The Eurasia Proceedings of Science, Technology, Engineering & Mathematics (EPSTEM)
ISSN: 2602-3199

The Eurasia Proceedings of Science, Technology, Engineering & Mathematics (EPSTEM)

Volume 1, Pages 353-358

ICONTES2017: International Conference on Technology, Engineering and Science

IMPLEMENTATION OF FORWARD 8x8 INTEGER DCT FOR H.264/AVC FREXT

Bunji Antoinette Ringnyu
University of Kocaeli

Ali Tangel
University of Kocaeli

Abstract: The H.264/AVC image encoding standard has been used in many systems, especially in HD(High Definition) devices because of the introduction of the FREXt standard, which leads to additional characteristics in this standard like the Higher Resolution and Higher Bit rates. With the introduction of the FREXt, a good number of amendments are added to the AVC standard, most importantly at the level of the transform block. In addition to the 4x4 Integer DCT, there is an 8x8 Integer DCT (Discrete Cosine Transform) matrix. This work focuses on the Forward 8x8 Integer Transform block implementation of the H.264 FREXt standard, exploring different methods of implementations, and examining how these methods affect the hardware and the maximum frequency. There is the 2D implementation (Matrix multiplication) using multipliers and adders and the 1D implementation(butterfly algorithm) using adders. These implementations are done using VHDL and MATLAB. The simulations are done in Vivado Design Suite.

Keywords: H.264 FREXt, integer DCT, image compression, VHDL

Introduction

The H.264/AVC standard, established by the Joint Video Team ITU-T VCEG and ISO/IEC MPEG is still the most used standard. Even though the H.265/HEVC has already been introduced, the AVC standard is still being used on mobile devices, video conferencing, multimedia streaming services and many other applications. Just like other standards, compression is accomplished through many steps in AVC. This is accomplished through a number of blocks which are Motion Estimation(ME), Motion Compensation(MC), Inter Prediction, Intra Prediction, Forward Transform, Forward Quantization, Inverse Quantization, Inverse Transform, Entropy coding, and Deblocking filter [1]. The block diagram for this AVC encoder is shown in Fig 1.

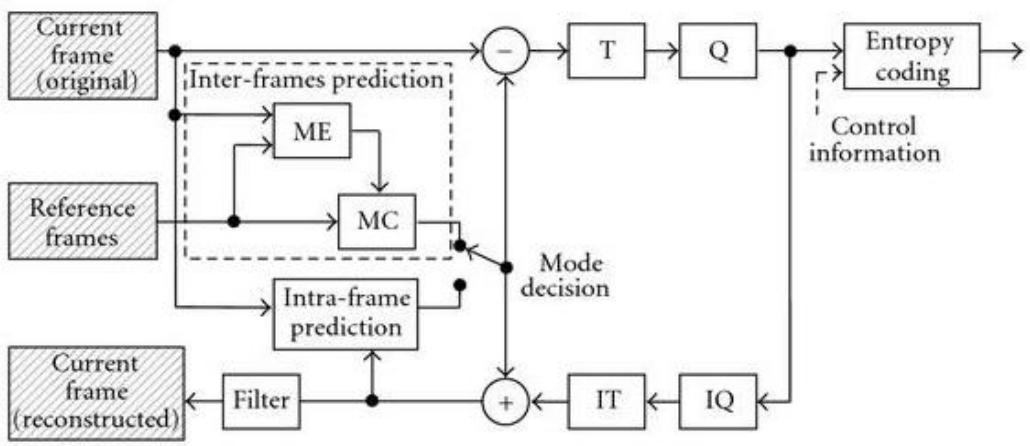

Figure 1.The block diagram of H.126/AVC encoder [2]

- Selection and peer-review under responsibility of the Organizing Committee of the conference

*Corresponding author: Bunji Antoinette Ringnyu -E-mail: bunjiantoinetteringnyu@gmail.com

The Motion Estimation (ME) 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 or 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 [3], [4].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 pictures. 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. The results of the inter prediction and intra prediction blocks are in the spatial domain, and the conversion of these results to the frequency domain is done at the level of the transform block, using Integer DCT. This is achieved with 4x4 Integer DCT and 4x4, 2x2 Hadamard Matrices for non-FRExt H.264 and 8x8DCT for FRExt H.264. The out of the transform block enters the quantization block where unimportant information is eliminated. The deblocking filter is the used to reduce blocking artifacts without reducing sharpness of the video [5]. The final output can then be encoded using encoders like the CABAC.

Some amendments and additions were applied to the H.264 to develop what was known as the H.264 FRExt to accommodate services like content distribution, content-contribution and studio editing. This extension has characteristics such as higher resolution, higher bit rates, very high fidelity, and RGB color representation. The features added to achieve the characteristics mentioned above included supporting an adaptive block-size for the residual spatial frequency transform, supporting encoder-specified perceptual-based quantization scaling matrices, and supporting efficient lossless representation of specific regions in video content [6]. The main difference between the H.264 FRExt and Non-FRExt is the use of 8x8 integer DCT, which is an approximation for the 8x8 2-D Discrete Cosine Transform as well as the original 4x4 and 2x2 matrices. This work is based on the implementation of the 8x8 Transform block using both 2-D methods and 1-D methods, with their hardware and frequencies. The rest of the paper consist of the overview of the Transform in H.264, the implementation, results, and conclusion.

Transform in H.264/Avc

The transform block converts residuals obtained from the spatial domain to the frequency domain. This is usually done using the equation below:

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

where X is the residual input of the transform block and C is the transform.
This equation is used for both 4x4 Integer DCT and 8x8 DCT. The coefficients of these matrices are integers. Due to this fact, Integer DCT can be implemented with shift adders and full adders. One of the advantages of the all integer coeffiecient aspect is the introduction of the butterfly algorithm generally known as the 1D implementation .

4x4 Integer DCT

For the Integer 4x4 DCT, the Transform matrix, (X) is given as:

$$C_{4x4} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix}$$

Equation (1) can be implemented using the 2D method (Matrix multiplication) and 1D implementation using the butterfly algorithm. The First 1D is applied to the rows and the second transform is applied to the columns. The 1D algorithm is illustrated in Table.1

Table 1. 4x4 forward 1D transform algorithm for H.264

Stage 1	Stage 2
Y(0) = X(0) + X(3)	V(0) = Y(0) + Y(1)
Y(1) = X(1) + X(2)	V(2) = Y(0) – Y (1)
Y(2) = X(1) – X(2)	V(1) = Y(2) + (Y(3)<<1)
Y(3) = X(0) – X(3)	V(3) = Y(3) – (Y(2)<<1)

8x8 Integer DCT

The 8x8 Integer DCT is also implemented using Equation (1). In this case, the transform matrix C is given by:

$$C_{8x8} = \begin{pmatrix} 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\ 12 & 10 & 6 & 3 & -3 & -6 & -10 & -12 \\ 8 & 4 & -4 & -8 & -8 & -4 & 4 & 8 \\ 10 & -3 & -12 & -6 & 6 & 12 & 3 & -10 \\ 8 & -8 & -8 & 8 & 8 & -8 & -8 & 8 \\ 6 & -12 & 3 & 10 & -10 & -3 & 12 & 6 \\ 4 & -8 & 8 & -4 & -4 & 8 & -8 & 4 \\ 3 & -6 & 10 & -12 & 12 & -10 & -6 & 3 \end{pmatrix} .1/8$$

Just like the 4x4 DCT, the 8X8 DCT can be implemented using both the 2D method (matrix multiplication) and the 1D method (the butterfly). Since the butterfly algorithm includes right shift operators, which can lead to loss of some information, it is applied in such a way that mismatch between the encoder and decoder is avoided. Unlike the ID for the 4x4 Integer DCT that is accomplished in two stages, the 1D for the 8x8 integer DCT is accomplished in 3 stages, and this is illustrated in Table 2.

Table 2. 8x8 forward 1D transform for H.264

Stage 1	Stage 2	Stage 3
Y(0) = X(0) + X(7) Y(1) = X(1) + X(6) Y(2) = X(2) + X(5) Y(3) = X(3) + X(4) Y(4) = X(0) - X(7) Y(5) = X(1) - X(6) Y(6) = X(2) - X(5) Y(7) = X(3) - X(4)	V(0) = Y(0) + Y(3) V(1) = Y(1) + Y(2) V(2) = Y(0) - Y(3) V(3) = Y(1) - Y(2) V(4) = Y(5) + Y(6) + ((a4>>1) + a4) V(5) = Y(4) - Y(7) - ((a6>>1) + a6) V(6) = Y(4) + Y(7) - ((a5>>1) + a5) V(7) = Y(5) - Y(6) - ((a7>>1) + a7)	Z(0) = V(0) + V(1) Z(1) = V(4) + (V(7) >>2) Z(2) = V(2) + (V(3) >>1) Z(3) = V(5) + (V(6) >>2) Z(4) = V(0) - V(1) Z(5) = V(6) - (V(5) >>2) Z(6) = (V(2)>>1) - V(3) Z(7) = -V(7) - (V(4)>>2)

Implementation

This work includes 3 different implementations of 8x8 integer DCT, which are 2D implementation using multipliers, 2D implementation using full adders and 1D (Butterfly) implementation using Full adders. The multiplier based implementation is generally discouraged because of the amount of area the chip occupies. In this case, it is used for demonstrative purposes.

2D implementation using multipliers

This is implemented as a normal 2D multiplication using Finite State Machines(FSM). This FSM consists of the states INITIALIZATION, then TRANSFORM1 which performs the first matrix multiplication of equation (1) which is *C*X*, and finally, state TRANSFORM2 which performs the second transform.

2D implementation using full adders

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

1-D implementation

This is accomplished by firstly performing the butterfly algorithm on the rows. The transpose of the resulting output is carried out, and this second butterfly is performed on the columns. A final transpose is taken to obtain the required output.

Results and Discussions

Implementation was done using both VHDL and MATLAB to ensure that the algorithm works as expected. The residue values were generated in MATLAB and written to a dumper (text file), then read by both the MATLAB and VHDL programs. This is used to ensure that, the VHDL program could handle the overflow. Then, the results of the algorithms were sent to dumpers and finally, compared to make sure that the results were the same, though the results of the 1D implementation were slightly different due to right shifts. The block diagram in Fig.2 shows an illustration of the realized process. The synthesis report of the 3 different architectures is presented in Table.3.

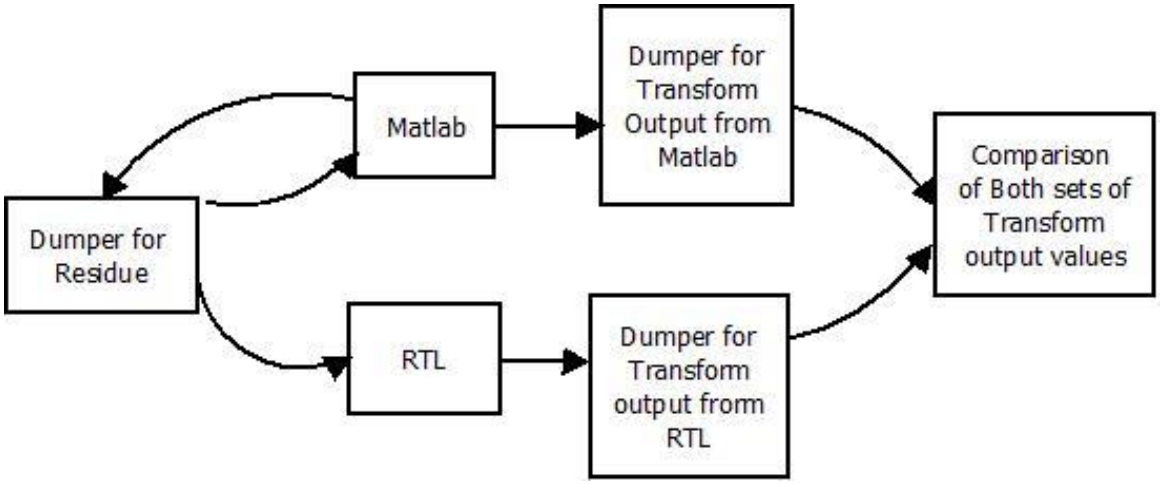

Figure 2. Block diagram of the synthesis and simulation processes

The simulation results for one of the sets of data are presented in Fig.3 (2D with Multiplication), Fig.4 (2D results with adders) and Fig.5 (1D implementation using adders).

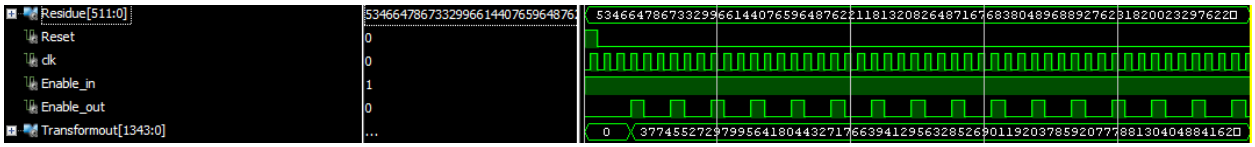

Figure 3. Waveform of the simulation results of the 2D implementation using multipliers

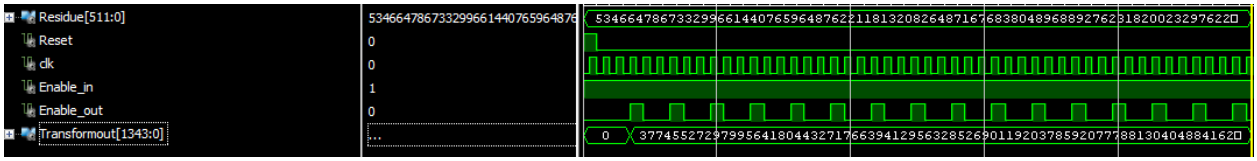

Figure 4. Waveform of the simulation results of the 2D implementation using adders

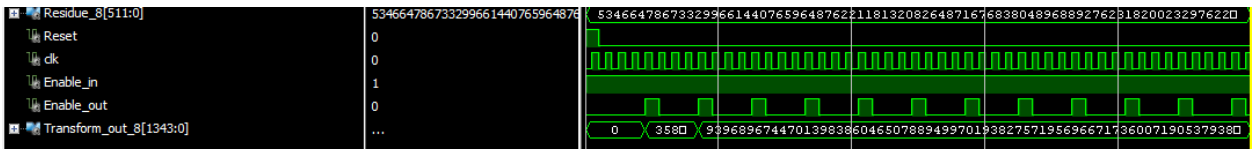

Figure 5. Waveform of the simulation results of the 1D implementation using adders

The results are the same except for the 1D implementation which has some slight differences. This is due to the right shifts in the butterfly algorithm, but the algorithm is designed to avoid mismatch errors. From the synthesis report, it can be seen that the architceture designed with multipliers occupies the largest area and the architecture with butterfly occupies less area. Also, the architecture implemented with butterfly has the highest maximum

operating frequency of 200MHz while the architecture with multipliers has least maximum operating frequencyof 83MHz. The 2D implemenation with adders has a maximum operation frequency of 100MHz, which is the highest but is still good enough for some devices.

Table 3. Synthesis report for the three different architectures

Architecture	CLB LUTS	CLB Registers	CARRY8	Maximum Operating Frequency
8x8 with Multipliers	19531	2091	3180	83MHz
8x8 with adders	14080	2168	1493	100MHz
8x8 with Butterfly	6027	4917	816	200MHz

Conclusion

From the results presented in Table 3, it can be seen clearly why using multipliers to implement transform block is greatly discouraged. This is because of the large area the architectures usually occupy, and consequently the low maximum frequency at which they operate. The importance of using adders in implementations can be seen from the results with 2D implementation with adders, with 20.5% increase in the maximum operating frequency and 28.4% decrease in the area. Also, the importance of using the butterfly algorithm with full adders is evident with the largest maximum frequency and lowest area achieved. Even though the 2D implementation is not always used, the one implemented here can still be used in low frequency systems like Video Compression systems.

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.

References

Gustavo A. Ruiz and Juan A. Michell 2011, Variable Bit-Depth Processor for 8×8 Transform and Quantization Coding in H.264/AVC, Recent Advances on Video Coding, Dr. Javier Del Ser Lorente (Ed.), InTech, DOI: 10.5772/16251. Available from: https://www.intechopen.com/books/recent-advances-on-video-coding/variable-bit-depth-processor-for-8-8-transform-and-quantization-coding-in-h-264-avc

M. Corrêa Marcel, T. Schoenknecht Mateus, S. Dornelles Robson, and Agostini Luciano 2011, A High-Throughput Hardware Architecture for the H.264/AVC Half-Pixel Motion Estimation Targeting High-Definition Videos, International Journal of Reconfigurable Computing.

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

H.S.Marver,A.Hallapuro,M.KarczewiczandL.kerofsky 2003, Low Complexity Transform and Quantisation in H.264/AVC, IEEE Transactions on circuits and systems for video technology,vol.13,No 7, pp. 560-576.

Gulistan Raja and Muhammad Javed Mirz 2006, In-loop Deblocking Filter for H.264/AVC Video, Second Internation Symposium on Communication and Signal Processsing.

Gary J. Sullivan, Pankaj Topiwala, Ajay Luthra 2004, The H.264/AVC Advanced Video Coding Standard:Overview and Introduction to the Fidelity Range Extensions, SPIE Conference on Applications of Digital Image Processing XXVII.

Detlev Marpe,Thomas Wiegand, and Stephen Gordon 2005, H.264/MPEG4-AVC Fidelity Range Extensions:Tools, Profiles, Performance, and Application Areas, International Conference on Image Process, Italy.

Hemika and Poornima Sharma 2015, A Review on an Efficient Implementation of H.264 Video Encoder DCT Transform and Quantization, Journal of Basic and Applied Engineering Research.

Ihab Amer, Wael Badawy, and Graham Jullien 2005, A High-Performance Hardware Implementation Of The H.264 Simplified 8x8 Transformation And Quantization, The International Conference on Acoustics, Speech, and Signal Processing.

Jia Su, Qin Liu, Satoshi Goto, Takeshi Ikenaga 2005, 8x8 Transformation Based All Zero Block Detection for H.264/AVC Encoder, The 23rd International Technical Conference on Circuits/Systems Computers and Communications (ITC-CSCC2008).

MarcelM. Corrˆea,Mateus T. Schoenknecht, Robson S. Dornelles, and Luciano V. Agostini 2011, A High-Throughput Hardware Architecture for the H.264/AVC Half-Pixel Motion Estimation, International Journal of Reconfigurable Computing.

Michael Michael and Kenneth Hsu 2008, A Low-power Design of Quantization for H.264 Video Coding Standard, Proceedings of the IEEE International SOC Conference, Newport Beach, CA, Pp. 201-204.

Mohammad Norouzi, Karim Mohammadi, Mohammad Mahdy Azadfar 2006. Multiplication and Error Free Implementation of H.264 like 4x4 DCT/Quan_IQuan/IDCT using Algebraic Integer Encoding, International Journal of Computer Science and Network Security, VOL.6 No.9B.