

Implementation of Context Adaptive Variable Length Coder for H.264 Video Encoder

N. Keshaveni¹, S. Ramachandran² and K.S. Gurumurthy³

¹Dr. MGR University, Dept. of Electronics and Communication, Chennai, India

Email: keshaveni@gmail.com

²National Academy of Excellence, Bangalore, India

³UVCE, BU, Bangalore, India

Email: {ramachandran_ns@yahoo.com, drksgurumurthy@gmail.com}

Abstract— This paper proposes the implementation of context adaptive variable length coder for H.264 video encoder. The implementation is capable of bringing about compression of video sequences as per user specification and is capable of processing high resolution pictures of sizes of up to 1024 x 768 pixels, encoding at a real time frame rate of 25 fps. The variable length coder has been realized using Verilog RTL coding. The compression achieved by the implementation is over 10 and the reconstructed picture quality is better than 35 dB. The quantized coefficient inputs to the context adaptive variable length coder is generated by the integer transform and quantization processor coded in Verilog reported in an earlier work. In order to get the reconstructed picture, context adaptive variable length decoder, inverse quantization and inverse integer transform have been realized using Matlab.

Index Terms— CAVLC, CAVLD, inverse quantization, inverse integer transform, compression, PSNR, Video Encoder and Verilog.

I. INTRODUCTION

Video compression plays an important role in enabling video processing on multimedia systems. Video compression is essential in order to reduce the storage requirements to manageable levels and to transmit data with the existing channel capacities. 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 [1-3]. An implementation of the context adaptive variable length coder (CAVLC), which uses two-stage block pipelining scheme for parallel processing of two 4x4 blocks using cell based design flow has been reported by Tung-Chien Chen “et al.” [4]. It has been reported as cell based design with 23.6 K logic gates operating at 100 MHz. A “structured truncated Golomb code” for the context based adaptive VLC by Sadaatsu Kato “et al.” [5] offers similar coding efficiency in comparison with the dedicated unstructured VLC, while reducing the size of memories to store code tables. Ihab Amer “et al.” [6] have reported VLSI prototype for Context-based adaptive variable length coding for lossless compression. Their architecture for the CAVLC has been realized using VHDL language and simulated using Modelsim. It is synthesized using Synplify Pro 7.1 and the critical path estimated by the synthesis tool is 31.3 ns. This corresponds to the maximum operating frequency of about 32 MHz. In

another paper, a simple and cost effective video encoder with memory efficient context adaptive variable length coder (CAVLC) is proposed for low cost multimedia applications by Yeong-Kang Lai “et al.” [7]. It is implemented on a Xilinx FPGA prototyping board. Its maximum operating frequency is 28 MHz and the gate count is 9171 (NAND2) in TSMC 0.35 technology.

In an earlier work, the present authors have designed and implemented integer transform and quantization processor for H.264 Video encoder on FPGA [8]. The FPGA implementation of that work is capable of processing moving pictures of sizes of up to 1024 x 768 pixels at a real time frame rate of 25 fps. The reconstructed picture quality obtained is better than 35 dB. The present work on CAVLC is a sequel in order to implement the complete H.264 video encoder.

This paper is organized as follows: The algorithm for processing CAVLC processor is presented in Section II. This is followed by the detailed hardware architecture in the next section. The verification methodology of CAVLC is described in Section IV, while the simulation results are discussed in Section V. The conclusions are presented in the last section.

II. ALGORITHM FOR PROCESSING CONTEXT ADAPTIVE VARIABLE LENGTH CODER

H.264 video encoder consists of various functional modules such as an intra-prediction processor, an integer transform and quantization processor and a CAVLC processor. Intra-prediction is done in order to exploit the spatial redundancy within a frame of a motion picture. Each pixel is predicted based on the values of its neighboring pixels that are available.

A video sequence is input to the integer transform and quantization processor in 4:2:0 format [1], where the Y, Cb and Cr components are intra-predicted, transformed and quantized. The quantized coefficients are then fed to the CAVLC module, which assigns variable length codes to get the desired compressed bit stream [9, 10]. The quantized coefficients are a 4x4 block of data which is scanned in a zig-zag manner and then fed to the CAVLC processor. After quantization, the data typical-

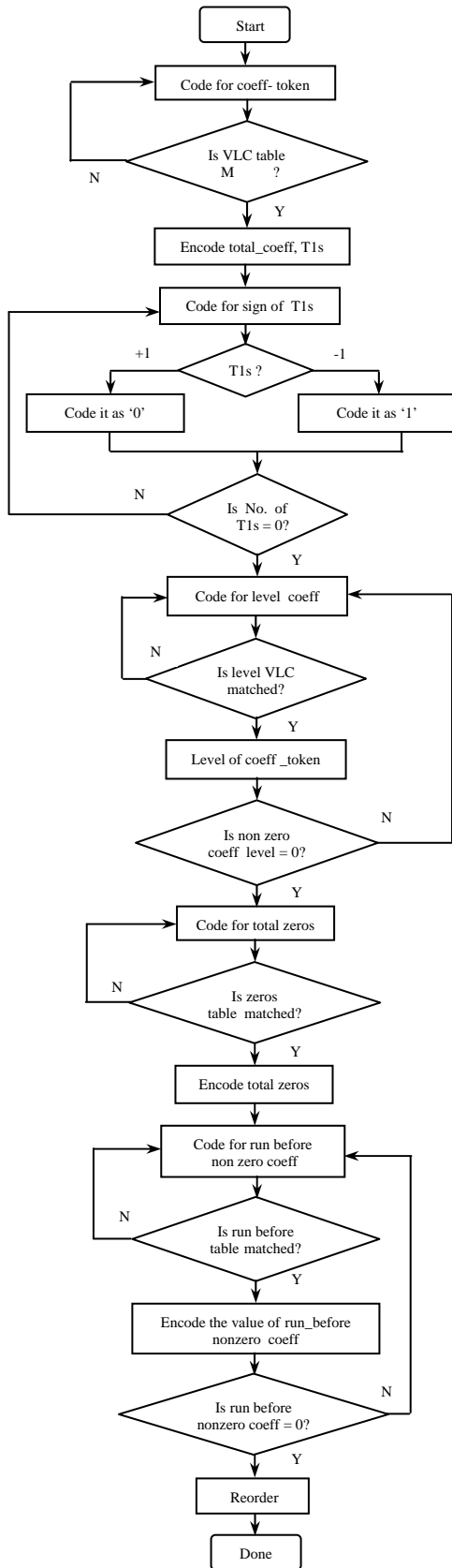


Figure 1. Flowchart for processing CAVLC

ly sparse containing more number of zeros than non-zero coefficients. The CAVLC is used to encode the quantized data. The algorithm for processing the CAVLC processor is shown in “Fig. 1”.

The quantized coefficients are first reordered and fed to the CAVLC processor. The first VLC, i.e., coeff_token, encodes both the total number of non-zero coefficients (Total Coeffs) and the number of trailing +/- 1 values (T1). Depending on the total number of coefficients, the VLC table is chosen to encode the total nonzero coefficients [2]. For each T1 (trailing +/-1) signaled by coeff_token, a single bit encodes the sign (+ = 0, - = 1). These are encoded in reverse order, starting with the highest-frequency T1. The level (sign and magnitude) of each remaining non-zero coefficient in the block is encoded in reverse order, starting with the highest frequency and working back towards the DC coefficient. The choice of VLC table to encode each level is adaptive depending on the magnitude of each successive coded level (context adaptive). Total zeros is the sum of all zeros preceding the highest non-zero coefficient in the reordered array. This is coded with a VLC. The number of zeros preceding each non-zero coefficient (run_before) is encoded in reverse order. A run_before parameter is encoded for each non-zero coefficient, starting with the highest frequency, with two exceptions:

- (a) If there are no more zeros left to encode, i.e., if [run_before] = Total Zeros, it is not necessary to encode any more run_before values.
- (b) It is not necessary to encode run_before for the final non-zero coefficient (lowest frequency). The VLC for each run of zeros is chosen depending on (i) the number of zeros that have not yet been encoded (Zeros Left) and (ii) run_before.

The following example illustrates the order of processing CAVLC and the encoded bit stream format, where the table Num-VLC0 is used to encode coeff_token.

A sample 4x4 blocks Values:

```
0 4 -1 0
0 -1 1 0
1 0 0 0
0 0 0 0
```

Reordered Sequence of the sub-block:

0,4,0,1,-1,-1,0,1,0...

In this example, Total Coeffs = 5 (indexed from highest frequency [4] down to the lowest frequency [0]); Total Zeros = 3 and T1s = 3.

In fact, there are 4 trailing 1s but standard provides for only a maximum of 3 that are encoded as a “special case”.

The encoding is as follows:

Since Total Coeffs=5 and T1s=3, the coeff_token (5, 3) from the Table Num_VLC0 yields the following bit-stream:
0000100

Next, the sign of trailing 1s are taken into account.

T1 sign (4) +: 0

T1 sign (3) -: 1

T1 sign (2) -: 1

In the next step, the level of the remaining non-zero coefficients is encoded as follows:

Level (1) +1 (use Level_VLC0): 1

Level (0) +4 (use Level_VLC1): 00010

After this, the total number of zeros is encoded.

Total Zeros 3: 111

Thereafter, we need to compute the “run_before” for every non-zero coefficient beginning from the highest frequency as follows.

run_before (4): Zeros Left=3; run_before=1: 10

run_before (3): Zeros Left=2; run_before=0: 1

run_before (2): Zeros Left=2; run_before=0: 1

run_before (1): Zeros Left=2; run_before=1: 01

run_before (0): Zeros Left=1; run_before=1

No code is required for the last “run_before” since it is the last coefficient.

The transmitted bit stream for this block is got by putting together all the encoded bits as:

0000100011100010111101101.

III. ARCHITECTURE OF CAVLC PROCESSOR

The video data are input to the Transform and Quantization processor [8] and, in turn is assigned the context adaptive variable length codes by the CAVLC Processor [9]. The proposed architecture of the CAVLC processor is shown in “Fig. 2”. The output of the quantizer is fed as the input “coeff_in” to the Dual RAM module “dram_nc”. Its validity is asserted by the signal “coeff_valid_in”. The “clk” is the system clock and “reset_n” is the global reset signal in order to reset the system during power on conditions. The “halt” is provided for incorporating rate control in future in order to regulate the compressed bit stream generated by the CAVLC Processor. The module “dram_nc” consists of double buffer in order to store the quantized coefficients.

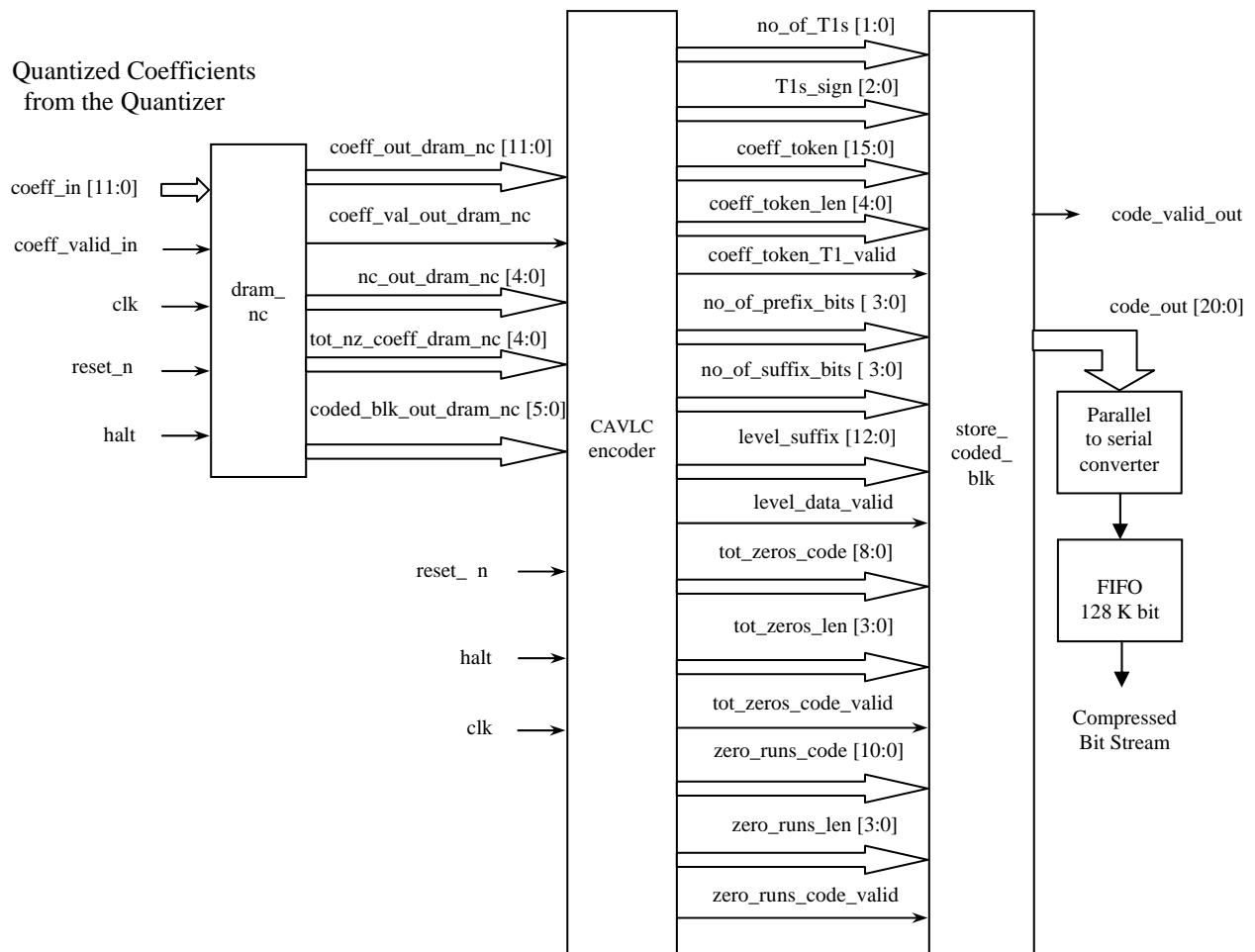


Figure 2. Architecture of CAVLC Processor

As one buffer gets filled with the quantized coefficients, the CAVLC is processed concurrently, fetching the coefficients from the other buffer in a zig-zag order. The reordered coefficients are output on the pins marked “coeff_out_dram_nc”. This is valid when the signal “coeff_valid_out_dram_nc” is asserted. From the reordered coefficients, the total number of nonzero

coefficients is thereafter found out. The total number of coefficients and the total number of non zero coefficients are output as signals “nc_out_dram_nc” and “tot_nz_coeff_dram_nc” respectively. The blocks of luminance and chrominance components are identified by the signal “coded_blk_out_dram_nc”. The outputs “coeff_out_dram_nc” and “coeff_valid_out_dram_nc” of

the “dram_nc” module are fed as the inputs to the next module called “CAVLC encoder”, where the context adaptive variable length codes are assigned. The total number of trailing +/- 1 value called “no_of_T1s” (also called T1s) is calculated. The sign of the T1s are assigned as “T1s_sign”. The total number of non-zero coefficients and the number of T1s are concatenated and the resulting signal is called as the “coeff_token”. The output marked as “coeff_token_len” gives the size of “coeff_token”. The validity of the signal is asserted by “coeff_token_T1_valid”.

The level (sign and magnitude) of each remaining non-zero coefficients in the block is encoded in reverse order, starting with the highest frequency and working back towards the DC coefficient. The choice of VLC table to encode each level adapts depending on the magnitude of each successive coded level (context adaptive). The number of prefix bits “no_of_prefix_bits” and suffix bits “no_of_suffix_bits” corresponding to each non zero coefficient is taken and the total level is assigned as “level_suffix” and is valid when the signal “level_data_valid” goes high. The total number of Zeros “tot_zeros_code” is the sum of all zeros preceding the highest non-zero coefficient in the reordered array. This is assigned with variable length codes. The size or the length of the total number of Zeros is known as “tot_zeros_len”. These signals are valid when the signal “tot_zeros_code_valid” is asserted. The number of zeros preceding each non-zero coefficient (run_before) is encoded in reverse order. The VLC for each run of zeros is chosen depending upon (a) the number of zeros that have not yet been encoded (Zeros Left) and (b) run_before. Depending upon the “run_before”, the “zero_runs_code” and “zero_runs_len” are assigned and their validity signaled by “zero_runs_code_valid”.

The context adaptive variable length codes issued out of the module “store_coded_blk” is the compressed output “code_out [20:0]” and is valid by asserting the signal “code_valid_out”. This output is in parallel form. In order to get the data in a serial format, a parallel to serial converter is used as shown in the architecture. The serial output is fed to a first in first out (FIFO) module before it is transmitted over a serial channel.

IV. VERIFICATION OF CAVLC

The Verilog implementation of transformation and quantization processor had been carried out in an earlier work [8]. The same modules were used in the present work and are shown in “Fig. 3”, which is a verification scheme in order to validate the CAVLC implemented. It may be noted that the hardware design implemented is marked as Verilog. The transformed and quantized coefficients are fed to the CAVLC processor, which generates the context adaptive variable length codes as described in the previous section on architecture. Since the reconstruction of the video sequence is possible only if we implement the inverse processors such as CAVLD, inverse quantization (IQ) and inverse transformation (IT),

the said modules were coded in Matlab as shown in “Fig. 3”. The quality of the reconstructed image is assessed by peak signal to noise ratio (PSNR) value defined as follows:

$$PSNR = 10 \log_{10} \{ 255^2 \times MN / (\sum_i \sum_j (i_o - i_r)^2) \} \text{ dB} \quad (1)$$

where i_o is the original pixel intensity and i_r is the corresponding reconstructed pixel intensity.

The inverse quantization is expressed as

$$W_{ij}^i = Z_{ij} * V_{ij} * 2^{\text{floor}(QP/6)} \quad (2)$$

where Z_{ij} are the quantized coefficients and V_{ij} are the rescaling factors dependent upon the coefficient position as specified in the H.264 standard. The inverse integer transform that follows the inverse quantization stage is as follows:

$$X = C_i' * W^i * C_i \quad (3)$$

where

$$C_i = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1/2 & -1/2 & -1 \\ 1 & -1 & -1 & 1 \\ 1/2 & -1 & 1 & -1/2 \end{pmatrix} \text{ and } C_i' \text{ is its transpose.}$$

The algorithm for performing inverse quantization and inverse transformation is the reverse process of transformation and quantization respectively [8]. The CAVLD, IQ and IT processors coded in Matlab were run in order to get the reconstructed picture, thus verifying the CAVLC output.

V. SIMULATION RESULTS

The H.264 video codec was first implemented in Matlab in order to estimate the quality of the reconstruct-

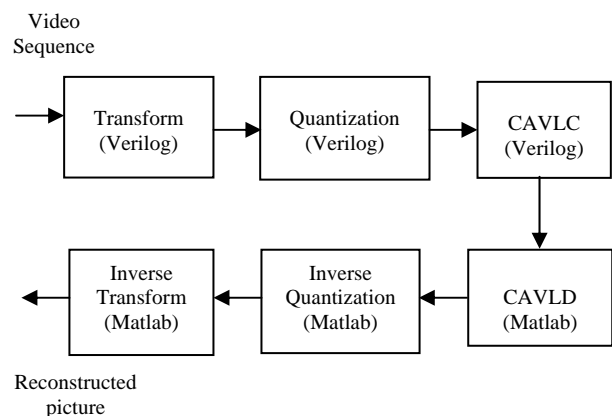


Figure 3. Block Diagram of Scheme for Verification



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

TABLE I.
 QUALITY OF THE RECONSTRUCTED LENA IMAGE AND
 COMPRESSION ACHIEVED FOR THE VERILOG
 IMPLEMENTATION OF CAVLC PROCESSOR

Q_{Step}	QP	PSNR in dB	Compression Ratio
4	16	39.85	6.08
8	22	37.30	10.09
16	28	32.06	11.96
32	34	25.80	14.14

ed image. The results obtained for the Matlab Codec for different values of quantization parameter step sizes and corresponding quantization parameter values are shown in Table I. The CAVLD processor, Inverse quantization and Inverse Transformation were coded using Matlab as described earlier. The CAVLC processor was coded in Verilog and integrated with the Transform and Quantization module developed earlier [8]. The compression desired is input to the TQ Processor at the Q_{Step} pins by the user. From the results tabulated, it may be seen that the quality of the reconstructed image obtained, as described in Section IV, is over 37 dB for Lena image and the compression achieved by Verilog design of TQ and CAVLC Processors is over 10 for a Q_{Step} size of 8. Usually, a PSNR value of 35 dB and above means that the reconstructed picture is indistinguishable from the original. Above 30 dB, the picture quality is accepted as good. Naturally, one needs to trade-off between compression and quality.

The TQ and CAVLC Processors implemented in Verilog were simulated using Modelsim. The compressed bit stream output of CAVLC was generated as text files with Y, Cb and Cr components. This output was subsequently fed to the CAVLD, IQ and IT modules coded in Matlab, thus reconstructing the picture. The original and the reconstructed pictures of Lena are shown in "Fig. 4" as an example. The Matlab program also computes the quality of the reconstructed picture. The PSNR obtained for Lena image for the complete Matlab codec was 37.1 dB and, for Verilog encoder it was 37.3 dB by using the scheme presented in Fig. 3, where Verilog CAVLC is used. The compression achieved is over 10 with the present scheme. This result thus validates the hardware implementation of CAVLC processor presented in this paper.

VI. CONCLUSIONS

The architectural design of context adaptive variable length coder for H.264 video encoder was presented. The CAVLC processor was coded in Verilog and integrated with other functional module such as the Transform and Quantization Processor developed in an earlier work. The complete video codec was also realized using Matlab for validating the Verilog CAVLC Processor. The results show that the reconstructed picture quality is indistinguishable from the original in spite of achieving high compression. The compression can be changed according to the user's choice. The end user may trade-off between compression and quality.

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 AVC, March 2005.
- [2] I. E. G. Richardson, H.264 and MEG-4 Video compression (Video coding for next Generation multimedia), John Wiley, January 2004.
- [3] D. LeGall, MPEG: A video compression standard for multimedia application, Communication, ACM, Vol. 34, pp. 46-58, Apr. 1991.
- [4] Tung-Chien Chen, Yu-Wen Huang, Chuan-Yong Tsai, Bing-Yu Hsieh, and Liang-Gee Chen, Dual block pipelined VLSI architecture of entropy coding for H.264/AVC baseline profile, pp. 271-275, IEEE 2005.
- [5] Sadaatsu KATO, Kazuo SUGIMOTO, Satoru ADACHI and Minoru ETOH, Structured truncated Golomb Code for context based adaptive VLC, pp. 405-408, ICASSP 2003.
- [6] Ihab Amer, Wael Badawy and Graham Jullien, towards MPEG-4 Part 10 system on chip: A VLSI prototype for context based adaptive variable length coding, pp. 275-279, SIPS 2004.
- [7] Yeong-Kang Lai, Chih-Chung Chou, and Yu-Chieh Hung, A Simple and Cost Effective Video Encoder with Memory-Reducing CAVLC, pp. 432-435, IEEE 2005.
- [8] Keshaveni, S. Ramachandran and K. S. Gurumurthy, Design and Implementation of Integer Transform and Quantization Processor for H.264 Encoder on FPGA, International conference on advances in computing, Control and Telecom Technologies, ACT 2009., in press.
- [9] JVT Document JVT-C028, G. Bjontegaard and K. Lillevold, "Context-Adaptive VLC Coding of coefficients", Fairfax, VA, May 2002.