An Efficient Hardware Architecture of CAVLC Encoder Based on Stream Processing

Milica Orlandić*, Kjetil Svarstad

Department of Electronic Systems
NTNU - Norwegian University of Science and Technology
Trondheim, Norway

Abstract

The paper presents an efficient implementation of Context-Adaptive Variable Length Coding (CAVLC) entropy encoder in H.264/AVC standard. The architecture is designed with a parallel structure targeting real-time video compression. The intensive memory access demand in the syntax element coding stage is lowered by using the proposed arithmetic table elimination technique. The packing stage implementation is interleaved with syntax element generation stage and includes fast methods for syntax elements concatenation. The register update method performs concatenation of the bitstream of previously processed sub-blocks and the syntax codewords of the currently processed sub-block. The CAVLC encoder processes 4×4 sub-block coefficients in parallel, introducing the initial latency of 12 clock cycles, after which the full pipeline of the data encoding on the sub-block level is performed, and 16 residuals are processed at each clock cycle. The achieved high throughput allows the encoding core to perform real-time processing of 8K UHD (4320p) video sequences with a frame rate of 30 frames/s.

Keywords: H.264/AVC; Entropy Encoding, CAVLC, FPGA, Bitstream packing, Hardware implementation.

*Corresponding author
Email address: milica.orlandic@ntnu.no (Milica Orlandić)

1. Introduction

Video compression is an intensive-computational application involving several stages such as transform coding, prediction algorithms and entropy coding. After the video sequence is compressed to a series of residuals in the prediction and transform stages, entropy encoding uses the statistical properties to compress data. The number of bits produced by entropy encoders is logarithmically proportional to the probability of the data. Entropy encoders are of serial nature due to data dependencies between the encoding elements, and achieving high throughput of entropy coding process represents a challenge nowadays. Existing video players, despite great efforts, cannot provide support for high bitrates dictated by increasing resolutions, and the design of high performance entropy encoder architectures on dedicated parallel hardware represents a solution.

The various types of information produced in the encoder stages, such as residuals from transform coding stage or mode flags from prediction stage, are referred as the syntax elements. In the H.264/AVC standard [1] a number of entropy encoding techniques are defined for different types of syntax elements and video standard profiles. Depending on the profile, residual encoding can be performed by Context-Adaptive Variable Length Coding (CAVLC) [2] and Context-Based Adaptive Binary Arithmetic Coding (CABAC) [3]. CABAC achieves bitrate savings when compared to CAVLC, but its higher complexity meets challenges in achieving efficient execution for high bitrate video content. Based on compression-complexity tradeoff compared to CABAC, CAVLC is deployed in the Baseline and Extended profile of H.264/AVC.

A number of hardware designs for CAVLC have been proposed in order to meet the high throughput requirements of high- definition coding. Recent works on CAVLC FPGA implementations [4, 5, 6, 7, 8] and ASIC designs [9, 10, 11, 12] are reported in literature. Moon *et al.* [4] propose a solution for decoding *Run_before* codeword in video length decoding (VLD) without the look-up tables. Lo *et al.* [5] combine two entropy decoding methods in H.264/AVC standard in the shared implementation. The shared components between CABAC

and CAVLC are context adaptation module, input and line buffers, whereas level computation in CAVLC level decoder and CABAC inverse binarization and look-up tables for other phases of entropy encoding within both standards are not adapted and merged into the common component. Ramos *et al.* [6] propose an implementation characterized by 2-pixel input parallelism, and the focus is on the speed up of syntax element encoding, in particular *Levels* encoding stage. Licciardo *et al.* [7] focus on minimizing area cost by using an arithmetic table elimination technique, however the operating frequency is the limiting factor for high definition content real-time processing. Hoffman *et al.* [8] present an implementation for surveillance applications with high frame rate characterized by a modified dual-coefficient scanning method. The level information and the number of zeros which run before each non-zero are determined during the scanning phase in order to reduce the number of clock cycles. Hsia *et al.* [11] propose a direct forward algorithm instead of backward tracking.

The paper is structured as follows: Section 2 presents details of CAVLC, and in particular, suggested adaptations of computationally and resource demanding phases. The details of implementation of both encoding and packing phases are presented in Section 3. The overall supporting SoC architecture for testing, logic utilization and performance analysis are presented in Section 4. Finally, the conclusions are given in Section 5.

2. CAVLC Encoding in H.264/AVC Standard

Entropy coders in video compression standards convert a series of elements of video sequence such as transform coefficients, headers or motion vectors into bitstream suitable for transmission or storage. The CAVLC coding is based on the common Variable Length Coding (VLC) method which defines a codebook by assigning a code to each symbol. Average size of each coded symbol can be minimised by assigning shorter codes to the frequent symbols. A variation of VLC which introduces context-based adaptivity is defined in H.264/AVC standard. Context-based adaptivity introduces strong inter-symbol dependency and lim-

its the use of parallelism in the encoder implementation. The data dependency exists among the coefficients on the 4×4 level, but also within the neighboring sub-blocks. The relationship between an actively encoded block and previously coded blocks is defined based on current block statistics. The CAVLC encoding process is partitioned into three phases: pre-processing including block scan and generation of flags and parameters, syntax element encoding and bitstream formation.

After the transform and the quantization stages, high-frequency regions typically contain coefficients with low values, whereas the levels of non-zero coefficients tend to be larger towards the low-frequency region. The zig-zag scan order proposed by the standard tends to group significant coefficients around DC coefficient as presented in Fig. 1. The coding parameters are extracted by backward tracking from the vector of coefficients obtained by zig-zag reordering.

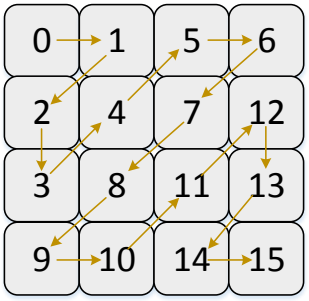

Figure 1: Zigzag scan in H.264/AVC

The five syntax element codewords are defined by the standard as follows:

- ***CoefToken*** - Encoding of the number of total coefficients *TotalCoef* and the number of high-frequency +/- coefficients also called 'Trailing 1s' (*NR_T1*),

- ***Sign*** - Encoding of the sign of each *NR_T1*, where signs of *NR_T1* elements are coded with a single bit in reverse order from the highest frequency in *NR_T1*, and the length of *Sign* codeword is *NR_T1*.

- ***Levels*** - Encoding of the levels of the remaining non-zero coefficients.

  Levels are values of non-zero coefficients. Encoding of this phase is context-adaptive since the successive level coding depends on the magnitude of the previously coded level.

- ***TotalZeros*** - Encoding of the total number of zeros before the last coefficient.

  Total number of zeros, *TotalZeros*, is the sum of zeros preceding the highest non-zero coefficient in the scan order array.

- ***Run_before*** - Encoding runs of zeros, where the codeword is determined by the number of zeros in between two consecutive non-zero coefficient together with the number of remaining zeros in the vector.

Parameter *TotalCoef* can take values in a range $[0-16]$, whereas *NR_T1* is in the range $[0-3]$. Standard defines that if there are more than three trailing values, only last three are considered as *NR_T1*, and the other coefficients are coded as normal coefficients. From the implementation viewpoint, *CoefToken* is obtained from 2D variable-length code tables inherited from variable length coding in MPEG-2 based on the number of non-zero coefficients and on the number of trailing coefficients. There are four context-dependent NUM_VLCx look-up tables for *CoefToken* codeword generation for Luma components (three variable-length code tables and one fixed-length code table). The choice of a table is determined by statistics of the neighboring blocks and depends on a number of non-zero elements in the left and upper coded blocks.

The vector *Levels* is built as follows:

$$Code_{Levels} = [\underbrace{0 \cdots 0}_{p\_length} \;\; 1 \;\; \underbrace{x \cdots x \;\; s}_{s\_length}], \tag{1}$$

where s is the sign of the coefficients in the vector *Levels*. The string of zeros followed by a stop bit '1' is *Prefix*, whereas the sequence of bits following the stop bit is *Suffix* and the sign of the coefficient. The number of bits for *Prefix* and *Suffix* are given by parameters *p_length* and *s_length*. There are defined

seven tables Lev_VLCx for coding levels syntax elements selected by an update step specified by parameter N and threshold values defined for each table.

The *TotalZeros* parameter counts the number of zeros in between the first non-zero coefficient and last non-zero coefficient. Depending on parameters $TotalCoef$ and $TotalZeros$ parameters, *TotalZeros* codeword is encoded using TZ_VLC look-up table - $\mathbf{TZ\_VLC}(TotalCoef, TotalZeros + 1)$.

The final codeword sequence, Run_before, is derived from a look-up table $\mathbf{RB\_VLC}(1 + Run(i), Zeros\_par)$, where the parameters *Run* and *Zeros_par* are the number of zeros between each non-zero coefficient, and the number of embedded zeros yet to be encoded until the last non-zero element, respectively.

3. Implementation of the CAVLC Entropy Encoder

The block diagram of the proposed CAVLC encoding system architecture is given in Fig. 2. The zig-zag reorder scan module receives sixteen coefficients corresponding to the 4×4 quantized residual sub-block as inputs and performs coefficient reordering within one clock cycle. A number of flags required for the encoding process of the syntax elements are generated:

- *Flag Nonzero* - Non-zero elements,
- *Flag Ones* - Coefficients with value ± 1,
- *Flag Sign* - Sign of the coefficients,
- *Flag FirstNonzero* - First nonzero element in reverse order,
- and *Flag TR1* - Trailing ones.

And a number of parameters used in the syntax element encoding phase are computed based on the set flags, such as:

- *Nonzero* - Non-zero coefficient vector,
- *TotalCoef* - Number of non-zero coefficients,

- *NR_TR1* - Number of trailing ones,
- and *TotalZeros* - Number of zeros embedded in between tail nonzero coefficients.

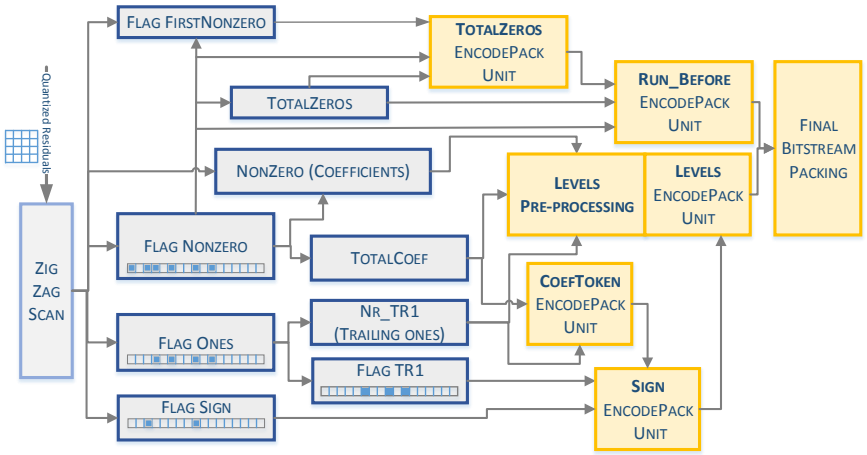

Figure 2: Block diagram of the proposed CAVLC encoder implementation

The traditional CAVLC encoder uses the encoding tables such as NUM_VLC, Lev_VLC, TZ_VLC and RB_VLC tables defined by the standard for encoding different syntax elements. The EncodePack module presented in Fig. 3 is the basic unit for syntax element generation in the proposed implementation. The module consists of look-up table or computational unit in the initial phase which are accessed and enabled by input parameters produced in pre-processing stage. The codeword and its length, computed in this initial stage or read from look-up table, are input to Barrel shifter which sends out the partial bitstream and its length for further packing.

The *EncodePack* unit for *CoefToken* syntax element is connected in pipeline to *Sign* generation module. The block diagram is presented in Fig. 4. After the sign codeword is generated by examining the trailing one and sign flags, length of sign codeword is added to the length of *CoefToken* codeword, and the sum is

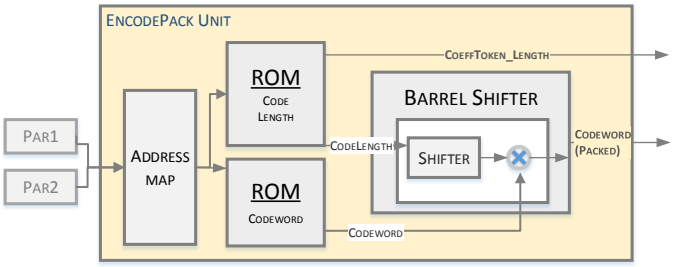

Figure 3: Block Diagram of Proposed Syntax Element Encoding and Packing Module

used for shifting operation in Barrel Shifter within *Sign* module. The encoding and packing module for *Sign* codeword includes the concatenation of two syntax elements at its final stage.

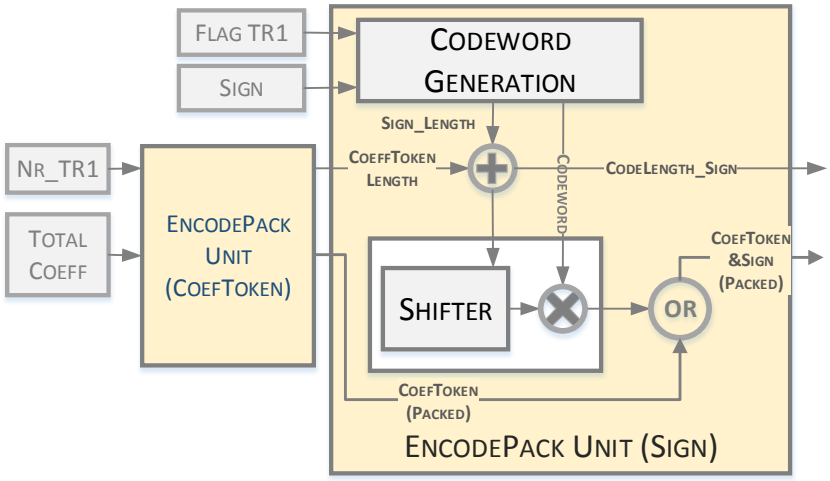

Figure 4: Block Diagram of *CoefToken* and *Sign* Syntax Element Encoding and Packing stages

150 Syntax elements *Levels* generation is computationally intensive and demanding due to data dependencies on the sub-block level. The complete block diagram of *Levels* codeword generation module is illustrated in Fig. 5. The level

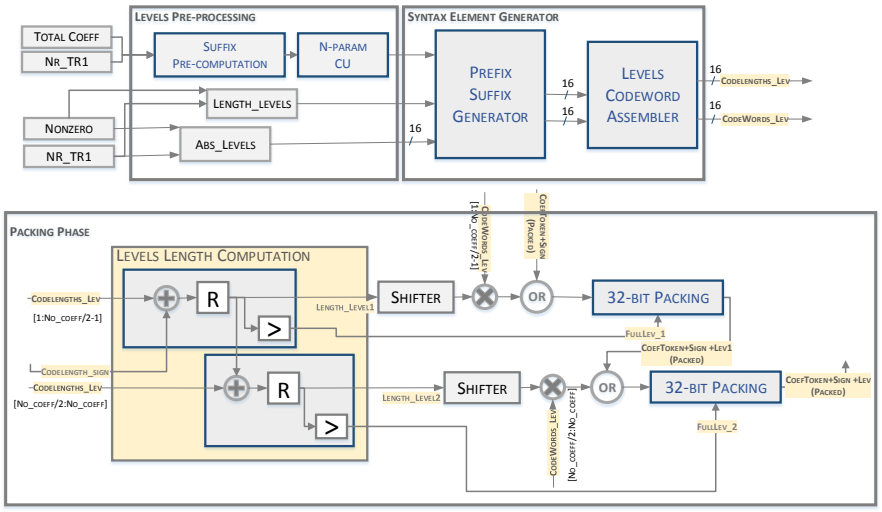

Figure 5: Block Diagram of Levels Pre-processing, Syntax Element Encoding and Packing stages

pre-processing is initialized by marking level coefficients, suffix pre-computation and initialization of N parameter. The selection step N initialized to zero ($N = 0$), corresponding to the look up table Lev-VLC0. The exception is the case when the total number of coefficients is higher than 10 while the number of trailing ones is lower than three, specifying the starting look-up table to be Lev-VLC1. The parameter *level_code* for each non-zero elements (*Levels*) is produced as part of *Prefix* and *Suffix* generation unit as follows:

$$level\_code(i) = |Levels(i)| << 1 - 2 + s. \tag{2}$$

where $<<$ is left-shift operator.

Standard defines seven Lev-VLCN tables for *Levels*. Techniques for reducing the size of used memory such as Arithmetic Table Expression (ATE) can substitute the use of tables by detecting relations between codewords in a table and by the use of arithmetic operations in order to reconstruct the data of interest from the table. Pseudo code of ATE technique for eliminating Lev-

VLCN tables proposed in this work is presented in Algorithm 1. In the proposed elimination technique, there are three defined subsets for computing *Prefix* and *Suffix* sequences for building *Levels* codewords. These subsets are chosen based on values of selection step parameter N and the value of $level\_code$. Furthermore, the selection step N with the defined range $0-6$ is incremented for every coefficient of *Levels* vector by comparison to the given threshold condition:

$$Threshold = (|Levels(i)| > (3 \ll (N-1)) \text{ and } N < 6). \tag{3}$$

The packing phase includes concatenation of generated *Levels* codewords. Update step of parameter N limits the parallelism of this phase. In order to avoid large critical path delays, by splitting the $Levels$ vector in two sub-vectors, an additional clock cycle is introduced and *Levels* codewords are generated in two consecutive clock cycles. The packing stage introduces 32-bit alignment of the partial bitstream. The codewords $code\_LEV(i)$ from two sub-vectors are concatenated and the summation of their lengths $length\_LEV(i)$ are given by:

$$code\_LEV = code\_LEV(1)\&...\&code\_LEV(M) \tag{4}$$

$$length\_LEV = \sum_{i=1}^{M} length\_LEV(i) \tag{5}$$

where M is the number of $Levels$ elements. If the length is larger than a 32-bit packet, the $Full\_flag$ is raised and a 32-bit packet is formed and forwarded to the final packing stage.

The last two syntax elements, *TotalZeros* and *Run_Before*, are processed as presented in Fig. 6. The generation of *TotalZeros* is performed by EncodePack unit with input parameters *TotalCoef* and *TotalZeros*. The pseudo-code for encoding *Run_before* code word is presented in Algorithm 2, where the vector *Run* contains the information of the locations of the embedded zeros in between non-zero coefficients, whereas the vector $Zeros\_par$ counts remaining zeros. The computation of vector *Run* and the investigation of nonzero significance map is performed in two clock cycles due to strong data dependencies. Variable length

Algorithm 1 Pseudo-Code for *Levels* Encoding

if $(TotalCoef > 10)$ and $(T1 < 3)$ **then**
  $N = 1$
else
  $N = 0$
end if
for $i = 1 : length(Levels)$ **do**
  $level\_code(i) = |Levels(i)| \ll 1 - 2 + s$
  if $(N = 0$ and $14 \leq level\_code(i) < 30)$ **then**
    $p\_length(i) = 14$
    $prefix(i) = zeros(p\_length(i))$
    $suffix(i)\&\mathbf{s} = Bin(level\_code(i)\%2^4)$
    $s\_length(i) = 4$
  else if $(level\_code(i) < 15)$ **then**
    $p\_length(i) = level\_code \gg N$
    $prefix(i) = zeros(p\_length(i))$
    $suffix(i)\&\mathbf{s} = Bin(level\_code(i)\%2^N)$
    $s\_length(i) = N$
  else
    $p\_length(i) = 15$
    $prefix(i) = zeros(p\_length(i))$
    $level\_code(i) = level\_code(i) - (15 \gg N)$
    $suffix(i)\&\mathbf{s} = Bin(level\_code(i)\%2^{12})$
    $s\_length(i) = 12$
  end if
  if $(N = 0)$ **then**
    $N = N + 1$
  end if
  $length\_LEV(i) = p\_length(i) + 1 + s\_length(i)$
  $code\_LEV(i) = prefix(i)$ & **1** & $suffix(i)$ & **s**
  if $(|Levels(i)| > (3 \ll (N - 1))$ and $N < 6)$ **then**
    $N = N + 1$
  end if
end for

vector generation is introduced so that information regarding number of zeros computed in the one sub-vector are synchronized with the data gathered in the second sub-vector.

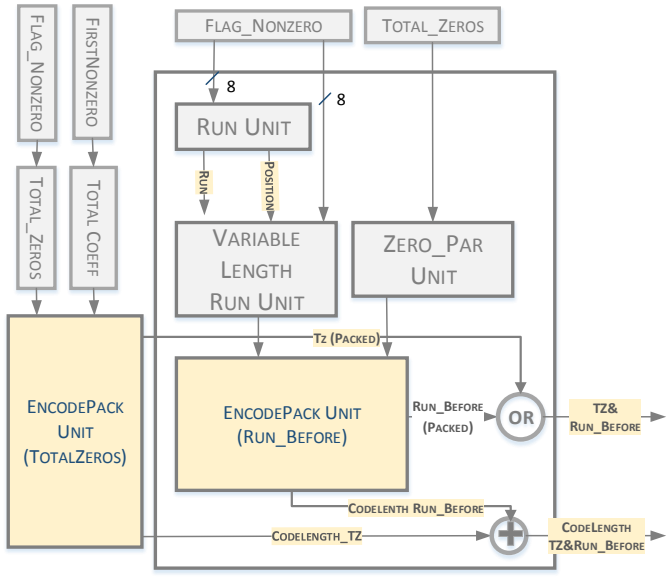

Figure 6: Block diagram of *TotalZeros* and *Run_before* codeword generation stage in CAVLC implementation

Algorithm 2 Pseudo-Code for Run_before Encoding

$Zeros\_left = TotalZeros$
for $i = 1 : Total\_Coeff - 1$ **do**
  if $(Zeros\_left > 0)$ **then**
    $Zeros\_par = min(Zeros\_left, 7)$
    $Run\_before(i) =$ **RB_VLC**$(1 + Run(i), Zeros\_par)$
    $Zeros\_left = Zeros\_left - Run(i)$
  end if
end for

3.1. Bitstream Packing Stage

In this work, parallel datapaths for each syntax element generation process are proposed. The packing process is performed in stages by following syntax elements generation. The individual syntax elements codewords are first concatenated into the partial bitstreams, whereas the complete bitstream packing is performed in the final stage. This stage includes also the byte alignment since the length of the produced bitstream for each sub-block is not constant.

There are seven Barrel shifters which are used to produce and concatenate five syntax elements. The barrel shifters *BS1* and *BS2* shift the code words *CoefToken* and *Sign* to the codeword lengths in the registers, where shifter *BS2* also simultaneously performs the concatenation of these two words. Due to the strong data-dependency in the production process of syntax element *Levels* which introduces a long critical path, the vector of the input coefficient for this syntax element is split into two parts and bitstreams for each sub-vector are generated in two consecutive clock cycles by Barrel shifters *BS3* and *BS4*. The barrel shifter *BS5* and *BS6* deal with syntax elements *TotalZeros* and *Run_before*. The generation of partial bitstream which consists of *TotalZeros* and *Run_before* codewords is performed in parallel with the creation of the partial bitstream of the first three codewords. After the data are shifted accordingly, the concatenation of two streams is performed by *BS7*.

The bitstream packing dataflow on the sub-block level proposed in this work is presented in Fig. 7. As the data on 4×4 block level is processed in parallel, the data is updated by the defined syntax elements order - *CoefToken*, *Sign*, *Levels*, *TotalZeros* and *Run_Before*.

Even though H.264/AVC defines a byte-stream format to transmit a sequence as the ordered stream of bytes in Network Abstraction Layer (NAL), the proposed implementation is characterized by 32-bit bitstream output. The final phase packing stage is presented in Fig. 8. In this phase, a 128-bits bitstream packer is proposed. This 128-bit bitstream packer concatenates variable-length codewords together and segments them into 32 bits words. The parameters to be monitored every time the data is concatenated into the bitstream registers

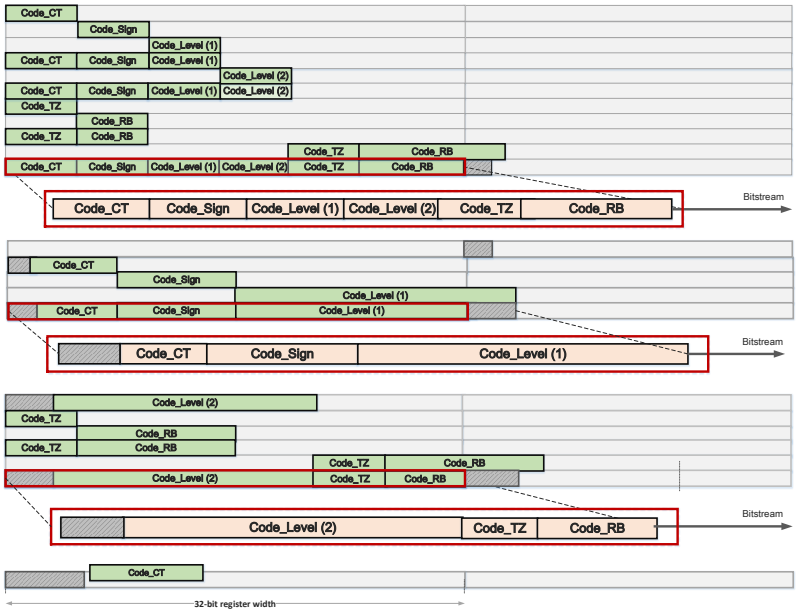

Figure 7: Dataflow of the Packing Stage of the proposed CAVLC encoder

are the number of integral 32-bit words and the number of the residual bits less than one 32-bit word. If the bitstream length does not reach 32 bits, a number of 4×4 sub-blocks are packed to form 32-bit word. When the 32-bit aligned bitstream is created, a number of 32-bit words ($Bit\_SB/32$) is sent as output and the remaining bits of the length $Bit\_SB\%32 < 32$, are forwarded as initial bits to the following sub-block packing process. The backward loop from the total sub-block bitstream register is used to concatenate the current sub-block bitstream with the residual $Bit\_SB\_R\%32$ of the previous one, since the bitstream residual and its length are sent back to the packing stage as peripheral product. The length of the residual is used as offset for the incoming sub-block bitstream, after which the produced bitstream of the active sub-block is concatenated with the residual bitstream from the previous sub-block. The flags *Full_lev1*, *Full_lev2* and *Full_Total* are inserted within the various syntax element encoding stages - *Levels* - *Bit_Lev1* and *Bit_Lev2*, and after the complete bitstream generation *Bit_Total*, respectively. These flags signal the presence of

32-bit words and the number of 32-bit chunks created for the current sub-block is known prior to the final bitstream packing stage. For example, if two out of three flags are raised, two 32-bit chunks are forwarded to FIFO register. The overview of the possible scenarios for syntax element concatenation logic behind the packing process based on raised flags is shown in Fig. 9.

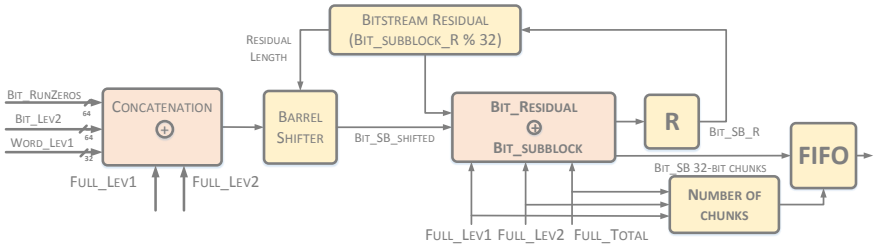

Figure 8: Block diagram of final Bitstream Packing Stage of CAVLC encoder

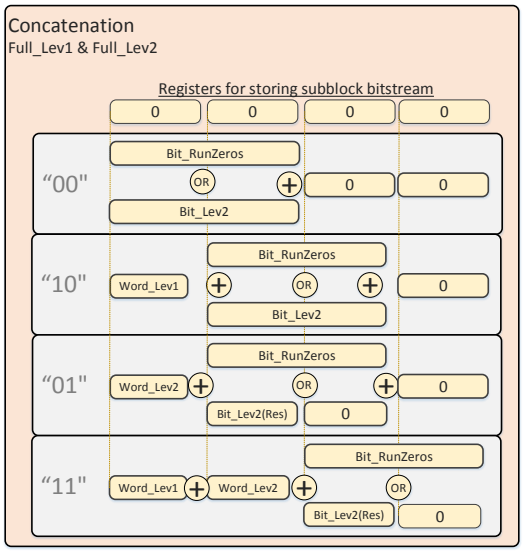

Figure 9: The concatenating process of bitstream on 4×4 sub-block level

4. Results

The proposed CAVLC entropy encoder module is implemented on Kintex-7 FPGA and the architecture is designed in the VHDL language. In the state of the art, the performance of entropy encoders clearly depends on the parameter configurations as the value of the parameter QP significantly impacts the performance. In the proposed implementation, however, the number of clock cycles for the encoding and packing stages is constant. The challenges met in order to achieve high operating frequency and limit the clock cycles are related to the long critical path in the *Levels* syntax encoding and packing stages due to the strong data dependency. The solution is found by splitting the computation over two clock cycles. Additional time savings can be achieve by skipping the partial bitstream *Bit_Lev1* in *Levels* encoding when it contains no nonzero elements.

The parallel creation of syntax elements can be performed but it requires an additional effort since the syntax element encoding generation process complexity and consequently processing time vary for each codeword. For example, *CoefToken* requires only one clock cycle for pulling codeword from the look-up tables, whereas only pre-processing stage of *Levels* codeword takes 5 clock cycles. In the proposed implementation, the concatenation of three syntax elements *CoefToken*, *Sign*, and two stages of *Levels* (*Lev1* and *Lev2*) is done after codeword generation processes for each syntax element. The partial bitstream is created after 10 clock cycles due to the long *Levels* codeword generation process. The creation and concatenation of *TotalZeros* and *Run_before* codewords are performed in parallel creating the second partial bitstream. After both partial bitstreams are created, the full sub-block bitstream is formed in the final packing stage.

The broader perspective on the timing with respect to both syntax element computational dependency and the processing order for the complete encoding and packing stages is given in Fig. 10. The initial latency of the CAVLC encoder is 12 clock cycles, after which the output data for each sub-block can

be streamed into the FIFO for each clock cycle. The number of 32-bit chunks ranging from 0-3 can be produced for a current sub-block. However, the data packing (number of chunks) and the compression performed in the previous quantization phase dictate the output rate of the FIFO. Higher Quantization Parameter (QP) sends less information to the entropy encoder, implying lower number of chunks produced in the encoding and packing stages.

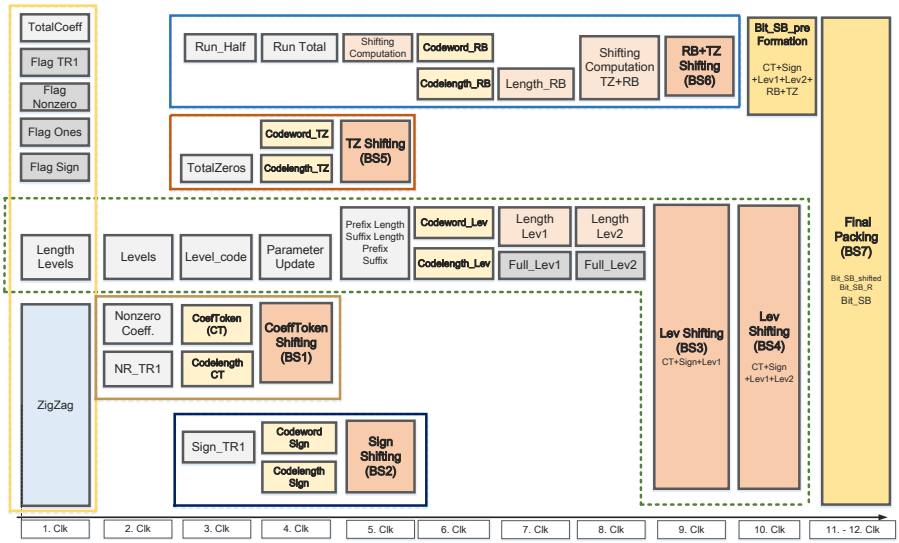

Figure 10: Complete Time diagram of the Encoding and the Packing stages of the proposed CAVLC encoder

Table 1 compares the proposed architecture to other selected FPGA and most recent ASIC solutions. Various information such as operating frequency, achieved throughput for average (worst) case in clk. cycles/MB and Mpixels/s, as well as the logic utilization and the technology are included in the comparison. State of the art CAVLC hardware implementations focus on the speed-up of syntax element generation and the packing stage has not been subjected to elaborated research or analysis in most of the cases.

Table 1: Comparison to the state of the art

	Ramos[6]	Licciardo[7]	Hoffman[8]	Hsia [11]	Proposed
Average case [clk.cycles]	244	172	141	200	16
Worst case [clk.cycles]	280	256	258	400	16
Operating Freq. [MHz]	180	80	200	125	110
Throughput [Mpixels/s]	188(164)	119(80)	363(193)	160(80)	1760
Slices	-	-	-	-	2352
FF	-	-	-	-	1485
LUTs	10791	1408	6549	-	6206
RAM [Kbits]	-	-	-	-	38.2
Gates	-	-	-	15K	-
Min. Freq [MHz]	180	80	200	125	110
Technology	Virtex 5	Virtex 5	Stratix I	0.18V	Kintex

In the syntax element encoding, scan phase is reported as a major limitation. The dual-scanning order proposed in [8] and [11] provides advances in achieved throughput but the efficiency of the implementation is highly dependent on quantization process and QP value. The reduction of overall clock cycle numbers for encoding process in [7] achieved by performing scan operation in one clock cycles led to long critical path and low clock speed. In the proposed work, the architecture efficiently decouples and pipelines the critical stages to address the bottlenecks in syntax element encoding phase. Our solution proposes the complete residual vector scanning in one clock cycle and the generation of two parallel syntax element encoding streams in order to increase the encoding speed on the sub-block level. Splitting the vector in two only for critical data-dependent stages such as *Levels* or *Run_Zeros* codewords encoding stages, lowers the critical path and enables high bitrates.

The proposed implementation outperforms the the state of the art FPGA implementations in terms of processing speed for high bitrate video content. The trade-off between high input parallelism and operating frequency gives 1760 Mpixels/s throughput which allows processing of 4320p resolutions at frame rate of 30 frames/s. The comparison on minimal frequency required for high

definition video content with other recent FPGA and ASIC implementations is given in Table 2, where it shows superior results compared to the state of the art.

Table 2: Minimal Frequency Comparison in State-of-the-Art Implementations

Min. Frequency [MHz]	1080@30	2160@30	4320@30	Max. Supported Resolution
[6]	59.3	-	-	1080@60
[7]	41.8	-	-	1080@30
[8]	34.3	137.2	-	2160@30
[11]	48.6	-	-	1080@60
Proposed	3.8	15.5	62.8	4320@30

5. Conclusion

This paper presents a fast high-throughput implementation of CAVLC encoder in H.264/AVC. The architecture is characterized by the balanced computational load within each clock cycle leading to the low clock cycle count for producing bitstream on the 4×4 sub-block level. The work targets compact syntax element computation with reduced requirements for memory access, and furthermore, an efficient data packing which enables a pipeline implementation of the complete entropy encoding process, where the incoming 4×4 sub-block data is processed in each clock cycle. The implementation can process 4320p UHD video sequences at 30 frames/s. The resulting processing time of 16 clock cycles for macroblock processing is significantly lower when compared with the state of the art of CAVLC encoder hardware implementation.

References

[1] J. V. Team, Advanced video coding for generic audiovisual services, ITU-T Rec. H 264 (2003) 14496–10.

[2] G. Bjoontegaard, K. Lillevold, Context-adaptive VLC (CAVLC) coding of coefficients. doc. JVT-C028, JVT of ISO MPEG&ITU VCEG, in: 3rd Meeting, Rairfax, Virginia, USA, 2002.

[3] D. Marpe, G. Blättermann, T. Wiegand, Adaptive codes for H.26L, ITU-T Telecommunications Standardization Sector (2001) 9–12.

[4] Y. H. Moon, G. Y. Kim, J. H. Kim, An efficient decoding of CAVLC in H.264/AVC video coding standard, Consumer Electronics, IEEE Transactions on 51 (3) (2005) 933–938.

[5] C.-C. Lo, S.-T. Tsai, M.-D. Shieh, Reconfigurable architecture for entropy decoding and inverse transform in H.264, Consumer Electronics, IEEE Transactions on 56 (3) (2010) 1670–1676.

[6] F. L. L. Ramos, B. Zatt, T. L. Silva, A. Susin, S. Bampi, A high throughput cavlc hardware architecture with parallel coefficients processing for hdtv h. 264/avc enconding, in: Electronics, Circuits, and Systems (ICECS), 2010 17th IEEE International Conference on, IEEE, 2010, pp. 587–590.

[7] G. D. Licciardo, L. F. Albanese, Design of a context-adaptive variable length encoder for real-time video compression on reconfigurable platforms, Image Processing, IET 6 (4) (2012) 301–308.

[8] M. P. Hoffman, E. J. Balster, W. F. Turri, High-throughput cavlc architecture for real-time h. 264 coding using reconfigurable devices, Journal of Real-Time Image Processing (2013) 1–8.

[9] T.-C. Chen, Y.-W. Huang, C.-Y. Tsai, B.-Y. Hsieh, L.-G. Chen, Architecture design of context-based adaptive variable-length coding for h. 264/avc, IEEE Transactions on Circuits and Systems II: Express Briefs 53 (9) (2006) 832–836.

[10] T.-H. Tsai, S.-P. Chang, T.-L. Fang, Highly efficient cavlc encoder for mpeg-4 avc/h. 264, IET circuits, devices & systems 3 (3) (2009) 116–124.

[11] S. C. Hsia, W. H. Liao, Forward computations for context-adaptive variable-length coding design, IEEE Transactions on Circuits and Systems II: Express Briefs 57 (8) (2010) 637–641.

[12] C. W. Chang, W. H. Lin, H. Yu, 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: 2014 IEEE International Symposium on Circuits and Systems (ISCAS), IEEE, 2014, pp. 2616–2619.