Low Cost and High Performance Implementation of Forward Transform and Quantization for an H.264/AVC Encoder

Van-Huan Tran, Ngoc-Mai Nguyen, Van-Mien Nguyen, and Xuan-Tu Tran
SIS Laboratory, The University of Engineering and Technology, Vietnam National University, Hanoi
114 Xuan Thuy road, Cau Giay, Hanoi 10000, Vietnam. Email: {huantv, main, miennv, tutx}@vnu.edu.vn

***Abstract*—This paper presents a low-cost and high-performance hardware design of forward transform and quantization (FTQ) for an H.264/AVC encoder. To minimize the hardware imple-mentation cost, in this design we use only one unified architecture of 1D transform engine to perform all transform processes. While the performance of the design is improved by using a fast architecture of the multiplier in the quantizer. Furthermore, this architecture also enables to share the common part among multipliers that have the same multiplicand. The design has been implemented using FPGA technologies for prototyping purpose. Experimental results show that our architecture can completely finish transform and quantization processes of a 4:2:0 macrob- lock in 228 clock cycles and the achieved performance is 115 MHz on a Virtex-II device (Xilinx) or 163 MHz on a Stratix-II device (Altera).**

Keywords: Integer Transform, Quantization, H.264/AVC Encoder

I. INTRODUCTION

The H.264/AVC is the newest and the most efficient video compression standard with capability of providing a good video quality at lower bit-rate than previous standards. To achieve high compression ratio, H.264/AVC has adopted several advances in coding technology. By specifying a set of integer transforms for small block-sizes, which are integer discrete cosine transform (DCT) and Hadamard transform, it has really reduced the computational complexity as important as blocking artifacts [1]. Thanks to new algorithms, these transforms can be easily realized with some required shift and add operations.

The size of transforms is variable, depending on the profile used in the encoder, where 4×4 block-size and 2×2 block-size transforms are primitive components. Larger transforms, which used in case of adaptive block-size transforms (ABT), are more suitable for High-Definition (HD) video.

Previous works have already been successes in hardware implementation of transform and quantization. Chih-Peng Fan and Yu-Lin Cheng [2] proposed a design with a high through- put and low latency architecture using Canonical Signed Digit (CSD) multiplier for shared quantization/inverse-quantization. In [3], Yu-Ting Kou presented an area-efficient architecture using direct 2D transform method. Whereas, in [4] is proposed a multi-transform architecture that used for variable adaptive block-size transforms. Generally, these works used two separate 1D transforms in cascading to carry out a 2D trans-form or implemented a direct 2D transform. Obviously, the advantage of these methods is that we can achieve a high throughput in transform. However, the bottleneck of encoders mostly comes from motion estimation and/or entropy coding modules rather than transform and quantization. Optimizing the design for throughput is therefore less important than other objectives such as performance or area, providing real-time processing. For this reason, to trade-off between throughput and area cost, we propose an architecture of transform using only one 1D transform module. With some improvements in control part, this architecture is able to perform integer DCT- based transforms as well as Hadamard transforms. In addition, to improve the system performance and more area-efficiency,we also present a particular architecture of multiplier in the quantizer. In where, a shared module (called pre-multiplier) is used for multipliers have the same multiplicands

The remaining part of this paper is organized as follows: Section II briefly recalls some background of transform and quantization algorithms; the proposed architecture for a forward transform and quantization will be presented in Section III. Experimental results will be presented in Section IV. Finally, conclusions and discussions will be given in Section V.

II. FORWARD TRANSFORM AND QUANTIZATION ALGORITHMS

In H.264/AVC standard, the residual frame of the predic- tion, which is the difference of the original frame and the predicted frame, is partitioned into fixed-size of macroblocks. As usually, a macroblock is composed of 16×16 luminance (Y) samples, 8×8 chroma

blue (C_b) samples and 8×8 chroma red (C_r) samples in case of 4:2:0 chroma subsampling format. At a smaller level, macroblocks are subdivided into blocks of 4×4 samples for coding. Each macroblock has its own information on quantization parameter (QP), coded type (Intra mode or Inter mode) and prediction mode. The flow of transform and quantization for those blocks is illustrated in .Figure 1.

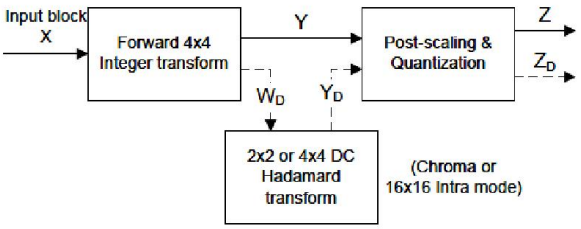

Figure 1. Transform and quantization flow diagram.

According to this flow, the input block X is first trans-formed using integer DCT-base method. The transformed coefficients are then post-scaled and quantized. In the 16×16 Intra-prediction mode, DC coefficients of all transformed residual blocks are grouped into an array of 4×4 before to be sent to Hadamard transform. Details of these processes are described in mathematical models later.

A. 4×4 forward transforms

• Integer DCT-based transform

The integer DCT-based transform which applied to a resi-dual 4×4 blocks (denoted by matrix *X*), is defined in H.264 as

the following:

$$Y = CXC^T = C(CX)^T$$

Where:

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

• Hadamard transform

The Hadamard transform which applied to a 4×4 luminance DC block (denoted by matrix W_D) in 16×16 Intra-prediction mode, is defined as the following:

$$Y_D = HW_DH^T = H(HW_D)^T$$

Where:

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

Generally, both integer DCT-based transform and Hada-mard transform are formed by two duplicated "cores" of 1D transform, where the "core" is a matrix multiplication either "*CT*" or "*HT*". The 2D transforms are carried out by applying the core to the input block; the immediate results are rear-ranged by transposing operation and re-applying the core. Obviously, the specification of the matrixes *C* and *H* in which only coefficients of ±1 and/or ±2 are available, these transfor- mations are multiplier-less and purely require a few of add and logical shift operations. On the other hand, the dynamic range of data is also estimated to reduce the overhead in computa-tions. With 8-bit precision of the pixel data, the dynamic range of outcome of integer DCT-based transform is 16-bit.

In here, we have already modified the matrix *H* by scaling of 1/2 to preserve the arithmetic operations of Hadamard transform in 16-bit precision as of integer DCT-based trans- form. Then, in quantization of the DC block, the result will be rescaled of 2. By this way, all 4×4 forward transforms are completely handled in 16-bit precision.

Figure 2. shows a hybrid and fast 1D transform diagram for processing 4 samples. The diagram is in the shape of butterfly diagram and is used for two types of transform. There are some multiplexers to select the shift factors (or scaled factors) in computations of each transform type. This diagram is great inspiration to design architecture for transform module.

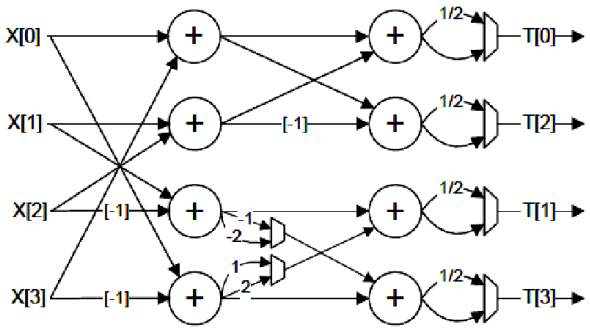

Figure 2. A hybrid 1D transform of integer DCT-based transform and Hadamard transform

B.Quantization

H.264/AVC standard defines a set of 52 values of quantiza- tion step (*Qstep*). These values are indexed by *QP* and to be determined in range of 0 to 51. As introduced above, the value of quantization parameter is associated with any maroblock (also blocks within a macroblock). Thanks to the wide range of *QP*, an encoder can be able to accurately and flexibly control the trade-off between bit-rate and quality .[9]..

Basically, forward quantization is defined as follows:

$$Z_{ij} = round\left(\frac{Y_{ij}}{Qstep}\right),\ 0 \le i, j \le 3 \qquad (5)$$

To avoid division operations, this equation can be represented in another way .[8].:

$$Z_{ij} = round\left(W_{ij}.\frac{MF}{2^{qbits}}\right) \quad (6)$$

Consequently, quantization can be computed as follows:

$$\begin{aligned} |Z_{ij}| &= \left(|W_{ij}|.MF + f\right) >> qbits \\ sign(Z_{ij}) &= sign(W_{ij}) \end{aligned} \quad (7)$$

Where $qbits = 15 + floor(QP/6)$, MF a matrix of multipli-cation factors (see.TABLE I.) and f is additional factor, f □$2^{qbits}/3$ if the block is coded in Intra mode, and $f = 2^{qbits}/6$ if the block is coded in Inter mode.

TABLE I. MULTIPLICATION FACTOR MF

QP%6	Positions (0,0),(2,0),(2,2),(0,2)	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

Especially, quantization for a DC block is implemented as follows (it has already rescaled by 2 due to scaling by 1/2 in transform):

$$\begin{aligned} |Z_{D(ij)}| &= \left(|Y_{D(ij)}|.MF_{00} + 2f\right) >> qbits \\ sign(Z_{D(ij)}) &= sign(Y_{D(ij)}) \end{aligned} \quad (8)$$

Where MF_{00} is the multiplication factor at position (0,0).

The innovation of quantization in H.264/AVC is the defini-tion of *Qstep*. In where, *Qstep* is non-uniform (or non-linear according to *QP*) and doubled in size if *QP* increases by 6. So, whenever *QP* is changed by the encoder, matrix of *MF* factor is also changed as consequence, but it absolutely depends on the value of *QP%6* (as shown in$_X$X). Besides, it does not require a lot of memory elements to store *MF* factors, only 18 values for full range of *QP*.

Similar to the transform part, the quantization has also simplified to obtain low-complexity in a manner of avoiding division and floating point operations.

III. PROPOSED ACHITECTURE

In this section, we present a novel architecture for trans- form and quantization of 4×4 blocks. While the design of transform is only intended to area-efficiency by using 1D transform module for all transformations of 4×4 block, the design of quantizer is estimated in order to improve on both performance and area usage. This design is able to process 4 samples at a time. The details will be described as follows

A. Transform module

With the proposing use one 1D transform module, the second 1D transform processes could not be started until the first 1D transform processes had finished on entire block. Therefore, it is necessary to have a memory buffer for storing and transposing the temporary data. Figure 3 shows the architecture of forward transform module with the sample-width of the datapath.

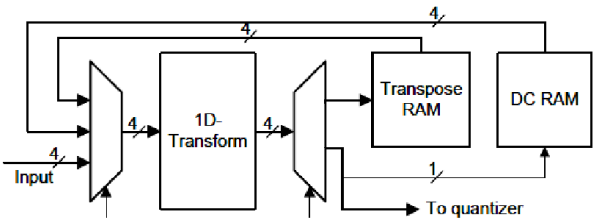

Figure 3. Architecture of forward transform.

The architecture is simply composed of three main compo- nents: 1D transform module, Transpose RAM module and DC RAM module, and other components such as multiplexer and de-multiplexer for arbitrating the dataflow. The input data and output data of the transform module are 4 samples, equivalent to 64-bit (4×16-bit). To have better view, some control signals have been hidden.

The activity of the module can be easily realized through the list of all states (corresponding to the dataflow):

1) *State_1: Input → 1D Transform → Transpose RAM*
2) *State_2: Transpose RAM → 1D Transpose → Output*
3) *State_3: DC RAM → 1D Transform → Transpose RAM*

These states have length of 4 clock cycles. By controlling the sequence of these states, a general block will be executed in the order of two states *{state_1; state_2}* while a DC block will be executed in the order of two states *{state_3; state_2}*.

- 1D transform

The 1D transform module is the hybrid transform as illu- strated in .Figure 2. All multiplexers in this module are controlled by a selector signal which configures the activity of the module as integer DCT-based transformation or Hadamard transformation. The responsibility of this module is one clock and the throughput therefore is 4 samples/clock. A higher throughput can be easily obtained by using several 1D trans-forms in parallel.

- Transpose RAM and DC RAM

The purpose of Transpose RAM as the meaning of the name is to store and transpose data. DC RAM is used

to store luminance DC coefficients of a transformed maroblock. Basically, Transpose RAM and DC RAM are matrices of 4×4 16-bit registers (as shown in .Figure 4.). The input and output of Transpose RAM are 4 samples width for read/write accessing a row/column data. Whereas the input of DC RAM is one sample width for writing one DC coefficient at a time and the output is 4 samples width, like Transpose RAM's.

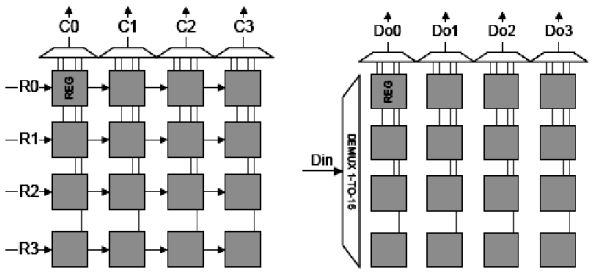

Figure 4. Transpose RAM (left) and DC RAM (right).

The writing operations of Transpose RAM are enabled whenever valid data are ready at the output of 1D transform module. These occur in 4 continuous clock cycles of the first 1D transform process. The reading operations occur in next 4 clock cycles to get out the column-wise data for the second 1D transform process. The registers in a row of Transpose RAM are connected in series. Thus, the data which stored in a register will be automatically shifted into the back register in next clock. By this means, Transpose RAM will be filled up new data in 4 clock cycles of the writing operations.

DC RAM is a bit different from Transpose RAM in their structures where the registers are independent with each other. This buffer is only useful and active in 16×16 Intra prediction mode only. In that case, the DC coefficients of any transformed luminance blocks are extracted and written into DC RAM.

Therefore, the writing operations of DC RAM only take place in one clock cycle at the time where the earliest data are valid. The reading operations are enabled in 4 clock cycles when the last block of a luminance macroblock is completely trans-formed. Besides, the read/write address signals of DC RAM are directly controlled by FTQ Controller module.

Compare with the architecture of cascading two separate 1D transform modules X, our architecture is required a bit challenging in designing the controller module but it has absolutely saved the hardware resource by the total cost of a 1D transform module.

B.Quantizer module

The quantizer can be easy realized from equations (7) and (8), as depicted in .Figure 5. It consists of four quantization cores and some common parts: MF_ROM module, DIVIDER_BY_6 module and F_CALC module. These common modules are shared to the 4 quantization cores. Actually, DIVIDER_BY_6 module is possible to share with de-quantizer module.

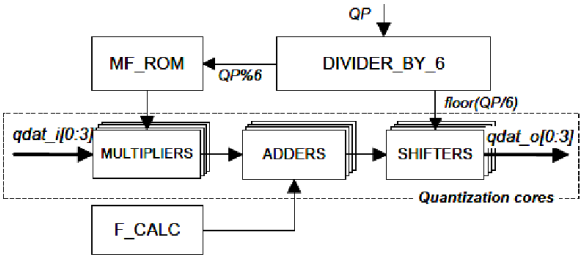

Figure 5. Architecture of quantizer.

DIVIDER_BY_6 module is a combinational block to calcu- late the value of *QP%6* and *floor (QP/6)* as well. In some related works, it was designed as common look-up-tables (LUTs), such as $_X[2]_X$. This design may take lots of memory utilization due to we have up to 52 values of *QP*. MF_ROM module is a ROM block for storing 18 constant values of MF factors. Accessing to a batch of *MF* factors is addressed by *QP%6* signal. F_CALC module is a combinational block to calculate the additional factor *f* based on the coded maroblock type (either Intra mode or Inter mode) and the block type (residual block or DC block).

Regarding to the multiplier design, when the size of multip-liers are large (15-bit of *qdat_i* and 14-bit of *MF*), it can mostly impact to the performance of the quantizer as a result of large latency. For this reason, we have deeply estimated the design of multiplier, which presented in next paragraph to minimize the latency.

- A fast and highly shared multiplier

The fast multiplier that we proposed is a conditional multip-lier. The idea is to build a basic element (called pre-multiplier) which is multiplier of MF factor with all possible 3-bit numbers (as shown in.Figure 6), where $A_i i.MF$, $0 \le i \le 7$. In fact, we do not need to carry out A0, A1, A2 and A4 on this module when these signals can be directly driven from MF signal.

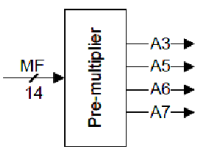

Figure 6. The pre-multiplier element.

Then, the 15-bit multiplier using the pre-multiplier element is explored as .Figure 7.. Each group of 3-bit vector (so it has 5 groups) is multiplied with the multiplicand *W* by controlling a multiplexer to select the equivalent result from the pre-multiplier element. There are some registers are inserted at the output of adders to cut down the combinational paths of the

multiplier. By this way, we have already improved the perfor-mance of the multiplier.

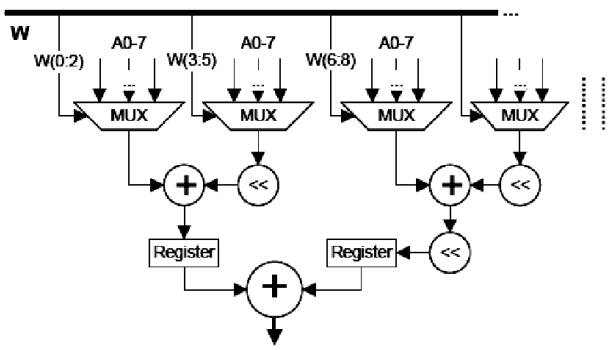

Figure 7. Multiplier for quantizer using the pre-multiplier element.

Obviously, the pre-multiplier can be shared among 5 groups of 3-bit multiplier. Since the pre-multiplier takes 4 adder blocks, we have already saved (4×4 = 16) adder blocks. In addition from the .TABLE I. of MF factors, *qdat_i[0]* and *qdat_i[2]* have the same MF factor, *qdat_i[1]* and *qdat_i[3]* are too. Therefore, the pre-multiplier is also possible to share between two quantization multipliers which have the same MF factor. Totally, our quantizer can be saved up to 2×(16×2+4)=72 adder blocks. Certainly, several multiplexers will be required to replace these adders but the hardware utilization is still reduced significantly.

C.Pipeline operation

To achieve better performance, the proposed architecture is controlled to operate in pipeline mode. Figure 8. shows the timeline of whole coding process.

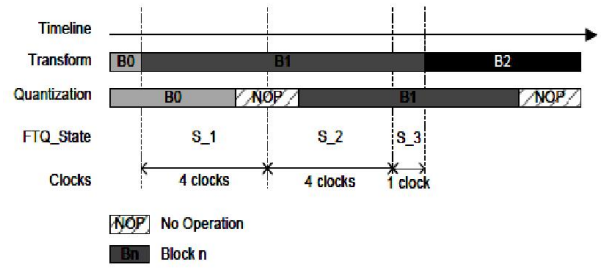

Figure 8. Three states of pipeline operation.

The pipeline has three states as identified by S_1, S_2 and S_3. In where, S_1 (4 clock cycles): launching the first stage of 1D transform on block B1 while previous block B0 is still quantizing to the end. S_2 (4 clock cycles): launching the second stage of 1D transform while start quantizing of block B1. S_3 (1 clock cycle): prepare loading new block B2 into transform module while block B1 is still quantizing, valid data at the output are ready. Therefore, it normally takes 12 clock cycles to complete transform coding of a block.

Summary, by the pipelined schedule, our design will take 9 clock cycles on average to process a block and this is equiva-lent to 228 clock cycles to complete the transform and quanti-zation processes of the entire 4×4 blocks within a 4:2:0 macroblock.

IV. VERIFICATION AND IMPLEMETATION

The architecture was modeled using VHDL language and simulated on Synopsys VCS tool to verify the functionality.

To verify the design (before and after the implementation), we developed a simple simulation environment as described in .Figure 9. In this environment, we developed a software model of FTQ architecture on Matlab, which is used for testing purpose only. The input data used in this simulation is a PGM (portable graymap format) image. This image is provided to both Matlab model and hardware model. Then, the outputs of the both hardware and Matlab models will be compared to each other by using the developed testbench.

It was implemented on different FPGA technologies, Altera and Xilinx, for rapid prototyping purpose. The implementation results are shown on .TABLE II. for both Xilinx XC2V1500-6 and Altera Stratix II devices.

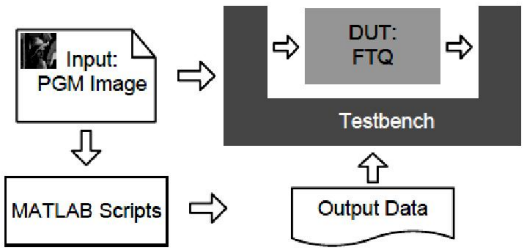

Figure 9. Verification model for the design.

.TABLE III. shows the comparison of different designs on two parameters: FGPA gate counts and performance. It is clear that our architecture has better performance than the others thanks to the improvement of multipliers, while the hardware implementation cost is less than the others.

TABLE II. SYNTHESIS RESULTS

Module	Xilinx C2V1500-6 (MHz)	Altera Stratix II (MHz)
Transform (included controller)	142	209
Quantizer	174	214
Max. Speed	115	163

TABLE III. HARDWARE COMPARISON ON FPGA TECHNOLOGY

Design name	FPGA device	FPGA gate counts	Max. speed (MHz)
Kordasiewicz [7] (area-optimized)	Xilinx XC2V7	260,520	90.70
Chih-Peng Fan [2] (using CSD multipliers)	Xilinx XC2V1500-6	135,306	99.15
Grzegorz Pastuszak [3]	Altera Stratix II	--	100
This work (optimized speed)	Xilinx XC2V1500-6	120,293	115
This work	Altera Stratix II	--	163

V. CONCLUSION

In this paper, we presented a low-cost and high-performance forward transform and quantization hardware implementation for an H.264/AVC encoder. In addition to proving the design, implementation results on different FPGA technologies were intently reported to help designers make decision when choosing the technology. As indicated above, the design gets a higher performance when implemented using Altera technology than using Xilinx technology. It is able to process a 4:2:0 macroblock in 228 clock cycles at a high frequency of 163MHz (Altera Stratix II). The area overhead reported in this work is also smaller than previous works, gained more than 10%.

VI. CKNOWLEDGMENT

This work is supported by Vietnam National University, Hanoi (VNU) through research projects, PUF.08.06 and VENGME. The authors would like to thank Synopsys' experts for their technical supports in order to invest and deploy this design to a real ASIC implementation.

VII. REFERENCES

[1] Detlev Marpe, Thomas Wiegand, and Gary J. Sullivan, "The H.264/MPEG4 Advanced Video Coding Standard and its Applications", IEEE Communications Magazine, pp. 134–143, August 2006.

[2] Chil-Peng Fan and Yu-Lin Cheng, "FPGA Implementations of Low Latency and High Throughput 4x4 Block Texture Coding Processor for H.264/AVC", Journal of the Chinese Institute of Engineers, Vol. 32, No. 1, pp. 33–44, 2009.

[3] Yu-Ting Kuo, Tay-Jyi Lin, Chih-Wei Liu, and Chein-Wei Jen, "Architecture for Area-Efficient 2-D Transform in H.264/AVC", in Proc. of the 2005 IEEE Int'l Conf. on Multimedia and Expo, July 2005.

[4] Javier D. Bruguera and Roberto R. Osorio, "A Unified Architecture for H.264 Multiple Block-Size DCT with Fast and Low", in Proc. of the 9th EUROMICRO Conf. on Digital System Design (DSD), August 2006.

[5] Tu-Chih Wung, Yu- Wen Huang, Hung-Chi Fang, and Liang-Gee Chen, "Parallel 4x4 2D Transform and Inverse Transform Architecture for MPEG-4 AVC/H.264", in Proc. of the 2003 Int'l Symposium on Circuits and Systems (ISCAS), pp. 800–803, May 2003.

[6] G. Pastuszak, "Transforms and Quantization in the High-Throughput H.264/AVC Encoder Based on Advanced Mode Selection", in Proc. of IEEE CS Annual Symposium on VLSI, pp. 203–208, April 2008.

[7] R. C. Kordasiewicz and S. Shirani, "ASIC and FPGA Implementations of H.264 DCT and Quantization Blocks," in Proc. of the 2005 IEEE Int'l Conf. on Image Processing (ICIP), pp. III-1020-3, September 2005.

[8] H.S. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, "Low-Complexity Transform and Quantization in H.264/AVC", IEEE Trans. on Circuits and Systems for Video Technology, pp. 598–603, 2003.

[9] I.E.G. Richardson, "H.264/MPEG-4 Part 10: Transform & Quantization", VCodex Ltd White Paper, March 2003.