Implementation of High Performance CAVLC for H.264/AVC Video Codec

Daeok Kim, Eungu Jung, Hyunho Park, Hosoon Shin, and Dongsoo Har, *Member, IEEE*

Abstract— **Context-based Adaptive Variable Length Coding (CAVLC) is an entropy coding for H.264/AVC video codec. Since the CAVLC is highly context-adaptive and of a block-based context formation, high coding efficiency is achieved. However, its high complexity causes various difficulties in full-hardware implementation. This paper presents a high performance hardware architecture of CAVLC. The proposed architecture is implemented in a FPGA device, and verified by RTL simulations. The implementation results show that the proposed architecture encodes a 4×4 block per 16 clock cycles, and achieves a real-time processing for 1920×1088 frame size with 30-fps video at 100MHz clock speed.**

I. Introduction

To date, video compression algorithms and their hardware implementation have continuously been studied by various research groups. H.264/AVC [1] is the latest international video coding standard, which improves significant coding efficiency up to 50% compared with previous MPEG4 video coding standard. In addition, the H.264 supports bit rate savings, high quality video, error resilience and network friendliness. Because of these advantages, it is used for manifold applications such as entertainment (Advanced TV, Digital cinema), education, medical images and personal communications (video-conferencing, video-phone) [2].

The H.264/AVC has three types of profile (i.e., baseline, main, extended) and two types of entropy coding (i.e., a combination of Exponential Golomb codes [3] and CAVLC or solely Context-based Adaptive Binary Arithmetic Coding (CABAC)). Particularly, the combination of Exponential-Golomb coding and CAVLC is used for all profiles [4]. The Exponential-Golomb coding is based on variable length codes with a regular construction, while the CAVLC is a block-based entropy coding.

Recently, extensive research on hardware implementation of CAVLC encoder has been carried out [5-7]. Amer *et al.* [5] proposed the pipeline architecture that is applied only to the quantized transform coefficients of the luminance component. Lai *et al.* [6] implemented the video encoder only encoding zig-zag ordered 4×4 blocks of DCT coefficients. Chen *et. al.* [7] proposed the first entropy coding engine for H.264/AVC baseline profile encoder as a hardware and software co-design. These papers have shown efficient architectures, satisfying requirements for target applications such as real-time constraint (high throughput) and low cost. However, their implementations are not full-hardware-based. Specifically, they did not explain encoding process for all components, e.g. luminance and chrominance, in a video sequence. Secondly, they did not describe the hardware implementation regarding the parameter nC which is a function of the number of non-zero coefficients in neighboring blocks and used for CAVLC when choosing an appropriate look-up table. In this paper, new full-hardware architecture to solve these problems to meet high speed requirement of CAVLC is proposed.

This paper is organized as follows. Section II provides the algorithm and an analysis for entropy coding. Section III explains the proposed architecture of the H.264/AVC entropy coding engine and also the solutions resolving hardware implementation issues. The implementation by an FPGA device is explained in Section IV. Finally, Section V concludes this paper.

Daeok Kim(e-mail: dokim@gist.ac.kr), Eungu Jung(e-mail: egjung@gist.ac.kr), Hyunho Park(e-mail: phhyun@gist.ac.kr), Hosoon Shin(e-mail: hosoon@gist.ac.kr), Dongsoo Har(corresponding author; phone: +82-62-970-2212; e-mail: hardon@gist.ac.kr) are with the Gwangju Institute of Science and Technology, Republic of Korea.

II. Entropy Coding for H.264/AVC Video Codec

The purpose of entropy coding is to represent source information using the least number of bits. The entropy coder typically uses statistical compression techniques that are not necessarily unique in their applications like image or video coding [8]. The right smaller square of dashed line in Fig. 1 shows that entropy coding is performed as the last step of the video coding process.

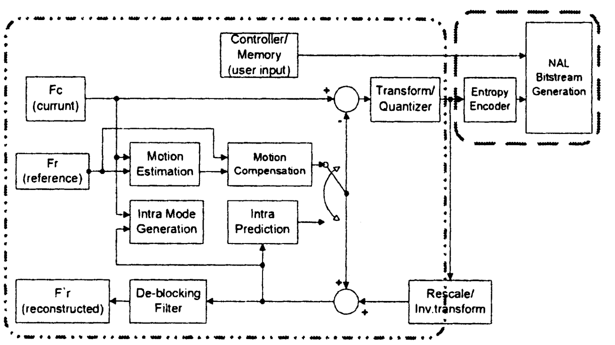


Fig. 1. Encoder architecture of H.264/AVC.

A. Features of entropy coding of H.264/AVC

A Variable Length Coding (VLC) improves coding efficiency by assigning small number of bits to frequently used symbols and relatively larger number of bits to less frequently used symbols. The CAVLC is characterized by several features; i)adoption of run-level coding to represent strings of zeros compactly, ii)utilization of Trailing ones (equal to ± 1), iii)choice of a look-up table depending on the number (nC) of non-zero coefficients in neighboring blocks, and iv)the choice of a VLC look-up table for the Level depending on a recently-coded value [9].

When the parameter entoropy_coding_mode_flag specified in the standard is set to 0, syntax elements are coded using the combination of Exponential-Golomb and CAVLC as shown in Fig. 2. The syntax elements mean inputs of entropy encoder including context information and residual data. The Exponential-Golomb coding encodes header information in Network Abstraction Layer (NAL) due to its strong error resilience, based on variable length codes with a regular construction. The CAVLC encodes only residual data (quantized transform coefficients) owing to its highly context-adaptive feature.

B. Algorithm of Exponential-Golomb coding

The Exponential-Golomb coding generates a codeword according to code_num parameter. The code generation method is carried out by following equations.

$$codeword = [M\ zeros][1][INFO]$$
$$M = floor(\log_2[code\_num + 1])$$
$$INFO = code\_num + 1 - 2^M$$

Value of the parameter code_num is taken according to four mapping types as shown in Fig. 3 [1].

C. Algorithm of CAVLC

Since the CAVLC is a block-based entropy coding, each pixel in one block is related to other pixels in the block. As shown in Fig. 2, the CAVLC encodes five variable length codes after zig-zag scanning of 4×4 DCT coefficients.

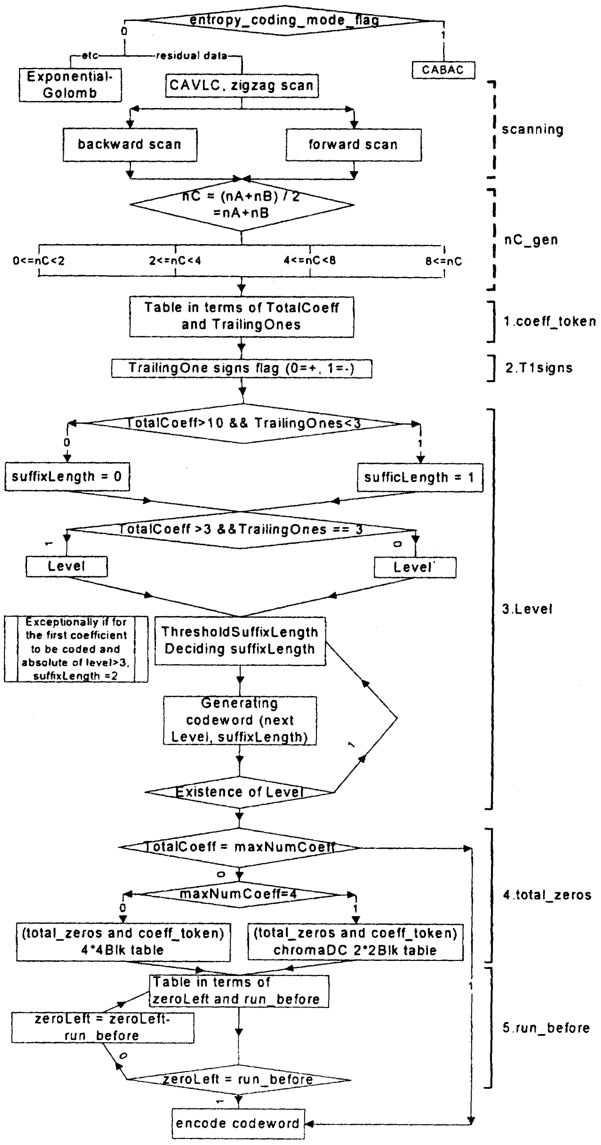
1) coeff_token

This variable is generated from the total number of non-zero coefficients (TotalCoeff), the number of TrailingOnes (T1s), and nC. The nC is determined by the number of non-zero coefficients in the left adjacent 4×4 block (nA) and the upper block (nB), which are already coded. The nC is calculated as follows.

$$nC = \begin{cases} round((nA+nB)/2), & block\ A,\ B\ are\ both\ available. \\ nA+nB, & otherwise. \end{cases}$$

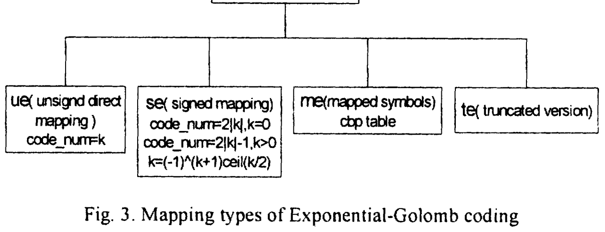
2) TrailingOne signs

The TrailingOne (T1) is the number of values ± 1, which represent high spatial frequency components. Its maximum value is 3. The TrailingOne signs (T1signs) is encoded by a single bit (0=+1, 1=−1) in reverse order.



entropy_coding_mode_flag
0
1
etc
residual data
Exponential-Golomb
CAVLC, zigzag scan
CABAC
backward scan
forward scan
nC = (nA+nB)/2
=nA+nB
0<=nC<2
2<=nC<4
4<=nC<8
8<=nC
Table in terms of TotalCoeff and TrailingOnes
TrailingOne signs flag (0=+, 1=-)
TotalCoeff>10 && TrailingOnes<3
suffixLength = 0
sufficLength = 1
TotalCoeff >3 &&TrailingOnes == 3
Level
Level'
Exceptionally if for the first coefficient to be coded and absolute of level>3, suffixLength =2
ThresholdSuffixLength Deciding suffixLength
Generating codeword (next Level, suffixLength)
Existence of Level
TotalCoeff = maxNumCoeff
maxNumCoeff=4
(total_zeros and coeff_token) 4*4Blk table
(total_zeros and coeff_token) chromaDC 2*2Blk table
Table in terms of zeroLeft and run_before
zeroLeft = zeroLeft-run_before
zeroLeft = run_before
encode codeword
scanning
nC_gen
1.coeff_token
2.T1signs
3.Level
4.total_zeros
5.run_before

Fig. 2. Flow chart of CAVLC.



Exponential-Golomb
ue(unsignd direct mapping) code_num=k
se(signed mapping) code_num=2|k|,k<=0 code_num=2|k|-1,k>0 k=(-1)^(k+1)ceil(k/2)
me(mapped symbols) cbp table
te(truncated version)

Fig. 3. Mapping types of Exponential-Golomb coding

3) Level

The Level means the sign and magnitude of non-zero coefficients other than TrailingOnes. The codeword for each Level is made up of a prefix and a suffix. The length of the suffix (suffixLength) is between 0 and 6 bits, and it is adapted depending on the magnitude of each successively coded Level. The encoding process involving the Level is shown in Fig. 2.

4) run_before

The number of zeros preceding each non-zero coefficient (run_before) is encoded in reverse order.

5) total_zeros

This variable is determined by the total number of zeros before the last non-zero coefficient and also by TotalCoeff. The reason for generating total_zeros is that many blocks include a lot of non-zero coefficients at the start pixels in a block, and this approach means that run_before at the start pixels in a block need not be encoded [4].

D. DCT coefficient blocks in a macrcblock

The quantized transform coefficients block has a luminance (Luma) component block and two chrominance component (ChromaCb, ChromaCr) blocks. These three component blocks are divided into two categories decided by the transform/quantization engine. The first type consists of DC component block and AC component block. The second type adds DC_AC component block to the first type. One notable fact is that the Luma DC component block is generated only in intra 16 × 16 prediction mode. Thus the prediction mode decides the order, the property, and the total number of pixels of the input component blocks coming into CAVLC. Another important point in the block order is that there are 2 × 2 blocks between 4 × 4 blocks. Fig. 4 shows the order of component blocks in a macroblock in detail. Fig. 4(b) implies inter prediction or intra 4 × 4 prediction.

III. FULL-HARDWARE IMPLEMENTATION

The proposed hardware architecture for the entropy coder in baseline profile of H.264/AVC is illustrated in Fig. 5. As explained in Fig. 5, the proposed architecture consists of Exponential-Golomb coding scheme and CAVLC method. The Exponential-Golomb coder generates four types (ue, se, me, and te) of codewords. The CAVLC is composed of three blocks: Scanning block, Arithmetic and Selection (AS) block, and PackData block. The other remaining blocks are the nC_generation (nC_gen) block and FIFO for Level (fifo_level) and run_before (fifo_run). The Scanning block

Luma_DC 4 × 4 (1 block)
↓
Luma_AC 4 × 4 (16 blocks)
↓
ChromaCbDC 2 × 2 (1 block)
↓
ChromaCrDC 2 × 2 (1 block)
↓
ChromaCbAC 4 × 4 (4 blocks)
↓
ChromaCrAC 4 × 4 (4 blocks)

(a) Intra 16 × 16 mode

LumaDC_AC 4 × 4 (16 blocks)
↓
ChromaCbDC 2 × 2 (1 block)
↓
ChromaCrDC 2 × 2 (1 block)
↓
ChromaCbAC 4 × 4 (4 blocks)
↓
ChromaCrAC 4 × 4 (4 blocks)

(b) otherwise

Fig. 4. Order of component blocks in a macroblock according to prediction mode.

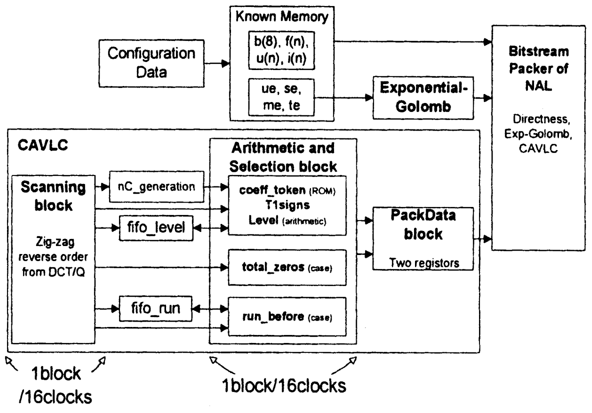


Fig. 5. Proposed architecture for full-hardware implementation of entropy coding for baseline profile of H.264/AVC.

requires 16 clock cycles to scan 16 pixels in 4 × 4 block, and the AS block also needs 16 clock cycles to generates a codeword. Due to the order of DCT coefficient blocks incoming to CAVLC engine, the 2 × 2 block should wait until the 4 × 4 block is encoded. Thus the 2 × 2 block as well as the 4 × 4 block needs 16 clock cycles in CAVLC process.

A. Scanning block

It receives zig-zag scanned and reverse ordered coefficients from the transform/quantization engine and extracts six values (TotalCoeff, T1s, T1signs, Level, total_zeros, run_before) to be stored.

B. Arithmetic and Selection block

The three blocks (coeff_token, total_zeros, run_before) generate the codeword by reading Look Up Tables (LUTs). The T1signs block encodes just a T1signs value calculated by the Scanning block. The Level block generates the codeword by an arithmetic method without using LUT.

C. PackData block

The simultaneous output data from the AS block are loaded into registers. Then the data are packed by using two 32-bit registers.

D. FIFO

To handle one block during 16 clock cycles, additional clock cycles can not be used for control of FIFO blocks located between the Scanning block and the AS block. So fifo_level block and fifo_run block adopt handshaking protocol [10] for data transfer.

E. nC_generation block

It generates the nC of every component (Luma or Chroma) block. The architecture of nC is illustrated in Fig. 6, where coeff_mode means type of component block. The nC_generation block includes pre_nC_gen block and main_nC_gen block. All component blocks except chromaDC component block are neccessary to calculate nC. Particularly, since a macroblock consists of LumaDC component block and LumaAC component block in intra 16 × 16 prediction mode,

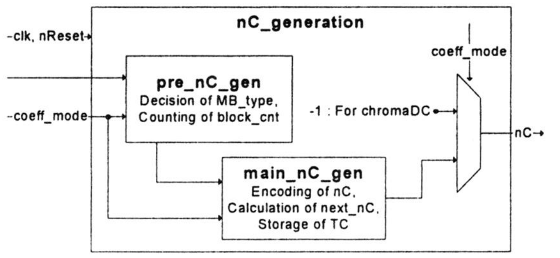


Fig. 6. Architecture of nC_generation block.

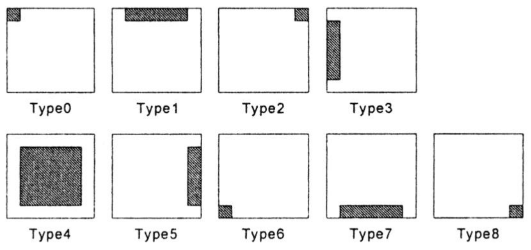


Fig. 7. Types of macroblock at the frame.

the nC of LumaDC is equal to the nC of first LumaAC component block. The nC of chromaDC component block is fixed at −1 regardless of prediction mode.

1) pre_nC_gen block

It numbers a block in the macroblock and divides a macroblock into 9 types as shown in Fig. 7. The black color box presents macroblocks belonging to each type.

2) main_nC_gen block

It has three functions. It encodes the nC and calculates next_nC (the number of nC of a next block), and saves TotalCoeff in current block. Due to Finite State Machine (FSM) architecture, this block generates the nC of Luma component blocks, ChromaCbAC component blocks, and ChromaCrAC component blocks in the same fashion.

IV. Implementation Results

The proposed architecture is designed by Verilog language, and implemented in an FPGA using Synplify Pro 8.2. Simulations are performed by ModelSim 6.0. The performance of the proposed architecture is calculated as follows.

$$\begin{aligned}
&\textit{The number of clock cycles needed for } 1920\times1080 \textit{ HD video with } 30-\textit{fps}\\
&= \textit{The number of clock cycles per block}\\
&\quad \times \textit{The number of blocks per macroblock}\\
&\quad \times \textit{The number of macroblocks per } 1920\times1088 \textit{ HD video frame}\\
&\quad \times \textit{The number of frames per second}\\
&= 16\times27\times8160\times30 \textit{ clock cycles}\\
&= 105{,}753{,}600 \textit{ clock cycles}
\end{aligned}$$

The required number of clock cycles is calculated, assuming the worst case without skip mode. It means that the proposed architecture can encode 1080 HD format at 30-fps video at about 100MHz. Also the equivalent gate count of hardware implementation is shown in Table I.

TABLE I
EQUIVALENT GATE COUNT OF ENTROPY CODER.

	Item	Equivalent Gate
	Exp-Golomb	2,108
	Scanning	1,362
	nC_gen	16,294
C	coeff_token	298
A	level	2,249
V L	total_zeros	564
C	run_before	1,361
	PackData	9,265
	Total	33,501

V. Conclusions

In this paper, full-hardware implementation of CAVLC for H.264/AVC is presented. The proposed architecture could encode one block per 16 clock cycles. Thus the designed hardware architecture enables real-time processing for 1920 × 1088 HD video frame with 30-fps, using 100MHz clock.

Acknowledgment

This work has been supported in part by the Center for Distributed Sensor Network (CDSN) at GIST, in part by the GIST Technology Initiative (GTI), in part by the MIC, Korea, under the ITRC support program supervised by the IITA" (IITA-2005-C1090-0502-0029), and in part by IC Design Education Center (IDEC).

References

[1] Joint Video Team of ITU-T and ISO/IEC, "ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC," May 2003.
[2] T. Wiegand, G. J. Sullivan, G. Bjontegaard and A. Luthra, "Overview of the H.264/AVC Video Coding Standard," IEEE Transaction on Circuits and Systems for Video Technology, Vol. 13, No. 7, pp.506 576, July 2003.
[3] S. W. Golomb, Run-length encoding, IEEE Trans, on Inf. Theory, IT-12, pp. 399 401, 1966.
[4] Iain E. G. Richardson, H.264 and MPEG-4 Video Compression, John Wiley and Sons LTD, England, 2003.
[5] Amer. I, Badawy. W, and Jullien. G, "Towards MPEG-4 Part 10 SoC : A VLSI Prototype for CAVLC," Signal Processing Systems, 2004.
[6] Yeong-Kang Lai, Chih-Chung Chou and Yu-Chieh Chung, "A Simple and Cost Effective Video Encoder with Memory-Reducing CAVLC," Circuits and Systems, ISCAS 2005.
[7] T. C. Chen, Y. W. Huang, C. Y. Tsai, B. Y. Hsieh and L. G. Chen, "Dual-Block-Pipelined VLSI Architecture of Entropy Coding for H.264/AVC Baseline Profile," VLSI Design. Automation and Test, 2005.
[8] Iain E. G. Richardson, Video Codec Design, John Wiley and Sons LTD, England, 2002.
[9] G. Bjontegaard and K. Lillevold, "Context-adaptive VLC coding of coefficients," JVT document JVT-C028, Fairfax, May 2002.
[10] Synchronous FIFO 5.0, Product Specification. Available: http://www.xillinx.com.