Design of High Speed CAVLC encoder for H.264 codec

AMEE S. PANDYA

ME Student, Department of VLSI & Embedded System Design,

Gujarat Technological University PG School, Ahmadabad, Gujarat

amee.45@gmail.com

***ABSTRACT*: -** *Context-adaptive variable length coding (CAVLC) is a part of entropy coding used in H.264/MPEG-4 AVC video codec. It is an inherently lossless compression technique. CAVLC is used to encode residual, zigzag ordered 4 × 4 blocks of transform coefficients. CAVLC is designed to take advantage of several characteristics of quantised 4 × 4 blocks: 1) After prediction, transformation and quantisation blocks are typically sparse (containing mostly zeros). CAVLC uses run-level coding to represent strings of zeros compactly. 2) The highest nonzero coefficients after the zigzag scan are often sequences of ±1 and CAVLC signals the number of high-frequency ±1 coefficients ('Trailing Ones') in a compact way. 3) The number of nonzero coefficients in neighbouring blocks is correlated. The number of coefficients is encoded using a look-up table and the choice of look-up table depends on the number of nonzero coefficients in neighbouring blocks.*

KEYWORDS: H.264, Context adaptive variable length coding, run-level coding, entropy coding, VLC.

1. INTRODUCTION

H.264 is the newest video coding standard and is currently one of the hot subjects of video processing technologies. Coding quality and compression ratio have been greatly improved in the new standard compared with the previous standards. The context-based adaptive technology is introduced into the new standard [3], which can be said to be a technology renovation of the video coding. The main entropy coding technologies of H.264 include VLC (Variable-Length Coding) and CABAC (Context-based Adaptive Binary Arithmetic Coding). CAVLC is VLC and adopts the context-based adaptive technology; therefore the coding efficiency is greatly improved. Variable Length Code (VLC) plays an important role in the H.264 video compression standard. Many of the codewords used by H.264 are stored in look up tables (LUTS). Choice of a codeword is based on the frequency of occurrence of the symbol it represents. In H.264, several characteristics and properties of a frame are tracked in order to minimize the length of future codewords. Essentially, each symbol uses multiple VLC tables that are adapted to the symbol's context to create a codeword. This is officially known as Context based Adaptive Variable Length Coding, or CAVLC. In my proposed CAVLC encoder RAM (random access memory) is available for storing previously coded 4×4 blocks non zero coefficients which is use for coding of coefficient token block and as it is internal it provides fast access so encoding becomes fast.

2 OVERVIEW OF H.264

The H.264/AVC standard was first published in 2003. It builds on the concepts of its predecessors MPEG-2 and MPEG-4 Visual and offers better compression efficiency and greater flexibility in compressing, transmitting and storing video. The H.264/AVC standard is designed for a broad range of applications including broadcasting, Interactive or serial storage, conversational services, multimedia streaming services, multimedia messaging services. H.264 encoder involves in the prediction, transform and encoding processes to produce a compressed bit stream. H.264 decoder involves in the complementary processes of decoding, inverse transform and reconstruction to produce a displayable video sequence. This paper concentrates on the encoding side. Intra prediction algorithms use neighbouring previously encoded macroblocks from the current frame. Inter prediction algorithms use macroblocks from previously encoded frames that have been filtered. The residual samples are transformed using a 4x4 integer transform that is an approximate form of the Discrete Cosine Transform. The output of the transform is a set of coefficients. Each of the coefficients is a weighting value for a standard basis pattern. When the coefficients are combined the weighted basis patterns re-create the block of residual samples. The block of transform coefficients are then quantized by dividing each coefficient by an integer value i.e. quantization parameter. After quantization the result is a block in which most or all of the coefficients are zero, with a few non-zero coefficients. If quantization parameter is set to a high value then

more coefficients are set to zero resulting in high compression but poor decoded image quality. Setting quantization parameter to a low value means that more non-zero coefficients remain after quantization, resulting in better-decoded image quality but lower compression.

3. CAVLC ALGORITHM

Encode the number of coefficients and trailing ones (coeff token)

The first VLC, coeff token, encodes both the total number of nonzero coefficients (TotalCoeffs) and the number of trailing ±1 values (TrailingOnes). [1] Totalcoeffs can be anything from 0 (no coefficients in the 4 × 4 block) 5 to 16 (16 nonzero coefficients) and TrailingOnes can be anything from 0 to 3. If there are more than three trailing ±1s, only the last three are treated as 'special cases' and any others are coded as normal coefficients. There are four choices of look-up table to use for encoding coeff token for a 4 × 4 block, three variable-length code tables and a fixed-length code table. The choice of table depends on the number of nonzero coefficients in the left-hand and upper previously coded blocks (nA and nB respectively). A parameter nC is calculated as follows. If upper and left blocks nB and nA are both available (i.e. in the same coded slice), nC = round ((nA + nB)/2). If only the upper is available, nC = nB; if only the left block is available, nC = nA; if neither is available, nC = 0. The parameter nC selects the look-up table (Table) so that the choice of VLC adapts to the number of coded coefficients in neighbouring blocks (context adaptive).

Table: Selection of coefficient token

nC	Table for coeff_token
0,1	Table 1
2,3	Table 2
4,5,6,7	Table 3
8 or above	Table 4

Encode the sign of each TrailingOne

For each TrailingOne (trailing ±1) signalled by coeff token, the sign is encoded with a single bit (0 = +, 1 = −) in reverse order, starting with the highest-frequency TrailingOne.

Encode the total number of zeros before the last coefficient

The sum of all zeros preceding the highest nonzero coefficient in the reordered array is coded with a VLC, total zeros. The reason for sending a separate VLC to indicate total zeros is that many blocks contain a number of nonzero coefficients at the start of the array and (as will be seen later) this approach means that zero-runs at the start of the array need not be encoded.

Encode each run of zeros.

The number of zeros preceding each nonzero coefficient (run before) is encoded in reverse order. A run before parameter is encoded for each nonzero coefficient, starting with the highest frequency, with two exceptions:

1. If there are no more zeros left to encode (i.e.∑[run before]=total zeros), it is not necessary to encode any more run before values.
2. It is not necessary to encode run before for the final (lowest frequency) nonzero coefficient.

The VLC for each run of zeros is chosen depending on (a) the number of zeros that have not yet been encoded (ZerosLeft) and (b) run before. For example, if there are only two zeros left to encode, run before can only take three values (0, 1 or 2) and so the VLC need not be more than two bits long. If there are six zeros still to encode then run before can take seven values (0 to 6) and the VLC table needs to be correspondingly larger.

Encode the levels of the remaining nonzero coefficients.

The level (sign and magnitude) of each remaining nonzero coefficient in the block is encoded in reverse order, starting with the highest frequency and working back towards the DC coefficient. The code for each level is made up of a prefix (level prefix) and a suffix (level suffix). The length of the suffix (suffixLength) may be between 0 and 6 bits and suffixLength is adapted depending on the magnitude of each successive coded level ('context adaptive'). A small value of suffixLength is appropriate for levels with low magnitudes and a larger value of suffixLength is appropriate for levels with high magnitudes.

4. CAVLC PROPOSED ARCHITECTURE:

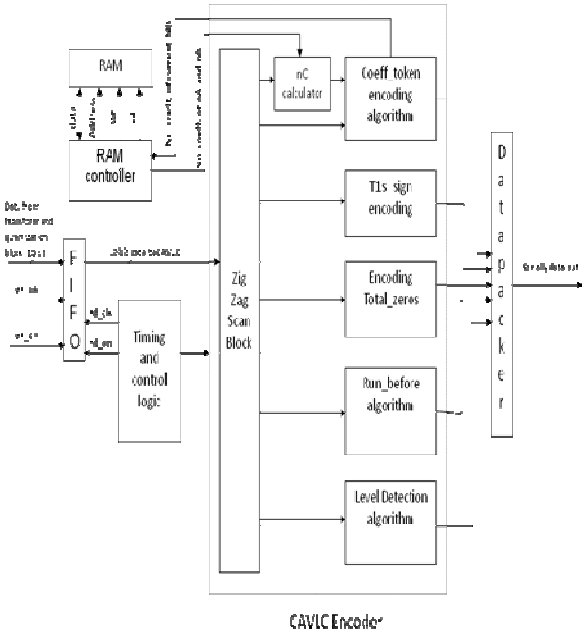

Figure: 1 CAVLC Architecture

FIFO (First in first out) :-

In this proposed architecture FIFO is used to receive and storing data coming from transform and quantization blocks of H.264 codec. We can say that FIFO will provide medium for coming data to pass to the CAVLC encoder. The incoming 10 bit data from transform and quantization blocks is writing to FIFO using write clock (wr_clk) and write enable (wr_en). When this data is needed by CAVLC encoder it will read from FIFO using read clock (rd_clk) and read enable (rd_en). This outgoing 10bit data is going to the zigzag scan module of CAVLC encoder.

Timing and control logic:-

This module will decide on which time which module is perform its function. Control module is used to control the overall operating sequence orders and generate the control signals those are necessary for each functional unit. When previous stage (transfers and quantize unit) finishes one block and writes data to FIFO, and then Control module will activate the CAVLC to start reading data from FIFO for encoding.

RAM and RAM controller:-

RAM is used to store number of coefficients of all encoded macroblock which are needed to calculate nC. This nC calculator will take stored number of coefficients of upper macroblock and left macroblock of current encoding macroblock.RAM controller controlling access of data from RAM according to read or write operation by providing data to nC calculator and taking data from coeff_token to store number of coefficient of current macroblock.

Zigzag Scan block:-

When the encoding processing starts, residuals and the type of blocks are sent to zigzag scan module under the control of control module. The backward scan technology is adopted, which scans 4×4 macroblock coefficients in following manners.

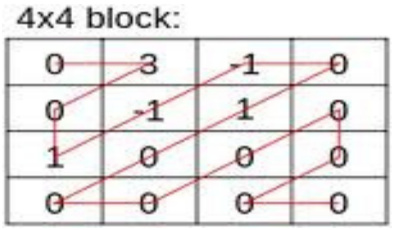

Figure:2 4x4 block

nC calculator:-

CAVLC also predicts the number of coefficients in the current array using the context variables nA and nB. These two variables represent the number of non zero coefficients in the neighboring 4x4 blocks, in particular the blocks that lie above (nA) and to the left (nB) of the current 4x4 block.

Coeff_token block :-

The Coeff_token generates the code by calculating the values of Totalcoeffs, number of T1 and nC value from the nC generator block. After taking the nC value from the nC generator block, the Coeff_token block implemented VLC tables and FLC table instead.

T1 sign encoding:-

This block takes input from zigzag scan block in reverse order. Then from last coefficient it finds trailing one and encodes is sign as 1 for negative and 0 for positive. Up to maximum three trailing ones it will encode sign and after that it takes coefficient as level coefficient.

Total Zeros block:-

Totalzeros block encodes the total number of zeros preceding the last nonzero coefficient by counting the Totalcoeffs and counting Totalzeros. Case by case code and length are stored in look up tables. Depending on the value of the Totalcoeffs and Totalzeros the code and length of the code are sent as outputs to the data packer block. For example if the value of Totalcoeffs is 7, the value of Totalzeros can range from 0 to 9. So the code and length for each case i.e. Totalzeros=0 to Totalzeros=9 is stored and the output of the Totalzeros block will be the code output and length depending on the value of the Totalzeros and Totalcoeffs.

Level encode block:-

The level block encodes the nonzero coefficients. Level gives the level of each nonzero coefficient. The format of the level code is the level prefix followed by bit 1 and that followed by level suffix. Level prefix is the array of zeros that are in level code and level suffix is the array of bits. The level input is taken for each nonzero coefficient and this block generates the level prefix, length of level prefix, level suffix and length of level suffix based on the look up tables. These are then sent to the data packer.

Run before block:-

The run before block encodes the number of zeros preceding each nonzero coefficient in reverse zigzag order by taking the inputs run before, Totalcoeffs and Totalzeros. The block calculates the run before for each nonzero coefficient. It determines the number of zeros left by subtracting the run before from Totalzeros. Depending on the value of the zeros left and the run before, the code and the length of the code are determined from the look up table.

Data Packer:-

The data packer receives all the codewords generated in the encoding phase, and when enabled, creates the bit stream. The Data Packer block gets T1's Sign inputs from trailing one sign encoding clock; Coefftoken and length of the token from the

Coefftoken block; Totalzeros code and its length from the Totalzeros block; Level suffix, level prefix and their lengths from the level block; Run before code and its length for each nonzero coefficient from run before block. The Data Packer generator reads in the inputs and their lengths and appends all the code based on their lengths together in order to generate the encoded bitstream.

5. SIMULATION RESULTS

Coeff_token block Simulation

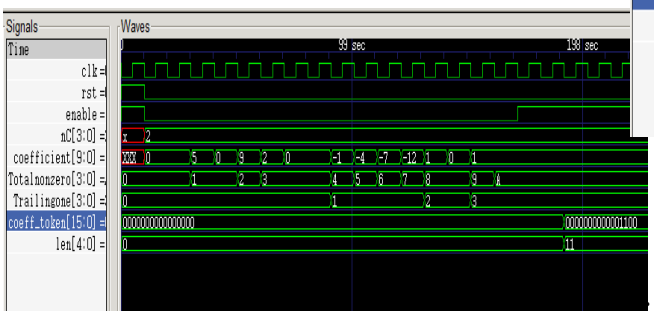

Figure : 3 Coefficient token simulation

T1 sign encoding simulation:

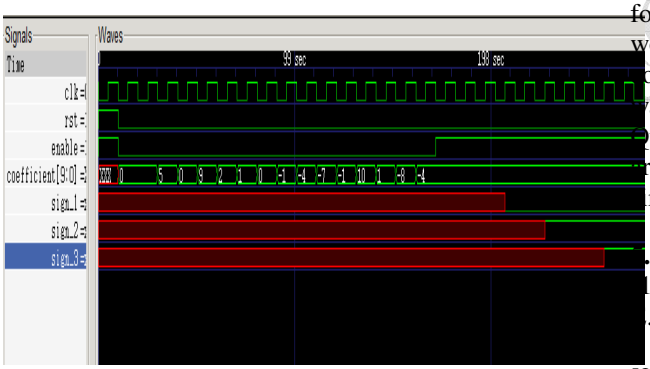

Figure : 4 Trailing ones sign encoding simulation

Total Zeros block Simulation

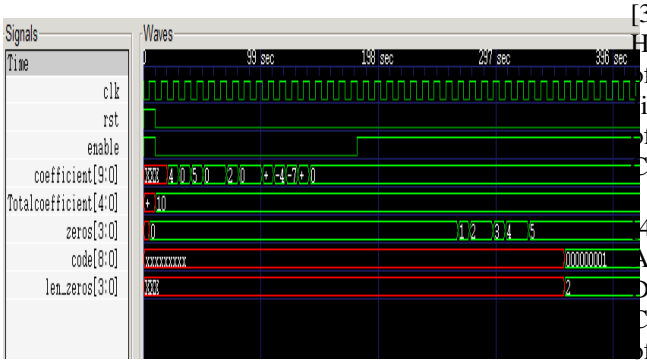

Figure : 5 Total embedded zero simulation

Run before Simulation:

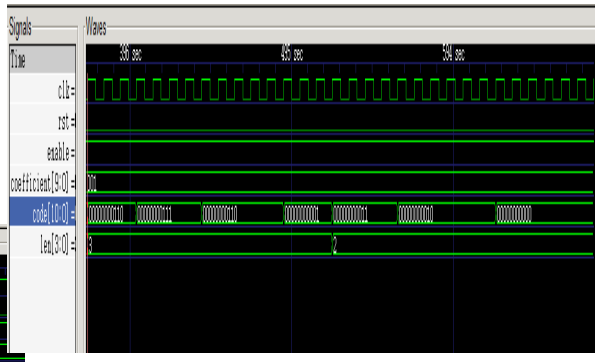

Figure : 6 Runbefore simulation

CONCLUSION

CAVLC is the entropy coding method adopted in H.264/AVC. In this paper, I present high speed technique for CAVLC VLSI implementation which has internal RAM so there is no need to use external for storing nA and nB. All modules of CAVLC are worked in parallel so there is no need to wait for completion of any module. The CAVLC architecture was designed and verified with simulations using the Quartus II. Verilog HDL description of the architecture was verified for functionality with simulation.

REFERENCES

[1] "H.264 and MPEG - 4 video compression" By Iain E. G. Richardson.

[2] "High-Speed CAVLC Encoder for 1080p 60-Hz H.264 Codec," Yi, Y., Song, B. C., Signal Processing Letters. IEEE.vol.15, pp.891-894(2008).

[3] "A VLSI Implementation for CAVLC for H.264/AVC " Li Luo, Zheying Li, Qinmei Yu School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing, China, 100044; College of Information, Beijing Union University, Beijing, China, 100101

[4] "Highly efficient CAVLC encoder for MPEG-4 AVC/H.264" T.-H. Tsai S.-P.Chang T.-L. Fang Department of Electrical Engineering, National Central University, Chung-Li 320, Taiwan, Republic of China"