Architecture of High-throughput Context Adaptive Variable Length Coding Decoder in AVC/H.264

Gwo Giun (Chris) Lee, Shu-Ming Xu, Chun-Fu Chen, Ching-Jui Hsiao
Department of Electrical Engineering, National Cheng Kung University, Tainan, Taiwan
E-mail: {clee, N26991441, N28991146, N26984559}@mail.ncku.edu.tw, Tel: +886-6-2360663

***Abstract*—In this paper, a High-throughput Context Adaptive Variable Length Coding decoder which is capable for supporting AVC/H.264 HP@level 4.2 has been presented. To increase throughput, multi-symbol decoders for "LEVEL" and "RunBefore" and architecture of fast zero insertion are presented to reduce processing cycles to reach high-throughput rate. Finally, the experimental results show that the throughput of presented Context Adaptive Variable Length Coding decoder achieves the level limitation of level 4.2 in AVC/H.264 and the synthesis result shows that the gate count is about 17.2K gates at a clock constrain of 108MHz.**

I. INTRODUCTION

Many people have great demand on images and videos of high quality. Several new techniques like high definition TV, blu-ray DVD have been developed and will become more and more popular in the future. A brand new compression standard, AVC/H.264 [1], is used for these products because of its better compression performance. For this purpose, we would like to design a CAVLC decoder for AVC/H.264 which provides higher throughput for HD 1080P video format. According to the level 4.2 in AVC/H.264, the supported resolution and frame rate is 1920x1080 at frame rate 60fps; therefore, the number of MBs should be decoded in one second is 489,600 MBs (1). In addition, the system clock rate of our decoder is 108 MHz; therefore, only 220 cycles (2) to decode one MB include the syntax elements. To increase the throughput of the CAVLD, multi-symbol decoder architectures and fast zero insertion are presented to reduce processing cycles to achieve required throughput rate.

(*Resolution / MB size*) x *frame rate* =
((1920 x 1088) / (16 x 16)) x 60 = 489,600 *MB/s* (1)

Clock Rate / *number of MB per second* =
$(108 \times 10^6) / 489{,}600 \fallingdotseq 220$ *cycles*/*MB* (2)

II. OVERVIEW OF CAVLC IN AVC/H.264

Entropy coding is a lossless compression scheme. According to the information theory, the entropy of source bounds the minimum required bits per symbol in average. The symbols with higher occurrence probability are assigned to shorter codeword length; conversely, the symbols with lower occurrence probability are assigned to longer codeword length.

Only quantized transform coefficient and DC block of intra-coded MB would be encoded by Context Adaptive Variable Length Coding (CAVLC) in AVC/H.264. CAVLC [2] was designed to take many advantages by exploiting the statistical characteristics of quantized blocks. The encoding flow [1] was illustrated as follows. Symbol *TotalCoeff* means the number of non-zero coefficient in a 2x2 or 4x4 block and *TrailingOnes* means number of trailing ones in reverse order; by the way, the largest value of *TrailingOnes* is three. First of all, the symbol *CoeffToken* in a 2x2 or 4x4 block is encoded and which is composed by *TotalCoeff* and *TrailingOnes* and then sign flags of *TrailingOnes* would be encoded. The other coefficients that not belong to *TrailingOnes* are encoded as symbol *LEVEL*. The number of total zeros before the first non-zero coefficient in reverse order array would be encoded to symbol *TotalZeros*. Finally, number of zeros between the two non-zero coefficients is coded as symbol *RunBefore* and the tables selected to encode *RunBefore* depends on the value of *Zeroleft*; *Zeroleft* represents the number of zeros that have not been encoded.

III. CONVENTIONAL DESIGN ANALYSIS

In the aspect of CAVLD architecture design, S.Y. Tseng *et al.* [3] gather many statistics and find some 4x4 block occurrence frequently with relative bitstream pattern; therefore, they use pattern search method to decode some particular 4x4 blocks; if the pattern was matched, the decoder directly reconstructs the relative 4x4 block within one cycle; otherwise, the normal CAVLC decoding process is executed. S. Park *et al.* [4] combined the decoding process of *CoeffToken* and sings flag of *TrailingOnes* into one and also proposed a memory-free architecture to decode *CoeffToken*. J. Bae *et al.* [5] proposed a high throughput *RunBefore* decoder that could decode at most four symbols when *RunBefore* equals to zero. T.L. da Silva *et al.* [6][7] and C.C. Lo *et al.* [8] used tree-based decoding method to decode *CoeffToken* because it didn't need access memory and could reduce power consumption. T.G. George *et al.* [9] used constant output rate decoder to decode *LEVEL* and *RunBefore* and it could decode two symbols per cycle. Unfortunately, the CAVLC decoder architecture described above is not enough throughputs rate to support HD 1080P sequences at frame rate 60 fps. In this paper, a high-throughput context adaptive variable length coding decoder which is capable for supporting AVC/H.264 HP@level 4.2 has been presented.

IV. PROPOSED ARCHITECTURE OF CONTEXT ADAPTIVE VARIABLE LENGTH CODING DECODER

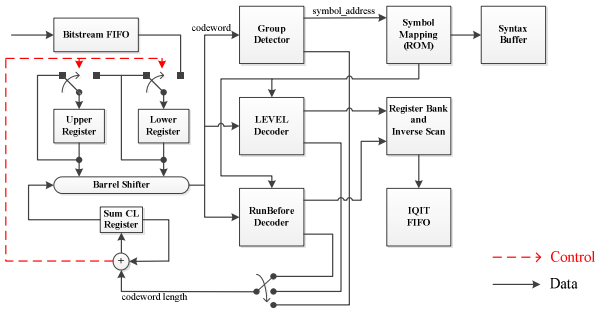

Fig. 1 High-level block diagram of CAVLD

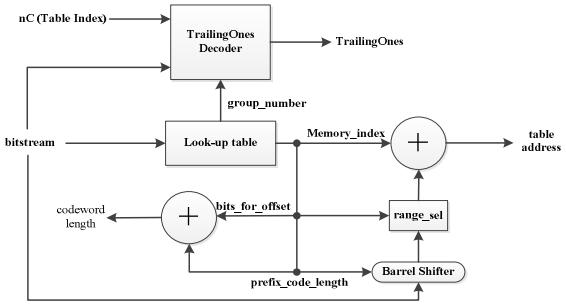

Fig. 2 Group detector and TrailingOnes decoder

The block diagram of context adaptive variable length coding decoder is shown in Fig. 1. The variable length code table in AVC/H.264 is unsuitable to be implemented by using group-based look up table method when the codeword length and similarity are different from each other. Hence, the arithmetic operation is used to decode the variable length code directly and hence the memory usage can be reduced concurrently. On the other hand, the variable length code tables can be merged efficiently by changing the order of symbols in variable length code tables and hence the memory usage is reduced, too. In the following subsections, the proposed architecture would be analyzed to fit the real-time constraint of level 4.2 in AVC/H.264.

A. Group-based Look Up Table and TrailingOnes Decoder

For the purpose of reducing the memory usage, B.J. Shieh *et al*. [10][11] have developed the group-based look up table methods because the group-searching scheme used numerical properties and arithmetic operation to perform efficiently with high speed and reduced the memory usage dramatically. To increase throughput and reduce memory usage, a group detector has been proposed for variable length code tables in AVC/H.264 that decoded by group-based look up table method and Fig. 2 is the architecture of group detector. After analyzing each variable length code table of CAVLD in AVC/H.264, the variable length code tables are grouped into corresponding groups depending on their codeword length. Table I shows the partial grouping result. According to codeword lengths of symbols, the variable length code tables are grouped into corresponding groups.

To output prefix_code_length, bits_for_offset, group_number and memory_index depending on prefix_code of input bitstream, a look up table module would be used in Fig. 2. In the proposed architecture, prefix_code_length and bits_for_offset can be used to determine the memory address offset and then add to memory_index to generate the memory address and then read the symbol stored in the memory. Except for decoding symbol *CoeffToken*, the codeword length of symbol is equal to prefix_code_length plus bits_for_offset. In *CoeffToken* decoding process, since the sign flag of *TrailingOnes* is decoded at the same cycle for processing cycle reduction, the codeword length would be the summation of prefix_code_length, bits for offset, and *TrailingOnes*. In addition, most of *TrailingOnes* can be decoded by inverting the last two bit of codeword as the rows colored by pink depicted in Table I. For other cases, we used the particular methods to decode *TrailingOnes* and the details are depicted in Table I The row colored by green means *TrailingOnes* is a particular output value, the rows colored by blue denotes *TrailingOnes* is obtained by inverting last one bit then output last two bits, the rows colored by orange means the last two bits is *TrailingOnes*. In addition, Group detector generates a *group_number* which selects the relative *TrailingOnes* decoder. Therefore, only *TotalCoeff* must be stored in the memory. The codeword length can be calculated at the first cycle and it did not be stored in the memory, and then the memory usage is decreased and furthermore the throughput is increased.

Table I Partial result of grouping

T1s	TotalCoeff	Codewords	prefix_code	Group	prefix_code_length	bits for offset	Codeword length	Memory index
0	0	1	0	0	1	0	1	35
1	1	01	01	1	2	0	2	34
2	2	001	001	2	3	0	3	36
3	3	00011	00011	3	5	0	5	38
1	2	000100	00010	4	5	1	6	58
0	1	000101						
3	4	000011	000011	5	6	0	6	37
3	5	0000100	000010	6	6	1	7	81
2	3	0000101						
3	6	00000100	000001	7	6	2	8	0
2	4	00000101						
1	3	00000110						
0	2	00000111						

B. *LEVEL Decoder*

To support the bit-rate required in level 4.2 of AVC/H.264, it is necessary to increase the throughput of CAVLD decoder; therefore, a multi-symbol *LEVEL* decoder is presented to decode two *LEVELs* once. However, in AVC/H.264 standard, *SuffixLength* of next *LEVEL* decoding process is depending on *LevelCode* in current *LEVEL* decoding process, therefore, the critical path is too long to accommodate targeted clock rate. A method to break the data dependence between *SuffixLength* of next *LEVEL* decoding process and *LevelCode* of current *LEVEL* decoding process have been proposed by H. Y. Lin *et al.* [12]. Hence, the modified level decoding process is presented as shown in Fig. 3. The *Bitstream_2* in Fig. 3 is the input bitstream by shifting the *LEVEL_1* codeword length bits. In addition, since *LEVEL_2* decoder never decodes the first *LEVEL* in decoding process and hence *SuffixLength* is never equal to zero in *LEVEL_2* decoder. Then, some decoding processes, such as checking *SuffixLength* equal to zero could be skipped in *LEVEL_2* decoder. *SuffixLength* calculator also can be reduced from *LEVEL_2_SuffixLength* to *LEVEL_1_SuffixLength* decoder. After reducing the critical path of *LEVEL* decoder from *Bitstream_1* to output of *LEVEL_2*, the proposed architecture of *LEVEL* decoder could decode at most two *LEVEL*s in one cycle with clock rate 108MHz and hence the throughput of CAVLD is increased.

C. *RunBefore Decoder*

After decoding symbol *TotalZeros*, according to the value of *TotalZeros*, *RunBefore* decoder decodes the number of zeros between the two nonzero coefficients depending on the value of *Zeroleft*. Inspired from consecutive occurrence of symbol *LEVEL*, *RunBefore* also has the similar property to *LEVEL*. Hence, to increase the throughput of the CAVLD, the multi-symbol decoder for *RunBefore* is also used in CAVLD.

By predicting the codeword length, it is feasible to decode all possible codewords simultaneously because the symbol with short code length means that it occurs with higher probability. Therefore, G.G. Lee *et al.* [13] gather many statistics on the relationship between *Zeroleft* and codeword length of the following occurred symbols and Table II shows the predicted codeword length scheme according to *Zeroleft*.

The architecture of multi-symbol *RunBefore* decoder is illustrated in Fig. 4. The initial value of *Zeroleft* is equal to the value of *TotalZeros*, after one of *RunBefore* has been decoded; the second value of *Zeroleft* would be updated to the value of *TotalZeros* minus the value of *RunBefore*.

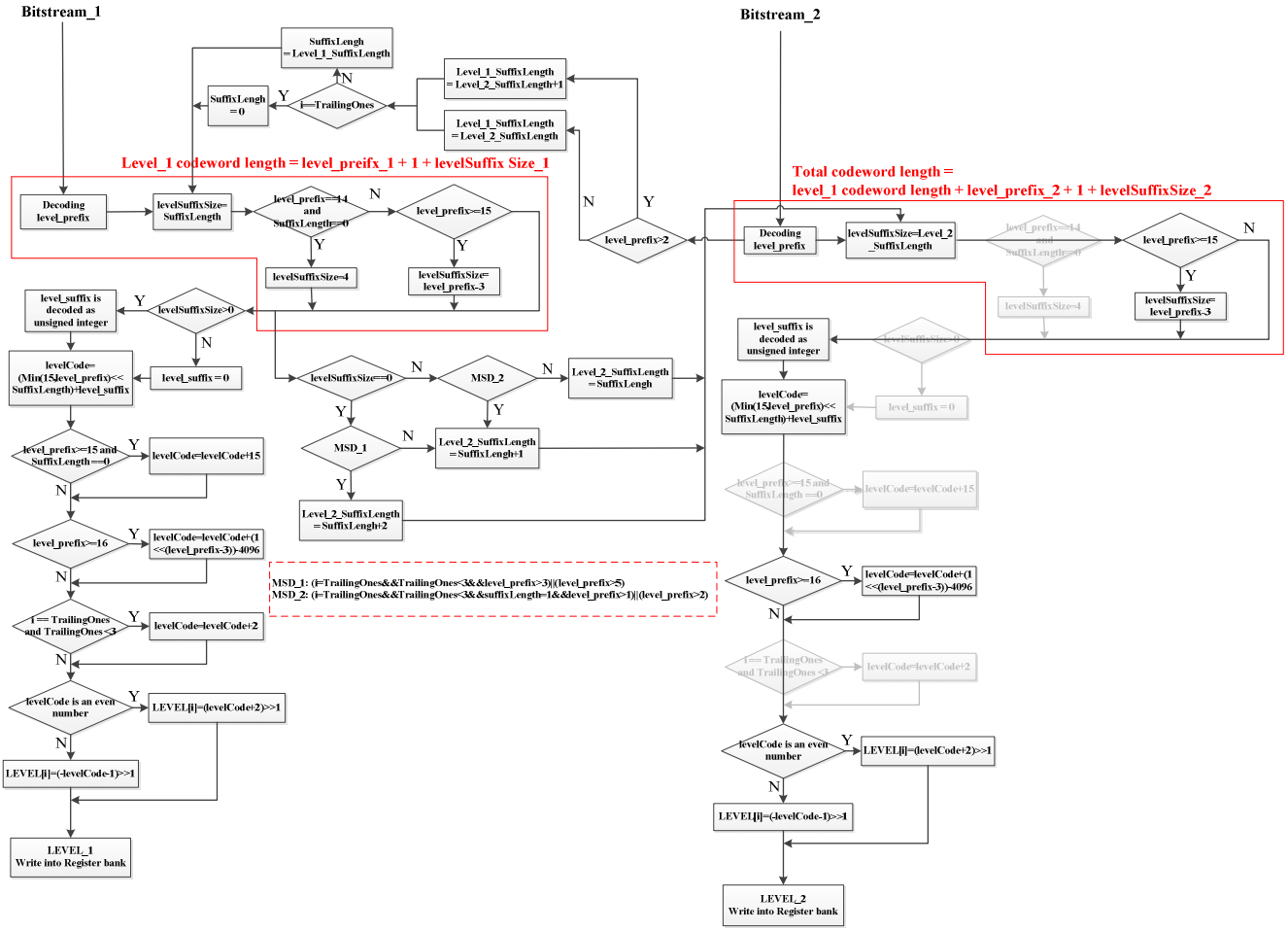

Fig. 3 Decoding process of Multi-symbol LEVEL decoder

Table II The prediction codeword length for different *Zeroleft*

Zeroleft	1	2	3	4	5	6	>6
First codeword length	1	1	2	2	2	3	3
Second codeword length	1	1	2	2	2	2	3

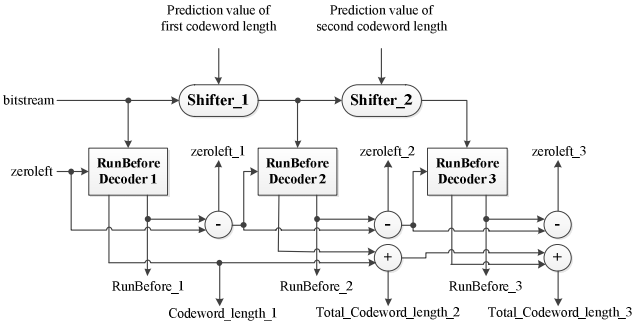

Fig. 4 Architecture of multiple symbols RunBefore decoder

The *RunBefore* decoding process would finish until *Zeroleft* is equals to zero. Subsequently, an availability check for *RunBefore* is needed. Hence, according to availability flag of each symbol, *RunBefore* decoder outputs the correct total codeword length and *Zeroleft* for the next cycle decoding process. The multi-symbols *RunBefore* decoder can decode at most three symbols at one cycle. The throughput rate would be increased when accurate prediction scheme is provided.

D. Fast Zero Insertion

The zeros need to be inserted into non-zero coefficients after decoding *RunBefore*. To decrease processing cycles of zero insertion, the proposed architecture is shown in Fig. 5 and Fig. 6 is an example of zero insertion. The initial value of index register is the value of *TotalCoeff* plus the value of *TotalZeros*, after the value of *RunBefore* has been decoded, the last non-zero coefficient would be move to the location that *RunBefore_Index_1* indexed and the next non zero coefficient would be moved to the location that *RunBefore_Index_2* indexed and etc. and hence there is no cycle would be spent to move the other coefficients with no zeros before the coefficients.

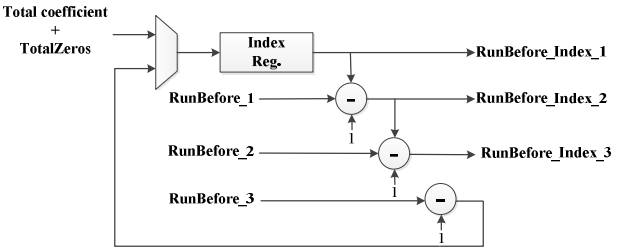

Fig. 5 Architecture of zero insertion

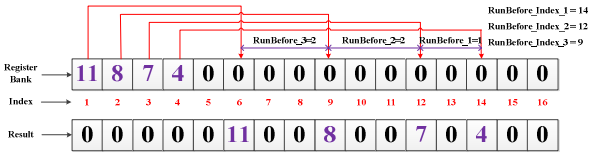

Fig. 6 Example of zero insertion

V. EXPERIMENTAL RESULTS AND COMPARISON

The proposed design is implemented using Verilog HDL and synthesized on 0.18um CMOS technology provided by TSMC. The gate count is 17.2K gates at a clock constrain of 108MHz and all the functionalities AVC/H.264 is verified by the conformance bitstream [14]. To test the capability of proposed CAVLD, the parser processing should be considered simultaneously. We encoded sequences in HEVC of MPEG by using AVC/H.264 reference software [15] because there is no conformance bitstream encoded whose resolution and frame rate could not achieve the worst case of level 4.2 limitation, i.e. 1920x1088@60fps.

In order to ensure the proposed architecture can support the level limitation (Maximum bits rate is 62.5 Mbits/s), the information of the encoded test bitstream and the corresponding required cycles per MB is shown Table III include 48 cycles for writing the data into FIFO between IQIT and entropy decoder. To guarantee the real-time decoding in worst cases, the bitstream with 59.7 Mbits/s should be decodable. After analyzing the results of Table III, the proposed architecture also meets the real time constraint in AVC/H.264 mode.

Table III Information of encoded test bitstream and required cycles per MB

Sequence	BQTerrance								
Resolution	1920x1088								
Frame Rate	60 fps								
GOP Structure	IBBP								
Initial QP	28			30			32		
Bitrate (bits/s)	59.7M			39.87M			26.74M		
Bitstream Name	BQTerrace_1			BQTerrace_2			BQTerrace_3		
	Parser	CAVLD	Total	Parser	CAVLD	Total	Parser	CAVLD	Total
Intra4x4	23	190	213	23	173	196	23	165	188
Intra8x8	11	162	173	11	151	162	11	136	147
Intra16x16	5	167	172	5	163	168	5	152	157
P	47	167	214	47	144	191	47	129	176
B	47	169	216	40	136	176	40	142	182

Table IV Comparison with related works

	[15]	[16]	[12]	Proposed
Technology	UMC 0.13um	TSMC 0.18um	TSMC 0.18um	TSMC 0.18um
Gate Count	17,202	9,943	6,117	17,225
Throughput (macroblocks/sec)	1.77×10^6			AVC/H.264: 2.16×10^6
Average cyclesper macroblocks	141			50
Clock Rate	250MHz	175MHz	213MHz	108MHz
Memory Usage (bit)	5,120	1,152	0	2816
Specification	HD 1080P Frame Rate: 60	HD 1080I Frame Rate: 30	HD 1080P Frame Rate: 30	HD 1080P Frame Rate: 60

The experimental results show that proposed architecture could process 2.16×10^6 MBs per second in AVC/H.264 mode and only need 50 cycles to decode one MB in AVC/H.264 mode in average when the numbers of cycles for writing the data into IQIT FIFO did not be considered.

Table IV shows the comparison results with the other existing design. M. Alle *et al.* [15] proposed architecture has lower gate counts but its memory usage and clock rate is larger than our design. H.C. Chang *et al.* [16] proposed architecture have lower gate counts and memory usage but it have higher clock rate and only support HD1080I with 30 fps. H.Y. Lin *et al.* [12] proposed architecture also have lower gate counts and memory usage but it also have higher clock rate and only support HD1080P with 30 fps. The experimental and comparison results shows that the presented CAVLD decoder has been shown to surpass the state-of-arts as stated in the literature and the supported specification is also higher than related works.

VI. Conclusions

In this paper, high-throughput context adaptive variable length decoders that can support AVC/H.264 HP@level 4.2 have been proposed. To achieve level limitation in AVC/H.264, the multi-symbol decoder for *LEVEL* and *RunBefore* is also used in reconfigurable variable length decoder to reach high throughput-rate.

The proposed design is implemented and verified using Verilog HDL and is synthesized adopting 0.18um TSMC technology. The synthesis result shows that the gate count is about 17.2K gates at a clock constrain of 108MHz. The experimental results show that our architecture working at 108 MHz can process 2.16×10^6 macroblock per second in AVC/H.264 mode and only 50 cycles are needed to decode one MB in AVC/H.264 mode. Hence, the proposed CAVLC decoder is presented with the features of high throughput.

References

[1] ITU-T Recommendation H.264 Advanced video coding for generic audiovisual services, Draft, March 2005.

[2] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H. 264/AVC video coding standard," *Circuits and Systems for Video Technology, IEEE Transactions on,* vol. 13, pp. 560-576, 2003.

[3] S. Y. Tseng and T. W. Hsieh, "A pattern-search method for H. 264/AVC CAVLC decoding," *IEEE Int. Conf. Multimedia Expo*, 2006, pp. 1073-1076.

[4] S. Park, K. Min, J. Chong, "The new memory-efficient hardware architecture of CAVLD in H. 264/AVC for mobile system," *ISCIT*, 2009, pp. 204-207.

[5] J. Bae, J. Cho, B. Kim, and J. Baek, "High performance VLSI design of run_before for H. 264/AVC CAVLD," *IEICE Electronics Express,* vol. 8, pp. 950-955, 2011.

[6] T.L. da Silva, J.A. Vortmann, L.V. Agostini, A.A. Susin, S. Bampi, "Low Cost and Memoryless CAVLD Architecture for H. 264/AVC Decoder," *SBCCI*, 2009, pp. 280-285.

[7] T. Silva, F. Pereira, A. Susin, S. Bampi, L. Agostini," High performance and low cost architecture for H. 264/AVC CAVLD targeting HDTV," *ACM*, 2009, pp. 41.

[8] C. C. Lo, C. W. Hsu, and M. D. Shieh, "Area-Efficient H. 264 VLC Decoder Using Sub-tree Classification," *IIH-MSP*, 2010, pp. 284-287.

[9] T.G. George, N. Malmurugan, "The Architecture of Fast H. 264 CAVLC Decoder and its FPGA Implementation," *IIH-MSP*, 2007, pp. 389-392.

[10] B.J. Shieh, Y.S. Lee, C.Y. Lee, "A new approach of group-based VLC codec system with full table programmability," *Circuits and Systems for Video Technology, IEEE Transactions on*, 11 (2001) 210-221.

[11] B.J. Shieh, T.Y. Hsu, C.Y. Lee, "A new approach of group-based VLC codec system," *ISCAS*, 2000, pp. 609-612 vol. 4.

[12] H.Y. Lin, Y.H. Lu, B.D. Liu, J.F. Yang, "A highly efficient VLSI architecture for H. 264/AVC CAVLC decoder," *Multimedia, IEEE Transactions on,* 10 (2008) 31-42.

[13] G.G. Lee, C.C. Lo, Y.C. Chen, H.Y. Lin, "M.J. Wang, High-throughput low-cost VLSI architecture for AVC/H. 264 CAVLC decoding," *Image Processing, IET*, 4 (2010) 81-91.

[14] H.264/AVC reference software JM16.2, http://iphome.hhi.de/suehring/tml/.

[15] M. Alle, J. Biswas, and S. Nandy, "High performance VLSI architecture design for H. 264 CAVLC decoder," *ASAP*, 2006, pp. 317-322.

[16] H. C. Chang, C. C. Lin, and J. I. Guo, "A novel low-cost high-performance VLSI architecture for MPEG-4 AVC/H. 264 CAVLC decoding," *ISCAS*, 2005, pp. 6110-6113 Vol. 6.