An Efficient Hardware Implementation for CAVLC Encoder in H.264/AVC

Milica Orlandić
Department of Electronics and Telecommunications
Norwegian University of Science and Technology, NTNU
Trondheim, Norway
orlandic@iet.ntnu.no

Abstract—**This paper presents an efficient FPGA implementation of the context-based adaptive variable length coding (CAVLC) in H.264/AVC. The proposed entropy encoder architecture includes three stages: scan stage, coding stage and bitstream packing. Its performance is optimized by eliminating or weakening data dependencies such as memory accessing and context based data, by computing codewords on-the-fly and by pipelining stages in the encoding process. The design has been synthetized and implemented on Kintex 7 FPGA board.**

Keywords - H.264/AVC, Entropy Encoding, CAVLC, FPGA

I. INTRODUCTION

Video compression is an intensive computational application involving several complex stages such as transform coding, prediction algorithms and entropy coding. Entropy encoders are characterized by serial nature, thus their parallel system implementation represents a challenge. This limits entropy encoders in existing video players to provide support for data rates of high resolution videos. In addition, real-time constraint for compression of high definition (HD) video sequences requires high-performance processors or dedicated hardware implementations, in particular in video standards characterized by large computational complexity such as H.264/AVC. The information produced in the encoder stages, such as transform coding residuals, are referred in entropy encoder as syntax elements. In H.264/AVC standard a number of entropy encoding techniques are defined for different types of syntax elements and video standard profiles. Depending on profile of H.264/AVC standard, the choice for encoding residual data consists of CAVLC and CABAC encoding modules. Entropy encoding is recognized as the bottleneck due to the inherent dependence among the syntax elements and, therefore, a number of hardware designs for CAVLC have been proposed in order to meet the high throughput requirement. The recent works on CAVLC implementations are reported in literature [1], [2], [3], [4].

In this paper, an efficient hardware implementation of the CAVLC encoder for H.264/AVC video coding standard is presented. The architecture proposes a number of optimization techniques in order to speed up the encoding process and to save the hardware area.

The paper is organized as follows. Details of the CAVLC algorithm are briefly reviewed in Section II. Section III presents the proposed CAVLC encoder architecture. In Section IV the achieved performance in terms of hardware resources is discussed. Finally, conclusions are presented in Section V.

II. CAVLC ALGORITHM

A CAVLC encoding process can be partitioned into three phases: pre-processing block scan, syntax element encoding and bitstream formation. The scan ordering tends to group significant coefficients by the proposed zig-zag order in the manner that non-zero elements are clustered around the DC coefficient. The syntax element encoding produces five syntax elements. Each syntax element contains several look-up tables for storing VLC tables. *Coeff_token* syntax element presents a pair of numbers: number of non-zero coefficients and number of trailing ones. Trailing ones parameter indicates the number of high frequency coefficients, and its range is [0-3]. Depending on the number of elements in the neighboring left and upper blocks, one among defined VLC tables for *Coeff_token* is chosen. String of trailing ones signs is the second syntax element. *Levels* is the vector of remaining non-zero coefficients within 4x4 block and its syntax element code is given by [0 . . . 0 1 *x* . . . *x s*], where the string of zeros followed by stop bit '1' is *Prefix level*, the sequence of bits after the stop bit is *Suffix Level* and *s* is the sign of the *Levels* coefficient. The level encoding is context adaptive since successive level coding depends on the magnitude of the previously coded level. The *Total_Zeros* syntax element counts the number of embedded zeros in between non-zero coefficients. The codeword *Total_Zeros* is encoded by using look-up table depending on total number of non-zero coefficients and total number of zeros. The final codeword of the encoding sequence is *Run_before*, encoded by a look-up table which depends on two derived parameters, the vector *Run* and the parameter *Zeros_par*. The vector *Run* counts zeros between each non-zero coefficients, whereas *Zeros_par* is the number of embedded zeros yet to be encoded until the last non-zero element.

III. THE PROPOSED CAVLC ARCHITECTURE

The design organization is given in Figure 1. The preprocessing stage reorganizes the coefficients within 4x4 block in the zig-zag order. In this stage the flags needed for encoding syntax elements, such as Sign flag, Non-zero coefficient flag and Ones flag, are computed. *Coeff_token*, *Total_zero* and *Run_before* codewords are extracted from look-up tables containing VLC tables. In *Level* encoding stage, the technique Arithmetic Table Expression (ATE) is employed for reducing memory utilization. It consists of detecting relations between codewords in a table and using arithmetic operation for codeword reconstruction. A packing stage concatenates

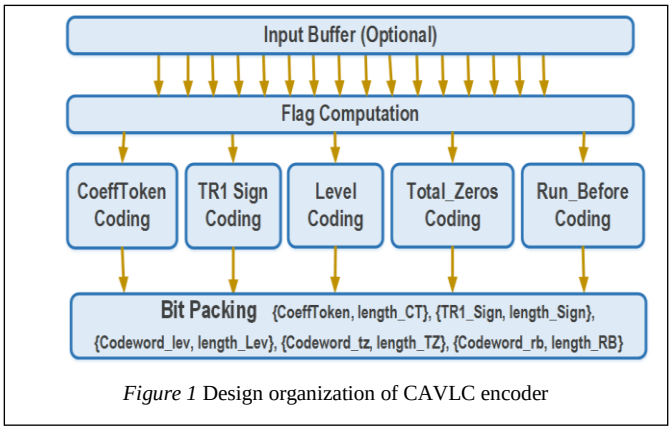

Figure 1 Design organization of CAVLC encoder

syntax element codewords within a block into a partial bitstream. Memory buffer for storing the partial bistream consists of four 32-bit words. The storing data operation includes also 32-bit alignment with unaligned data from the previous blocks. If partial bitstream for current block is larger than 32-bit chunks, these chunks are sent out. The time diagram for each phase of the proposed implementation is given in Figure 2. As presented, the critical path is lowered by performing the syntax encoding and the bit packing stages in parallel when data dependency allows.

IV. RESULTS

The proposed architecture has been modeled in VHDL and synthetized and implemented in Xilinx ISE 14.7 on Kintex 7 FPGA board. Preliminary synthesis results in terms of hardware resources utilization are reported in Table I. Operating frequency for the proposed design is 110 MHz. The data from reference syntax element encoder in MATLAB are compared with the simulation data of the proposed implementation produced in Active HDL.

TABLE I. UTILIZATION OF HARDWARE RESOURCES

Technology	FPGA, KINTEX 705-XC7K325T
LUTS	6210 (3%)
FPGA Slices	2442 (4%)
FF Utilization	1485 (1%)
RAM (kbits)	38.2

V. CONCLUSIONS

In this paper a hardware implementation of entropy encoding algorithm CAVLC in H.264/AVC video standard has been presented. The development process was focused on achieving a high throughput performance in syntax element encoding and bit packing. Future work includes implementation of the output FIFO and improved pipeline alignment among blocks in bit packing phase, further testing of bit packing, operating frequency improvement, statistical analysis of bitsream in HD video sequences and adaptation of the proposed architecture as hardware accelerator within larger SoC which includes components such as external video sources, processor and large memory for storing video data.

REFERENCES

[1] G. D. Licciardo and L. F. Albanese, "Design of a context-adaptive variable length encoder for real-time video compression on reconfigurable platforms," *Image Processing, IET*, vol. 6, pp. 301-308, 2012.

[2] N.-M. Nguyen, X.-T. Tran, P. Vivet, and S. Lesecq, "An efficient Context Adaptive Variable Length coding architecture for H. 264/AVC video encoders," in *Advanced Technologies for Communications (ATC), 2012 International Conference on*, 2012, pp. 158-164.

[3] C.-W. Chang, W.-H. Lin, H.-C. Yu, and C.-P. Fan, "A high throughput CAVLC architecture design with two-path parallel coefficients procedure for digital cinema 4K resolution H. 264/AVC encoding," in *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*, 2014, pp. 2616-2619.

[4] M. P. Hoffman, E. J. Balster, and W. F. Turri, "High-throughput CAVLC architecture for real-time H. 264 coding using reconfigurable devices," *Journal of Real-Time Image Processing*, vol. 11, pp. 75-82, 2016.

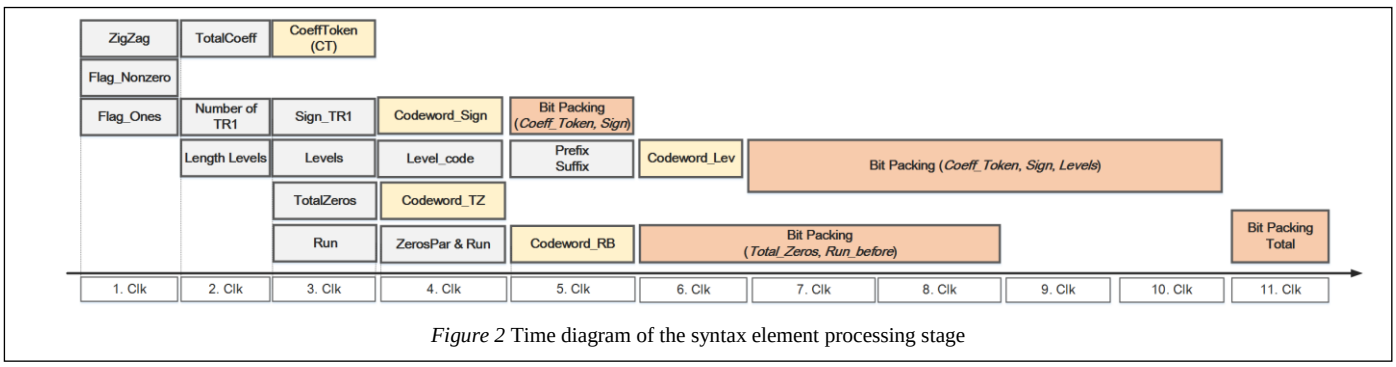

Figure 2 Time diagram of the syntax element processing stage