An Efficient Context Adaptive Variable Length Coding Architecture for H.264/AVC Video Encoders

Ngoc-Mai Nguyen, Xuan-Tu Tran
VLSI Systems Design Group, SIS Laboratory
VNU University of Engineering and Technology
144 Xuan Thuy road, Hanoi, Vietnam
{mainn, tutx}@vnu.edu.vn

Pascal Vivet, Suzanne Lesecq
CEA-LETI, MINATEC
17 rue des Martyrs
Grenoble, France
{pascal.vivet, suzanne.lesecq}@cea.fr

***Abstract* — In this paper, we present an efficient hardware implementation of a Context Adaptive Variable Length Coding (CAVLC) module for an H.264/AVC video encoder. To improve timing performance, a three-stage pipeline architecture is proposed including: input data statistical analysis, encoding and packing. The context information and coding tables are stored in memory elements. To minimize the hardware implementation overhead and increase the system performance, in some sub-encoders, the codewords are calculated on-the-fly instead of being stored in look-up tables. The proposed architecture has finally been implemented using a low power CMOS 65*nm* technology from STMicroelectronics. The design is able to operate up to 715*MHz*. At 550*MHz*, the design complexity is 33*Kgates* for a power consumption of 20*mW*. The design is initially targeted to CIF video format; however, it is obviously suitable for real-time HD 1080p video format.**

Keywords- *Video encoding, H.264/AVC, CAVLC, entropy coding, pipeline architecture, VLSI architecture for H.264 codec.*

I. INTRODUCTION

The H.264/AVC video coding standard proposed by the Joint Video Team (JVT) has introduced significant compression performance. To save approximately 50% bit-rate for equivalent perceptual quality compared with the performance of previous standards, many advanced techniques have been introduced with a large computational complexity [1]. Therefore, H.264/AVC codecs are often implemented in full hardware or hardware/software co-design.

In video compression, after being transformed and quantized [2], video data enters the entropy encoding process to remove statistical redundancy. One main idea of entropy encoding in video compression is Variable Length Coding (VLC) using shorter codewords for more frequent symbols and longer codewords for less frequent symbols. The new techniques used in H.264 are context adaptive entropy encoding. One of them is Context Adaptive Variable Length Coding (CAVLC) which is applied in the baseline and main profiles of the H.264/AVC standard. In CAVLC, many variable length coding tables are used and the table selection depends on the context of the input data.

Several previous works have presented the implementation of CAVLC encoder. Most of them tried to implement a high throughput CAVLC encoder, using pipelining techniques [3] or parallel symbols encoding [4][5]. Two-stage pipeline architecture can halve the time to process a block in average, but requires double buffer size to store all the statistic information of one block before the data is encoded. Therefore, Chien [5] and Tsai [6] introduced a mechanism that scans the coefficients in inversed zigzag order and proposed a special buffer structure to maintain the buffer size at the size of one block.

Parallel symbols encoding is an efficient method in terms of performance, however, it obviously double the area cost of symbol encoders. This method also needs to handle the data dependency, that is, with two symbols encoded in parallel, the encoding of the later symbol depends on the previous one. Thus, the statistical information of both symbols is pre-calculated before encoding process [5]. In [4], the later symbol is encoded in two parallel encoders, then, the results are selected based on the output of the previous symbol encoder. This solution triples the hardware cost of the symbol encoder. Ramos et al. [7] presented an efficient method to overcome the bottleneck at the scan phase by scanning coefficients in parallel. This method halves the required time of the scan phase. Parallel coding of level and run-before is also applied.

Some other authors aimed at designing low cost CAVLC encoders by calculating the coding level variables on-the-fly [8][9] or by reducing the size of look-up tables [5][10]. In [10], instead of storing 16-bit word for each symbol in coeff_token encoding, Kim has modified the VLC tables into tables of 9-bit words. Finally, Tsai [6] has implemented a low power CAVLC encoder which can reduce up to 69% the power consumption but the total gate count is somewhat high (about 27*Kgates* without the implementation of coefficient token VLC table selector).

In this paper, we propose an efficient hardware implementation of the CAVLC encoder for H.264/AVC video coding. In order to achieve high performance encoding, we propose a three-stage pipeline architecture using various techniques to reduce the number of codewords to be packed into the bit-stream in the last stage. Zero-skipping technique is intently used at 8×8 block level to skip encoding the all zero blocks. This technique reduces the encoding time for low-bit rate video data where lots of coefficients are zero. The VLC table selector for encoding coeff_token, including the reference memory, is integrated in our design to minimize the load of the H.264/AVC global processor. Finally, in order to reduce the cost of the table selector and its associated memory area, two main techniques are applied: to re-encode the VLC tables and

to calculate the codewords arithmetically. For example, the codewords are calculated on-the-fly instead of being stored in look-up tables at the level encoding and run_before information encoding. The proposed CAVLC architecture has been fully implemented in RTL and synthesized using a low power CMOS 65*nm* technology from STMicroelectronics.

The rest of the paper is organized as follows. In Section II, the basic principles of CAVLC coding is briefly reviewed. Section III presents in detail the proposed CAVLC encoder pipeline architecture. In Section IV, we discuss the achieved performance in comparison to related works. Finally, conclusion will be given in Section V.

II. CAVLC ENCODING PRINCIPLES

A. Main principles of CAVLC

CAVLC is entropy encoding used to encode residual data in 4×4 or 2×2-coefficient blocks. The coding techniques are applied according to statistical characteristic of the block:

The blocks of quantized transform coefficients contain mostly zero coefficients. Thus, run-length coding is applied to encode the zero strings. In the zigzag order, non-zero coefficients are gradually lower. The last non-zero coefficients of the blocks are normally +1 or −1. The last consecutive coefficients with magnitude 1 are called trailing ones. The maximum number of trailing ones is three. These coefficients are encoded in a special way. The other non-zero coefficients are called "level". Levels are encoded in inversed zigzag order because in this inversed order, the magnitude of the previous level is used to predict the value of the next level. This prediction is then used to select the appropriate coding table. Because the numbers of non-zero coefficients in neighboring blocks are correlated, these numbers in upper and left blocks are used to predict the current block [11].

The ITU-T recommendation [12] has introduced the five syntax elements to be encoded in the CAVLC. They are coefficient token (coeff_token), trailing ones' signs, level, total zero, and run before.

B. Coefficient Token encoding

Coefficient token (coeff_token) is a syntax element presenting a pair of numbers: the number of non-zero coefficients and the number of trailing ones in a block. The VLC table selection depends on the number of non-zero coefficients in the upper and the left blocks. If the number of non-zero coefficients in the upper block is nU and the number of non-zero coefficients in the left block is nL, then, the parameter nC used to decide which VLC table will be selected is calculated as follows:

$$nC = ceiling(\frac{nU + nL}{2})$$

The coeff_token of a 4×4 luma block is encoded using the table VLC0 if nC is less than 2. The table VLC1 is selected if nC is greater than 1 and less than 4. If nC is greater than 3 and less than 8, the VLC2 is selected. If nC is greater than 7, the Fixed Length Coding (FLC) table is used. We also have a special table for encoding the coeff_token of 2×2 chroma DC blocks.

C. TrailingOnes Sign flag encoding

Each TrailingOne is encoded in one bit presenting its sign. If the coefficient is 1, the sign bit is zero ('0'). If it is −1, the sign bit is one ('1'). The TrailingOne signs are encoded in the bitstream in inversed zigzag order.

D. Level encoding

Levels are encoded in inversed zigzag order. In the level encoding, seven VLC tables are used. The next VLC table is selected according to the current VLC table and current magnitude of level. The first level is encoded using VLC0. Then the VLC number is increased if the magnitude of the current level is larger than the correlate threshold of the current VLC. TABLE I shows the thresholds of VLC tables.

One exception is that if there are more than 10 nonzero coefficients and less than three trailing ones, the first level will be coded using table VLC1. Other exception is that if there are less than three trailing ones, the first level coded with the magnitude is decreased by 1 as mentioned in [12].

TABLE I. THE THRESHOLDS TO INCREASE THE VLC NUMBER

Current VLC table	Threshold
Level_VLC0	0
Level_VLC1	2
Level_VLC2	3
Level_VLC3	12
Level_VLC4	24
Level_VLC5	48
Level_VLC6	-

The ITU-T recommendation [12] introduces the level encoding using three variables: suffix length, level prefix and level suffix. The coding tables can be found in the JVT document JVT-C028 [13]. From this presentation of VLC tables, we can figure out the formats of the codeword in three cases: one general case and two escape code cases. Signed level is converted into unsigned code-number.

- In general case, a codeword contains prefix and suffix parts. The prefix is a string of zero bits followed by one '1'. Suffix length is equal to the VLC number. The suffix value is equal to code-number minus the number of zero in the prefix. The maximum prefix is 13 in VLC0. In the baseline and main profile, maximum prefix is 14 in the other VLC tables [1].
- If the VLC number is VLC0 and the code-number is greater than 13 and smaller than 31, the codeword is in the first escape format: prefix is equal to 14, suffix length is 4.
- If in VLC0, the code-number is greater than 30, or in the other VLC tables, the code-number is greater than maximum prefix plus maximum suffix, the second escape format is applied: the prefix is 15; the suffix length is 12 in baseline and main profile.

E. TotalZero encoding

TotalZero is the number of zero coefficients standing before the last non-zero coefficient in the zigzag order. Total zero is encoded using 15 VLC tables selected by the number of non-

zero coefficients in the luma blocks. Three other tables are used for encoding chroma DC blocks.

F. Run_Before encoding

Run_before is a sequence of numbers of zero coefficients standing before levels in zigzag order. However, run_befores are encoded in inversed zigzag order. The VLC table selection is done based on "zero left" information, that is, the number of zero left after each run_before is encoded. Next zero left is equal to current zero left minus current run_before. There are 7 VLC tables used for run_before encoding.

Finally, the syntax structure of output bitstream for one block data is in the following order: coeff_token, TrailingOne signs, Levels, TotalZero, and run_before. A conventional CAVLC encoder is composed of five encoders to generate five syntax elements of the current block. Each encoder contains several look-up tables to store VLC tables. Table selection is done by the previous coded syntax elements with some exceptional cases. There are totally up to 41 VLC tables in the CAVLC encoder.

III. THE PROPOSED CAVLC ARCHITECTURE

In this section, the proposed architecture of the CAVLC encoder will be presented. Pipelining and other techniques are used to achieve high performance. In order to reduce the hardware implementation cost, Level and run_before are encoded using dedicated combinational logic circuits while coeff_token and TotalZero are encoded by using look-up tables containing re-encoded VLC tables.

A. CAVLC pipelined architecture overview

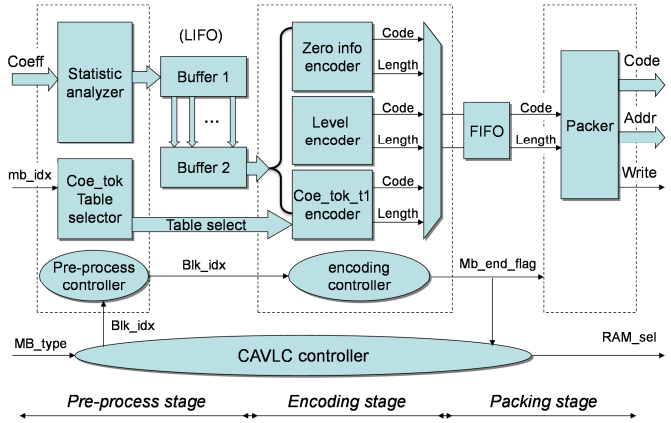

Figure 1. The proposed three-stage pipeline architecture of CAVLC encoder.

Figure 1 illustrates the three-stage pipeline architecture of our proposed CAVLC encoder design. The pre-process stage scans the sixteen (or four) coefficients, and then analyzes the statistical characteristic of the input blocks. In this stage, the coeff_token table selector is also integrated. This sub-module stores the reference information of the neighboring blocks and calculates the parameter *nC* to select the coding table for coeff_token encoding. In the encoding stage, various encoders operate in parallel to encode syntax elements of the current block at the same time. The three encoders (coe_tok_t1, level and zero info encoders) generate codewords in 32-bit bus and code length in 5-bit bus. The encoding controller synchronizes these three encoders, controls output data flow according to the syntax structure of the output bitstream. Finally, the packing stage concatenates the codewords and aligns them into 32-bit words. These words are written into an outer memory block.

The pipeline between pre-process and encoding stages is performed at block level. Due to data dependency, encoders start to encode a block after the scanning process is complete. Two buffers are implemented between two first stages. By doing this way, two blocks can be processed in parallel, one is being scanned, while the previous one is being encoded, thus increasing the throughput of CAVLC module.

The pipeline between two last stages is realized at codeword level. Each codeword generated by the encoder is pushed into the FIFO. Whenever there is a codeword in the FIFO, the packing stage pops it and concatenates with the previous codes. Therefore, the codewords can be processed in parallel; one is being encoded while the previous one is being packed.

The packing stage will stop packing and write out the remaining data of a macroblock into the outer memory when the mb_end flag is set. Hence, the architecture of the packing stage is simple.

B. Reduce number of codewords entering the packing stage

The preprocessing stage scans the input coefficients and stores the statistic information in the first buffer. After the scanning is complete, all the information such as number of non-zero coefficients, number of trailing ones, levels, run before…, is copied into the second buffer. During this copy process, the signs of three first levels are extracted and written into the second buffer (see Figure 2). Using the number of trailing ones, the encoding stage can construct the syntax element trailing one sign flag in just one clock cycle.

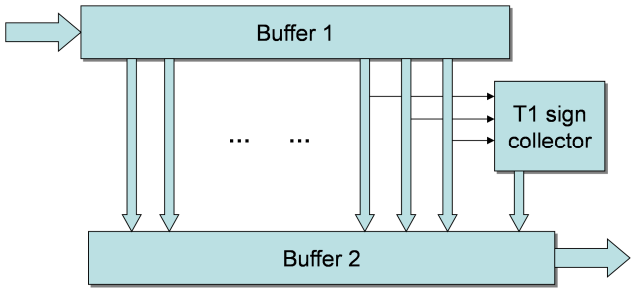

Figure 2. T1 signs extraction during buffer copy.

In the encoding stage, the coeff_token and TrailingOne signs are 19 bits long at maximum. In one clock cycle, the two syntax elements are merged into only one codeword pushed into the FIFO to reduce the task of packing stage, thus increases the throughput of the module. Figure 3 presents the encoding of coeff_token and the combining of two syntax elements into one.

In a block, the number of non-zero coefficients and the number of run_befores are equivalent. Although run_befores and levels are encoded simultaneously, the time for the encoding stage to encode and push all the syntax elements into the FIFO is still twice time greater than the time for encoding levels.

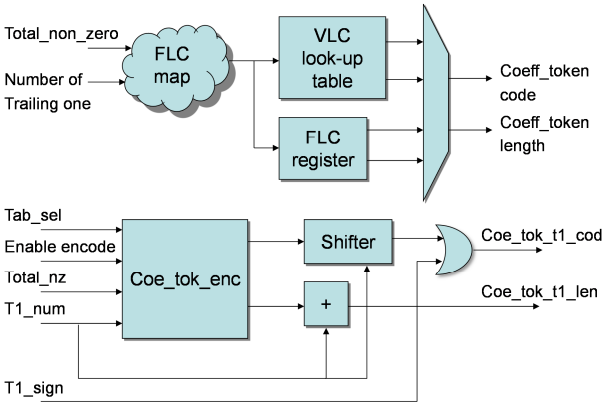

Figure 3. Coeff_token and T1 signs encoding into one codeword.

Besides, the total number of bits of TotalZero and run_befores after being encoded is less than 32 bits. Thus, we can solve the timing process problem by packing all the zero information of a block into a 32-bit codeword while the run_befores are being encoded. By this solution, we reduce the number of zero information codewords entering the packing stage into only one codeword. Figure 4 describes the position and architecture of the zero information packer (small packer) in the zero information encoder.

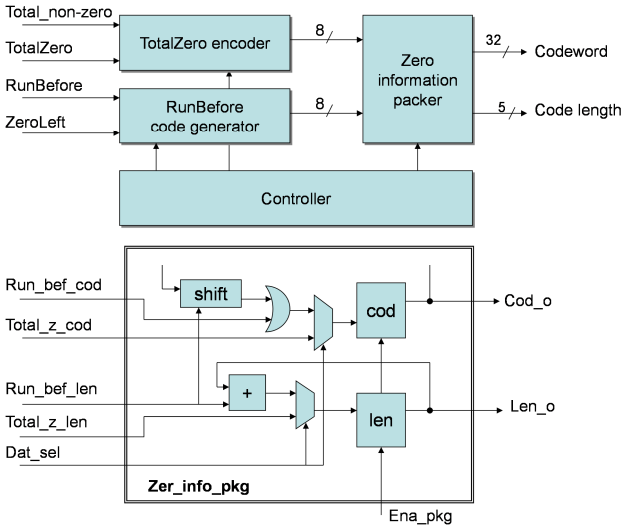

Figure 4. Position and architecture of the zero information packer.

C. Zero-skipping at block level

Another method to enhance the performance of the CAVLC encoder is zero-skipping where all residual zeros are flagged in order to skip the complete encoding process. Chen [3] and Chien [5] adopt zero skipping at 8×8-block level, using Coded Block Pattern. Then, in the following work [6], Chen has applied zero skipping at coefficient level to achieve high throughput and low power for CAVLC encoding.

In our design, zero skipping is also used at 8×8 block level. Before being scanned in pre-process stage, an 8×8 zero block is detected in zigzag scan phase. If the flag of 8×8 zero block is set, the encoder needs only two clock cycles to store the information into the reference memory, all the other encoding processes are then omitted. At 4×4 block level, zero blocks are also detected and flagged by zigzag scanner to reduce tasks of the CAVLC encoder. When a 4×4 block is flagged as zero block, the pre-processing does not have to re-scan it and coeff_token encoder is the only encoder operating in the encoding stage.

D. VLC table selector for coeff_token encoding

As mentioned above, the coeff_token VLC table selector including the reference memory is implemented in the pre-process stage. Without this integration, the global processor of the H.264 encoder would have to access the global memory three times per block to read the *nL*, *nU* and write back the number of non-zero coefficients of the current block. Therefore, with the VLC selector integrated in the CAVLC, the global performance of the H.264 encoder is increased.

However, the hardware cost of this table selector is very high. The work presented in [14] is known as the only design including the *nC* generator in the CAVLC encoder. The *nC* generator is reported to occupy more than 50% of the total hardware overhead of CAVLC encoder.

The significant cost of this sub-module leads us to optimize the hardware cost of the design. For blocks whose lower neighbors are in the same macroblock, reference memory is reused after each macroblock to be encoded. For the other blocks, the reference memory is reused line-by-line. The data stored in the reference memory is the number of non-zero coefficient in blocks which is in range 0 to 16. According to the CAVLC principle mentioned above, *nL* (or *nU*) equal to 16 or 15 yield different *nC* parameters, however, the VLC table decisions are the same. Indeed, the ceiling of $((15 + n)/2) \geq 8$ and the ceiling of $((16 + n)/2) \geq 8$, the FLC table is selected in both cases. To reduce the number of bit stored in reference memory from five to four bits, we store all data equal to 16 as 15 with no influence on the table selection. In other parts of the CAVLC encoder, the two main techniques are codeword calculation and VLC table re-encoding.

E. Codeword arithmetic calculation

Every logic/arithmetical relation between the input data and the output codeword is utilized. In the coeff_token encoding, the fixed length coding (FLC) table can be constructed by forming the codeword as: 4 bits presenting the number of non-zero coefficients minus 1 followed by 2 bits indicating the number of trailing ones. Hence, we remove the FLC look-up table. To reduce the area of address generator, the calculated FLC codeword is used as the address to access the VLC tables.

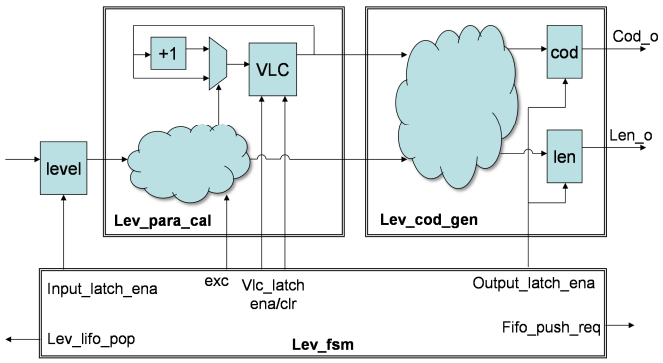

Figure 5. Architecture of level encoder.

In level encoding, the arithmetical relation between code-number and codeword is also exploited. The architecture of level encoder is presented in Figure 5. The lev_para_cal calculates two parameters: VLC number and code-number for each level. The lev_cod_gen generates the codeword and code length in one of three formats presented above. All the computations are arithmetic and logical. We even optimized some of the mathematical equations to have a circuit with minimum resource.

For example, the circuit in Figure 6 calculates the parameter code-number. The original relation between code-number and level is:

$$\text{Cod\_num} = \begin{cases} |\text{level}| \times 2 - 2 \text{ if level} > 0 \text{ and no exception} \\ |\text{level}| \times 2 - 1 \text{ if level} < 0 \text{ and no exception} \\ (|\text{level}| - 1) \times 2 - 2 \text{ if level} > 0 \text{ and exception} \\ (|\text{level}| - 1) \times 2 - 1 \text{ if level} < 0 \text{ and exception} \end{cases}$$

The relation is simplified as follows.

- If level > 0 and no exception, $\text{cod\_num} = (\text{level} - 1)$ concatenates with '0'.
- If level < 0 and no exception, then $\text{cod\_num} = (-\text{level}) \times 2 - 1 = (\text{inverse(level)} + 1) \times 2 - 2 + 1 = \big(\text{inverse(level)}\big) \times 2 + 1$. It means that $\text{cod\_num} = \text{inverse(level)}$ concatenates with '1'.

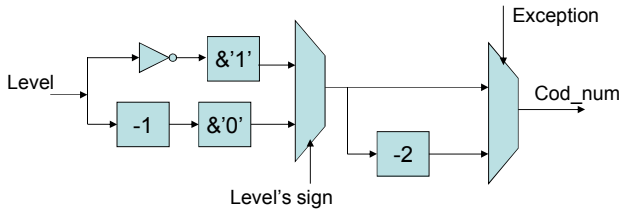

Figure 6. Code-number arithmetic expression optimization.

In run_before encoding, the codewords are also calculated to remove the look-up table for VLC in the design.

F. Re-encoding the VLC tables

The rest syntax elements are encoded using loop-up table. However, conventional look-up tables require large memory size to store the whole codewords and code length. We proposed a simple method to re-encode the coeff_token codeword into a format of length and value information as shown in TABLE II. Because the coeff_token codewords have length in range 1 to 16 and the values are in range of 0 to 15, a 8-bit word is enough to store the information of one coeff_token. The TotalZero coding table is re-encoded in the same method.

TABLE II. AN EXAMPLE OF A 8-BIT WORD IN VLC TABLES

Original codeword	**Proposed codeword**	
	Length - 1	Value
0000000000000010	1111	0010
16 bits	4 bits	4 bits

G. CAVLC controller

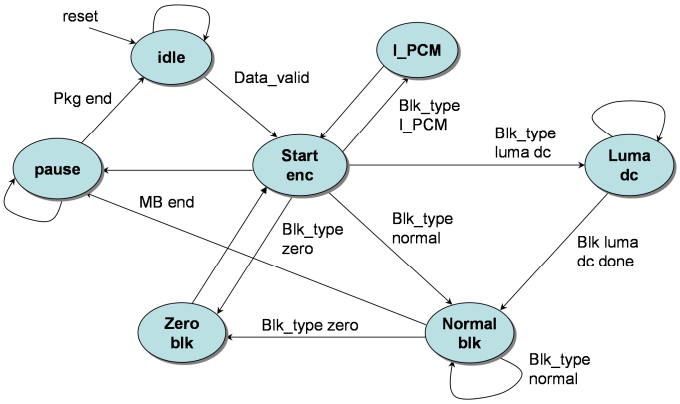

Figure 7. The main controller of the proposed CAVLC encoder.

The main controller of the proposed CAVLC encoder is illustrated in Figure 7. As the data transfer protocol between stages is defined in Figure 1, the major task of this controller is to generate input conditions for the pre-process stage and synchronize the three stages at macroblock level.

Whenever an input macroblock is available, the controller switches from "**idle**" state to "**start encoding**" (start_enc) state to start encoding a block. In "**start_enc**" state, based on the current block number and the input macroblock type, the controller decides the current block type and switches into a correlate state. The controller stays at "**Zero block"** (Zero_blk) and "**I_PCM**" (one mode of encoding in H.264 standard) states for only one clock cycle to write the number of non-zero coefficients into the reference memory.

After the last block of the current macroblock is processed in the first two stages, the controller stays at "**Pause**" state until the last information of the current macroblock is written into the outer memory. Then, the next macroblock can be encoded.

IV. PERFORMANCE ANALYSIS AND COMPARISON

A. Implementation and verification

The proposed architecture has been modeled in VHDL at RTL level and implemented using low power CMOS 65*nm* technology from STMicroelectronics. Verification is then done by using a testbench modeled in VHDL. The test cases of a macroblock are defined in the testbench. The input-output block data templates are the block examples presented in [11]. The reference output data is generated manually to make the comparison with the simulation results of the CAVLC encoder. In terms of CAD tools, the simulation is done using ModelSim from Mentor Graphics, and the synthesis is done using DC Compiler Topographical from Synopsys. Synthesis includes automatic clock gating for low power purpose and scanable flip-flops for testability purpose.

Two types of scenario are used. The first one represents the high-quality video with 33% non-zero coefficient of the total coefficients. The other represents the low-bit rate quality video where the coefficients are mostly zeros. As an example, the waveform in Figure 8 shows the simulation waveforms at the end of a macroblock coding process. After the last block is encoded, the last information of the macroblock (the output bits

left after data is aligned into 32-bit codes and data size information) is also written into the memory. The two first stages in the pipeline have to wait until the packing stage complete its task to start the next macroblock.

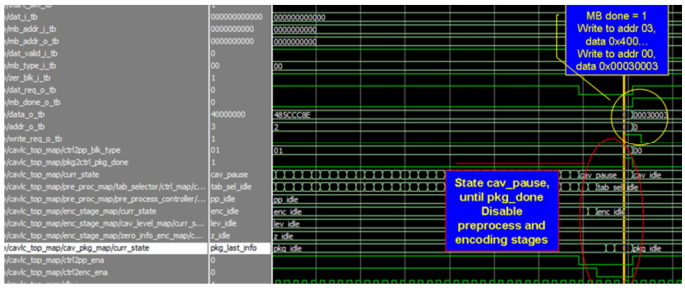

Figure 8. Simulation waveforms at the end of the macroblock encoding.

B. *Performance analysis*

Regarding performance results, Figure 9 presents the relationship between power consumption (in *mW*), area cost (in μm^2) versus the (synthesized) operating frequency of the design. The design is able to achieve a maximum frequency of about 715*MHz* in worst case corner (worst-case process, 1.1V, 105C). At 715*MHz*, more than 100 violating paths (with less than a few *ps*) are observed.

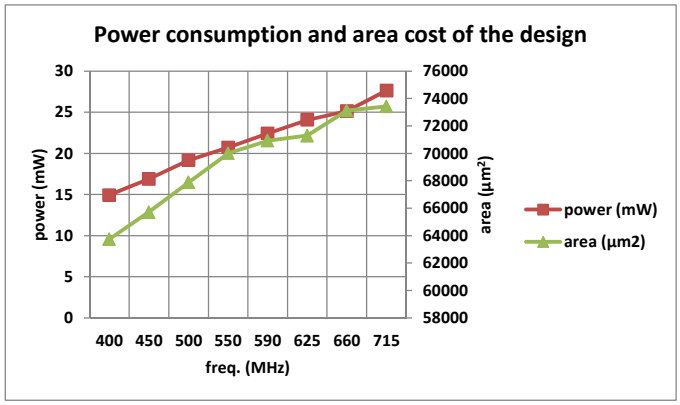

Figure 9. Power consumption and area at targeted frequencies.

At 550*MHz*, the CAVLC encoder consumes approximately 20.7*mW* including 2.4*mW* of clock tree and occupies approximately 32.6*Kgates*. Hardware implementation cost and power consumption of individual modules is reported in TABLE III containing the coeff-token table selector, the pre-process stage occupies the largest amount of area (51.3%) and consumes most energy (62.1%). FIFO is the second largest module, in terms of both area (19.1%) and power consumption (15.1%). In the current design, the size of FIFO is 19 elements, which is the maximum number of codewords per block.

TABLE III. AREA COST AND POWER CONSUMPTION PER SUB-MODULE (@ 550*MHz*)

Item	Pre-process stage	Encoding stage	FIFO	Packing stage	Total
Area cost (gate count)	16734	3611	6248	5747	32636
Area ratio (%)	51.3	11.1	19.1	17.6	100
Power (*mW*)	12.8	1.1	3.1	0.8	20.7
Power ratio(%)	62.1	5.4	15.1	3.8	100

However, the simulation indicates that, even in high-quality video data, a few elements of FIFO are used. Because the FIFO is generic, it should be resized into a more adequate size to achieve better synthesis results. Due to the data dependency, the processing time per macroblock exhibits a large variability. For a high quality test case, in average, it takes around 450 clock cycles to encode a macroblock.

In the case that all coefficients are non-zero, it takes at max 540 cycles. In the low-bit rate test case, number of clock cycles required is about 100 cycles. The minimum operating time to process all zero macroblocks is 52 cycles.

The proposed CAVLC encoder initially targets the CIF video format (resolution: 352×288; frame rate: 30*fps*; colour encoded using *YCbCr* 4:2:0) which is widely used for video teleconferencing. However, at the operating frequency of 550*MHz*, the proposed CAVLC design can process at least 1019000 macroblocks (~2573 CIF frames, ~126 HD1080p video frames) per second. Thus, the design is obviously suitable for real-time application with HD1080p HD video format.

C. *State of the art comparison*

To compare with other CAVLC designs, we use our proposed CAVLC design, running at a frequency of *550MHz*, which is a good area/power tradeoff.

In the previous designs, different techniques and architectures are used. Some modules/phases are not integrated in the CAVLC encoder. Hence, it is not fair enough if only the total gate counts are compared. In the TABLE IV, hardware cost is counted in individual items to be able to give a fair comparison.

TABLE IV. HARDWARE COST COMPARISON

	Implementation cost (*gates*)						
Design	**Pre-proc**	**Enc**	**Fifo**	**Pkg**	**Total**	**Tech.**	**Target**
Prop. at 550*MHz*	16734	3611	6248	5747	32636	65*nm*	CIF@30fps YCBCr 4:2:0
[3]	12283 (scan buffer)	5352	N/A	4796	22611	180*nm*	720p30 YCbCr 4:2:0
[5]	5325 (scan buffer)	2614	N/A	N/A	9724	180*nm*	HD1080p
[10]	5279 (scan buffer)	N/A	N/A	N/A	12276	180*nm*	N/A
[14]	17656 (scan + table select)	4472	N/A	9265	31393	FPGA	HD1080p
[9]	N/A	N/A	N/A	-	6850	FPGA	CIF
[16]	-	7389	-	-	-	65*nm*	HD1080p

N/A: the item is implemented, but there is no number in detail.

'-': the item is not implemented in the design or not counted in the total cost.

Only the CAVLC encoder of D. Kim [14] integrates the coeff-token table selector inside the pre-processing phase, as presented in [15]. In the table, the total cost of scanning and nC_gen [14] and the proposed pre-proc are mostly equal, while the others report only the cost of statistic buffer. Re-encode LUTs and codeword calculation are applied in the proposed

encoding stage. Thus, this module has a fairly low cost, 3611 gate count compared to 7389 gate count of [16] with the same technology 65*nm*.

Packing stage is omitted in some CAVLC designs. In the table, our packing stage has an equivalent area cost. The packing stage in [14] seems to be costly compared to the proposed and [3]. However, in [3] and the proposed design, the codewords enter the packing stage in syntax elements order while in [14], the packing module has to classify and order the codewords.

TABLE V presents other synthesis results from many CAVLC encoders. Zero skipping at block level can reduce the number of cycles per MB only in low-bit rate video. To actually reduce the processing time, zero skipping at coefficient level [6] and parallel scanning [7] actually break the bottleneck at the scanning phase, hence achieve outstanding throughput. The power consumption of [6] is also very promising. However, the frequency is low which reduces the performance of the design. The reason is that in enhancing throughput, their design uses fewer buffers than the others but this causes longer critical path. [6] also adds non-zero &abs-one flags and SLA modules which cost totally 14717 gate counts. The two modules improve the performance however increase the total gate counts to 26598 gates (without table selector and residual SRAM).

TABLE V. CAVLC RESULTS COMPARISON

Design	Cycles/MB	Average *(MBs/sec)*	Freq. *(MHz)*	Techno	Power *(mW)*
Proposed	540-52	5798×10^3	550	65*nm*	20.7
[8]	N/A	N/A	66	TSMC 0.35	21.8
[3]	500-200	350×10^3	100	180*nm*	12.0
[6]	350-100	174×10^3	27	180*nm*	3.7
[5]	413-166 ~300	417×10^3	125	180*nm*	N/A
[14]	432	231×10^3	100	FPGA	N/A
[7]	244	738×10^3	180	FPGA	N/A

As shown in TABLE IV and TABLE V, the proposed design has better performances in comparison to other designs while the area cost is slightly higher (with table selector).

V. CONCLUSIONS

Thanks to many advanced coding techniques equipped, the H.264/AVC has recently become as the most efficient video compression standard with high video quality at a low bit-rate. However, it is very difficult to implement the hardware architecture in order to get high performance while keeping low the overhead due to the computational complexity.

We have presented in this paper an efficient hardware implementation of CAVLC encoder being used in H.264/AVC video codec. Pipelining, zero-skipping, table selector integration and many other techniques are applied to improve the total performance of the design while re-encoded LUT and codeword calculating techniques are used to reduce the area cost. The proposed architecture has been fully modeled, verified, and synthesized using low power CMOS 65*nm* technology from STMicroelectronics. The synthesis results show that at the operating frequency of 550*MHz*, the design occupies about 32.6*Kgates* and consumes 20*mW*. However, the maximum operating frequency can reach to 715*MHz*. The target of the design was initially targeted to CIF video format; but it is also suitable for real-time HD 1080p video format.

VI. ACKNOWLEDGEMENT

This work is supported by Vietnam National University, Hanoi (VNU) through research project No. QGĐA.10.02 (VENGME).

REFERENCES

[1] T. Wiegand, G.J. Sullivan, G. Bjntegaard, and A. Luthra. Overview of the H.264/AVC Video Coding Standard. In IEEE Transactions on Circuits and Systems for Video Technology, Vol. 13, No. 7, July 2003, pp. 560 – 576.

[2] Xuan-Tu Tran, Van-Huan Tran. An Efficient Architecture of Forward Transforms and Quantization for H.264/AVC Codecs. In REV Journal on Electronics and Communications (JEC), Vol. 1, No. 2, pp. 122-129, April – August, 2011, ISSN: 1859 – 387X.

[3] Tung-Chien Chen, Yu-Wen Huang, Chuan-Yung Tsai, Bing-Yu Hsieh and Liang-Gee Chen. Architecture Design of Context-Based Adaptive Variable-Length Coding for H.264/AVC. In IEEE Transactions on Circuits and Systems – II: Express Briefs, Vol. 53, No. 9, September 2006, pp. 832-836.

[4] Yongseok Yi and Byung Cheol Song. High-Speed CAVLC Encoder for 1080p 60-Hz H.264 Codec. In IEEE Signal Processing Letters, Vol. 15, 2008, pp. 891-894.

[5] Chih-Da Chien, Keng-Po Lu, Yi-Hung Shih, and Jiun-In Guo. A High Performance CAVLC Encoder Design for MPEG-4 AVC/H.264 Video Coding Applications. In Proceedings of ISCAS 2006, pp. 3838-3841.

[6] Chuan-Yung Tsai, Tung-Chien Chen and Liang-Gee Chen. Low Power Entropy Coding Hardware Design For H.264/AVC Baseline Profile Encoder. In Proceedings of IEEE International Conference on Multimedia and Expo, 2006, pp. 1941-1944.

[7] F.L.L. Ramos, B. Zatt, T.L. Silva , A. Susin, and S. Bampi. A High Throughput CAVLC Hardware Architecture with Parallel Coefficients Processing for HDTV H.264/ AVC Enconding. In Proceedings of the 17th IEEE International Conference on Electronics, Circuits, and Systems (ICECS), 2010, Dec 2010, pages 587 – 590.

[8] Yeong-Kang Lai, Chih-Chung Chou, and Yu-Chieh Chung. A Simple and Cost Effective Video Encoder with Memory-Reducing CAVLC. In Proc. IEEE Int. Symp. Circuits and System, 2005, vol.1, pp. 432–435.

[9] Choudhury A. Rahman and Wael Basawy. CAVLC Encoder Design for Real-Time Mobile Video Applications. In IEEE Transactions on Circuits and Systems II, Vol. 54, No. 10, Oct. 2007, pp. 873-877.

[10] Yong-Jun Kim, Kyu-Yeul Wang, Sang-Seol Lee, Byung-Soo Kim, Bo-Keun Choi, and Duck-Jin Chung. Implementation of High Efficient CAVLC Encoder for H.264/AVC. In proceedings of the 1st International Conference on Pervasive Computing, Signal Processing and Applications, 2010, pp. 912-915.

[11] Iain E. Richardson. The H.264 Advanced Video Compression Standard, 2nd edition. John Wiley & Son, 2010.

[12] ITU-T, H.264 Advanced Video Coding for Generic Audiovisual Service, March 2005.

[13] G. Bjontegaard and K. Lillevold. Context-adaptive VLC (CVLC) Coding of Coefficients. JVT Document JVT-C028. Fairfax, VA, 2002.

[14] Daeok Kim, Eungu Jung, Hyunho Park, Hosoon Shin, and Dongsoo Har. Implementation of High Performance CAVLC for H.264/AVC Video Codec. In Proceedings of the 6th International Workshop on SoC for Real-Time Applications, 2006, pp.20-23.

[15] X.H. Tian, T.M. Le, X. Jiang anf Y. Lian. Implementation Strategies for Statistical Codec Designs in H.264/AVC Standard. In Proceedings of the 19th IEEE/IFIP International Symposium on Rapid System Prototyping, 2008, pp. 151-157.

[16] C.S. Han, J.H. Lee. Area Efficient and Throughput CAVLC Encoder for 1920 1080@30p H.264/AVC. In Digest of Technical Papers International Conference on Consumer Electronics, Jan 2009.