VHDL Implementation of an Efficient Context Adaptive Variable Length Coding Algorithm of H.264 Video Codec

Waleed Ahmed El-Ghobashy
Department of
Electronics and Electrical
Communications,
Ain Shams University, Egypt

Hossam O. Ahmed
Department of Electronics and
Electrical Communications,
El Shorouk Academy, Egypt

Osama EL-Mowafy
Department of
Electronics and Electrical
Communications
High institute of Engineering
and Technology, Egypt.

Abdelhalim Zekry
Department of
Electronics and Electrical
Communications,
Ain Shams University, Egypt

ABSTRACT
H.264/MPEG-4 AVC video compression standard uses Context Adaptive Variable-Length Coding (CAVLC) for encoding the transformed coefficients after quantization. The CAVLC is an important technique that used for reducing the bit stream realization of the coefficients. It is used for coding both Luminance and chrominance blocks. In this paper, an efficient VHDL implementation and verification of CAVLC coding is proposed. It increases the throughput and reduces the time of bit stream generation. The proposed VHDL architecture has been synthesized and simulated based on the cyclone II FPGA EP2C35F672C6 from Altera.

Keywords
H.264/AVC, entropy coding, CAVLC, Luminance coding , Chrominance coding, FPGA, cyclone II, video compression.

1. INTRODUCTION
In order to achieve the suitable compression ratio for data link communication, the communication devices uses many techniques to do that. One of the major techniques used is Context-Based Adaptive Variable Length Coding. CAVLC is adopted to be used as one of the H.264 standard techniques. The CAVLC is an entropy coding method which encodes the transformed quantized of residual data. It can achieve better coding efficiency, but the algorithm complexity is higher [1,2].

In figure 1, the CAVLC lies after transformation and quantization which in turn the CAVLC receives the 4x4 blocks that contain most of its coefficients zeros.

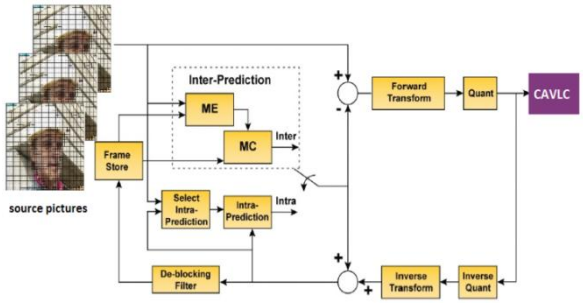

Fig.1 H.264 encoder block diagram

Most of previous works had introduced the implementation of CAVLC encoder.

Arun et al [3] introduced a mechanism that scans the coefficients in inversed zigzag order and proposed a special buffer structure to maintain the buffer size at the size of one block. Parallel symbols encoding is an efficient method in terms of performance, however, it obviously double the area cost of symbol encoders.

N. Keshaveni et al [4] proposed the implementation of CAVLC encoder by Veriloge, and integrated with other functional module such as Transformation and Quantization realized using Matlab this did not give an obviously area cost and power consumption of encoder.

Some other authors presented an efficient method to overcome the bottleneck at the scan phase by scanning coefficients in parallel. This method halves the required time of the scan phase. Parallel coding of level and run-before is also applied.

Finally, Chuan-Yung Tsai [5] has implemented a low power CAVLC encoder which can reduce up to 70% the power consumption but the total gate count is somewhat high.

In this paper, an efficient implementation CAVLC algorithm is proposed for H.264/AVC video codec. This method will reduce the processing time of the frame and increase the throughput by reducing the waste time consumed during the input quantized transformed residual data of the 4x4 block and its output coded data. It exploits the time between the input coefficients of the current block and the output codes by scanning the next 16 coefficients of the next block sequentially

This paper is organized as follows: the CAVLC entropy coding is introduced and the related works are surveyed in section 1. An overview on CAVLC technique is mentioned in section 2. Our proposed hardware architecture of CAVLC is described in Section 3. The implementation and simulation results are discussed in Section 4. The results and conclusions are mentioned in the next two sections.

2. OVERVIEW ON CAVLC

The Macroblocks are ordered Luma and Chroma for encoding. CAVLC algorithm is used to encode transformed and quantized residual luminance and chrominance blocks in a macroblock. All the transformed and quantized 4x4 and 2x2 blocks for a macro block are given as inputs to CAVLC algorithm. CAVLC algorithm processes each 4x4 block in zig-zag scan order and each 2x2 block in raster scan order. The main parameters for encoding block:

1) Nonzero coefficients and Trailing ones.
2) The pattern of Trailing ones.
3) The nonzero coefficient.
4) Number of zeros embedded in the non zero coefficients.
5) The location of those embedded zeros.

In the Fig.2, The quantized transformed 4X4 block introduced as an example of CAVLC input block.

1	1	0	0
-2	0	1	0
-4	1	1	0
1	0	0	0

Fig. 2 4x4 quantized block

The reordered data (zig-zag scanning) of the block are in the form (1, 1, -2, -4, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0)

The five main parameters are encoded in sequence; and finally, the transmitted bit stream for this block is: Table 1

(01101-000-1-00011-111-100-100-11-10-11-11-00)

Table 1. CAVLC Coding Processes Example

Syntax element	value	code
Coeff_token	Total coeff=8 Traillingones=3	01101
T1sign	+,+,+	000
Level(4) Level(3) Level(2) Level(1) Level(0)	1 -4 -2 1 1	1 00011 111 100 100
Total zeros	4	11
Runb(7) Runb(6) Runb(5) Runb(4)	1 0 0 3	10 11 11 00
Code (01101-000-1-00011-111-100-100-11-10-11-11-00)		

3. PROPOSED SCHEME

3.1 CAVLC calculation Counters

The CAVLC module contains a number of counters and registers units to collect the information exists in a block as a first step before encoding. Non-Zero Coefficients counter counts the number of non-zero coefficients (TotalCoeff). TrailingOnes counter stores the number of trailing ±1 values (TrailingOnes). TotalZeros counter counts the total number of zeros before the last non-zero coefficient (Total_Zeros). Level counter gets the number of non-zero coefficients other than the TrailingOnes. TrailingOnes register file is used to store the sign of each Trailing One coefficient. Level register file is used to store the level (sign and magnitude) of each non-zero coefficient other than the TrailingOnes. RunBefores register file is used to store the number of zeros preceding each non-zero coefficient. CAVLC hardware begins the encoding for a 4x4 or 2x2 block by reading the coefficients from the input buffer in reverse zig-zag order.

In each clock cycle (during rising edge), CAVLC reads one coefficient from the input buffer then analyzes the coefficient and updates the information stored in the related counter and register file. After all inputs entered, the counters and register files contain all the information for the current block that will be encoded. Reverse zig-zag scanning enables us to determine the necessary information for encoding a 4x4 block by reading and analyzing each coefficient only once.

3.2 Steps of generating CAVLC codes

The CAVLC has five steps to encode one block which are:[4]

3.2.1 Coding Coeff_Token step:

The CAVLC gets the number of non-zero coefficients (TotalCoeff) and the number of trailing ±1 values (TrailingOnes) in a block. It generates coeff_token variable length code depending on TotalCoeff, TrailingOnes and the VLC tables. Since the highest non-zero coefficients after the zig-zag scan are often sequences of ±1, CAVLC algorithm encodes the number of high-frequency ±1 coefficients (TrailingOnes) in coeff_token. Since the number of non-zero coefficients in neighboring blocks is correlated, CAVLC algorithm generates coeff_token for a block context adaptively. Table 2

Table 2. Coeff_token mapping to TotalCoeff and TrailingOnes

Trailingones (coeff_token)	Totalcoeff (coeff_token)	0<=nc<2	2<=nc< 4	4<=nc<8	8<nc
0	0	1	11	1111	0000 11
0	1	0001 01	0010 11	0011 11	0000 00
1	1	01	10	1110	0000 01
0	2	0000 0111	0001 11	0010 11	0001 00
1	2	0001 00	0011 1	0111 1	0001 01
2	2	001	011	1101	0001 10
-	-	-	-	-	-

It uses one of the four different VLC tables for generating the coeff_token for a block based on the number of nonzero coefficients in the neighboring blocks as follows. Fig.3 shows the calculation of nC parameter based on the number of non-zero coefficients in the left-hand and upper previously coded blocks, nA and nB respectively [6].

nC = round ((nA + nB) /2).	If both block is available.
nC = nB;	If upper block is available.
nC = nA;	If left block is available.
nC = 0	If neither is available.

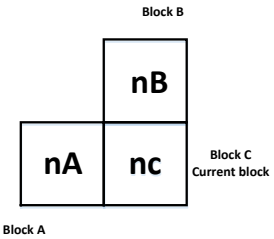

Fig.3 The relationship between block A, B and C

After calculation of nC parameter the VLC table is selected that will be used for generating the coeff_token based on the value of nC .

If nC = (0 or 1) the first column of table 2 will be used else if nC= (2 or 3) the second column will be used else if nC = (4 or 5 or 6 or 7) the third column will be used else if nC = 8 or more the forth column will be used.

As a special case, for 2x2 dc chroma blocks, nC is set to -1.

3.2.2 Coding TrailingOne Sign step:

It encodes the sign of each TrailingOne with a single bit in reverse order starting with the highest-frequency TrailingOne. If the sign is positive the sign code = 0, if the sign is negative the sign code =1.

3.2.3 Coding LEVEL coefficient step:

It encodes the level (magnitude and sign) of each remaining non-zero coefficient in the block in reverse order starting with the highest frequency coefficient and working back towards the DC coefficient. The code word for a level consists of a prefix and a suffix. It sets the suffix length for the first level to 0. It then increments the current suffix length, if the magnitude of the current level is larger than a predefined threshold for this suffix length. CAVLC algorithm generates the code length and the codeword for the current level based on its suffix length. When the suffix length for a level is 0, its codeword does not include a suffix. Otherwise, the codeword for the level includes a suffix. The next suffix length based on present non-zero coefficient. as shown in Table 3.

Table 3. Suffix Length Value

Non zero coefficient	Suffix length to be set
0	0
1,2,3	1
4,5,6	2
7,8,9,10,11,12	3
13-24	4
25-48	5
>48	6

The codeword for a level always includes a prefix, but the prefix for a level is generated using different equations in the two cases; when the suffix length for the level is 0 and when the suffix length for the level is greater than 0.

3.2.4 Coding Total_Zeros step:

During this step the CAVLC counts the total number of zeros between the non-zero coefficients (Total_Zeros). It extracts the output code and its length depending on the total number of zeros and the total number of coefficients using a VLC tables.

3.2.5 Coding Run_Before step:

It encodes the number of zeros preceding each non-zero coefficient in reverse order starting with the highest frequency coefficient. The blocks contain zeros at highest frequency after transformation and quantization and the number of zeros decrease when directed to the DC level. CAVLC algorithm uses run level coding to represent strings of zeros compactly see Table 4.

Table 4. run_before code with number of zerosleft

Run_before	Zerosleft						
	1	**2**	**3**	**4**	**5**	**6**	**>6**
0	1	1	11	11	11	11	111
1	0	01	10	10	10	000	110
2	-	00	01	01	011	001	101
3	-	-	00	001	010	011	100
4	-	-	-	000	001	010	011
5	-	-	-	-	000	101	010
-	-	-	-	-	-	-	-
14	-	-	-	-	-	-	00000000001

The proposed input /output architecture is shown in fig. 4. The video encoder stores the residual data of Macroblock in a two types of buffers one for four luminance blocks and the other for two chrominance blocks. The luminance block is arranged by reverse scanning ordering and is fed to the CAVLC module. The CAVLC counters calculate the five main parameters of the block. The CAVLC module uses these parameters to get out the codes. After coding the luminance blocks, the two chrominance blocks is encoded consecutively.

The CAVLC architecture is pipelining architecture in which when processing the first block it scanning the second block. Two-stage pipeline architecture requires double buffer size to store all the statistic information of one block before the data is encoded.

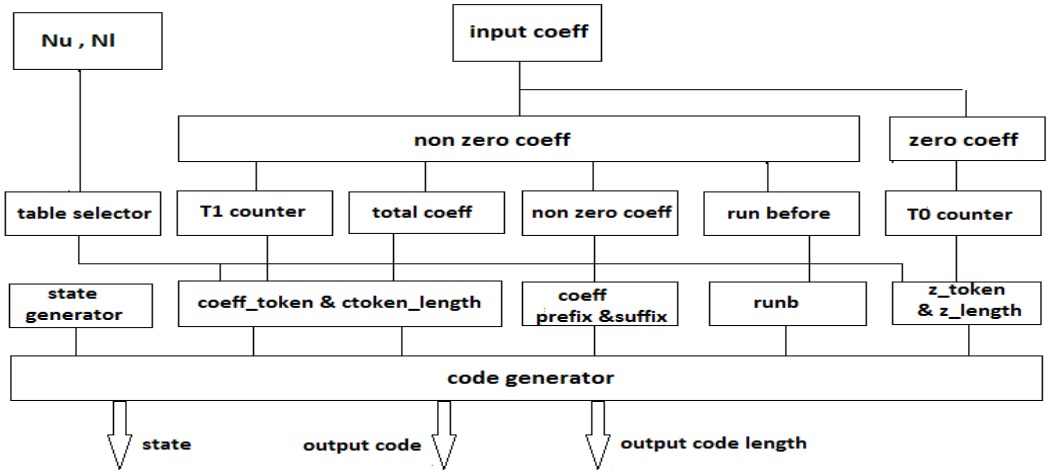

Fig.4 the CAVLC calculation stages

In the proposed method there are three stages and two clocks (CLOCK_2) and (CLOCK_1= 2 CLOCK_2) used for calculation. The first stage is the input stage which receive the input coefficients and the second stage is the processing stage and the third stage is the output stage which get out the output codes. The 4x4 luminance block, is entered by 16 CLOCK_2 cycles for calculating and initializing the internal counters. As shown in fig.5.

The processing stage uses about 21 CLOCK_2 cycles for calculating all the coefficients needed.

The output stage gets the output codes about 8 CLOCK_1 cycles (or 16 CLOCK_2). So the total number of cycles used for encoding 4x4 blocks is about 42 or 44 cycles. The 2x2 luminance block, is entered by 4 CLOCK_2 cycles for calculating and storing, so the total number of cycles used for encoding 2x2 blocks is about 20 or 21 CLOCK_2 cycles.

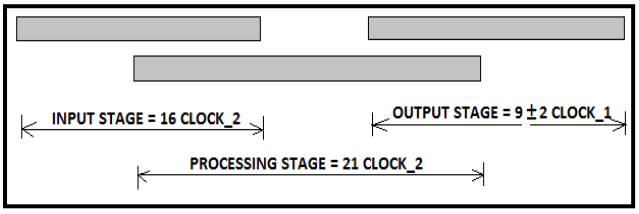

Fig.5 CAVLC processing stages

The proposed method can handling with the first 16 coefficients for the first block to get the code during scanning the second 16 coefficients for the second block. This method will lead to reduce the time of calculation and increase the throughput of coding CAVLC [7,8].

4. PROPOSED SCHEME SIMULATION

The CAVLC architecture is implemented by VHDL language and simulated using Modelsim 6.5f and Quartus 13 from Altera. Fig.6 shows the timing simulation of CAVLC architecture. The delay time between rising edge clock and its output is 8 ns. The proposed architecture can work with max frequency 100 MHz [9].

The proposed scheme tested by different types of videos. High texture sequences (such as Mobile and container), low motion sequences (such as mother-daughter and Hall) and high motion sequences (such as foreman and coastguard). The bitstream and compression ratio results of coding the first frame (352x288) of six types is organized in tables and graphs as shown in figures (6, 7).

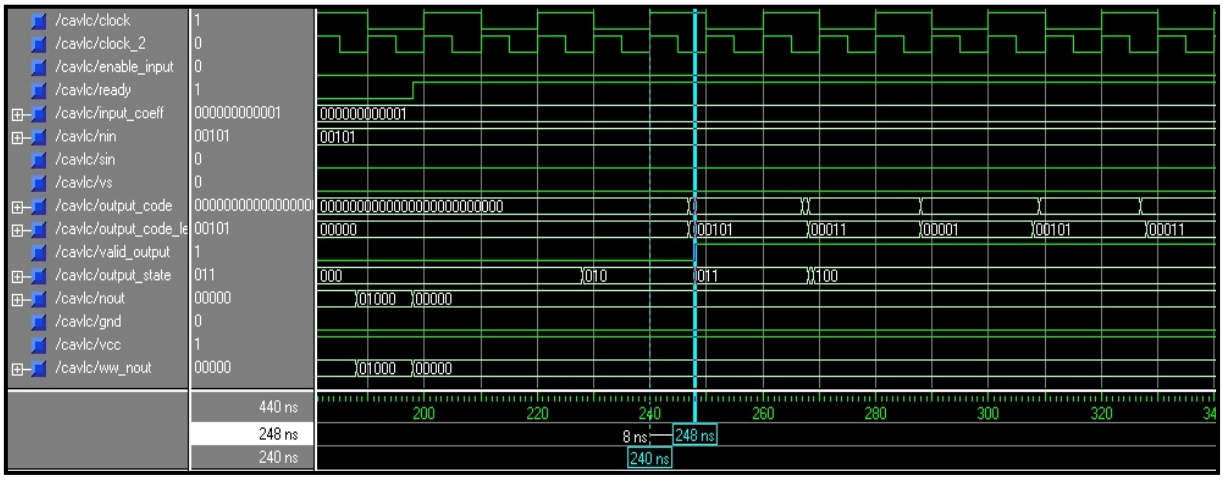

Fig.6 Timing simulation waveform

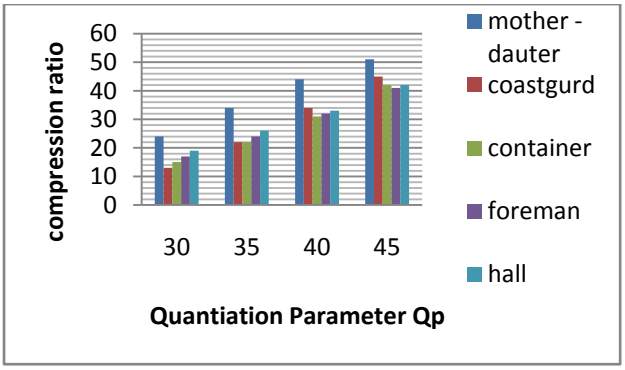

Fig 6. Compression ratio vs. quantization parameter

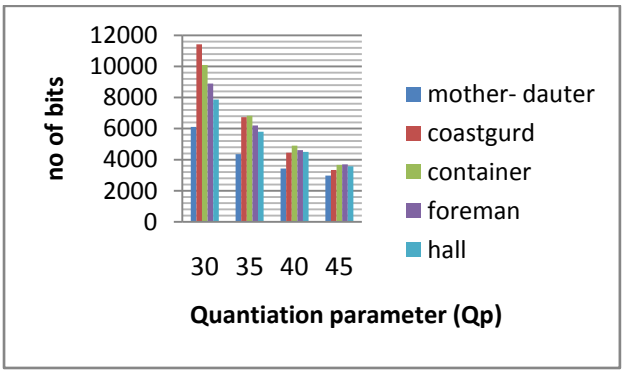

Fig 7. Bit stream vs. quantization parameter

5. RESULTS

The proposed entropy coding engine with pipelined architecture is synthesized and simulated process technology by using Quartus 13 and Modelsim 6.5f tools.

To achieve full hardware utilization by the dual-buffer architecture, two block statistic buffers are required. Two types of memories are required. The coefficient memory and bit stream memory are used as input and output buffers for system consideration. The 2x2 block for chrominance is used to process 2x2 block. The 4x4 block for luminance is used to process the 4x4 block. Table 5 listed the utilization parameters from Altera FPGA. Table 6 mention the power analyzer summary[10].

Table 5. Utilization Parameters Of Altera Device

Flow summary	Available	Our proposal	
		Used	Utilization
Total logic elements	33216	1961	6%
Total combinational functions	33216	1692	5%
Dedicated logic registers	33216	696	2%
Total pins	475	51	11%
Total memory bits	483840	384	<1%

Table 6. Power analyzer summary

Power parameters	The value
Thermal power dissipation	74.26 mW
Core static thermal power dissipation	47.36 mW
I/O thermal power dissipation	26.90 mW

6. CONCLUSIONS

In this paper, the CAVLC implementation algorithm and its timing simulation is proposed. This implementation focused on the high throughput of the CAVLC encoder. It is achieved with a low-cost memory requirement and hardware complexity. A complex CAVLC hardware encoder due to the data dependency in CAVLC.

Reverse zig-zag scanning from the lower right to the upper left of the block is used to reduce the encoding data. The CAVLC architecture had three stages reading, processing and output stages. This reduces the power consumption by reducing the switching activity on the input buffer address and data signals. The proposed architecture is implemented in VHDL based on the latest cyclone II FPGA EP2C35F672C6 from Altera. The VHDL RTL code is verified to work at 100 MHz.

7. REFERENCES

[1] ITU-T, H.264 'Advanced Video Coding for Generic Audiovisual Service', JAN 2012.

[2] Iain E. Richardson. 'The H.264 Advanced Video Compression Standard', 2nd edition. John Wiley & Son, 2010.

[3] Arun Kumar Pradhan, Lalit Kumar Kanoje, Biswa Ranjan Swain, 'FPGA based High Performance CAVLC Implementation for H.264 Video Coding', International Journal of Computer Applications (0975 – 8887) Volume 69– No.10, May 2013.

[4] N. Keshaveni, S. Ramachandran and K.S. Gurumurthy 'Implementation of Context Adaptive Variable Length Coder for H.264 Video Encoder' , International Journal of Recent Trends in engineering, Vol 2, No. 5, November 2009.

[5] Tung-Chien Chen, Yu-Wen Huang, Chuan-Yung Tsai, Bing-Yu Hsieh, and Liang-Gee Chen : 'Architecture Design of Context-Based Adaptive Variable-Length Coding for H.264/AVC' , IEEE Transactions on circuits and systems—ii: express briefs, vol. 53, no. 9, September 2006.

[6] Ngoc-Mai Nguyen, Xuan-Tu Tran, Pascal Vivet, Suzanne Lesecq, 'An Efficient Context Adaptive Variable Length Coding Architecture for H.264/AVC Video Encoders', The International Conference on Advanced Technologies for Communications (ATC 2012).

[7] 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', International Symposium on VLSI Design, Automation and Test, Conference, 2005

[8] P.Soundarya, Solomon Gotham, 'CAVLC Video coding Technique for MPEG-4', International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-1, Issue-6, November 2012.

[9] Asma Ben Hmida, Salah Dhahri, and Abdelkrim Zitouni, 'A High Performance Architecture Design of CAVLC Coding Suitable for Real-Time Applications' , Association of Computer Electronics and Electrical Engineers, 2013.

[10] Chuan-Yung Tsai, Tung-Chien Chen and Liang-Gee Chen 'Low Power Entropy Coding Hardware Design For H.264/AVC Baseline Profile Encoder'. In Proceedings of IEEE International Conference on Multimedia and Expo, 2006, pp. 1941-1944.