CAVLC ENCODING TECHNIQUE FOR H.264

Afreen Hamza L, Manasa M G, Payal Basak, Shikha Khandelwal, Vinod B Durdi

***Abstract*— The latest advancement in H.264 video compression technique is CAVLC method used for entropy coding. After the prediction, transformation and quantization a block of 4X4 is obtained containing mostly zeros. The zig zag scan of block is done to obtain the following for better compression in this lossless technique. 1)Non zero coefficients from neighboring blocks are correlated and encoded using look up table. 2) The main sequence obtained by zig zag scan is ±1 coefficients which represent the highest frequencies. 3) Zeros and the run level coding of zeroes after the scanning are encoded by referring the look up tables. 4) Levels in zig zag block are the representation of the highest to lowest frequencies.**

***Index Terms*— CAVLC,H.264 video compression, entropy coding**

I. INTRODUCTION

Context based adaptive technique is a method in which the non zero coefficients are compared with the previous coefficients before encoding. The result of prediction which contains the residual data of the two frames (reference frame and the current frame) contains less information to encode than the original raw data. The data is converted into a frequency representation during block transformation. Here the low frequency information is stored near the top left corner of macroblock, while the high frequency information is stored in bottom right corner of each block. The entropy coding involves the conversion of the data needed to recreate the video file into 0s and 1s.

CAVLC is chosen because of the following advantages. It is less complex than CABAC and therefore encodes faster. Hence, CAVLC is preferred when speed is required. CAVLC can be used in all profiles but CABAC can be used only in certain profiles. VLC allows sources to compress (close to its entropy) and decompress with zero error and still be read back symbol by symbol. But for data compression with large data blocks, this method fails.

II. PROCEDURE FOR ENCODING

A. Zig zag scan

Magnitude of low frequency coefficients tends to be greater than high frequency coefficient. The Zig zag scan is used to rearrange the coefficients of 4×4 block into a vector that(varies from low to high frequency) has coefficients in increasing order from reverse side. Hence, by using CAVLC encoder in reverse order i.e zig zag scanning, we can minimize the number of required bits[1]. For reading, a 16:1 mux is used. All 16 coefficients are given to mux as input. For this encoder a 4 bit counter is used as select line to select each input coefficient given to mux. Output of this mux is stored in memory.

B. Sign and total number of trailing ones

This module represents the number of ±1 that appear at the end of zig zag reordered vector.

Calculation: From the first non zero coefficient start calculating the number of 1s in the reverse zig zag vector(+1 or -1). If the first non zero coefficient is other than 1, trailing one count will be zero. If the encountered first non zero coefficient in the reverse zigzag vector is ±1, start counting the succeeding number of 1s(irrespective of its sign) and start the trailing ones counter. The maximum trailing one count can be 3. If succeeding non zero coefficient is other than 1, stop trailing ones counter. The overview of the hardware for this module is shown in Fig 1.

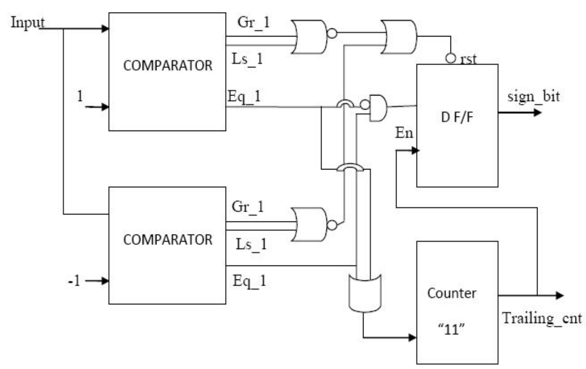

Fig 1. Trailing ones module

C. Nc calculation

Na= number of non-zero coefficient of previous 4×4 CAVLC coded block.

Nb=number of non-zero coefficient of 4×4 CAVLC coded block above the current block.

Nc calculation[4]:

Nc=Na (if only left block is available).

Nc=Nb (if only upper macroblock is available).

Nc=(Na+Nb)/2 (if both upper and left blocks are available).

Nc=0 for the first block of every frame.

D. Coefficient token

Coefficient token depends on total coefficient, trailing ones and Nc value as shown in Fig 2. Refer TABLE III[2] for the value of coefficient token.

Manuscript received May 15, 2015

Afreen Hamza L, Dept. of Telecommunication Engineering, Dayananda Sagar College of Engineering, Bangalore, India

Manasa M G, Dept. of Telecommunication Engineering, Dayananda Sagar College of Engineering, Bangalore, India

Payal Basak, Dept. of Telecommunication Engineering, Dayananda Sagar College of Engineering, Bangalore, India

Shikha Khandelwal, Dept. of Telecommunication Engineering, Dayananda Sagar College of Engineering, Bangalore, India

Vinod B Durdi, Associate Professor, Dept. of Telecommunication Engineering, Dayananda Sagar College of Engineering, Bangalore, India

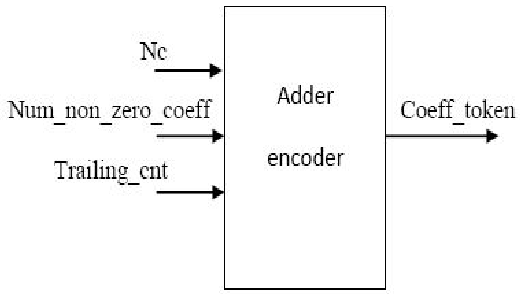

Fig 2. Coefficient token module

E. Total zeroes

Referring to Fig 3 find the total number of zero coefficient in 4×4 CAVLC coded block between first non zero coefficient to the last coefficient of block in reverse order coded zigzag vector. Compute total zero code referring the look up table TABLE III[2] which includes the total zeros and total coefficient.

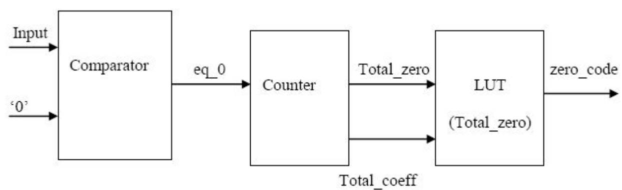

Fig 3. Total zeroes

F. Levels

Except the trailing ones, all non-zero coefficients are considered as levels. Number of levels= (total no. of non-zero coefficient – trailing ones) present in CAVLC 4×4 coded block. As number of levels depends on the number of non zero coefficients present in CAVLC 4×4 coded block. Level increases, as number of non zero coefficiesnts increases[6].
In Fig 4 the flow chart for calculating level code is shown by concatenating suffix length and prefix length.

Algorithm for calculating prefix suffix

For level calculation there will be <prefix><suffix>. Prefix is calculates as <zeros 1>, calculation of no. of zeros in prefix will be discussed in Algorithm[5]. Suffix has the sign bit as its LSB, the remaining bits of suffix is calculated from the nonzero coefficient which will be discussed in Algorithm. Number of bits of suffix is called as Suffixlength.

Algorithm for level:

For a given nonzero coefficient, 'a'
Step 1: If (numCoeff > 10) and (T1 < 3), Suffixlength = 1 or else Suffixlength = 0

Step 2: If (Suffixlength = 0) and (numCoeff ≤ 3 or T1 < 3), change |'a'| = |'a'| – 1 and sign is same. Or else keep 'a' same.

(i) If |'a'| < 8, there is no suffix. Prefix will be found for 'a'. Prefix is <zeros 1>. No. of zeros before 1 in prefix = 2 x (|'a'| – 1) + sign. (If a < 0, sign = 1, else sign = 0). Go to Step 12.

(ii) If |'a'| < 16, there is suffix of length, suffixlength = 4. The LSB of suffix = sign bit. The remaining bits (3 bits) is (|'a'| – 8). The prefix is <14 zeros 1>. Go to step 12.

(iii) If |'a'| > 15, there is <Prefix><Suffix>. Diff = |'a'| – 16. Go to Step 9.

Step 3: Else (Suffixlength = 1), change |'a'| = |'a'| – 1 and sign is not changed. There will be <prefix><suffix> = <zeros 1><suffix>.

Step 4: If (|a| – 1) > 15 x 2suffixlength-1, Diff = (|a| – 1) – (15 x 2suffixlength-1), Then go to Step 9.

Step 5: If 'a' is positive, the LSB of suffix = 0, or else LSB of suffix = 1

Step 6: If suffixlength > 1, the remaining bits of suffix = the (Suffixlength – 1) LSBs of (|a| – 1)

Step 7: No. of zeros in prefix = value of remaining MSBs of (|a| – 1)

Step 8: The code for <prefix><suffix> is ready. Go to Step 12.

Step 9: Suffix length = 12 + (Diff >> 11) bits Step 10: Prefix = < (15 + 2*(Diff >> 11)) zeros 1 >

Step 11: LSB of Suffix = sign bit of 'a'. Remaining bits of Suffix = Binary form of Diff (Right Aligned).
Step 12: Based on present nonzero coefficient 'a', set the next Suffixlength (Ref: TABLE IV)
Note 1: If first nonzero coefficient (other than trailing ones), and if the present |'a'| > 3, then new suffixlength = 2.
Note 2: Else if the new Suffixlength > previous Suffixlength, Suffixlength will be incremented.
Note 3: Else keep the same previous Suffixlength as new Suffixlength. (Previous Suffixlength means the Suffixlength calculated previously in the same step, not in any other steps).

Step 13: If any nonzero coefficient is available next, read 'a' and then go to Step 4.

Step 14: Stop.

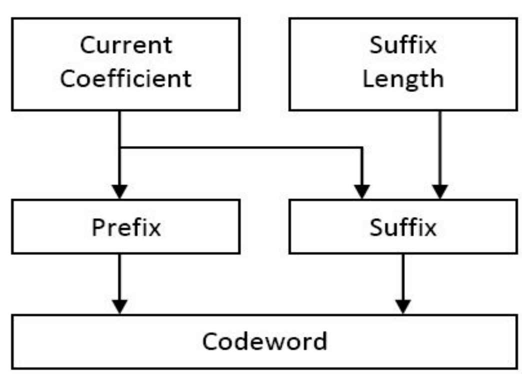

Fig 4. Prefix-suffix module

G. Run before

Run before provides the information about the location of zeroes in CAVLC 4X4 coded block.
The calculation is as follows:
Starting from the first non-zero coefficient from the last (the very first value of the 4X4 matrix) after reordering, the zeroes

are calculated to get the total zeroes. Taking every non-zero value the number of zeroes preceding the value is calculated and refered as run-before. The value of zeroes left is equated to the total zeroes at the initial stage. Later stages we subtract the run before value from total zero to obtain zeroes left shown in Fig 5. With help of run-before and zeroes left the code is obtained from the run before TABLE V as given below.

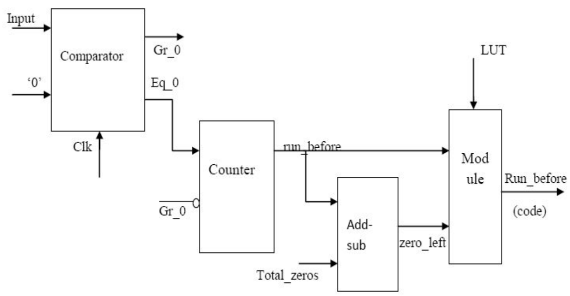

Fig 5. Run before module

III. RAM AND ASSOCIATED TABLES

In CAVLC encoder, RAM stores the look up tables for all modules and the number of coefficients of all encoded macroblock needed to calculate Nc. For Nc calculation we take the stored values i.e. number of coefficients of upper macroblock (Na) and left macroblock (Nb) of current encoding macroblock. RAM access the data from memory according to the read or write operations needed for the Nc calculation.

TABLE I: COEFFICIENT TOKEN

T1	numCoeff	0 ≤ nC < 2	2 ≤ nC < 4	4 ≤ nC < 8	8 ≤ nC
0	0	1	11	1111	000011
0	1	000101	001011	001111	000000
0	2	00000111	000111	001011	000100
0	3	000000111	0000111	001000	001000
0	4	0000000111	00000111	0001111	001100
0	5	00000000111	00000100	0001011	010000
0	6	0000000001111	000000111	0001001	010100
0	7	0000000001011	00000001111	0001000	011000
0	8	0000000001000	00000001011	00001111	011100
0	9	00000000001111	000000001111	00001011	100000
0	10	00000000001011	000000001011	000001111	100100
0	11	000000000001111	000000001000	000001011	101000
0	12	000000000001011	0000000001111	000001000	101100
0	13	0000000000001111	0000000001011	0000001101	110000
0	14	0000000000001011	0000000000111	0000001001	110100
0	15	0000000000000111	00000000001001	0000000101	111000
0	16	0000000000000100	00000000000111	0000000001	111100
1	1	01	10	1110	000001
1	2	000100	00111	01111	000101
1	3	00000110	001010	01100	001001
1	4	000000110	000110	01010	001101
1	5	0000000110	0000110	01000	010001
1	6	00000000110	00000110	001110	010101
1	7	0000000001110	000000110	001010	011001
1	8	0000000001010	00000001110	0001110	011101
1	9	00000000001110	00000001010	00001110	100001
1	10	00000000001010	000000001110	00001010	100101
1	11	000000000001110	000000001010	000001110	101001
1	12	000000000001010	0000000001110	000001010	101101
1	13	000000000000001	0000000001010	000000111	110001
1	14	0000000000001110	00000000001011	0000001100	110101
1	15	0000000000001010	00000000001000	0000001000	111001
1	16	0000000000000110	00000000000110	0000000100	111101
2	2	001	011	1101	000110
2	3	0000101	001001	01110	001010
2	4	00000101	000101	01011	001110
2	5	000000101	0000101	01001	010010
2	6	0000000101	00000101	001101	010110
2	7	00000000101	000000101	001001	011010
2	8	0000000001101	00000001101	0001101	011110
2	9	0000000001001	00000001001	0001010	100010
2	10	00000000001101	000000001101	00001101	100110
2	11	00000000001001	000000001001	00001001	101010

TABLE II: COEFFICIENT TOKEN

T1	numCoeff	0 ≤ nC < 2	2 ≤ nC < 4	4 ≤ nC < 8	8 ≤ nC
2	12	000000000001101	0000000001101	000001101	101110
2	13	000000000001001	0000000001001	000001001	110010
2	14	0000000000001101	0000000000110	0000001011	110110
2	15	0000000000001001	00000000001010	0000000111	111010
2	16	0000000000000101	00000000000101	0000000011	111110
3	3	00011	0101	1100	001011
3	4	000011	0100	1011	001111
3	5	0000100	00110	1010	010011
3	6	00000100	001000	1001	010111
3	7	000000100	000100	1000	011011
3	8	0000000100	0000100	01101	011111
3	9	00000000100	000000100	001100	100011
3	10	0000000001100	00000001100	0001100	100111
3	11	00000000001100	00000001000	00001100	101011
3	12	00000000001000	000000001100	00001000	101111
3	13	000000000001100	0000000001100	000001100	110011
3	14	000000000001000	0000000001000	0000001010	110111
3	15	0000000000001100	0000000000001	0000000110	111011
3	16	0000000000001000	00000000000100	0000000010	111111

TABLE III: TOTAL ZEROES

No. of zeros (total_zeros)	Number of non zero coefficients (numCoeff)														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	111	0101	00011	0101	000001	000001	000001	000001	00001	0000	0000	000	00	0
1	011	110	111	111	0100	00001	00001	0001	000000	00000	0001	0001	001	01	1
2	010	101	110	0101	0011	111	101	00001	0001	001	001	01	1	1	
3	0011	100	101	0100	111	110	100	011	11	11	010	1	01		
4	0010	011	0100	110	110	101	011	11	10	10	1	001			
5	00011	0101	0011	101	101	100	11	10	001	01	011				
6	00010	0100	100	100	100	011	010	010	01	0001					
7	000011	0011	011	0011	011	010	0001	001	00001						
8	000010	0010	0010	011	0010	0001	001	000000							
9	0000011	00011	00011	0010	00001	001	000000								
10	0000010	00010	00010	00010	0001	000000									
11	00000011	000011	000001	00001	00000										
12	00000010	000010	00001	00000											
13	000000011	000001	000000												
14	000000010	000000													
15	000000001														

TABLE IV: SUFFIX LENGTH

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

TABLE V: RUN BEFORE

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
6	-	-	-	-	-	100	001
7	-	-	-	-	-	-	0001
8	-	-	-	-	-	-	00001
9	-	-	-	-	-	-	000001
10	-	-	-	-	-	-	0000001
11	-	-	-	-	-	-	00000001
12	-	-	-	-	-	-	000000001
13	-	-	-	-	-	-	0000000001
14	-	-	-	-	-	-	00000000001

IV. SIMULATION RESULTS

A. Coefficient token module

The input variables of clock, reset, enable are given with the nA and nB calculated value to give out Nc, shown in Fig 6. The calculated values of total non zero, trailing one are included in finding out the code for coefficient token.

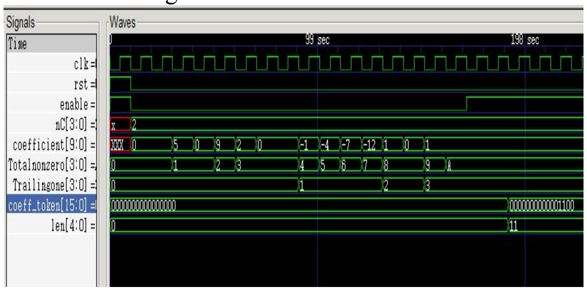

Fig 6. Simulation of coefficient token module

B. Sign of trailing ones module

Fig 7 shows clock, reset, enable and total coefficient are given as input to get the out as the signs of the trailing ones.

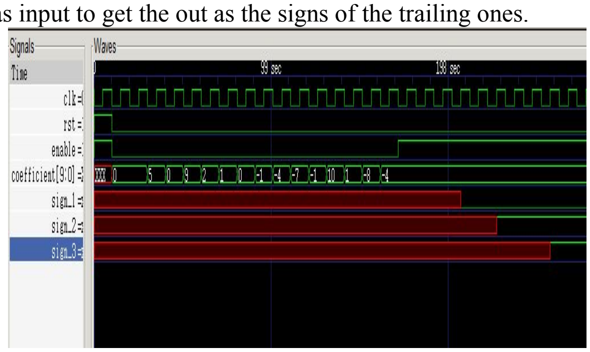

Fig 7. Simulation for sign of trailing ones module

C. Total zeroes module

Total coefficient, clock, enable, reset are used to find the code for the total zeroes in the code, shown in Fig 8.

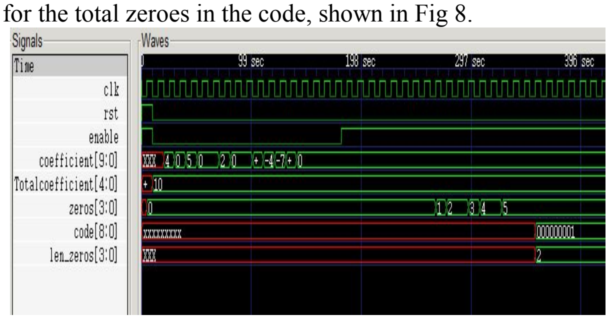

Fig 8. Simulation for total zeroes module

D. Run before module

The Fig 9 shows run before code with total zeroes left and run before value (indicates the number of zeroes before each non-zero coefficients).

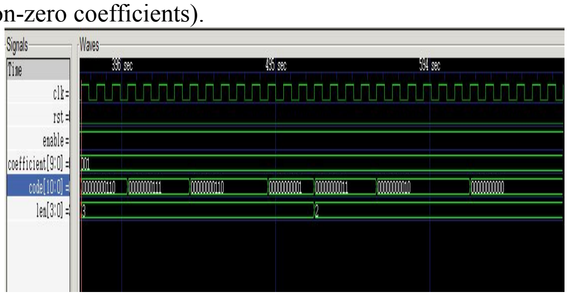

Fig 9. Simulation for run before module

CONCLUSION

CAVLC in H.264 provides the advantage of good and efficient performance. It is a high speed algorithm providing fast compression. The RAM included in the hardware design stores the nA and nB values. The paper showcases the advantage of using CAVLC encoder with zig-zag.

REFERENCES

[1] Iain E. Richardson, The H.264 Advanced Video Compression Standard, 2nd edition, Vcodex limited, UK.

[2] ITU-T TELECOMMUNICATION STANDARDIZATION SECTOR OF ITU SERIES H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS Infrastructure of audiovisual services – Coding of moving video

[3] Iain. E.Richardson, Video Codec design, Developing Image and Video Compression Systems

[4] 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, september2006

[5] High Performance Context Adaptive Variable Length Coding Encoder for MPEG-4 AVC/H.264 Video Coding

[6] Asma Ben Hmida, Salah Dhahri, and Abdelkrim Zitouni, "A High Performance Architecture Design of CAVLC Coding Suitable for Real-Time Applications," ACEEE, Proc. of World Cong. on Multimedia and Computer Science, pp. 69-74, 2013