A New Hardware Implementation Of The H.264 8×8 Transform And Quantization

Jeoong Sung Park
Department of Computer Engineering
Santa Clara University
Santa Clara, CA 95053, USA

Tokunbo Ogunfunmi
Department of Electrical Engineering
Santa Clara University
Santa Clara, CA 95053, USA

***Abstract*—H.264/AVC is the most powerful technology in video compression/transmission area because of its high coding efficiency and robustness. In this paper, we propose a new hardware architecture of 8x8 integer transform and quantization for H.264 which promises very low resource utilization. In the architecture, each pixel is processed one by one on a simplified pipeline without multiplication. Thus, redundant modules, which are used for block-based or row-based parallel processing, can be reduced. Experimental results show that it can reduce resource usage 30% compared to previously proposed models. It can be used for mobile applications. It covers a wide range of parameters as well.**

***Index Terms*—H.264, integer transform, quantization**

I. Introduction

The continuing development of digital video coding has produced ITU-T H.264/MPEG-4 (Part 10) Advanced Video Coding (commonly referred as H.264/AVC). It provides gains in compression efficiency of up to 50% over a wide range of bit rates and video resolutions compared to previous standards [1]. Besides, network friendliness and good video quality at high and low bit rates are important features that distinguish H.264 from other standards.

Actually, the initial H.264 specification adopted an integer approximation of 4×4 [2]. But, the 4×4 block is not enough for SD resolutions and above. That is, when larger than 4x4 transforms are used, significant compression performance gains have been reported at Standard Definition (SD) and High Definition (HD) resolutions [3]. Thus, a new integer transform of 8×8 was proposed in the Fidelity Range Extensions (FRExt) to be added to the existing specification. Using an 8×8 transform in addition to the 4×4 transform in H.264/AVC, a roughly 10% bit-rate reduction can be achieved across a wide range of content and coding parameters [10]. However, these advantages have resulted in additional complexity to the H.264/AVC. To avoid the large complexity, [9] proposed a simplified way of introducing larger transforms into the JVT specification, while maintaining the rate-distortion performance gain.

This ongoing technical evolution has accelerated the development of system-on-chip (SoC) platform to support compactness, low power, robustness, cheap cost, and most importantly, real-time operation [4]. To implement DCT and quantization blocks for H.264 on SoC, many efforts have been carried out. References [5],[6] and [7] provide FPGA implementation of 4×4 transform and quantization which are targetted on the initial H.264. In [4], simplified 8×8 transform and quantization are implemented on FPGA. An architecture for adaptive block size transform for 8×8 and 4×4 is developed in [8].

Most of previous research have deployed parallel computation of either block data or row(column) data to make system speed up to satisfy real-time constraints. For example, [6] and [4] process 16 block data (for 4×4 transform) and 64 block data (for 8×8 transform) in full parallel, respectively. In [7], 4 concurrent row data (for 4×4 transform) is processed in parallel. All those implementations result in system speeds much faster than real-time requirements. However, they can not avoid deploying N redundant transform and quantization blocks, where N can be 4, 8, 16, or even 64 as the number of parallelism. Intuitively, N redundant blocks require N hardware area.

In this paper, we start from the idea that N hardware area can be reduced to just one area by adjusting N to one. As a result, we propose a minimum complexity architecture that processes 64 sequential pixel data for 8×8 transform and quantization with reduced parallelism. It does not only satisfy real-time constraints but also cover all range of parameters to support accuracy and flexibility without requiring the same redundant blocks.

This paper is organized as follows. Section II presents a brief overview of 8×8 integer transform and quantization used in this paper. Section III describes the proposed architecture and performance analysis. In Section IV, we provide implementation results. We conclude in Section V.

II. BACKGROUND

A. 8×8 Integer Transform

We follow the 8x8 integer approximation of DCT proposed in [9]-[11]. It consists of only adds and shifts without any multiplication.

The 2-D forward 8x8 transform is computed in a separable way as a 1-D horizontal (row) transform followed by a 1-D vertical (column) transform, where the corresponding 1-D transforms are given by Equation (2) and Matrix (1),

$$W = CXC^T \tag{1}$$

Matrix C is specified as

$$\begin{bmatrix} 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\ 12 & 10 & 6 & 3 & 8 & 8 & -10 & -12 \\ 8 & 4 & -4 & -8 & -8 & -4 & 4 & 8 \\ 10 & -3 & -12 & -6 & 6 & 12 & 3 & -10 \\ 8 & -8 & -8 & 8 & 8 & -8 & -8 & 8 \\ 6 & -12 & 3 & 10 & -10 & -3 & 12 & -6 \\ 4 & -8 & 8 & -4 & -4 & 8 & -8 & 4 \\ 3 & -6 & 10 & -12 & 12 & -10 & 6 & -3 \end{bmatrix} .1/8 \quad (2)$$

Each of these 1-D transforms can be computed via fast butterfly operations as follows [10]:

Data-path 1:
$a[0] = x[0] + x[7]$
$a[1] = x[1] + x[6]$
$a[2] = x[2] + x[5]$
$a[3] = x[3] + x[4]$
$a[4] = x[0] - x[7]$
$a[5] = x[1] - x[6]$
$a[6] = x[2] - x[5]$
$a[7] = x[3] - x[4]$

Data-path 2:
$b[0] = a[0] + a[3]$
$b[1] = a[1] + a[2]$
$b[2] = a[0] - a[3]$
$b[3] = a[1] - a[2]$
$b[4] = a[5] + a[6] + ((a[4] >> 1) + a[4])$
$b[5] = a[4] - a[7] - ((a[6] >> 1) + a[6])$
$b[6] = a[4] + a[7] - ((a[5] >> 1) + a[5])$
$b[7] = a[5] - a[6] + ((a[7] >> 1) + a[7])$

Data-path 3:
$w[0] = b[0] + b[1];$
$w[2] = b[2] + (b[3] >> 1);$
$w[4] = b[0] - b[1];$
$w[6] = (b[2] >> 1) - b[3];$
$w[1] = b[4] + (b[7] >> 2);$
$w[3] = b[5] + (b[6] >> 2);$
$w[5] = b[6] - (b[5] >> 2);$
$w[7] = -b[7] + (b[4] >> 2);$

B. 8×8 Quantization and Scaling

Based on [10], quantization and scaling is performed according to the following equation:

$$Z_{ij} = (W_{ij} * MF + f * 2^{16+n}) >> (16 + n) \qquad (3)$$

where *n* is *QP*/6. *QP* is the quantization parameter, and *f* is the deadzone/offset parameter with an absolute value ranging between 0 and 1/2 and with the same sign as the coefficient that is being quantized. MF is a multiplication factor that depends on m (= *QP* mod 6) and the position (i,j) of the element as follows.

$$MF[m; i, j] = \begin{cases} M_{m0} \ for(i,j) \ with \ (i,j) = G_0 \\ M_{m1} \ for(i,j) \ with \ (i,j) = G_1 \\ M_{m2} \ for(i,j) \ with \ (i,j) = G_2 \\ M_{m3} \ for(i,j) \ with \ (i,j) = G_3 \\ M_{m4} \ for(i,j) \ with \ (i,j) = G_4 \\ M_{m5} \ for(i,j) \ with \ (i,j) = G_5 \end{cases}$$

where

$$\begin{cases} G_0 : i = [0,4], \ j = [0,4] \\ G_1 : i = [0,3,5,7], \ j = [1,3,5,7] \\ G_2 : i = [2,6], \ j = [2,6] \\ G_3 : i = [0,4], \ j = [1,3,5,7]) \cap (i = [1,3,5,7], \ j = [0,4]) \\ G_4 : i = [0,4], \ j = [2,6]) \cap (i = [2,6], \ j = [0,4]) \\ G_5 : i = [2,6], \ j = [1,3,5,7]) \cap (i = [1,3,5,7], \ j = [2,6]) \end{cases}$$

The matrix M is specified as:

$$\begin{bmatrix} 13107 & 11428 & 20972 & 12222 & 16777 & 15481 \\ 11916 & 10826 & 19174 & 11058 & 14980 & 142990 \\ 10082 & 8943 & 15978 & 9675 & 12710 & 11985 \\ 9362 & 8228 & 14913 & 8931 & 11984 & 11259 \\ 8192 & 7346 & 13159 & 7740 & 10486 & 9777 \\ 7282 & 6428 & 11570 & 6830 & 9118 & 8640 \end{bmatrix} \qquad (4)$$

III. PROPOSED ARCHITECTURE

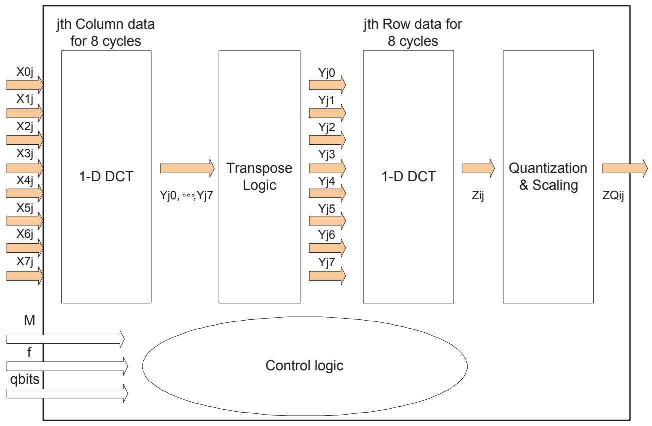

Fig. 1. Architecture of the proposed system

We adopt architectures presented in [5] and [7], which are used for a 4×4 integer DCT and quantization system. The block diagram of the proposed architecture for 8×8 integer transform and quantization is presented in Figure 1, which contains 1-D integer transform blocks, transpose logic, quantization block, and control logic. 16-bit word length is defined in this system. This architecture clearly reflects two separate 1-D integer transforms and quantization as mentioned in Section II. *QP* is not used as an input, which is different from [4] which suggest the hardware implementation of QP-processing block that computes *f, qbits, P0, ..., P5* from input signal *QP*. However, that block is not necessary in terms of hardware implementation because the block does not process data but calculate parameters used for data processing in quantization block. Once the *QP* is fixed, other parameters are also not changed. So, we assume that the QP-processing is previously done by a software.

A. 1-D integer transform block

The architecture of 1-D integer transform block is shown in Figure 2. Each input column vector of 8 pixels is input to the 1-D DCT block for 8 cycles, so transformed outputs

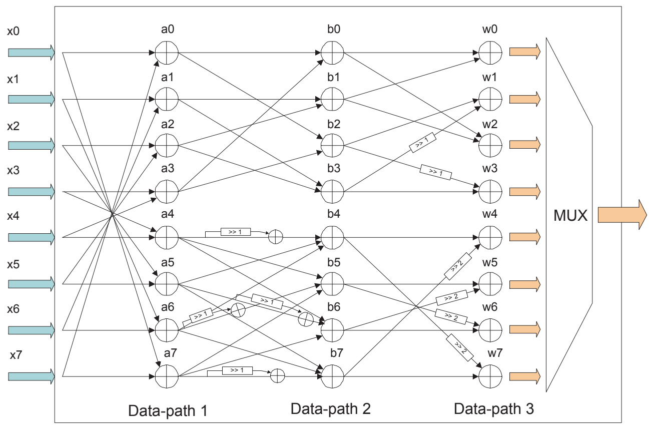

Fig. 2. 1-D row (column) transform block

w0-w7 are also held for 8 cycles. However, just one out of w0-w7 is output through MUX. That is, w0, w1,..., w7 go out of the 1-D DCT in sequential order for 8 cycles. After all, 64 cycles are required to process all pixel elements in one 8×8 block. Since each pixel in one column is processed one by one, the transpose logic as shown in Figure 3 is very simple. F/F indiciates a 16-bit register that consists of 16 flip flops. It converts the output column vector to the input row vector to the next 1-D DCT while working as a pipe-line memory.

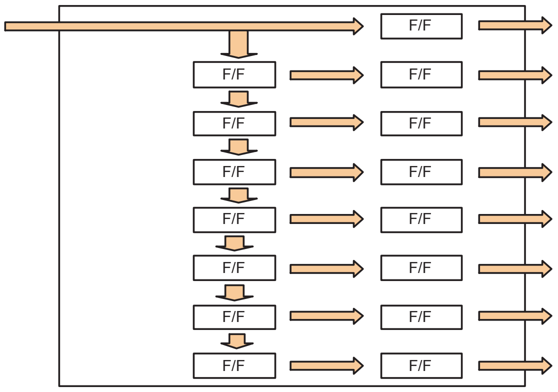

Fig. 3. Architecture of the transpose logic

B. *Quantization block*

Quantization block does the function of quantization and scaling presented in Section II-B. Since integer approximations of 4×4 and 8×8 DCT were proposed in [12] and [10], respectively, critical complexity of DCT were gone and forwarded to the quantization block. Even though previous researches have focused on low complexity implementation of DCT transforms with adds and shifts, there were still parallel multipliers for processing parallel output data from 2-D integer transform block. Output data from 2-D transform block have longer bit widths than system input data. Therefore, architecture of the quantization block needs to be considered to reduce hardware utilization of the total system. In our proposed model, only one multiplication is required because the input data are sequential. We also prevent the large multiplier by replacing it with adders and shifters as shown in Fig 4. This architecture is designed to cover all multiplication factors (*MF*) in Equation 3. Among 36 *MF*s, 12,222 (10111110111110) has the largest number (11) of non zero partial products. So, eleven left shifters are needed. Control logic assignes the shift index that corresponds to each *i, j, m* from its look-up table. The binary tree architecture of eleven additions occupies less hardware utilization than a real multiplier even though its latency is 5.

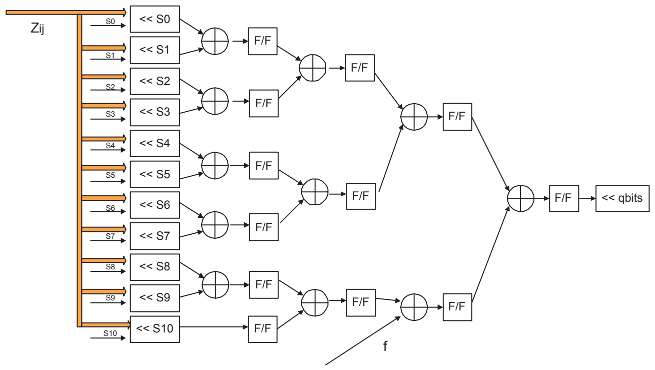

Fig. 4. Quantization block

IV. Simulation and Results

The architecture is implemented in Verilog HDL and simulated with NC-Verilog 6.2. Syhthesis are performed using Synplify Pro 9.2.4, and place and route using Xilinx ISE 10.1. The target device chosen is Xilinx Virtex-II Pro XC2VP30 FPGA. The implementation results are as shown in Table I.

TABLE I
Implementation Result of the proposed architecture

Technology	Xilinx XC2VP30-7FF896
Critical Path delay (ns)	8.943 ns
CLK Frequency (MHz)	111.8 MHz
IOs	553
Slices	1624
Slice Flip Flops	251
LUTs	2887
Global Clocks	1

The critical path measured by the place and route tool is 8.943 ns. It takes sixty four clock cycles to process a 8×8 block through the integer transform and the quantization block. Therefore the time required to process a whole frame is as follows:

$$\begin{aligned} T_{frame} &= N_{block\_per\_frame} \times T_{block} \\ &= \frac{N_{pixel\_per\_frame}}{N_{pixel\_per\_block}} \times T_{block} \\ &= \frac{N_{pixel\_per\_frame}}{N_{pixel\_per\_block}} \times N_{cycle} \times T_{cycle} \end{aligned}$$

where *N_cycle* and *T_cycle* indicate the number of cycles and time required per cycle, respectively. Times required to encode a full HD frame of $1,920 \times 1,080$ and a HDTV frame of $1,024 \times 768$ are:

$$T_{HD} \quad = \quad \frac{1,920 \times 1,080}{8 \times 8} \times \; 8.943\,ns \times \; 64 = 18.54\,ms$$

$$T_{HDTV} \quad = \quad \frac{1,024 \times 768}{8 \times 8} \times \; 8.943\,ns \times \; 64 = 7.03\,ms$$

T_{HD}(18.54 *ms*) is 1.8 times faster than the 33.3 *(*ms) time required for processing each frame (assuming a refresh rate of 30 frames/sec). In the same way, T_{HDTV}(7.03 *ms*) is 4.7 times faster. Hence, the proposed architecture meets the real-time constraints for HD and HDTV as well as HD of 704×480.

As mentioned in Section I., we compare our results with [4] and [8] which implement the 8×8 transform and quantization on FPGA. As denoted in Table II, the number of LUTs used in the proposed design is 9.9% and 70% of that in [4] and [8], respectively. Hence, 90% and 30% can be saved by using the proposed design. The main difference between our proposed design and [4] is that we use a reduced parallel architecture without multiplication and QP-processing block included in [4]. Table II also explains that less parallelism can save more area in spite of longer latency.

TABLE II
PERFORMANCE COMPARISON WITH PREVIOUS DESIGNS

	[4]	[8]	Proposed
Critical path delay (ns)	14.598	12.930	8.943
CLK frequency (MHz)	68.5	77	111.8
LUTs	29018	4124	2887
Parallelism	64	8	1
Latency	1	16	64

V. CONCLUSION

We presented a new implementation of a 8×8 integer transform, quantization, and scaling for H.264 on a FPGA. To reduce hardware resource utilization, a reduced parallel architecture that processes sequential pixel data was developed. Experiments show that our reduced parallel architecture requires from 30% up to 90% less resources than existing designs. In the architecture, each pixel is processed one by one on a simplified pipeline without multiplication. The pixel-by-pixel processing can remove redundant modules used for block-based or row-based processing in not only the integer transform block but the quantization block. In addition, our quantization block is designed to cover all multiplication factors without using a real multiplier.

The proposed design supports most of video formats by satisfying real-time constraints. Even though it has slower speed and longer latency than previous designs, there is other remarkable advantage of lower hardware area as a trade-off. Longer latency and less hardware area can be suitable for mobile applications.

REFERENCES

[1] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, T. Wedi Video coding with H.264/AVC: tools, performance, and complexity. *Circuits and Systems Magazine IEEE*, Vol. 4, Issue 1, pp. 7-28, 2004.

[2] T. Wiegand, G. J. Sullivan, G. Bjontegaard and A. Luthra Overview of the H.264/AVC Video Coding Standard. *IEEE Trans. Circuits Syst. Video Technol.*, Vol. 13, no. 7, pp. 560-576, July 2003.

[3] S. Gordon, D. Marpe, and T. Wiegand ABT for Film Grain Reproduction in High Definition Sequences. Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, doc. JVT-H029, Geneva, Switzerland, May 2003.

[4] I. Amer, W. Badawy and G. Jullien High Performance Hardware Implementation of the H.264 Simplified Transform and Quantization. *Proceedings of International Conference on Acoustics, Speech and Signal Processing*, Philadelphia, Pennsylvania, USA, Vol. 2, pp. 1137-1140, March 2005.

[5] R. Korah, J. Raja Paul Perinbam FPGA implementation of integer transform & quantizer for H.264 Encoder. *Journal of VLSI signal processing systems - Springer*, 2008.

[6] R. Kordasiewicz, S. Shirani Hardware implementation of the optimized transform and quantization blocks of H.264. *Canadian Conference on Electrical and Computer Engineering*, Vol. 2, no. 7, pp. 943 - 946, May 2004.

[7] A. Mendez Patifio, M. A. Martinez Peiro, F. Ballester, G. Paya Architectures for ICT on FPGA. *Field-Programmable Technology, 2004. Proceedings. 2004 IEEE International Conference on*, Vol., no., pp. 403-406, 6-8 Dec. 2004.

[8] Y. Li, Y. He, and S. Mei A Highly Parallel Joint VLSI Architecture for Transforms in H.264/AVC. Journal of Signal Processing Systems, vol. 50, No. 1, pp. 19-32, Jan. 2008.

[9] S. Gordon, D. Marpe, and T. Wiegand Simplified Use of 8x8 Transform. *Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG*, doc. JVT-I022, San Diego, USA, September 2003.

[10] S. Gordon, D. Marpe, and T. Wiegand Simplified Use of 8x8 Transform - Proposal. *Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG*, doc. JVT-J029, Waikoloa, USA, December 2003.

[11] S. Gordon, D. Marpe, and T. Wiegand Simplified Use of 8x8 Transform - Updated Proposal & Results. *Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG*, doc. JVT-K028, Munich, Germany, March 2004.

[12] H. Malvar, A. Hallapuro, M. Karczewicz and L. Kerofsky Low-Complexity Transform and Quautization in H.264/AVC. *IEEE Trans. Circuits and Systems for Video Technology*, vol. 13, No. 7, Jul. 2003.

[13] S. Kim and W. Sung Fixed-point error analysis and word length optimization of 8x8 IDCT architectures. *IEEE Trans. Circuits and Systems for Video Technology*, vol. 8, No. 8, pp. 935-940, Dec. 1998.