International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-1 Issue-5, October 2012

CAVLC Video coding Technique for MPEG-4

P.Soundarya, Solomon Gotham

***Abstract:** This paper presents architecture for context adaptive variable length coding (CAVLC) used in the MPEG-4 AVC/H.264 video coding. The proposed design implements the fine-grained zero skipping at the 4x4 block level and the individual coefficient level. This design saves more than half of cycle count and 41% of area cost when compared with the other designs. The total gate count of the design is 10.2K gates. The design is implemented on Xilinx Spartan 3E Xc3s500 fpga and the total power consumption is estimated to be 0.081W.*

***Keywords:** CAVLC, MPEG-4, AVC/A.264.*

I. INTRODUCTION

Context-based Adaptive Variable Length Code (CAVLC) has been adopted in MPEG-4 AVC/H.264 video coding standard as one of the entropy coding methods The abundant zero data in H.264 video coding can be divided into two levels. First, in the 8x8 block level, if all four 4x4 blocks are zero blocks, the CAVLC process will be skipped according to the coding flow. However, if any of the 4x4 blocks is nonzero, the process cannot be skipped. However, even in such nonzero 8x8 block, we still find plenty of zero 4x4 blocks. Second, in a nonzero 4x4 block, only small portion of data, 13% for Qp = 28 case, is nonzero During the development of H.264 standard, none of previous designs have explored this feature to speedup the encoding cycle and they use one-by-one coefficient coding method ignoring of the data value. This will result in the waste of redundant coding cycles and power consumption. To avoid such problem while still explore the fine-grained zero skipping, this proposed CAVLC design adopts a dedicated zero-block codeword table for fast all-zero block encoding and the nonzero index table for direct nonzero coefficient encoding.

II. OVERVIEW OF CAVLC PROCESS

The CAVLC coding steps are shown as below:

Scan 4x4 transformed coefficients in reverse zigzag scan order. Encode the number of total nonzero coefficients, *Total Coefficients*, and the trailing ones (up to 3), *Trailing Ones* (T1s). Encode the sign of each trailing one. Encode the level of each remaining nonzero coefficient. Encode the number of total zeros, *Total Zeros*, after the first nonzero coefficient. Encode each run of zeros, *Run Before*, between the nonzero coefficients, which will also depend on the number of zeros that have not yet been coded (*Zeros Left*) During the encoding process, if all coefficients for an 8x8 block are zero, the CAVLC encoding will be skipped and only a special flag, coded block pattern (CBP), is used and set to zero.

Revised Manuscript Received on 30 October 2012
*****Correspondence Author(s)
P. Soundarya*, M.Tech Department of ECE, JNTU Kakinada University, Kaushik College of Engineering/Visakhapatnam, India.
Solomon Gotham, Professor & Head, Department of ECE, Kaushik College of Engineering, India.

II. ANALYSIS OF ZERO COEFFICIENTS

A. Simulation Settings

To justify our fine-grained zero skipping approach, we test two types of statistics. One is the distribution of nonzero coefficients per 4x4 block and the other is the distribution of detected zero 4x4 blocks beyond detected zero 8x8 blocks. The results show large percentage of zero 4x4 block in a onzero 8x8 blocks and zero coefficients in a nonzero 4x4 block.

B. Statistics of nonzero coefficients per 4x4 block

For the Qp=28 case, the average number of nonzero coefficients is less than 2. That implies that over 87% of coefficients in a 4x4 block is zero, which shows a large possibility to speed up by the zero coefficient skipping method.

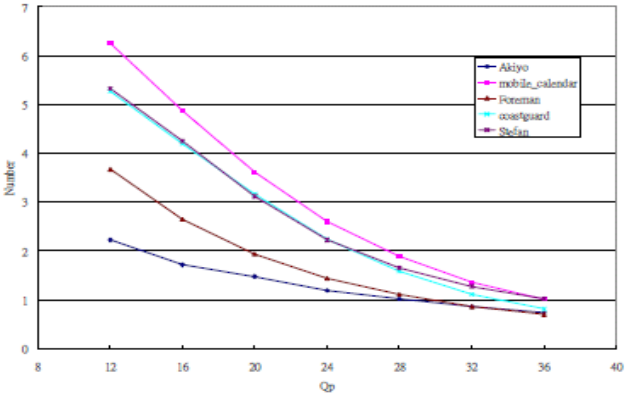


Figure 1 Average number of *Total Coefficients* per4x4 block

C. Statistics of detected zero 4x4 blocks beyond the detectedzero 8x8 block

Fig. 2. shows the percentage of extra detected zero 4x4 blocks beyond the detected zero 8x8 block, which is defined as [the number of zero 4x4 block/ (the number of zero 8x8block × 4) - 1].

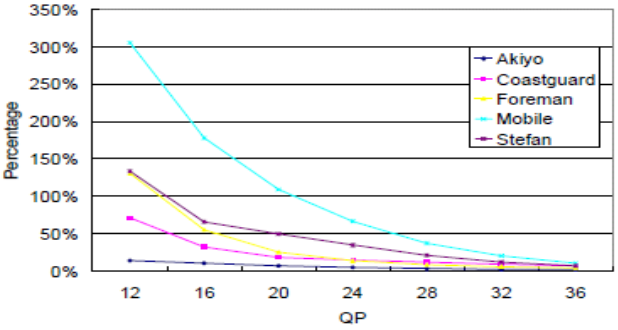


Fig. 2. Percentage of extra detected zero 4x4 blocks beyond the detected zero 8x8 blocks.

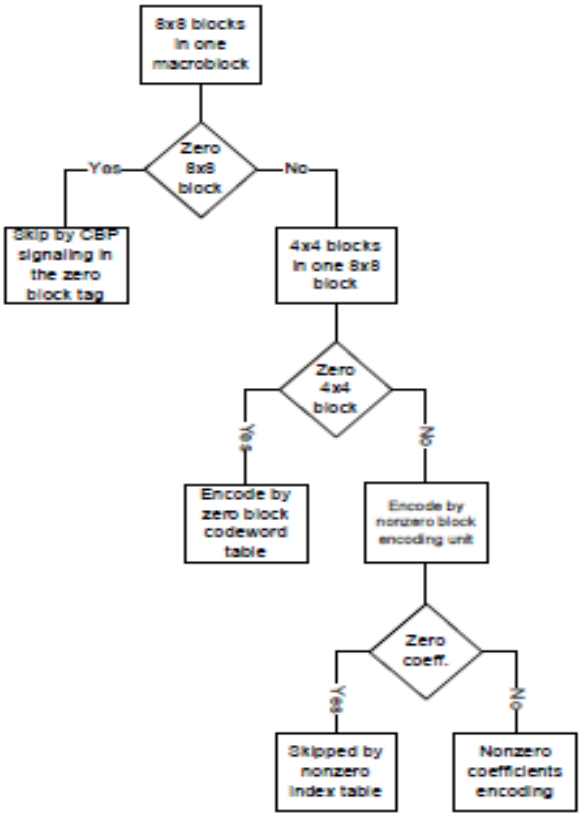
Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

Fig. 2. Shows that at least 10% of extra zero 4x4 blocks cannot be detected by zero 8x8 blocks even at high Qp. Thus, just using the CBP signaling as in [4] for zero 8x8 block skipping could not explore the possibility to skip these extra 4x4 zero blocks.

IV. EXISTING DESIGN OF CAVLC

A. The zero block encoding flow

And Architecture of CAVLC encoder The encoding flow of the proposed CAVLC encoder illustrated in Fig. 3. Figure 3 Encoding flow of the proposed CAVLC encoder.



The encoding path is divided into two, one for fast zero block encoding and one for nonzero block encoding. Then after the four input clock cycles for a 4x4 block, if a zero 4x4 block is detected, the nonzero block encoding path will be turned off. The corresponding codeword generation will be generated by the zero block codeword table directly. The zero block codeword table is a dedicated codeword table for zero 4x4 blocks in a nonzero 8x8 block that is extracted from the CAVLC codeword table. With this approach, no extra cycles are required for encoding and this also provides a fast and low power path for zero 4x4 blocks. Figure 4 shows the architecture of CAVLC encoder

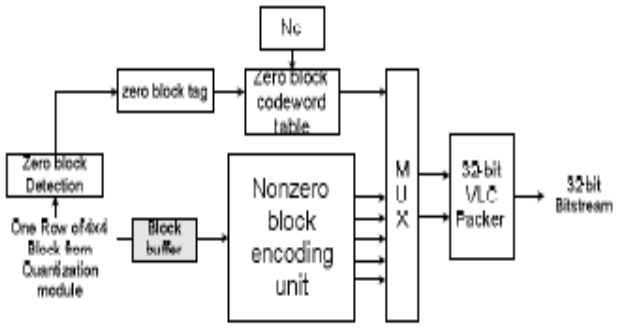
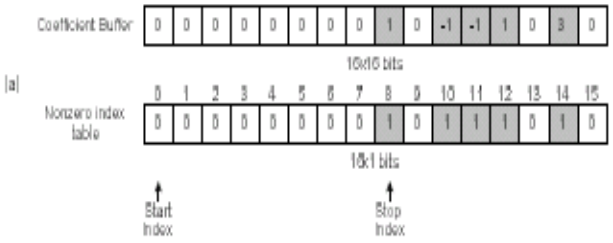


Figure 4 Architecture of CAVLC encoder

Fig.5. shows a simple example to encode the nonzero coefficients and skip the zero coefficients. Fig. 5 (a) shows the content of coefficient buffer and the nonzero index table after all coefficients are loaded. When the encoding is started, the Find Leading One logic first finds the leading one for first nonzero index coefficient from left to right by skipping the run of zeros, and generates one set of start and stop index. The irst run of zeros is skipped and not encoded according to the standard. Then, the value of the corresponding nonzero coefficients is checked if it equals to one to decide the T1 and its sign.



(a)

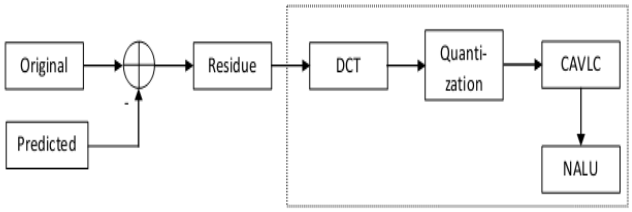
In Fig. 6. (b), the second nonzero coefficient will be encoded according to its value. At the same time, the run of zeros specified by the difference of start and stop indexes is also calculated and encoded by the run table. This process is iterated until the stop index is 15 or there is no leading one remained. Thus, in the whole encoding process, we directly encode the levels of nonzero coefficients and the run of zeros at the same time. The cycles to encode zero coefficients are efficiently skipped.

(b)

Fig. 6. An example of zero coefficient skipping method: (a) the initial status after all coefficients are loaded and (b) the status after first iteration of leading one detection.

III. PROPOSED DESIGN OF CAVLC

H.264, the latest video compression standard, uses CAVLC for encoding the coefficients after quantization. CAVLC encodes the coefficients (the coefficients may be positive or negative) into binary bit stream. The design is implemented on Xilinx Spartan 3E,XcS500E fpga device.



In H.264 Encoder, the predicted submacroblock (of size 4x4) is subtracted from the original submacroblock (of size 4x4), resulting the residue submacroblock (of size 4x4).

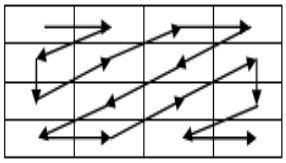
Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

This residue is transformed by modified Discrete Cosine Transform. The transformed residue is also of size 4x4. Then this transformed residue is quantized (dividing by some defined number, QStep). This quantized-transformed residue is encoded as follows.

Encoding process:

There will be 16 coefficients in a 4x4 submacroblock. The total number of coefficients is tot_coef = 16.

(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)
(3,0)	(3,1)	(3,2)	(3,3)



The 16 coefficients are taken in a zig-zag manner as shown in the above figure, and written in an array.

(0,0) – (0,1) – (1,0) – (2,0) – (1,1) – (0,2) – (0,3) – (1,2) – (2,1) – (3,0) – (3,1) – (2,2) –(1,3) – (2,3) – (3,2) – (3,3)

The important parameters to be encoded.

1. Number of nonzero coefficients (numCoef) and Trailing Ones (T1)
2. The pattern of Trailing Ones (T1)
3. The nonzero coefficients (Levels)
4. Number of zeros embedded in the nonzero coefficients (Total_zeros)
5. The location of those embedded zeros (run_before)

1. Number of nonzero coefficients (numCoef) and Trailing Ones (T1)

Calculate number of nonzero coefficients (numCoeff). For this array (or sub macroblock), allot Nc Present = numCoeff. [0 ≤ numCoeff ≤ tot_coef]

Remove the zero coefficients from the last, one – by – one, till the nonzero coefficient is reached. This reduced array may have zeros in between the first and last nonzero coefficients.

If the last nonzero coefficient is not equal to '±1', then Trailing ones (T1) = 0. If the last nonzero coefficients is equal to '±1', then calculate the number of continuous ones (+1 or -1) from last nonzero coefficient '±1' towards first nonzero coefficient. T1 includes the last nonzero coefficient also, if the last nonzero coefficient is '±1'.

Restrict the value of T1 to 3. Maximum value of T1 = 3 i.e., consider only three '±1's only from the last nonzero coefficient inclusive. If there is any other '±1' still, consider those '±1' as normal numbers, not as T1s. So, now T1 is ready. i.e., [0 ≤ T1 ≤ 3].

For Luma, Chroma DC (4:4:4) and Chroma AC (T1, numCoeff) will be coded based on nCContext. nC Context = (nCA + nCB + 1) >> 1. nCA is the number of nonzero coefficients available in the left 4x4 submacroblock and nCB is the number of nonzero coefficients available in the up 4x4 submacroblock. If any of nCA or nCB is not available, then nC Context = (nCA + nCB). Note that nCPresent has no role here. Based on nCContext, the corresponding column will be selected in the Table 1. For (T1, numCoeff), the code will be selected. If nCContext < 8, the codes will be variable length codes. If nCContext > 7, the codes will be fixed length codes (6-bits). The first four bits of fixed length code is (numCoeff – 1) and the last two bits of fixed length code is T1. But for T1 = 0 and numCoeff = 0, the code is different.

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	000001 11	00001 11	001000	001000
0	4	0000000111	00000111	0001111	001100
0	5	00000001 11	00000100	0001011	010000
0	6	000000001111	00000011 1	0001001	010100
0	7	000000001011	000000011 11	0001000	011000
0	8	000000001000	000000010 11	0000111	011100
0	9	0000000001111	000000001111	0000101 1	100000
0	10	0000000001011	000000001011	000001111	100100
0	11	0000000000111 1	000000001000	000001011	101000
0	12	0000000000101 1	000000000111 1	000001000	101100
0	13	0000000000001111	000000000101 1	0000001101	110000
0	14	0000000000001011	000000000111	0000001001	110100
0	15	0000000000000111	000000000001001	0000000101	111000
0	16	0000000000000100	000000000001111	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	000000001110	000000110	001010	011001
1	8	0000000001010	000000001110	0001110	011101
1	9	00000000001110	0000000001010	0000111 0	100001
1	10	0000000000010 10	000000000111 0	0000101 0	100101
1	11	0000000000001110	000000000010 10	0000011 10	101001
1	12	0000000000001010	000000000011 10	0000010 10	101101
1	13	0000000000000001	000000000001010	0000001 11	110001
1	14	00000000000001 110	000000000001011	0000001 100	110101
1	15	0000000000000 1010	000000000000 1000	0000001000	111001
1	16	00000000000001 10	000000000001 10	0000000 100	111101
2	2	001	011	1101	000110
2	3	0000101	001001	01110	001010
2	4	0000010 1	000101	01011	001110
2	5	0000001 01	0000101	01001	010010
2	6	0000000 101	00000101	001101	010110
2	7	000000010 1	000000101	001001	011010
2	8	0000000001101	0000000 1101	0001101	011110
2	9	000000000 1001	000000001001	0001010	100010
2	10	0000000000 1101	000000001101	0000 1101	100110
2	11	0000000000 1001	000000001001	0000 1001	101010

T1	numCoeff	0 ≤nC < 2	2 ≤nC < 4	4 ≤nC < 8	8 ≤nC
2	12	00000000000011 01	00000000001 101	00000 1101	101110
2	13	00000000000010 01	00000000001 001	00000 1001	110010
2	14	00000000000001 101	00000000000 110	00000 01011	110110
2	15	00000000000001 001	00000000000 1010	00000 00111	111010
2	16	00000000000000 101	00000000000 0101	00000 00011	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	000000000 1100	000000001100	0001100	100111
3	11	000000000011100	000000001000	0000 1100	101011
3	12	000000000001000	000000000 1100	0000 1000	101111
3	13	000000000000 1100	000000000 1100	0000 1100	110011
3	14	000000000000 1000	000000000 1000	0000 01010	110111
3	15	000000000000 1100	000000000000 1	0000 00110	111011
3	16	000000000000 1000	000000000000 100	0000 00010	111111

Table 1 Code for (T1, numCoeff) for Luma coefficients and Chroma AC Coefficients

T1	num Coeff	nC = -1	nC = -2	T1	num Coeff	nC = -1	nC = -2
0	0	01	1	1	7	-	000000000110
0	1	000111	0001111	1	8	-	000000000101
0	2	000100	0001110	2	2	001	001
0	3	000011	0000011 1	2	3	0000010	0001011
0	4	000010	0000001 10	2	4	0000010	0001010
0	5	-	0000000 111	2	5	-	000000100
0	6	-	0000000011	2	6	-	000000101
0	7	-	0000000011 1	2	7	-	0000000101
0	8	-	0000000001 11	2	8	-	0000000001 00
1	1	1	01	3	3	000101	00001
1	2	000110	0001101	3	4	0000000	00001
1	3	0000011	0001100	3	5	-	0001001
1	4	000000 11	0000001 01	3	6	-	0001000
1	5	-	0000000 110	3	7	-	0000000100
1	6	-	0000000 0110	3	8	-	0000000100

Table 2 Code for Chroma DC Levels

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

2. The pattern of Trailing Ones (T1)

If T1 > 0, those T1 nonzero coefficients should be coded from last T1 nonzero coefficient towards the first T1 nonzero coefficient (like reading Urdu / Arabic !). If the T1 is '+1', it is coded with '0'. Else it is coded with '1'. Writing the code should be from left to right (? □). Maximum 3 bits will be written.

3. The nonzero coefficients (Levels)

If numCoeff > T1, then the remaining nonzero coefficients are called levels and those levels are coded by Prefix – Suffix method which is explained next. Else there is no coding of levels.

Prefix - Suffix Method :

For any level, there will be <prefix><suffix>

<prefix> will have <zeros 1>, calculation of no. of zeros in prefix will be discussed in Algorithm given next.

<suffix> will have sign bit as LSB, the remaining bits will be derived from the nonzero coefficient which will be discussed in Algorithm. No. 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 13.

(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 13.

(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 same. There will be <prefix><suffix> = <zeros 1><suffix>.

Step 4: If (|a| – 1) > 15 x 2 suffixlength-1 , Diff = (|a| – 1) – (15 x 2 suffixlength-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 present nonzero coefficient 'a' is <prefix><suffix> ready. Go to

Step 13.

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'

Step 12: Remaining bits of Suffix = Binary form of Diff (Right Aligned).

Step 13: Based on present nonzero coefficient 'a', set the next Suffixlength (Ref: Table 3)

Note 1: If the nonzero coefficient is first nonzero coefficient (other than trailing ones), and if the present |'a'| > 3, then new suffixlength = 2.

Note 2: Else if the new Suffixlength = 2 and previous Suffixlength = 0, then

Suffixlength = new Suffixlength (i.e., 2)

Note 3: Else if the new Suffixlength > previous Suffixlength, Suffixlength will be incremented (it will not be changed drastically).

Note 4: Else keep the same previous Suffixlength as new Suffixlength.

(Note that the previous Suffixlength means the Suffixlength calculated previously in this step only, not in any other steps.)

Step 14: If any nonzero coefficient is available next (reverse reading!), read 'a' and then go to Step 4. Step 15: Stop

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 3 Suffix length

4. Number of zeros embedded in the nonzero coefficients (Total_zeros)

If numCoeff < tot_coef, by reading the array between the last nonzero coefficient and the first coefficient, calculate number of zeros. It is represented as total_zeros. For Luma, Chroma DC (4:4:4) and Chroma AC, the (total_zeros, numCoeff) in Table 5, will give the code for total_zeros. (0 □≤ total_zeros ≤ 15). For Chroma DC (4:2:0), the code for total_zeros is found in Table 4 and for Chroma DC (4:2:2), the code is found in Table 6.

No. of zeros (total_zeros)	Number of non zero coefficients (numCoeff)		
	1	2	3
0	1	1	1
1	01	01	0
2	001	00	
3	000		

Table 4 Total zeros table for 4x4 blocks for Chroma DC 2x2 block (4:2:0 subsampling)

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	100	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	0001							
8	000010	0010	0010	011	0010	0001	001	000000								
9	000011	00011	00011	0010	00001	001	000000									
10	000010	00010	00010	00010	0001	000000										
11	000001 1	000011	000001	00001	00000											
12	000001 0	000010	00001	00000												
13	000000 11	000001	000000													
14	000000 10	000000														
15	000000 01															

Table 5 Total zeros table for 4x4 blocks

No. of zeros (total_zeros)	Number of non zero coefficients numCoeff)						
	1	2	3	4	5	6	7
0	1	000	000	110	00	00	0
1	010	01	001	00	01	01	1
2	011	001	01	01	10	1	
3	0010	100	10	10	11		
4	0011	101	110	111			
5	0001	110	111				
6	0000	1	111				

Table 6 Total zeros table for 4x4 blocks for Chroma DC 2x4 block (4:2:2 subsampling)

5. The location of those embedded zeros (run_before) If total_zeros > 0, then runbefore is calculated. 'runbefore' will find a code for the location of zeros between last nonzero coefficient and first coefficient.

Algorithm for run before:

Step 1: Starting from last nonzero coefficient

Step 2: zerosLeft = total_zeros.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

Step 3: Find the number of zeros present before that nonzero coefficient (run_before).
Step 4: Referring Table 7, find the code for (run_before, zerosLeft).
Step 5: zerosLeft = zerosLeft – run_before
Step 6: If zerosLeft = 0 or number of coefficient remaining = zerosLeft, go to Step 11.
Step 7: If the present nonzero coefficient = first nonzero coefficient, go to Step 11.
Step 8: If number of coefficients left = zerosleft – run_before, go to Step 11.
Step 9: Go to previous nonzero coefficient.
Step 10: Go to Step 3
Step 11: Stop

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

Table 7 Tables for run_before

V. HARDWARE IMPLEMENTATION AND PERFORMANCE ANALYSIS

The presented CAVLC design is implemented with Verilog HDL and synthesized on Xilinx Spartan 3E XC3s500 fpga. TABLE I. shows the device utilization summary.

RTL VIEW

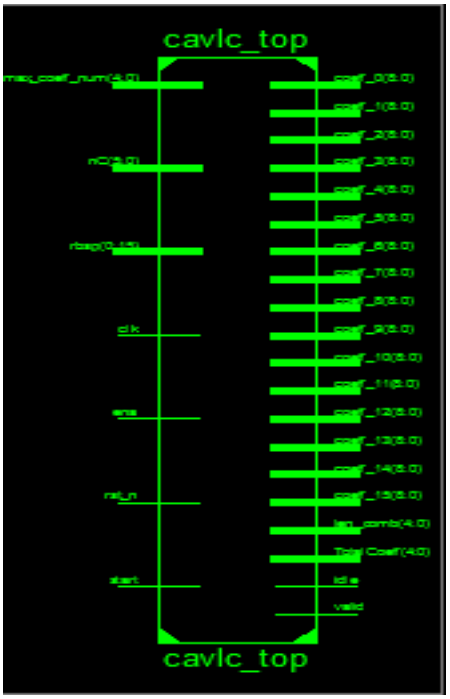
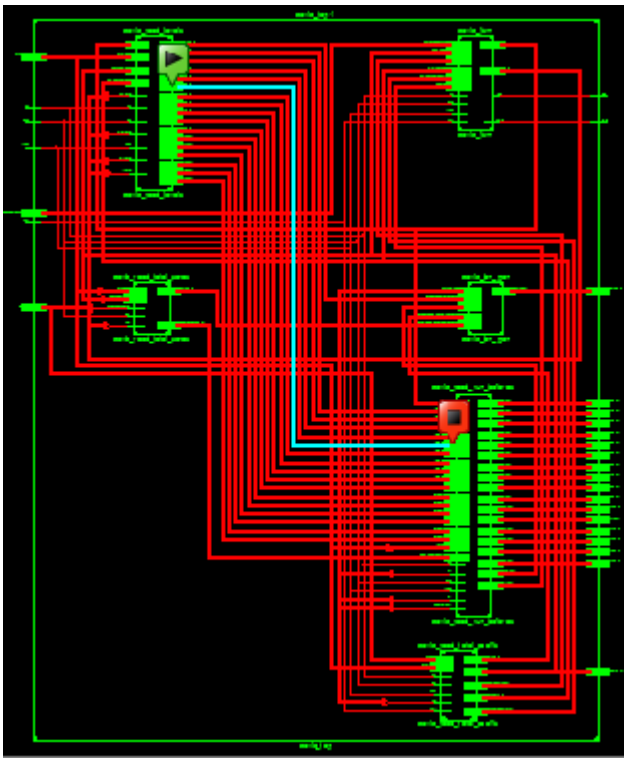


Table I

Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	402	9,312	4%
Number of 4 input LUTs	1,483	9,312	15%
Number of occupied Slices	806	4,656	17%
Number of Slices containing only related logic	806	806	100%
Number of Slices containing unrelated logic	0	806	0%
Total Number of 4 input LUTs	1,500	9,312	16%
Number used as logic	1,483		
Number used as a route-thru	17		
Number of bonded IOBs	187	232	80%
Number of BUFGMUXs	1	24	4%

RTL Schematic:



POWER COSUMPTION REPORT:

Device	
Family	Spartan3e
Part	xc3s500e
Package	fg320
Grade	Commercial
Process	Typical
Speed Grade	-4

On-Chip	Power (W)	Used	Available	Utilization (%)
Clocks	0.000	1	---	---
Logic	0.000	1490	9312	16.0
Signals	0.000	1506	---	---
IOs	0.000	187	232	80.6
Leakage	0.081			
Total	0.081			

Environment	
Ambient Temp (C)	25.0
Use custom TJA?	No
Custom TJA (C/W)	NA
Airflow (LFM)	0

Thermal Properties	Effective TJA (C/W)	Max Ambient (C)	Junction Temp (C)
	26.1	82.9	27.1

VI. CONCLUSION

This paper presents a high performance CAVLC encoder for H.264 video coding. With the efficient fine-grained zero skipping, the encoder design just needs one cycle for all-zero block encoding and the average fan-out of Non-clock nets 4.29 cycles for nonzero block coding with the area cost of 1483 LUT's .The total area cost reduced to 41% and the total gate count in the design is 10.2K gates. Further area optimization is possible by minimizing the codeword table as in previous VLC designs.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

REFERENCES

1. Joint Video Team (JVT), Draft ITU-T recommendation and Final Draft International Standard of Joint Video Specification. ITU-T Rec. H.264 and ISO/IEC 14496-10 AVC, May 2003.
2. I. Amer, W. Badawy, and Graham Jullien, “Towards MPEG4 part 10 system on chip VLSI prototype for context-based adaptive variable length coding (CAVLC),” in Proc. SIPS, pp. 275–279, 2004.
3. Y. K. Lai, C. C. Chou, and Y. C. Chung, “A simple and cost effective video encoder with memory-reducing CAVLC,” in Proc. ISCAS, pp. 432–435, May 2005.
4. T. C. Chen, and et al,“Dual-block-pipelined VLSI architecture of entropy coding for H.264/AVC baseline profile,” in Proc. VLSI-DAT, pp, 271–274, April, 2005
5. Joint Video Team reference software JM8.2

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)