Implementation of Different Architectures of Forward 4x4 Integer DCT For H.264/AVC Encoder

Bunji Antoinette Ringnyu[1], Ali Tangel[2], Emre Karabulut[3]

[1]Kocaeli University, Institute of Science and Technology, Kocaeli, Turkey
bunjiantoinetteringnyu@gmail.com
[2]Kocaeli University, Department of Electronics and Communication Engineering, Kocaeli, Turkey
atangel@kocaeli.edu.tr
[3]YONGATEK, Teknopark Istanbul, Turkey
emre.karabulut.dde@gmail.com

Abstract

This paper presents an overview and different implementations of the 4x4 Integer Discrete Cosine Transform (DCT) used for the H.264 standard, also presenting the Utilization Report and the Maximum Operating Frequency for each implementation. The H.264 standard specifies the use of Integer DCT to decompose the results of inter prediction and intra prediction from spatial to Frequency Domain. We implemented the 2-D direct multiplication, the 1-D (butterfly) method and the 2-D multiplication with adders. With these implementations, the results show that 2-D with adders and 1-D implementations outperform the direct 2-D multiplication. However, the 2-D with adders uses fewer resources than the 1-D butterfly and still achieves a relatively high frequency.

Keywords—H.264/AVC, Integer DCT, Image Compression, VHDL

1. Introduction

Since the 1990s, image compression has experienced a significantly high progress. From the H.261 and JPEG used in the 1980s, to the MPEG-1 and MPEG-2 used in the 1990s. With the popularity internet got in the 2000s, standards like MPEG-4, H.264 or MPEG-Part 10/AVC, MP4 and HEVC sprung up. Of all the standards mentioned above, H.264 standard is the most widely used codec. Compared to previous standards, H.264 provides a higher compression of about 50% over a wide range of bit rates and high video resolutions. Although its coding algorithms are based on the same block-based motion compensation and transform based spatial coding framework of prior video coding standards, the H.264 has many innovations compared to the older standards [1]; such as hybrid predictive/transforms coding of intra frames and integer transforms. Same as the existing standards, AVC encoding is accomplished through many blocks which include Motion Estimation and Motion Compensation, Intra prediction, Transform and Quantization, Inverse Transform and Quantization, and Entropy Encoder. Fig. 1 shows a block diagram of H.264/AVC encoder scheme [1].

The Motion Estimation module is used to identify and eliminate temporary redundancies that exist between individual frames. It involves use of motion vectors that describes the transformation of the video /image from one dimension to the next. Motion vectors may be applied to the whole image in which case we have global motion estimation or on parts of the image in which it becomes local motion estimation or even per pixel Motion Compensation (MC) will decode the image that is encoded by Motion Estimation [2], [3]. The input to the inter prediction and intra prediction blocks are macroblocks. These blocks are encoded in either inter or intra mode. In inter mode, prediction is formed by motion-compensated prediction or two reference picture(s) selected from the set of list 0 and/or list 1 reference pictures [4]. In instances where motion estimation cannot be exploited, intra mode is used to eliminate spatial redundancies by attempting to predict the current block by extrapolating the neighboring from adjacent blocks in a defined set of adjacent directions.

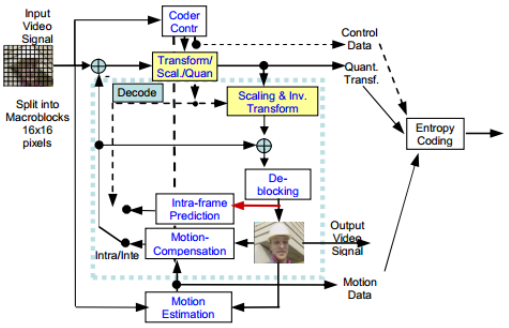

Fig. 1. Diagram of AVC encoding scheme. [3]

To decompose the results of inter prediction and intra prediction from spatial to Frequency Domain, integer DCT is used. This is usually achieved through the use of 4X4 DCT. In the case of a macroblock coded in 16×16 intra prediction mode, its luma pixels are first transformed using the 4×4 DCT and as a second step, a gathered 4×4 DC coefficients block is transformed again using a 4×4 Hadamard transform [5].The coefficients from the Transform block are quantized to remove unimportant information, allowing only significant coefficients for representing the residual frame. At the level of the transformation block, over all precision of integers coefficients are reduced, leading to the elimination of high frequency coefficients. There are several papers [6, 7, 9, 10] discussing hardware implementation of Transform and Quantization.

In this paper we will focus on the Transform block. The core transform matrix is a 4x4 matrix which can be implemented using 2-D matrix multiplication or the popular butterfly algorithm which is 1-D using additions, subtractions and shift operations along rows then along columns. In addition to the two methods used above, implement the 4x4 DCT using 2-D multiplications with addition operations only and the final results from the three architectures compared to see which of them uses less resources. These architectures are implemented using VHDL. The rest of the paper is made of section 2, which presents an overview of H.264 Transform algorithm. Section 3 presents the implementation of the three different architectures for the 4x4 Integer DCT Block. The synthesis and results are presented in section 4. The paper is concluded with section 5.

2. Overview of H.264 Transform

In the AVC standard, the residual frame of the prediction, which is the difference of the original frame and the predicted frame, is partitioned into fixed-size of macroblocks. A macroblock is composed of 16×16 luminance(*Y*) samples, 8×8 chroma blue(*Cb*) samples, and 8×8 chroma red(*Cr*) samples in the case of 4:2:0 chroma subsampling format. A bock diagram of these three sample blocks is shown in Fig. 2

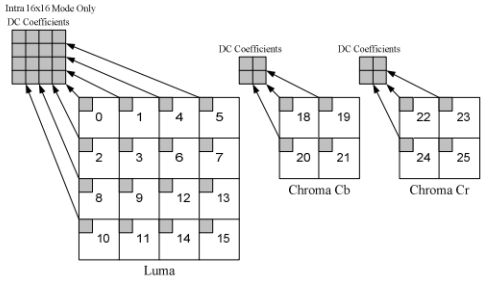

Fig. 2. Processing order of blocks in a macroblock[3]

At a sub-macroblock level, macroblocks are subdivided into sub-blocks of 4×4 samples for encoding. Due to the 3 different samples, there are three different transforms used in H.264/AVC. According to the block diagram of H.264 transform component as illustrated in Fig. 4, the residue is transformed using integer DCT. In the 16 × 16 Intra-prediction mode, DC coefficients of all transformed residual blocks are grouped into an array of 4× 4 before being sent to Hadamard transform. Details of these processes are described in mathematical models in section 2.1.

2.1 4 × 4 Forward Transforms

DCT has been used in both previous standards (like the 8x8 DCT) and existing standards of image compression (like the Integer DCT used in H.264 and H.265). The H.264/AVC is based on 4x4 Integer DCT

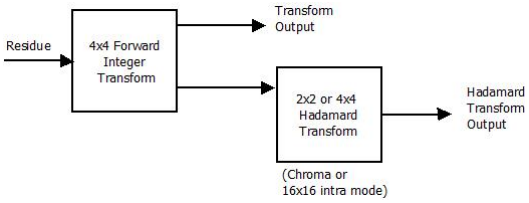

Fig. 3. Block diagram of H.264 Transform Component

that can be computed exactly with integer arithmetic in order to avoid inverse transform mismatch problems. There are two types of 4x4 integer transforms for the residual coding. The first one is for luminance residual blocks and is described by (1) [11].

$$Y = CXC^T \qquad (1)$$

Where X is the 4x4 residual input of the Transform block and C is specified by

$$C = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & \_c \end{bmatrix}$$

Where:

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

This can be factorized as

$$Y_I = (CXC^T) \otimes E \qquad (2)$$

Where:

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

Where E is a matrix of scaling factors. The symbol ⊗ means that each component of CXC^T is multiplied by the corresponding coefficient in E. To reduce hardware implementation of the transform, the constant d is approximated by 0.5 and the constant b by √2/5. The final forward transform expression becomes [3], [12]:

$$Y_2 = (C_f X C_f^T) \otimes E_f \qquad (3)$$

Where:

$$C_f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \qquad E_f = \begin{bmatrix} a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \\ a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \end{bmatrix}$$

So, the scaling matrix E_f can be incorporated into the quantization process. Then $C_f X C^T_f$ becomes the core of a 2-D integer forward transform. This can be computed by using two 1-D transforms. The first 1-D is applied to the rows of the incoming residue. The second 1-D is then applied on the columns. This is what is popularly known as butterfly algorithm. Since the 4x4 transform matrix has only 1, -1, 2, -2 as coefficients, its butterfly is as shown below:

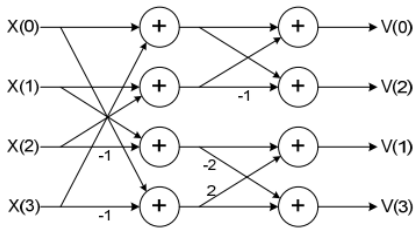

Fig. 2. Butterfly Diagram of 4x4 Integers DCT.

2.2 4 × 4 Hadamard Transform

The other kind of transform is Hadamard Transform (HT). It is applied to the luminance DC terms in 16x16 intra prediction mode. The Hadamard transform is defined by equation (4)

$$Y = HXH^T \qquad (4)$$

Where:

$$H_f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$$

The Hadamard transform matrix is very similar to the Forward transform matrix with the only difference being, the 2 in forward transform is being replaced with a 1 in the Hadamard transform. The butterfly of the Hadamard is as shown below :

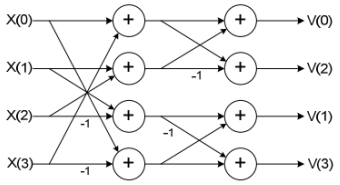

Fig. 2. Butterfly Diagram of 4x4 Integers DCT.

2. Implementation of 4x4 DCT Transform

There are several papers discussing on the VLSI implementation of 2-D integer transform for H.264.Thus, implementation of fast 2-D transform can be classified into two categories: row/column decomposition (1-D) approach and direct 2-D approach. Though direct 2-D requires more resources, it is implemented to be used for comparison with other architectures. Also, the 2-D is implemented with full adders only. Then, the Butterfly is also implemented.

3.1 Direct 2-D Multiplication

This is implemented by using normal Matrix multiplication with a Finite State Machine (FSM). This FSM consists of the states INITIALIZATION, then MULT1 which performs the first matrix multiplication of equation (1) which is $C*X$ and finally, state MULT2 which performs the second transform. All the other architectures are implemented using similar states.

3.2 2-D Multiplication with Adders

Unlike performing multiplication directly, this architecture is implemented by replacing multiplications (*) with full adders. This is accomplished with the help of the concatenation operator in VHDL.

3.3 Implementation of 2-D-Transform using 1-D approach.

This is accomplished by firstly performing the butterfly algorithm on the rows. The transpose of the resulting out is taken and the send butterfly algorithm is taken. This second butterfly is performed on the columns. A final transpose is taken to obtain the required output. The block diagram of this stage is as shown below:

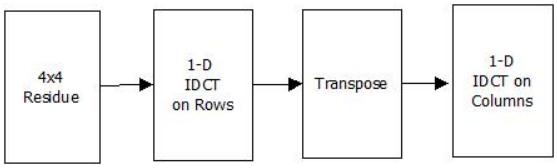

Fig. 2. Block of the Integer DCT using the 1-D approach.

4. Results and Discussions

The implementation is done both with VHDL and Matlab as explained below. Firstly, the Residue values are generated in Matlab and written to a dumper (a text file). The Residue values are then read from the dumper to Matlab and VHDL, then the algorithms executed. The results from the Transform block are again dumped to two separate text files and lastly, compared to be sure that the results were the same. The whole process of generating Residue values to comparing is as depicted in Fig. 7. This process was repeated for different sets of data, just to confirm the algorithms were working correctly and no bits were lost especially in VHDL. Finally, a synthesis was done to view the Utilization Analysis and Maximum Frequency supported by each architecture. The results are presented in Table 1. The 2-D direct multiplication does the operation with multipliers using 1210 Slice LUTs and 451 Slice Registers. The architecture also has a maximum operating frequency of 125 MHz. This confirms the reasons why multipliers are

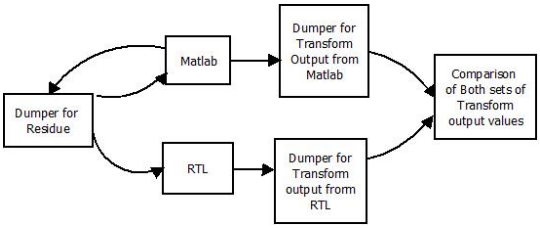

Fig. 2. Block diagram of the synthesis and simulation processes

discouraged in the implementation of Integer DCT. They use more resources and operate at low frequencies .The 1-D just as expected uses less resources (661 Slice LUTs and 730 Slice Registers) and maximum frequency of 200MHz). The 2-D with adders on the other hand achieves the same maximum frequency as the 1-D method (200 MHz), using even the least resources (794 Slice LUTs and 411 Slice Registers).

Table 1. Synthesis Report of the Three Different Architectures

Architecture	Slice LUTs	Slice Registers	Maximum operating Frequency
Direct 2-D Multiplication	1210	451	125MHz
2-D Multiplication With Adders	794	411	200MHz
The 1-D Approach	661	730	200MHz

The use of 2-D integer DCT is generally not advisable because for the hardware resources they consume. But it can be seen from the table above that, 2-D integer DCT implemented with full adders can still be used in H.264/AVC encoders.

5. Conclusion

This paper has given the Maximum Frequency and Resource usage improvements for the 2-D Integer DCT in a H.264 encoder. The 2-D with adders and 1-D architectures achieved a higher Maximum frequency (about 33.3 %), which is higher than the 2-D with multipliers. Even though the two architectures have relatively higher frequencies, the 2-D with adders uses less hardware resources. With this maximum frequency achieved and low hardware utilization capabilities, the 2-D architectures with adders can also be used in systems where the 1- D architecture especially in real-time applications such as mobile communication and video broadcasting. Even though the 2D implementation is not always used, the one implemented here can still be used in low frequency systems like Video Compression for storage devices like DVDs, and Digital signal processing.

6. Acknowledgement

We will like to thank Mr. Muhammed Aslam for his constant support during this project and the entire staff of YONGA TEK, TEKNOPARK Istanbul, for providing a suitable environment for this research.

7. References

[1] Meihua Gu et al, “Hardware Prototyping for Various Transforms in H.264 High Profile”, Journal of Information & Computational Science, 2011, pp. 119–128.

[2] Thomas Wiegand and Gary J. Sullivan ,” The H.264/MPEG4 Advanced Video Coding Standard and its Applications”, IEEE SIGNAL PROCESSING MAGAZINE, Agust 2006, pp. 134-143.

[3] H.S.Marver, A.Hallapuro, M.Karczewicz and L.kerofsky, “Low Complexity Transform and Quantisation in H.264/AVC” IEEE Transactions on circuits and systems for video technology, vol. 13, No 7, July, 2003, pp. 560-576.

[4] I.E.G Richardson, “H.264 and MPEG-4 Video Compression”,published by John Wiley and sons, West Sussex, UK, 2003

[5] Meihua Gu et al, “Hardware Prototyping for Various Transforms in H.264 High Profile”, Journal of Information & Computational Science, 2011, pp. 119–128.

[6] H.S.Marver, A.Hallapuro, M.Karczewicz and L.kerofsky, “Low Complexity Transform and Quantisation in H.264/AVC” IEEE Transactions on circuits and systems for video

technology, vol. 13, No 7, July, 2003, pp. 560-576.

[7] Meihua Gu et al, “Hardware Prototyping for Various Transforms in H.264 High Profile”, Journal of Information & Computational Science, 2011, pp. 119–128.

[8] H. Kalva and J.B.Lee, “The VC-1 and H.264 Video Compression Standards for Broadband Video Services”, Springer, New York, USA, 2008.

[9] Charles S. Lubobya, Mqele M. Dlodlo, Gerhard De. Jager, and Keith L.Ferguson, “Optimization of 4x4 Integer DCT in H.264/AVC Encoder”, Council for Scientific and Industrial Research.

[10] Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification, ITU-TRec.H.264 and ISO/IEC14496-10 AVC,2003.

[11] I.E.G.Richardson,“H.264 and MPEG4 Video Compression-Video Coding for Next Generation Multimedia”,NewYork:Wiley,2003.

[12] Xuan-Tu Tran and Van-Huan Tran, “An Efficient Architecture of Forward Transforms and Quantizationfor H.264/AVC Codecs” , Journal on Electronics and Communications, Vol. 1, No. 2, April – June, 2011, pp. 122- 129.