# Design and Implementation of Integer Transform and Quantization Processor for H.264 Encoder on FPGA

N. Keshaveni
Electronics and Communication
MGR University, Chennai, India
keshaveni33@yahoo.com

Dr S. Ramachandran
National Academy of Excellence
Bangalore, India
ramachandran_ns@yahoo.com

Dr K. S. Gurumurthy
Electronics and communication
UVCE, Bangalore, India
drksgurumurthy@gmail.com

***Abstract*—This paper proposes a novel implementation of the core processors, the integer transform and quantization for H.264 video encoder using an FPGA. It is capable of processing the frames with the desired compression controlled by the user input. The algorithm and architecture of the components of the video encoder namely, integer transformation, quantization were developed, designed and coded in Verilog. The complete H.264 video encoder was coded in Matlab in order to verify the results of the Verilog implementation. The processor is implemented on a Xilinx Vertex – II Pro XC2VP30 FPGA. The gate count of the implementation is approximately 1,057,000 working at a frequency of 208 MHz. It can process 1024x768 pixel color images in 4:2:0 format at 25 frames per second. The reconstructed picture quality is better than 35 dB.**

***Keywords*— Integer transform, quantization, video encoder, Verilog, FPGA.**

## I. INTRODUCTION

With the widespread use of technologies like digital television, internet streaming video and DVD video, video compression has become an inevitable component of broadcast and entertainment media. Currently, the video codec that achieves the highest data compression without sacrificing on the picture quality is the MPEG-4 Part 10 Advanced Video Coding, also known as the H.264 [1]. This codec has many new features such as intra-frame prediction, 4x4 integer transform, quantization, context adaptive entropy coding, deblocking filter etc., which were not available in the earlier standards.

The present work has realized some of the above features such as 4x4 integer transform and quantization. The implementation conforms to the baseline, main as well as extended profiles since only Intra frames are used. Qiang Peng and Jin Jing [2] have reported an implementation of the H.264 encoder using a 32-bit RISC CPU on a single chip running on Linux operating system, which can process PAL, SECAM or NTSC video at 80 MHz.

This paper is organized as follows: A parallel algorithm is presented in Section II for evaluating the transform and quantization suitable for high speed implementation on FPGA/ASIC. Section III deals with the architecture of the proposed schemes and Section IV with the implementation of the design in the FPGA. Results are discussed in Section V and conclusions are presented in Section VI.

## II. ALGORITHM FOR PARALLEL MATRIX MULTIPLICATION OF INTEGER TRANSFORM AND QUANTIZATION

A parallel algorithm that is capable of being highly pipelined has been developed for the DCT/IDCT by one of the authors earlier [3]. Basically, the same algorithm is adapted for computing the integer transform in the present work so that it is suitable for FPGA/ASIC implementation. The core integer transform is expressed as two-stage matrix multiplication as shown in "(1)". The scaling factors specified in the standard are not shown in this equation since the same will be absorbed in the computation of quantization as MF. The values $X_{00}$ to $X_{33}$ are the pixel inputs. C and C' (the transpose of C) are constant matrices. W, containing elements $W_{00}$ to $W_{33}$ is a matrix of coefficients after transforming the matrix X.

Each of the transformed coefficients $W_{ij}$ is quantized by a scalar quantizer specified as in [4]. A total of 52 values of quantization step size (Qstep) are supported by the standard and these are indexed by a Quantization Parameter, QP. Qstep doubles in size for every increment of 6 in QP. The wide range of quantizer step sizes makes it possible for an encoder to accurately and flexibly control the trade-off between bit rate

$$\begin{pmatrix} W_{00} & W_{01} & W_{02} & W_{03} \\ W_{10} & W_{11} & W_{12} & W_{13} \\ W_{20} & W_{21} & W_{22} & W_{23} \\ W_{30} & W_{31} & W_{32} & W_{33} \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{pmatrix} \begin{pmatrix} X_{00} & X_{01} & X_{02} & X_{03} \\ X_{10} & X_{11} & X_{12} & X_{13} \\ X_{20} & X_{21} & X_{22} & X_{23} \\ X_{30} & X_{31} & X_{32} & X_{33} \end{pmatrix} \begin{pmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{pmatrix}$$

Or in short, $W = C * X * C'$ (1)

and quality. The quantized coefficients are computed as follows:

$$Z_{ij} = \text{floor}(W_{ij} * MF / 2^{qbits}) \quad (2)$$

where qbits = 15 + floor(QP/6) and MF is a multiplication factor specified in the H.264 reference model software of the standard. In the present implementation, we have chosen (QP mod 6) = 4, i.e., the encoder can use QP values of 4, 10, 16, 22 etc. depending on the user's choice. This requires that a coefficient be multiplied by 8192, 3355 or 5243 depending on its position (0, 0), (0, 2), (2, 0), (2, 2); (1, 1), (1, 3), (3, 1), (3, 3) or others in the matrix W.

The proposed algorithm for the integer transform and quantization is as follows:

1. Multiply/add the first row of C with each column of X one after another to generate the first row of partial products, $P_{00} – P_{03}$. Multiplications involved are trivial since 1, -1, 2, -2 are the multiplying constants.
2. Multiply/add the second row of C with each column of X one after another to generate the second row of partial products, $P_{10} – P_{13}$. Concurrently multiply the first row of partial products $P_{00} – P_{03}$ (generated in the previous step) with each of the columns of C' one after another to generate the first row of integer transformed coefficients. Pipeline the quantization (multiplication with MF) as per "(2)" immediately after each integer coefficient Wij is generated. It may be noted that the computation $2^{qbits}$ is just right shift operation dispensing with division. In this step the quantized coefficients Zij are generated.
3. Repeat the step 2 for the third and fourth rows of C to generate the rest of the sixteen quantized coefficients.

## III. Architecture of the Integer transform and Quantization processor

An image or a frame in a sequence of motion pictures is processed macro block after macro block in the raster scan order as shown in "Fig. 1". A macro block is organized as a 16 x 16 pixel information for Y (luminance) samples, 8 x 8 pixels for Cb and 8 x 8 pixels for Cr samples in the 4:2:0 color picture format. The integer transform is applied on 4x4 pixel sub blocks of a picture. Sub blocks are processed in the order shown in "Fig. 1", meeting the H.264 standard.

The basic architecture of a Format Converter, integer transform and quantization (TQ) processor is shown in "Fig. 2". As shown therein, the interface that receives the video input 'Y_in' and 'C_in' concurrently from a camera decoder (not shown in the figure) is a format converter which converts 4:2:2 format into 4:2:0 format. The input is written at every rising edge of 'write_clk' and is valid if 'data_in_valid' signal is asserted. The converted signals Y, Cb and Cr are stored in on-chip memory 'Dual ram' at the rising edge of 'write_clk' when the corresponding valid signals are asserted.

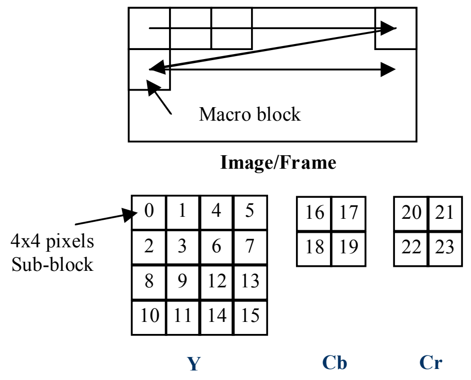

Figure 1. Order of processing the Macro block

We are interested in processing pictures of widths up to 1024 pixels. Therefore, we need 16 KB of on-chip RAM for storing 16 lines of picture for Y. In order to maintain continuous processing, we need to have a dual RAM instead of a single RAM, thus requiring a RAM size of 32 KB for Y component. Also since in 4:2:0 format, there are 4, 1 and 1 block(s) for Y, Cb and Cr components respectively, we need one-fourth this size of RAM, each for Cb and Cr components. Thus, the dual RAMs store two numbers of 16 lines, i.e., two macro block rows. As one RAM buffer gets filled, the transformation is processed concurrently from the other buffer previously filled, taking pix_Y, pix_Cb and pix_Cr as inputs. This is transformed, quantized as per the algorithm explained earlier using the respective modules. The qstep [1:0], which determines the desired compression, is input by the user. The signals, pix_Y_valid, pix_Cb_valid and pix_Cr_valid are the valid signals of pix_Y, pix_Cb and pix_Cr respectively.

The entire Transformation and Quantization modules are heavily pipelined with highly parallel circuits such that every pixel is processed in 3 'read_clk' cycles on an average. Thus a picture of size 1024x768 pixels in 4:2:0 format can be processed in 35 ms, thereby at a frame rate of 25 per second if the FPGA implementation permits 100 MHz operation.

The Y, Cb and Cr components and their valid signals read from the dual RAM are applied to the Integer Transform. The transformed coefficients are fed to the next stage, namely, the quantizer using 'coeff [15:0]' signals. Validity of these signals is reckoned by the corresponding valid signal, 'coeff_valid'. The quantization desired by the user is supplied by the input, 'qstep [1:0]'. The signal 'read_clk' serves as the system clock for the transform and quantization processor modules. Finally, the quantized coefficients are available as the 'quant_coeff [11:0]' output. Its validity is asserted by the signal 'quant_coeff_valid'.

## IV. FPGA Implementation

The integer transform and quantization algorithm and various modules described in previous sections were coded in Verilog, simulated using ModelSim, synthesized and place and routed using Xilinx Project Navigator ISE 8.2. The target device chosen was Xilinx Vertex-II Pro XC2VP30 -7 FF896 FPGA since the board available in our laboratory is based on this FPGA. The core parts of the H.264 video encoder design (Transform and Quantization) described earlier utilize 1,057,000 gates and 192 numbers of dual port RAMs with 828 numbers of occupied slices. The maximum frequency of operation is 208 MHz. This works out to a frame rate of 50 per second for a picture size of 1024x768 pixels. Since we need to implement the inverse quantization and inverse transform later on, the frame rate for a complete H.264 video encoder will ultimately reduce to 25 frames per second. Thus, the claim of processing video sequences of size 1024x768 pixels at 25 frames per second is feasible.

## V. Results and Discussions

The H.264 Video Codec was first implemented in Matlab in order to estimate the quality of the reconstructed image and the compression that can be achieved. In addition, Matlab output serves as a reference for verifying the Verilog output.

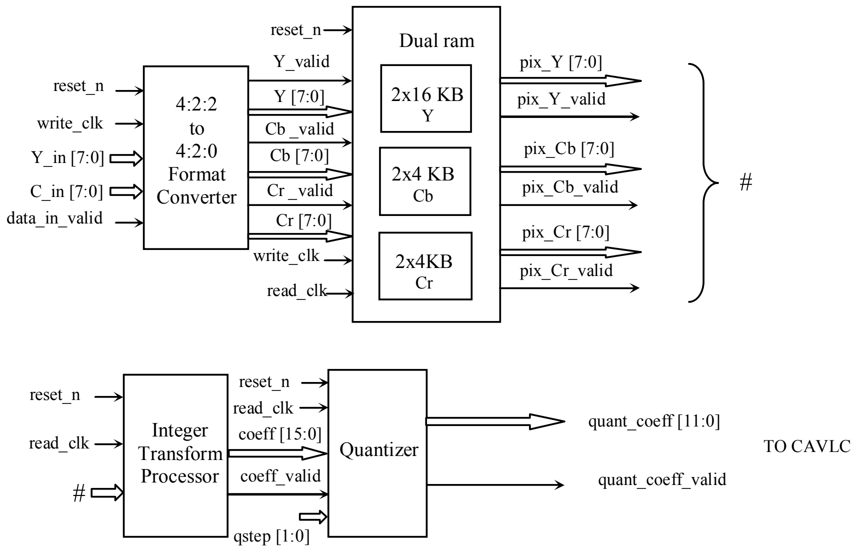

Figure 2. Architecture of Format Converter, Integer Transformation and Quantization Processor

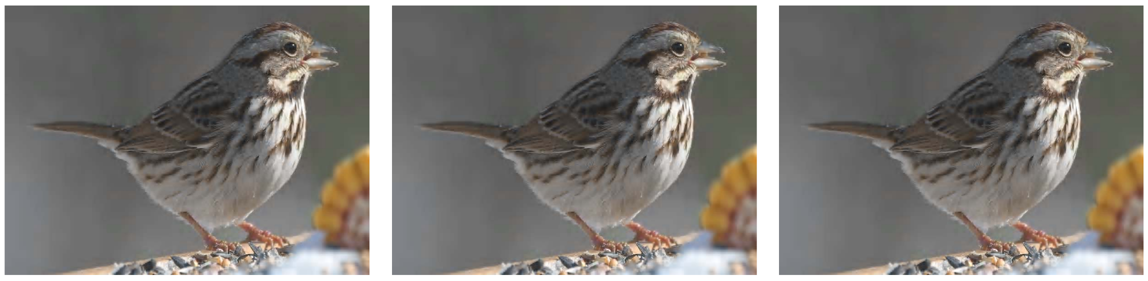

Figure 3. Simulation Results of Transform and Quantization Processor for H.264 Video Encoder

a. Original bird Image (800x512 pixels)
b. Reconstructed bird Image using Matlab, PSNR: 38.1 dB (Q step =16)
c. Reconstructed bird Image using Verilog, PSNR: 38.3 dB (Q step =16)

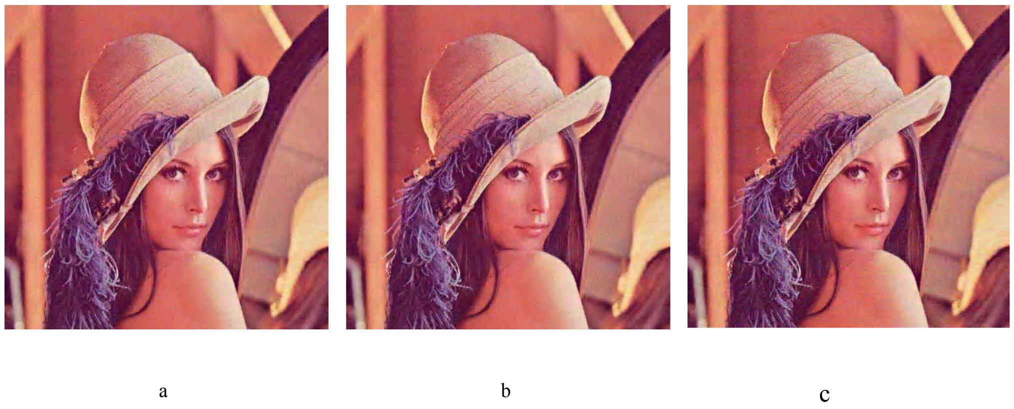

Figure 4. Simulation Results of Transform and Quantization Processor for H.264 Video Encoder
a Original Lena Image (512x512 pixels)
b Reconstructed Lena Image using Matlab, PSNR: 37.1 dB (Q step =8)
c Reconstructed Lena Image using Verilog, PSNR: 37.3 dB (Q step =8)

Subsequently, the core modules of the encoder such as the transform and quantization processor as described earlier were realized using Verilog for ASIC/FPGA implementation. The resulting qualities of the reconstructed images obtained using Matlab and Verilog compare favorably as can be seen from "Fig. 3" and "Fig. 4" for two sample images. It may be seen from the notes "b" and "c" that Verilog result is very close to the Matlab result (within 0.4 dB) since the Verilog codes use at least 16-bits precision. The results shown in "Fig. 4" were obtained for Qstep = 8 (i.e., QP = 22). The peak signal to noise ratio (PSNR) for 4:2:0 format was 37.1 with the Matlab coding and 37.3 by using the Verilog code. Verilog result however, offered a higher PSNR value, namely, 37.3 dB compared to Matlab. The proposed FPGA implementation, which processes motion pictures of size 1024x768 pixels at 25 frames/sec., is faster by about two times the SOC/ASIC implementation reported by Qiang Peng and Jin Jing [2].

## VI. Conclusions

An FPGA implementation of the core processors of H.264 video encoder, namely, the transform and quantization processors has been presented. The results were compared by considering the images which are of different sizes. The gains obtained vary with the video sequence. The 4x4 integer transform used is significantly simpler and faster than the 8x8 DCT used in MPEG 2. A significant improvement over MPEG 2 is the reduction of blocking artifacts, especially for high compression. The desired compression can be selected by the user in the implemented processor. The implementation produces high resolution, high quality reconstructed pictures and compares favorably with another implementation. The processor is implemented on a Xilinx Vertex – II Pro XC2VP30 FPGA. The gate count of the implementation is approximately 1 Million working at a frequency of 208 MHz. It can process 1024x768 pixel color images in 4:2:0 format at 25 frames per second. The reconstructed picture quality is better than 35 dB. Currently, work is under progress for coding the intra-prediction, inverse quantization and inverse transform.

## References

[1] Joint Video Team, Draft ITU-T Recommendation and final draft international Standard of Joint video specification, ITU-T Rec H.264 and ISO/IEC 14496-10 AVC, Mar. 2005.

[2] Qiang Peng and Jin Jing, "H.264 system on chip design and verification, the IEEE 2003 Workshop on Signal Processing Systems (SIPS'03), 2003.

[3] S. Ramachandran and S. Srinivasan, "A fast, FPGA-based MPEG-2 video encoder with a novel automatic quality control scheme," Elsevier, Journal of Microprocessors and Microsystems, UK, 25, pp. 449-457, 2002.

[4] I. E. G. Richardson, "H.264 MPEG4 Part 10 Overview," www.vcodex.com, 20/12/2002.

[5] Gulistan Raja, Sadiqullah Khan, Muhammad Javed Mirza, "VLSI architecture and implementation of H.264 integer transform," The IEEE 2005 Workshop on Signal Processing Systems (SIPS'05), 2005.

[6] Liu Ling-zhi, Qiu Lin, Rong Meng-tian, Jiang Li, "A 2-D forward/inverse integer transform processor of H.264 based on highly parallel architecture," Proceedings of the 4th IEEE International workshop on System-on-Chip for Real-Time Applications (IWSOC'04), 2004.