree in VLSI System Design from CVR College of Engineering, JNTU University, Hyderabad in 2009 respectively. He is currently an Associate Professor with the CVSR College of Engineering with 6 years of experience. He was with the Department of Electronics and Communication Engineering, JNTU University. His research interests include image processing and H.264 video compression.