A HIGH PERFORMANCE AND LOW POWER HARDWARE ARCHITECTURE FOR H.264 CAVLC ALGORITHM

Esra Sahin and Ilker Hamzaoglu

Faculty of Engineering and Natural Sciences, Sabanci University
34956, Orhanli, Tuzla, Istanbul, TURKEY
phone: + (90) 216 483-9577, fax: + (90) 216 483-9550, email: hamzaoglu@sabanciuniv.edu
web: www.sabanciuniv.edu/~hamzaoglu

ABSTRACT

In this paper, we present a high performance and low power hardware architecture for real-time implementation of Context Adaptive Variable Length Coding (CAVLC) algorithm used in H.264 / MPEG4 Part 10 video coding standard. This hardware is designed to be used as part of a complete low power H.264 video coding system for portable applications. The proposed architecture is implemented in Verilog HDL. The Verilog RTL code is verified to work at 76 MHz in a Xilinx Virtex II FPGA and it is verified to work at 233 MHz in a 0.18µ ASIC implementation. The FPGA and ASIC implementations can code 22 and 67 VGA frames (640x480) per second respectively.

1. INTRODUCTION

Video compression systems are used in many commercial products, from consumer electronic devices such as digital camcorders, cellular phones to video teleconferencing systems. These applications make the video compression hardware devices an inevitable part of many commercial products. To improve the performance of the existing applications and to enable the applicability of video compression to new real-time applications, recently, a new international standard for video compression is developed. This new standard, offering significantly better video compression efficiency than previous video compression standards, is developed with the collobaration of ITU and ISO standardization organizations. Hence it is called with two different names, H.264 and MPEG4 Part 10.

The video compression efficiency achieved in H.264 standard is not a result of any single feature but rather a combination of a number of encoding tools. As it is shown in the top-level block diagram of an H.264 Encoder in Figure 1, one of these tools is the Context Adaptive Variable Length Coding (CAVLC) algorithm used in the baseline profile of H.264 standard [1, 2, 3].

CAVLC algorithm is used to encode transformed and quantized 4x4 residual luminance and chrominance blocks. CAVLC algorithm uses multiple VLC tables for a syntax element. It adapts to the current context by selecting one of the VLC tables for a given syntax element based on the already transmitted syntax elements. This context-adaptivity provides better entropy coding performance in comparison to an entropy coding algorithm using a single VLC table. In addition, CAVLC algorithm improves the entropy coding performance by using coding techniques such as run-level and trailing ones coding that are designed to take advantage of the characteristics of the 4x4 blocks of transformed and quantized residual data [2, 3, 4].

H.264 CAVLC algorithm achieves better coding results than the entropy coding algorithms used in the previous video compression

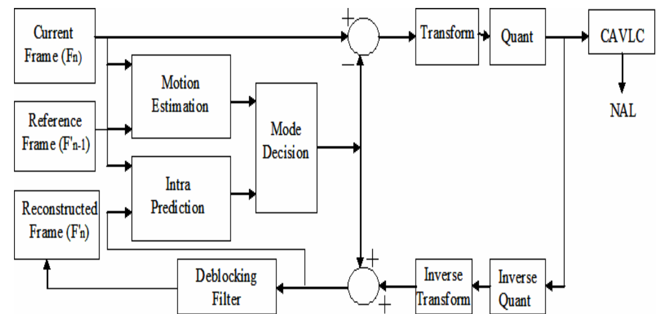


Figure 1 H.264 Encoder Block Diagram

standards. This coding gain, however, comes with an increase in encoding complexity which makes it an exciting challenge to have a real-time implementation of this algorithm.

In this paper, we present a high performance and low power hardware architecture for real-time implementation of H.264 CAVLC algorithm. This hardware is designed to be used as part of a complete low power H.264 video coding system for portable applications. The proposed architecture is implemented in Verilog HDL. The Verilog RTL code is verified to work at 76 MHz in a Xilinx Virtex II FPGA and it is verified to work at 233 MHz in a 0.18µ ASIC implementation. The FPGA and ASIC implementations can code 22 and 67 VGA frames (640x480) per second respectively.

A hardware architecture for H.264 CAVLC algorithm similar to our design is presented in [5]. However, that architecture doesn't have the low-power techniques we have used in our design.

The rest of the paper is organized as follows. Section II gives a brief overview of the CAVLC algorithm used in H.264 standard. Section III describes the proposed architecture in detail. The implementation results are given in Section IV. Finally, Section V presents the conclusions.

2. H.264 CAVLC ALGORITHM OVERVIEW

CAVLC algorithm is used to encode transformed and quantized residual luminance and chrominance blocks in a macroblock in the order shown in Figure 2. Block -1 is formed by the DC coefficients of 4x4 luminance blocks only for the macroblocks that are coded in 16x16 Intra Mode. Blocks 16 and 17 are formed by the DC coefficients of 4x4 chrominance blocks for all the macroblocks. All the transformed and quantized 4x4 and 2x2 blocks for a macroblock are given as inputs to CAVLC algorithm in the order shown in Figure 2. CAVLC algorithm processes each 4x4 block in zig-zag

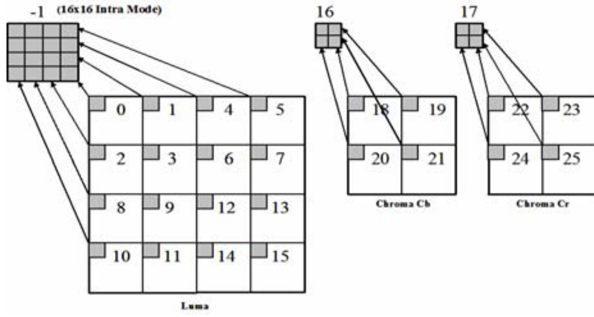


Figure 2 Coding Order of Blocks in a Macroblock

scan order and each 2x2 block in raster scan order. It encodes each block in the following five steps [2, 3, 4].

Step 1. It generates coeff_token, the variable length code that encodes both the number of non-zero coefficients (TotalCoeff) and the number of trailing ±1 values (TrailingOnes) in a block.
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 neighbouring blocks is correlated, CAVLC algorithm generates coeff_token for a block context adaptively. It uses one of the four different VLC tables for generating the coeff_token for a block based on the number of non-zero coefficients in the neighbouring blocks as follows. It first calculates a parameter nC based on the number of non-zero coefficients in the left-hand and upper previously coded blocks, nA and nB respectively. 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. As a special case, for 2x2 dc chroma blocks, nC is always set to -1. It, then, selects the VLC table that will be used for generating the coeff_token based on the value of nC as shown in Table 1.

Table 1

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

Step 2. It encodes the sign of each TrailingOne with a single bit in reverse order starting with the highest-frequency TrailingOne.

Step 3. It encodes the level (sign and magnitude) 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 codeword for a level consists of a prefix and a suffix. Since the magnitude of non-zero coefficients tends to be larger near the DC coefficient and smaller towards the higher frequencies, CAVLC algorithm adapts the suffix length for the level parameter depending on recently-coded level magnitudes. It sets the suffix length for the first level, except in some special cases, 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 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 versus when the suffix length for the level is greater than 0 [4].

Step 4. It encodes the total number of zeros before the last non-zero coefficient (Total_Zeros) using a VLC table.

Step 5. It encodes the number of zeros preceding each non-zero coefficient (Run_Before) in reverse order starting with the highest-frequency coefficient. Since after transformation and quantization, blocks typically contain mostly zeros, CAVLC algorithm uses run-level coding to represent strings of zeros compactly.

3. PROPOSED HARDWARE ARCHITECTURE

The proposed hardware architecture for H.264 CAVLC algorithm is shown in Figure 3. The proposed hardware performs context-adaptive variable length coding for a macroblock, in the worst case, in 2880 clock cycles. The worst-case occurs for the macroblocks that have no zero coefficients and trailing ±1 coefficients. Therefore, the proposed hardware can process 30 VGA frames per second at 104 MHz. In the following subsections, we will explain the hardware architecture in detail.

3.1 VLC Counters and Reverse Zig-zag Ordering

CAVLC hardware contains a number of counters and register files to store the information for a block that will be encoded by variable length codes. Non-Zero Coefficients counter is used to store the number of non-zero coefficients (TotalCoeff). TrailingOnes counter is used to store the number of trailing ±1 values (TrailingOnes). TotalZeros counter is used to store the total number of zeros before the last non-zero coefficient (Total_Zeros). Level counter is used to store the number of non-zero coefficients other than the TrailingOnes. TrailingOnes register file is used to store the sign of each TrailingOne. 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 block by reading the coefficients from the input buffer in reverse zig-zag order. In each cycle, it reads one coefficient from the input buffer, analyzes the coefficient and updates the information stored in the related counter and register file. At the end of this process, the counters and register files mentioned above contain all the information for the current block that will be encoded with variable length codes.
Reverse zig-zag scanning enables us to determine the necessary information for encoding a 4x4 block by reading and analyzing each coefficient only once. This reduces the power consumption by reducing the switching activity on the input buffer address and data signals.
It takes 16 cycles to read the coefficients and store the corresponding information in the counters and register files for each 4x4 luminance block in the macroblocks that are not coded in 16x16 Intra Mode. The same process takes 16 cycles for block -1 and 15 cycles for the other 4x4 luminance blocks in the macroblocks that are coded in 16x16 Intra Mode. Because DC coefficients in 4x4 luminance blocks in these macroblocks are coded in block -1. The same process takes 4 cycles for 2x2 chrominance blocks 16 and 17, and 15 cycles for the 4x4 chrominance blocks in all the macroblocks due to the same reason.

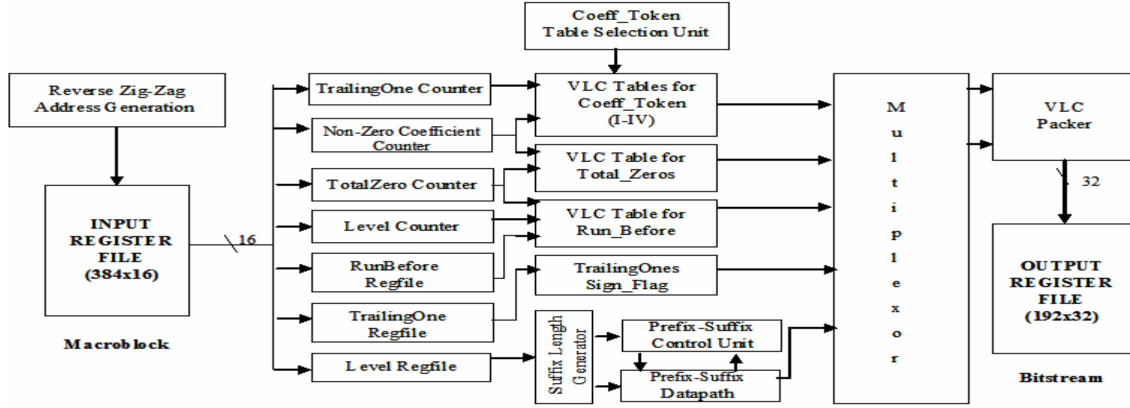


Figure 3 Context Adaptive Variable Length Coder Hardware Architecture

3.2 CAVLC Hardware for Generating Coeff_Token

CAVLC hardware generates coeff_token, the variable length code that encodes both the number of non-zero coefficients (TotalCoeff) and the number of trailing ±1 values (TrailingOnes) in a block, by a VLC table lookup based on the values of Non-Zero Coefficients and TrailingOnes counters. CAVLC hardware uses one of the four different VLC tables for generating the coeff_token for a block based on the number of non-zero coefficients in the neighbouring blocks. Coeff_Token Table Selection Unit (CT_TSU) shown in Figure 3 determines the VLC table that will be used for the current block as follows.

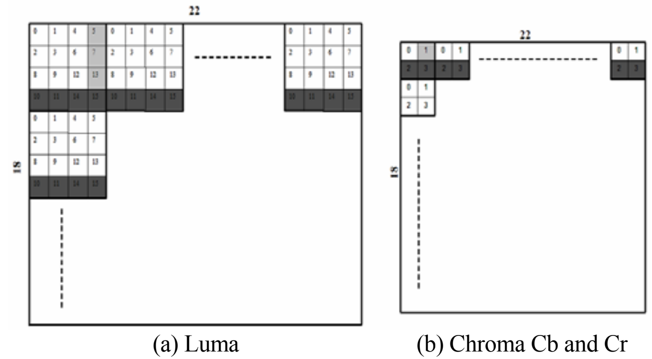
CT_TSU first calculates the parameter nC based on the number of non-zero coefficients in the left-hand and upper previously coded blocks, nA and nB respectively. It uses three internal SRAMs and six internal register files to determine the number of non-zero coefficients in the left-hand and upper previously coded neighbouring blocks of a 4x4 block in a frame. The organization of the luminance and chrominance components of the macroblocks in a CIF frame is shown in Figure 4. The blocks within a macroblock are organized and numbered as shown in Figure 2.

CT_TSU uses three internal register files to store the number of non-zero coefficients in each 4x4 block in a macroblock; one for luminance, one for chrominance Cr, and one for chrominance Cb component. After CT_TSU processes a 4x4 block in the current macroblock, it updates the number of non-zero coefficients entry for this block in the corresponding register file. When it is later processing a new 4x4 block (cblk), if the neighbouring 4x4 block (nblk) of cblk is in the same macroblock as cblk, CT_TSU determines the number of non-zero coefficients in nblk using the corresponding register file entry.

CT_TSU uses three internal register files to store the number of non-zero coefficients in the blocks 5, 7, 13 and 15 of the previously coded macroblock; one for luminance, one for chrominance Cr, and one for chrominance Cb component. It uses this data to determine the number of non-zero coefficients in the left-hand previously coded neighbouring block of the blocks 0, 2, 8 and 10 in the current macroblock.

CT_TSU uses three internal SRAMs to store the number of non-zero coefficients in the blocks 10, 11, 14, and 15 of the macroblocks in the previously coded macroblock row of the frame; one for luminance, one for chrominance Cr, and one for chrominance Cb component. It uses this data to determine the number of non-zero coefficients in the upper previously coded neighbouring block of the blocks 0, 1, 4 and 5 in the current macroblock.

We have disabled the SRAMs when they are not accessed in order to reduce power consumption in CT_TSU. In addition, instead of using one large external SRAM, we have used three internal SRAMs and six internal register files to store the number of non-zero coefficients in previously coded blocks in order to further reduce power consumption in CT_TSU. This is achieved in two ways. First, instead of accessing one large external SRAM, one internal SRAM and two register files are used for a block. The other internal SRAMs and register files are not accessed saving power. Second, instead of using complex address generation logic for an external SRAM, much simpler address generation logic is used for the internal SRAMs and register files. The blocks in the same locations of different macroblocks write to and read from the same SRAM and register file locations. So, if the left-hand and upper previously coded neighbouring blocks of a 4x4 block are available, the read addresses for the number of non-zero coefficients in these blocks are generated by a table lookup based on its macroblock and block number.



(a) Luma (b) Chroma Cb and Cr

Figure 4 Macroblocks in a CIF Frame

Consequently, CT_TSU calculates the parameter nC for the current block in just one cycle. It then selects the VLC table for the coeff_token based on the value of nC as shown in Table 1. It then generates coeff_token for the current block by a table lookup to the selected VLC table based on the values of Non-Zero Coefficients and TrailingOnes counters.

3.3 CAVLC Hardware for Encoding Level

The codeword for a level consists of a prefix and a suffix. Suffix length generator shown in Figure 3 determines the suffix length for the current level based on the magnitude of previously coded levels. The prefix-suffix datapath and control unit shown in Figure 3 generate the code length and the codeword for the current level based on the suffix length provided by the suffix length generator. The prefix for a level is generated using different equations when

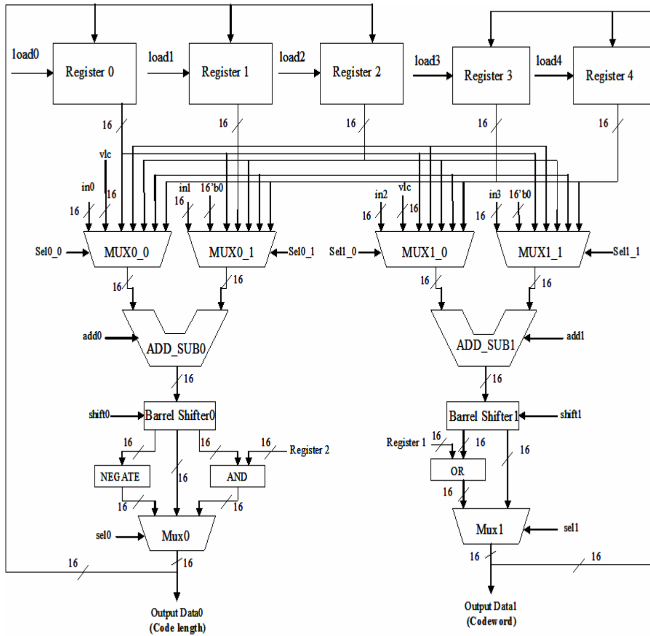


Figure 5 Datapath for Coding Level Prefix and Level Suffix

the suffix length for the level is 0 versus when the suffix length for the level is greater than 0. The equations used in both cases are given in the Joint Model (JM) Reference Software Version 8.2 [4]. We have proposed the dual purpose prefix-suffix datapath shown in Figure 5 to implement these equations. In each case, the prefix-suffix control unit sends the appropriate control signals to the datapath. In the worst case, our suffix length generator, prefix-suffix datapath and control unit take 6 cycles to generate the code length and codeword for a level.

3.4 VLC Packer

Since CAVLC generates variable length codewords, consecutive codewords should be packed into fixed size words before being written to output register file. The datapath shown in Figure 6 is used to pack the variable length codewords into 32-bit words [6]. When a new codeword is sent to the VLC packer, the barrel shifter places this codeword next to the end of the bitstream stored in the 32-bit lower register. If the length of the resulting bitstream is larger than 31, the carry-out bit of the adder in the datapath is set to one. This indicates that a 32-bit bitstream is packed into the upper register. Thus, VLC packer outputs the content of the upper register, and it moves the content of the lower register into the upper register. This process is repeated for each codeword.

4. IMPLEMENTATION RESULTS

The proposed architecture is implemented in Verilog HDL. The implementation is verified with RTL simulations using Mentor Graphics ModelSim SE. The Verilog RTL is then synthesized to a 2V8000ff1157 Xilinx Virtex II FPGA with speed grade 5 using Mentor Graphics Leonardo Spectrum [7]. The resulting netlist is placed and routed to the same FPGA using Xilinx ISE Series 5.2i.
The FPGA implementation including input and output register files as well is placed and routed at 76 MHz under worst-case PVT conditions. Since, in the worst-case, it takes 2880 clock cycles to process a MB, the FPGA implementation can code 22 VGA frames (640x480) per second. The FPGA implementation is verified to work in a Xilinx Virtex II FPGA on an Arm Versatile Platform development board.

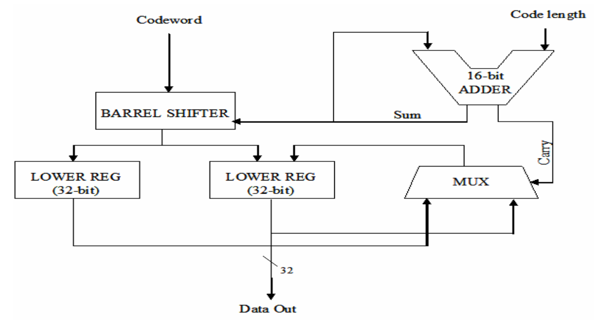


Figure 6 VLC Packer Datapath

The FPGA implementation including input and output register files as well used the following FPGA resources; 3946 Function Generators, 1973 CLB Slices, 719 Dffs /Latches, and 6 Block RAMs, i.e. 4.23% of Function Generators, 4.23% of CLB Slices, 0.75% of Dffs /Latches, and 3.57% of Block RAMs. The FPGA implementation excluding input and output register files used the following FPGA resources; 3849 Function Generators, 1925 CLB Slices, 719 Dffs /Latches, and 6 Block RAMs, i.e. 4.13% of Function Generators, 4.13% of CLB Slices, 0.75% of Dffs /Latches, and 3.57% of Block RAMs.
The Verilog RTL is also synthesized to Virtual Silicon UMC 0.18µ standard-cell library using Synopsys Design Compiler. The netlist excluding input and output register files has an area of 32K gates. The netlist including input and output register files has an area of 96K gates and it is verified to work at 233 MHz under worst-case PVT conditions with post synthesis simulations. This 0.18µ ASIC implementation can code 67 VGA frames (640x480) per second.

5. CONCLUSIONS

In this paper, we presented a high performance and low power hardware architecture for real-time implementation of H.264 CAVLC algorithm. This hardware is designed to be used as part of a complete low power H.264 video coding system for portable applications. The proposed architecture is implemented in Verilog HDL. The Verilog RTL code is verified to work at 76 MHz in a Xilinx Virtex II FPGA and it is verified to work at 233 MHz in a 0.18µ ASIC implementation. The FPGA and ASIC implementations can code 22 and 67 VGA frames (640x480) per second respectively.

REFERENCES

[1] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra *“Overview of the H.264/AVC Video Coding Standard*”, IEEE Trans. on Circuits and Systems for Video Technology vol. 13, no. 7, pp. 560–576, July 2003

[2] I. Richardson, *H.264 and MPEG-4 Video Compression,* Wiley, 2003

[3] Joint Video Team (JVT) of ITU-T VCEG and ISO/IEC MPEG, *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

[4] Joint Video Team (JVT) of ITU-T VCEG and ISO/IEC MPEG, *Joint Model (JM) Reference Software Version 8.2*, http://bs.hhi.de/suehring/

[5] Y. W. Huang, B. Y. Hsieh, T. C. Chen, and L. G. Chen, “*Hardware Architecture Design for H.264/AVC Intra Frame Coder*”, Proc. of IEEE ISCAS, 2004

[6] V. Bhaskaran and K. Konstantinides, *Image and Video Compression Standards: Algorithms and Architectures*, Kluwer Academic Publishers, 2nd Edition, 1997