Science Journal of Circuits, Systems and Signal Processing
2013; 2(1) : 6-15
Published online February 20, 2013 (http://www.sciencepublishinggroup.com/j/cssp)
doi: 10.11648/j. cssp.20130201.12

Context adaptive variable length decoding optimization and implementation on tms320c64 dsp for h.264/avc

Taheni Damak[1,*], Imen Werda[2], Mohamed Ali Ben Ayed[3], Nouri Masmoudi[4,*]

[1]Higher Institute of Computer Sciences and Communication Technologies of Hamam Sousse, University of Sousse, Sousse, Tunisia
[2]Higher Institute of Applied Sciences and Technology, University of Sousse, Sousse, Tunisia
[3]Higher Institute of Electronics and Communication of Sfax, University of Sfax, Sfax, Tunisia
[4]National School of Engineers of Sfax, University of Sfax, Sfax, Tunisia

Email adress:
damak.taheni@gmail.com (T. Damak), Nouri.Masmoudi@enis.rnu.tn (N. Masmoudi)

To cite this article:
Taheni Damak, Imen Werda, Mohamed Ali Ben Ayed, Nouri Masmoudi. Context Adaptive Variable Length Decoding Optimization and Implementation on Tms320c64 Dsp for H.264/Avc, *Science Journal of Circuits, Systems and Signal Processing*. Vol. 2, No. 1, 2013, pp. 6-15. doi: 10.11648/j.cssp.20130201.12

Abstract: Context Adaptive Variable Length Decoding (CAVLD) module takes the lion chair of the H.264/AVC video decoder time due to its complexity. In order to ameliorate decoding speed, a new CAVLD algorithm and an efficient internal memory design were implemented on Digital Signal Processor (DSP). The proposed CAVLD algorithm, Zero Length Prefix (ZLP), was designed to optimize the first syntax element: the CoeffToken. ZLP implementation reduces CAVLD execution time to 21% instead of 41% from decoding time with a throughput of 1.28 MegaMB/s. In addition, the decoder speed was increased from 36 frames per second (fps) to 44 fps for a CIF compressed bitstream.

Keywords: H.264 Video Coding Standard, Decoder, CAVLD, Coefftoken, TMS320C64 DSP

1. Introduction

H.264/AVC[1] is an advanced digital video codec standard that can deliver high resolution video even at low bit rates. To provide high throughput and high compression ratio context-based, an adaptive variable length decoding (CAVLD) is adopted as an entropy decoding method in H.264/AVC video compression standard for baseline profile. Variable length coding (VLC) is a widely technique used in many image and video compression applications. The main idea of VLC is to reduce data redundancy and to minimize the average codeword length by exploiting data statistics. Shorter codewords are assigned to frequently occurred data while longer codewords are assigned to less occurred symbols.

In the CAVLD decoder, look-up tables are used to decode syntax element. Typically decoding process, based on look-up table, requires multiple memory accesses until the desired codeword is found. It is well known that memory access induces a higher power consumption in hardware platform, and reduces the speed in software platform[2]. In addition, CAVLD process uses variable-length technique. Consequently there is no boundary information for detecting the end or the beginning of the codeword[3]. This characteristic substantially complicates the decoder design and decreases its performance. Thus efficient CAVLD algorithm and implementation optimizations are necessary to improve decoder speed.

In this paper, an efficient CAVLD decoding algorithm based on TMS320C6416 DSP is suggested. The remaining of this paper is organized as follows: In section 2, state of art is presented. Section 3 describes briefly the decoder processing and especially the CAVLD module. Section 4 details DSP platform and implementation. The proposed ZLP CoeffToken decoding algorithm steps are explained in section 5. The memory design of CoeffToken parameters is proposed in section 6, followed by performance results and a comparative study with related works in section 7. Finally, conclusion and perspectives are given in section 8.

2. Previous Works

Most CAVLD implementations were implemented on hardware architectures because of its complexity and the repetitive cheeking of look up tables. Reference[4] is a typical example of a hardware implementation. An optimized architecture for CAVLD on 0.13 µm CMOS technology including CoeffToken decoder, Level decoder,

Total_zeros decoder and Run_before decoder was detailed. In[5] also, a description of hardware architecture of the five syntax elements of CAVLD on 0.18 μm CMOS technology was proposed. The design detailed in[5] exploits five different techniques in order to reduce both hardware cost and power consumption.

In literature, many other works focused on the algorithmic approaches that optimized methods to decode one or several syntax elements, such as[6],[7],[8] and[9]. In[6], new look up tables are introduced to decode TotalCoeff and TrailingOnes in one codeword of 19 bits instead of 16 bits. The Cofftoken step determines also the following skip steps, if they exist, to reduce the number of decoding steps. In addition the design of[6] reduced the number of cycles for computing the codeword length of the current decoding block. This parameter is used to determine the next input bitstream (valid bits).

Both[7] and[8] optimize only Level syntax. Reference[7] suggested a very large-scale integrated implementation under 90-nm CMOS technology. Since decoding the Level syntax element depends on the previous one, pipeline structure cannot be exploited. Moreover, the inter-codeword dependency and the succession of the arithmetic operations to decode a Level lead to an unavoidably long critical path. Proposed solution in[7] broke the inter-Level dependency and the inter-codeword dependency by a delay balanced two-Level architecture. Reference[8] presented a pipelined architecture of CAVLD on FPGA. Level module was optimized by multi syntax decoding method that decodes two Levels in parallel. The pipelining possibilities were also explored between syntax elements modules. In addition, the syntax element processing could begin at the same time with reading and fixing the length of the next syntax element.

Reference[9] proposes two methods to improve the throughput of CAVLC decoder. The first one was called Multiple Level Decoding (MLD). It could decode two Levels in one cycle on most situations. The second one was nonzero skipping for Run_before decoding (NZS). It is able to detect coefficients whose Run_befores are null. These Run_befores were decoded in the same clock cycle. The architecture was also proposed on very large scale integration circuits.

Other optimisation approach is based on statistic and analysed method. Proposed algorithm in[10] analyzes the correlation between bit patterns and 4x4 (or 2x2) blocks and has an idea of a pattern-search method before CAVLC decoding. If a pattern is matched in the proposed look-up table, the standard CAVLD procedure is skipped and the block is directly reconstructed. However, if there is no any pattern matched in the table, CAVLD processing is used to reconstruct the block. This method was applied on a software implementation on ARM-based embedded system.

When analyzing these previous works, it is clear that all implementations were on hardware platform, expected of[10] that was on standard processor. In fact, ASIC solutions are not flexible. It can be changed to improve or to update architecture. However, software implementation on specific processor, such as DSP, can be often updated. In addition DSP architecture is able to attend real time execution.

Furthermore, all the proposed algorithms optimized syntax element processing and not the reading's technique data from bitstream. The bottleneck of CAVLD is the multiple access memory to found corresponding codword from bitstream. The optimization of this step is primordial to speed up CAVLD processing.

3. H.264 Decoding Process

The H.264/AVC decoder receives a compressed bitstream as input for the entropy decoding module that includes CAVLD and Exp-Golomb. The prime focus in this work is CAVLD, witch is highlighted in Figure 1. Outputs data of CAVLD are then reordered and will be inverse quantized and inverse transformed. Resulting block is added to the previously predicted one in order to reconstruct frame blocks after a de-blocking filter process. Two kinds of prediction can be used in H.264: intra prediction and inter prediction.

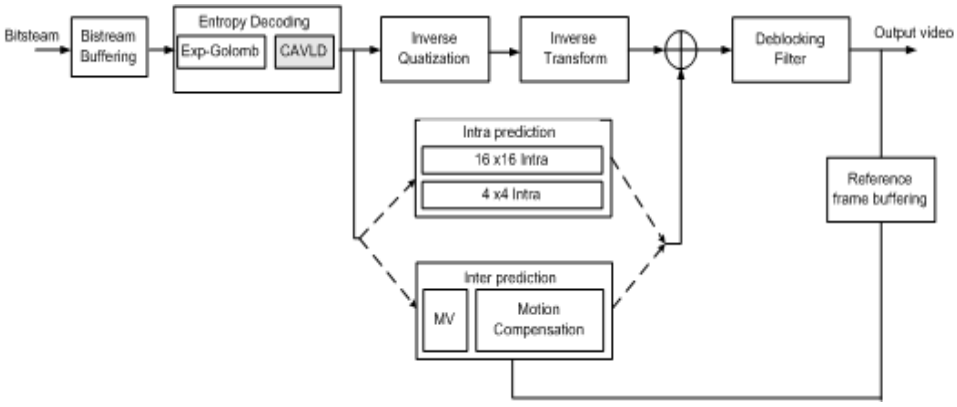

Figure 1. *H.264/AVC decoder block diagram.*

3.1. Entropy Decoding Process

The H.264 entropy decoding uses two ways to

reconstruct compressed value in bitstream. According to the type of parameter, CAVLD or Exp-Golomb is used. The Exp-Golomb decoding is a special VLC Huffman code based on a regular construction. It is used to decode macroblock header information such as prediction mode, motion vector difference and macroblock type.

For residual coefficients, CAVLD is the appropriate used technique. It decodes bitstream symbols to provide a zig-zag ordered 4x4 blocks of transformed coefficients. CAVLD is also a special VLC Huffman method but it is based on look-up tables.

According to the previously decoded data, the CAVLD module can adaptively choose one of several VLC tables, and then decodes the current input codeword efficiently by using the five syntax elements for each 4x4 block[11] as follows:

1. First syntax element is CoeffToken wich encloses two parameters: The number of none zeros coefficients of a 4x4 block (TotalCoeff) and the number of coefficients that the absolute values are equal to one (TrailingOnes). The VLC look-up table for CoeffToken is fixed by the mean of a parameter N. Since neighbouring blocks in a frame have similar number of none zero coefficients, for each decoded 4x4 block, N is computed from the number of none zero coefficients of the up and left corresponding blocks. If up and left blocks of the current block are available, N presents the average of Nup and Nleft as given in equation (1):

$$N = (Nup+Nleft)/2 \quad (1)$$

Nup and Nleft of a block are respectively the TotalCoeff of the upper decoded block and the TotalCoeff of the left decoded block. If only up or left neighbouring block is available, N takes its value (Nup or Nleft).If no block is available N takes zero.

1. Then a one bit parameter (Sign) is decoded to inform about TrailingOnes signs. It is null if the TrailingOnes sign is positive. Otherwise, it is equal to one.
2. To reduce bit rate, only none zero coefficients are decoded in CAVLD. The value of each non zero coefficient, except of TrailingOnes, is decoded in inverse zig-zag scan order as a Level. Then position of each Level is indicated by the mean of two following syntax elements: Total_zeros and Run_before.
3. Total_zeros syntax element presents the total number of zeros preceding non-zero coefficients. The VLC tables selected to decode the Total_zeros are decided according to the total number of the none-zero coefficients in the current block.
4. Run_before syntax element is the number of zeros preceding each non-zero coefficient. It is decoded in reverse order. The VLC table for each Run_before is chosen depending on the previous number of zeros which was not decoded yet.

3.2. Inverse Prediction

Using header information decoded from the bitstream for each macroblock, the decoder creates a predicted block that is identical to the original resulted by the encoder prediction. According to the predictor parameter, Intra or Inter mode prediction may be selected. Inter coding uses motion vectors to exploit temporal statistical dependencies between different pictures. Intra coding uses various prediction modes to exploit spatial statistical dependencies in the same picture. Motion vectors and intra prediction modes may be specified for a variety of block sizes in the picture.

The predicted block is added to the residual block generated by inverse transform and inverse quantization to reconstruct a version of the original block.

3.3. Inverse Transform and Quantization

The result block of CAVLD is inverse quantized through a quantization parameter (Qp) deducted from bitstream. H.264 defines 52 quantization parameters in order to manage the trade of between video quality and bitrate. Then, an inverse Integer Cosines Tranfrom (ICT) is applied to remove spatial correlation inside a block of samples.

The purpose of the transform stage is to convert residual data into frequency domain. The inverse transform reconstructs the block into temporal domain. The ICT is an evolution of the Discrete Cosines Transform (DCT). Like ICT, inverse ICT is also a 2D integer matrix multiplication. It is applied on each 4x4 block output of the inverse quantization. A non-integer matrix multiplication part of transform is interned in the inverse quantization step to keep an integer transform.

H.264/AVC defines also a supplementary Hadamard transform for DC coefficients in 16x16 intra mode prediction case.

3.4. De-blocking Filter

The de-blocking filter is applied to each decoded macroblock to reduce blocking distortion. It is applied after the inverse transform and the inverse quantization. It is the last step to reconstruct and display the macroblock. In order to improve the appearance of decoded video frames, the de-blocking filter smoothes edges of blocks that transform and quantization process can cause.

Filter is applied to vertical and horizontal edges of 4x4 blocks in a macroblock. First filtered boundaries are the 4 vertical one of luminance component in the following order: a, b, c, d, as shown in Figure 2. Then 4 horizontal boundaries of the luminance component proceed as per the following order: e, f, g, h of Figure 2. Boundaries of chrominance component are also filtered, starting by edge “i” to edge “l”.

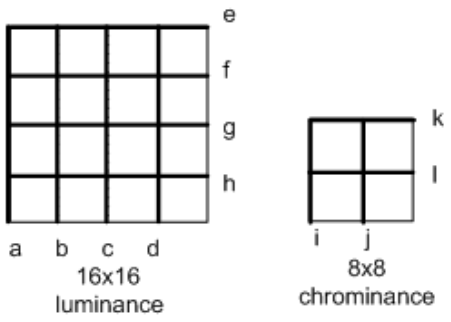

Figure 2. *The order of Edge filtering in a macroblock.*

According to the Boundaries Strength parameter (BS), Standard or Strong filter is applied on adjacent samples of a boundary. BS depends on the current Qp, the decoding modes of neighbouring blocks and the gradient of image samples across the boundary. Each filtering operation affects up to three samples on each side of the boundary.

Figure 3 shows four samples on each side of vertical and horizontal boundaries of adjacent blocks p (p0, p1, p2, p3) and q (q0, q1, q2, q3). For more details, a previous presented work[12] describes the de-blocking filter development and implementation.

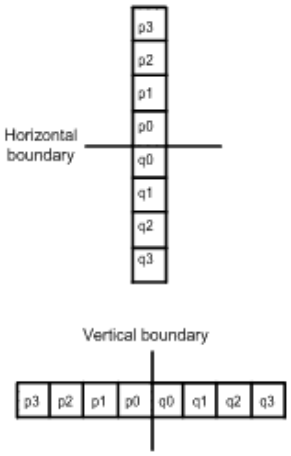

Figure 3. *Horizontal and vertical boundaries of adjacent samples.*

4. DSP Implementation

The TMS320C64x[13] is a fixed point DSP. It features very long instruction word (VLIW) architecture. As shown in Figure 4, C6416 Central Processing Unit is characterized by two sets of functional units. Each set contains four units which can execute parallel instructions. This architecture is adapted to multi-function applications. The C6416 is a 1GHz cache-based architecture. The Level-one cache (L1) is 16 Kbytes in size; it is only accessible by the CPU.

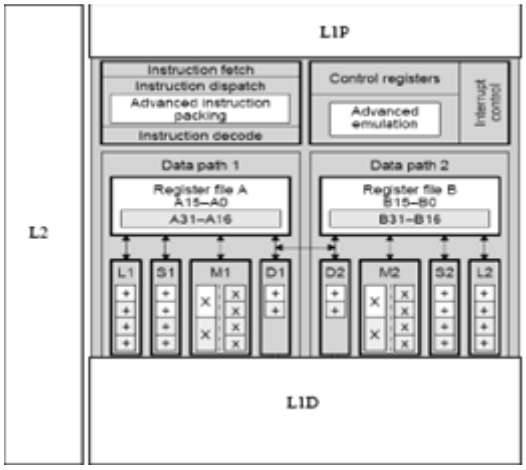

Figure 4. *Architecture of TMS320C6416 core.*

The Level-two cache (L2) is 1Mbytes in size, and it may be configured as all memory mapped SRAM, all cache, or a combination of the two.

Intrinsic instructions are allowed to program the DSP, such as _abs, _dotp4, and _pack. Intrinsic instructions combine the assembly programming language efficiency and the simplicity of high abstraction programming language level.

Preceding the proposed implementation, earlier works were elaborated to implement an optimized H.264 baseline encoder[14] on TMS320C6416 platform. Then, many structural, algorithmic and architectural optimization approaches presented in[15], were applied to attend real time execution. Additional serious efforts were investigated to develop the corresponding H.264 baseline decoder on the same DSP platform[16]. First CAVLD implementation gave the profile result of Figure 5. CAVLD block took 41% of the total running time of the decoder[17]. It was the decoder bottleneck. To improve the CAVLD complexity, algorithmic and structural optimizations were elaborated. Details of the two optimizations are given in the following sections.

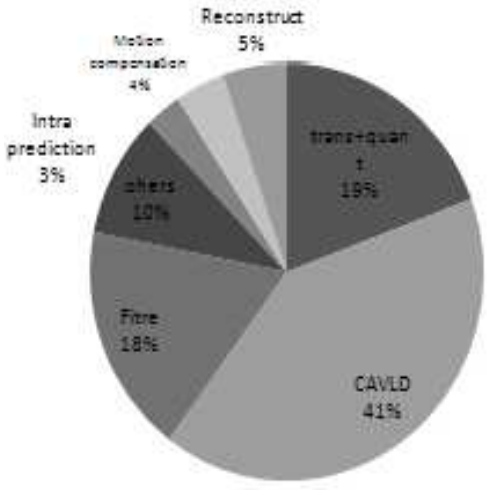

Figure 5. *First Foreman decoder profile.*

5. Proposed CAVLD Algorithm

CoeffToken look up tables are composed of 262 codewords[18]. This syntax element involves TotalCoeff and TrailingOnes parameters. Many combinations of those two parameters are possible: TotalCoeff can vary from 0 to 16 and TrailingOnes can take as value: 0, 1, 2 or 3. When VLC table is fixed by N, the subsequent step is needed to check the actual code in the selected table and to determinate its corresponding value. As defined in H264 coding standard, CAVLD applies variable length decoding method to code syntax elements. VLC technique gives an advantage to reduce the bitstream length and to decrease the bit rate, but at the same time, it represents a handicap to the CAVLD time decoding. In fact, to find the value of the actual codeword, all look up tables have to be checked for different code length. Starting by the shorter possible, if this codeword doesn't correspond to any code in the table, next possible length is applied to the codeword. This standard method, shown in Figure 6, is not suitable for a real time application since a computational time is needed to decode a long codeword (the longest codeword for CoeffToken is a 16 bits code).

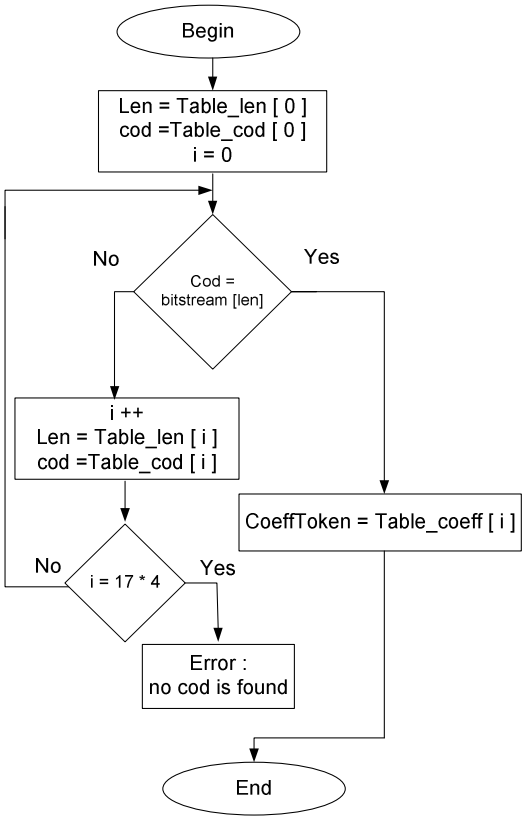

Figure 6. *Standard algorithm to decode CoeffToken.*

To reduce CAVLD time execution, a new ZLP technique which is detailed in Figure 7 was applied.

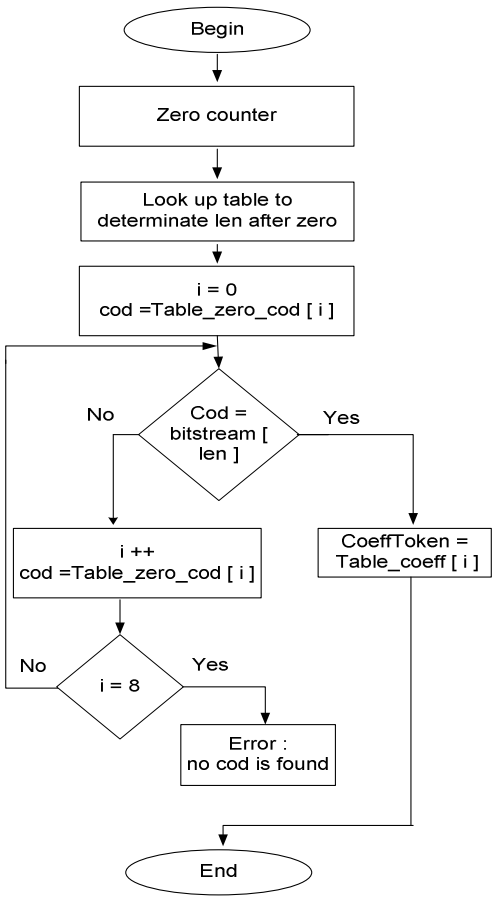

Figure 7. *ZLP proposed algorithm to decode CoeffToken.*

When examining CoeffToken VLC table, it is clear that the number of zero at the beginning of each code makes the difference between frequent and less frequent coefficients. ZLP algorithm takes advantage from this propriety. It makes use of a pre–analysis step to determinate the number of zeros at the beginning of each code.

According to zero prefix number, the possible corresponding code can be limited. New look up tables (Zero_tab) are deduced from the VLC table. Instead of VLC_0, VLC_1 and VLC_2 tables that H.264 standard defines to determine CoeffToken, Zero_tab_0, Zero_tab_1 and Zero_tab_2 are the respectively used tables. Those tables are given by table 1, table 2 and table 3.

Table 1. *Zero_tab_0.*

Zero length	Code word length	TotalCoeff	TrailingOnes
3	5,4	1,2	0,1
4	5,4	3,5	2,3
5	7,6,5,4	2,3,4,6	0,1,2,3
6	7,6,5,4	3,4,5,7	0,1,2,3
7	7,6,5,4	4,5,6,8	0,1,2,3
8	7,6,5,4	5,6,7,9	0,1,2,3
9	15,11,8,14,10,13,9,12	6,7,8,7,8,8,9,10	0,0,0,1,1,2,2,3
10	15,11,14,10,13,9,12,8	9,10,9,10,10,11,11,12	0,0,1,1,2,2,3,3
11	15,11,14,10,13,9,12,8	11,12,11,12,12,13,13,14	0,0,1,1,2,2,3,3
12	15,11,14,10,13,9,12,8	13,14,14,15,14,15,15,16	0,0,1,1,2,2,3,3
13	7,4,6,5	15,16,16,16	0,0,1,2

Table 2*. Zero_tab_1.*

Zero length	Code word length	Total Coeff	Trailing Ones
2	7,6,3,2,1,0	2,5,1,3,3,6	1,3,0,1,2,3
3	7,6,5,4	2,4,4,7	0,1,2,3
4	7,6,5,4	3,5,5,8	0,1,2,3
5	7,4,6,5	4,5,6,6	0,0,1,2
6	7,6,5,4	6,7,7,9	0,1,2,3
7	15,11,14,10,13,9,12,8	7,8,8,9,8,9,10,11	0,0,1,1,2,2,3,3
8	15,11,8,14,10,13,9,12	9,10,11,10,11,10,11,12	0,0,0,1,1,2,2,3
9	15,11,14,10,13,9,12,8	12,13,12,13,12,13,13,14	0,0,1,1,2,2,3,3
10	7,6,1,3,0,2	14,14,15,14,15,15	0,2,0,1,1,2
11	7,6,5,4	16,16,16,16	0,1,2,3

Table 3. *Zero_tab_2.*

Zero length	Code word length	TotalCoeff	TrailingOnes
0	15,14,13,12,11,10,9,8	0,1,2,3,4,5,6,7	0,1,2,3,3,3,3,3
1	15,12,10,8,14,11,9,13	2,3,4,5,3,4,5,8	1,1,1,1,2,2,2,3
2	15,11,8,14,10,13,9,12	1,2,3,6,7,6,7,9	0,0,0,1,1,2,2,3
3	15,11,9,8,14,13,10,12	4,5,6,7,8,8,9,10	0,0,0,0,1,2,2,3
4	15,11,14,10,13,9,12,8	8,9,9,10,10,11,11,12	0,0,1,1,2,2,3,3
5	15,11,8,14,10,13,9,12	10,11,12,11,12,12,13,13	0,0,0,1,1,2,2,3
6	5,1,4,0,3,2	13,14,14,15,14,14	0,0,1,1,2,3
7	5,4,7,6	15,16,15,15	0,1,2,3

The new maximum code length is 4 instead of 16 in the original tables. Zero_tab is partitioned into 11 sub-tables organized as follows: codes that have similar prefix number of zeros are grouped in the same sub-table Sub_Zero_tab. Most codes in the Sub table have the same length. Consequently, when the zero prefix is fixed, the remaining bits (suffix) became a fix length code and resolve the CAVLD decoding problem of variable length code. Hence, only one codeword length is cheked in tables.

Furthermore, in each Sub_Zero_tab, only 8 CoeffToken values are possible. Consequently, a CoffToken decoding process needs a maximum of 8 memory accesses. For standard algorithm, since 17 NumCoeff and 4 TrailingOnes solutions are possible in a VLC table, 68 memory accesses can take place to decode one CoeffToken. For example, to decode CoeffToken (NumCoeff = 8, TrailingOnes = 3) in VLC1 the following code has to be cheeked: “0000000101”. To find a result using the standard method, 59 memory accesses are necessery (8+3x17). The same result can be found in only 3 memory accesses to Zero_tab. The ZLP algorithm fixes the codeword length from the beginning to avoid multiple bitstream tests. It reduces also the memory access by replacing the VLC tables by an adaptive Zero tables.

6. Proposed Memory Design

In addition to the intensive complexity of the CAVLD algorithm, the use of look up tables intensified the appropriate time execution. In fact, memory access not only causes additional computations for generating memory addresses but also limits the DSP speed. The primitive solution was to avoid loading tables for each decoding parameter in order to reduce memory access. Therefore, the internal memory (L2) was configured as 256 Kbytes cache and 754 Kbytes ISRAM. Look up tables were then designed as persistent memories which alleviated the data transfer rate.

The memory design optimization was also applied to the first decoded VLC symbol in CAVLD, the CoeffToken. To decode this syntax element, one of the three look up tables is previously fixed: the adaptive VLC table (or *Zero_tab*) is selected by using N parameter computed from the CoeffToken value for the previously decoded neighbouring blocks. As mentioned in equation 1, Nup and Nleft of a block are respectively TotalCoeff of the earlier decoded block and TotalCoeff of the block in the same position at previously decoded row of block in frame. Consequently, TotalCoeff of each block must be stored to be used later as neighbours for other blocks. With a CIF (352x288) sequence, (22 x 18 x16) TotalCoeff values should be stored for each frame, since the frame contents (22x18x16) blocks. Hence, an overpass of DSP internal memory can be produced. To optimize memory size needed to calculate N, two different cases of block neighbouring are emphasized as follows and as illustrated in Figure 8 and figure 9:

Figure 8. *First case : neighbouring block are in the same macroblock.*

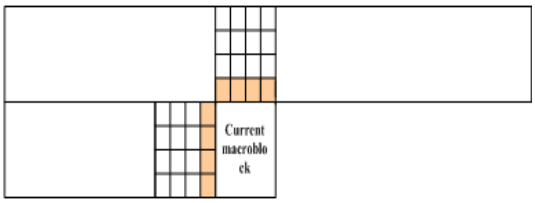

Figure 9. *Second case : neighbouring block are from a different macroblocks.*

- If neighbouring blocks (Up and Left) are in the same macroblock, the case of blocks 5, 6, 7, 9, 10, 11, 13, 14 and 15 as shown in figure 8.There is no need to use supplementary buffer since TotalCoeff of all blocks are already stored for the quantization.
- If neighbouring blocks (Up and Left) are from different macroblocks, the case of figure 9, such blocks 0, 1, 2 and 3 in figure 8. Corresponding up block is localised in the upper macroblock. Also for blocks 0, 4, 8 and 12, left block is taken from the previously decoded one, the left one. Therefore, two buffers are reserved for luminance coefficients in this case to save up and left neighbours. The first one, Numcoef_tab_up, is a (22x4) bytes buffer size. After each macroblock decoding, TotalCoeff of blocks 12, 13, 14 and 15 are stored in the corresponding places in Numcoef_tab_up buffer in order to be used as up neighbours for the macroblock at the same position in the following line of macroblock. The second table is called Numcoef_tab_left. It saves TotalCoef of the blocks 3, 7, 11 and 15 of each macroblock in order to be used as left neighbours for the next macroblock. For Chrominance components, two tables are reserved. Each table sizes 44 values. For Left neighbours, a buffer of two values for each chrominance is allocated.

7. Results

H.264 LETI baseline decoder was first implemented with standard CAVLD method and then with ZLP algorithm, on a TMS320C6416 DSP. Table 4 compares CAVLD execution time between the two implementations, benchmarked with popularly known CIF sequences such as Foreman, Mobile, Paris and Akiyo which are respectively encoded at 384, 512, 128, 1024 Kbps streams. ZLP effects appear clearly on the CAVLD execution time. In fact, more than 20% of CAVLD execution time is reduced for the worst case of Forman bitstream. For Mobile bitstream, the gain could attain 60%.

Table 4. *ZLP effect on CAVLD execution time.*

bitstream Sequences	Standard CAVLD(%)	CAVLD using ZLP(%)	CAVLD gain(%)
Foreman	41	21	20
Paris	37.83	27.43	42.3
Akiyo	25	19.44	30
Mobile	68.56	46.2	60

Throughput of CAVLD in term of mega macroblock per second for Forman bitstream was 0.55 MegaMB/s with the classic CAVLD algorithm. It became 1.28 MegaMB/s with ZLP algorithm.

Consequently, ZLP algorithm affected the overall decoder time execution. Table 5 lists the decoding throughputs in terms of frame per second for different test sequences, on TMS320C6416 DSP. For all test bitstreams, decoder speed presents an apparent amelioration with the ZLP algorithm.

Table 5. *Decoder performances.*

bitstream Sequences	Pre CAVLD optimization	Post CAVLD optimization	Speed gain(%)
Foreman	36 fps	44 fps	18 %
Paris	37 fps	47,5 fps	22 %
Akiyo	45 fps	50 fps	10 %
Mobile	20 fps	34.7 fps	42 %

In addition to the speed results amelioration, the decoder blocks partition becomes equilibrated due to the ZLP algorithm. In fact, the comparison of profiles illustrated in Figure 5 and in Figure 10 for Foreman test sequence at 384 Kbps, demonstrates that the transform and quantization modules consume most of the decoder time execution instead of CAVLD module. Based on[19] decoder profile, proposed decoder with ZLP method gives more balanced profile results.

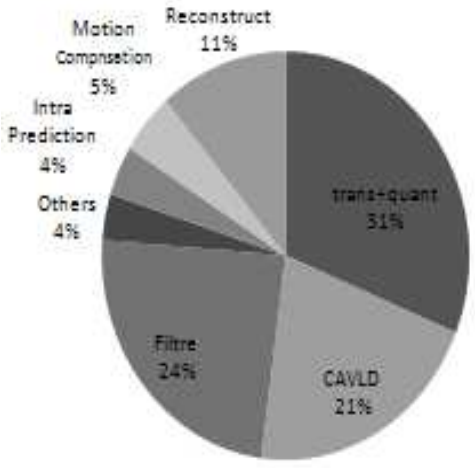

Figure 10. *Foreman Decoder profile with ZLP algorithm.*

Table 6 summarizes previous works related to CAVLD optimization and implementation where different architectures are revealing.

Design presented in[20] is a full software implementation. Therefore performance could be compared to the proposed implementation on DSP TMS320C6416. Architecture developed in[20] was applied on a CPU-Pentium4 frequented at 3.6 Ghz and then on a CPU-Dual Zeon with 3.06 Ghz. The first implementation gave a throughput of 0.61 MegaMB/s which wass less than the second one (3.44 MegaMB/s). This result is obvious since it was implemented on a dual processor platform.

Table 6. *Comparison with State of art.*

	technology	**Cycles/MB**	**Frequency**	**megaMB/second**
[4]	0.13 μm	141	250 Mhz	1.77
[5]	0.18 μm	294	175 Mhz	0.59
[6]	0.18 µm	126	30.8 Mhz	0.24
[7]	90 nm	127.13	125 Mhz	0.98
[8]	Altera StratixIII FPGA	412.5	74.25Mhz	0.18
[9]	0.18μm	141	160 Mhz	1.13
[20]	CPU-Pentium4	5.9x10^3	3.6 Ghz	0.61
	CPU-Dual Zeon	889	3.06 Ghz	3.44
Proposed	DSP TMS320C6416	781	1 Ghz	1.28

The proposed implementation in a simple core platform led to a throughput of 1Ghz. Therefore the number of macroblocks per second is less than those in the implementations proposed in[20] using dual core platform. However the number of cycles per macroblock of the proposed work is better than those in both implementations of[20]. This parameter illustrates the impact of the algorithm optimization. In fact, if the proposed CAVLD algorithm was implemented on a higher frequency platform, its throughput would clearly increase.

As mentioned in previous works section, most of CAVLD implementations are based on hardware architectures, as references[4],[5],[6],[7],[8]and[9]. Performance comparison is not possible since the huge architectural difference. However, throughput results could be taken on consideration to compare the final CAVLD results. Only the algorithm in reference[4] is speeder than the proposed implementation in term of macroblock per second.

This can be explained by the fact that architecture in[4] optimized each syntax elements. However, the proposed design optimizes only CoeffToken syntax element.

Reference[6] optimized CoeffToken syntax element. Compared to the proposed architecture, it cannot decode more macroblock per second.

For reference[7],[8] and[9], they don't assure better throughput compared than the proposed architecture.

8. Conclusion

In this paper, a ZLP algorithm and an optimized memory design is suggested to ameliorate CAVLD performance and LETI decoder speed. This new method of decoding CoeffToken parameter reduced CAVLD complexity and limited the memory access that present the bottleneck of decoder. Since CAVLD is a variable length coding technique, multiple accesses to look up tables are necessary until founding corresponding codeword. ZLP method suggested a pre-analyses step to fix code length. The advantages of DSP architecture were also exploited to design efficiently the internal buffers in order to reduce memory accesses. Finally throughput of CAVLD was 1.28 MegaMB. Thus, decoder speed was improved to be 44 fps and to attend real time. As perspectives, other decoder blocks can be optimized using optimal algorithm to attain real time decoder with SD sequences.

References

[1] 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.

[2] Yong Ho Moon, Gyu Yeong Kim, and Jae Ho Kim, "An Efficient Decoding of CAVLC in H.264/AVC Video Coding

Standard", Transactions on Consumer Electronics, Volume: 51 , Issue: 3, pp 933 – 938, 2005.

[3] Jari Nikara, Stamatis Vassiliadis, Jarmo Takala, Mihai Sima, and Petri Liuha, "Parallel Multiple-Symbol Variable-Length Decoding". International Conference on Computer Design. Freiburg, Germany. ICCD 2002.

[4] MythriAlle,JBiswas,S.K.Nandy "High Performance VLSI Architecture Design for H.264 CAVLC Decoder", Application-Specific Systems, Architecture and Processors. Colorado, USA. ASAP 2006.

[5] Hsiu-Cheng Chang, Chien-Chang Lin, and Jiun-In Guo, "A Novel Low-Cost High-Performance VLSI Architecture for MPEG-4 AVC/H.264 CAVLC Decoding", Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on, volume 6, pp 6110 - 6113, May 2005.

[6] Myungseok Oh, Wonjae Lee, Yunho Jung, and Jaeseok Kim, "Design of High-Speed CAVLC Decoder Architecture for H.264/AVC". ETRI 2008 Journal, Volume 30, Number 1, February 2008.

[7] Yuan-Hsin Liao, Gwo-Long Li, and Tian-Sheuan Chang, "A 385MHz 13.54K Gates Delay Balanced Two-Level CAVLC Decoder for Ultra HD H.264/AVC Video ", IEEE transactions on circuits and systems for video technology, volume 22, no. 11, pp 1604-1610. , November 2012.

[8] Tony Gladvin George, Dr.N.Malmurugan, "A New Fast Architecture for HD H.264 CAVLC multi-syn Decoder and its FPGA Implementation". International Conference on Computational Intelligence and Multimedia Applications. Sivakasi, India. ICCIMA 2007.

[9] Tsung-Han Tsai, Member, IEEE, Te-Lung Fang, and Yu-Nan Pan, "A Novel Design of CAVLC Decoder with Low Power and High Throughput Considerations", IEEE transactions on circuits and systems for video technology, volume 21, no. 3, pp 311-319, March 2011.

[10] Shau-Yin Tseng, Tien-Wei Hsieh, "A Pattern-Search Method for H.264/AVC CAVLC decoding", Multimedia and Expo, 2006 IEEE International Conference on, pp 1073 – 1076, Jully 2006.

[11] Iain E.G. Richardson, "H.264 and MPEG-4 Video Compression – video coding for next generation multimedia". John Wiley & Sons, pp.187-207. 2003.

[12] Taheni Damak, Imen Werda, Sébastien Bilavarn, Nouri Masmoudi, "Fast prototyping H.264 deblocking filter using ESL tools", 8th International Multi-Conference on Systems, Signals & Devices. Tunisia. SSD 2011.

[13] Texas Instuments, 2001.TMS320 C¬64x Technital overview. spru395b, Janvier 2001.

[14] I. Werda, H. Chaouch, A. Samet, M.A. Ben Ayed and N. Masmoudi, "Optimal DSP-Based Motion Estimation Tools Implementation For H.264/AVC Baseline Encoder". International Journal of Computer Science and Network Security, IJCSNS, VOL.7 No.5, May 2007.

[15] T.Damak, I.Werda, A.Samet, N.Masmoudi, "DSP CAVLC implementation and Optimization for H.264/AVC baseline encoder", IEEE International Conference on Electronics, Circuits and Systems. Malte. ICESC, 2008.

[16] Werda. I, Dammak. T, Grandpierre. T, Ben Ayed MA, Masmoudi N, " Real-time H.264/AVC baseline decoder implementation on TMS320C6416", Springer-Verlag 2010, J Real-Time Image Processing, Volume 7, Issue 4, pp 215-232.

[17] Damak T., Werda I., Ben Ayad M-A, Masmoudi N, "An Efficient Zero Length Prefix Algorithm for H.264 CAVLC Decoder on TMS320C64", International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS). Tunisia, 2010.

[18] Yanmei Qu, Yun He and Shunliang Mei, "A Novel Cost-Effective and Programmable VLSI Architecture of CAVLC Decoder for H.264/AVC". Journal of Signal Processing Systems, Springer Science + Business Media, LLC. Volume 50, pp 41- 51 , 2008.

[19] Klaus Schoffmann, Markus Fauster, Oliver Lampl, and Laszlo Bosz ormenyi , " An Evaluation of Parallelization Concepts for Baseline-Profile Compliant H.264/AVC Decoders". LNCS 4641, pp. 782–791, 2007. Springer-Verlag Berlin Heidelberg. 2007.

[20] Sangyoon Park, Kyeongyuk Min, Jongwha Chong, "The New Memory-Efficient Hardware Architecture of CAVLD in H.264/AVC for Mobile System", Communications and Information Technology, 2009. 9th International Symposium on, pp 204-207. ISCIT 2009.