EURASIP Journal on Image and Video Processing
a SpringerOpen Journal

RESEARCH **Open Access**

Fast CAVLD of H.264/AVC on bitstream decoding processor

Jung-Han Seo[1], Hyun-Ho Jo[1], Dong-Gyu Sim[1*], Doo-Hyun Kim[2] and Joon-Ho Song[2]

Abstract

This paper presents a fast context-based adaptive variable-length decoding (CAVLD) method of H.264/AVC with a very long instruction word-based bitstream processing unit (BsPU) designed for entropy decoding of multiple video formats. A new table mapping algorithm for the coeff_token, level, and run_before syntax elements of the quantized transform coefficients is proposed, and many branch operations are removed by utilizing several designated instructions in the BsPU. By applying designated instructions and the proposed table mapping algorithm to CAVLD, we found that the proposed fast CAVLD method achieves an increase of approximately 47% in the decoding speed and a reduction of approximately 59% in memory requirements for the table mapping.

Keywords: CAVLD, H.264/AVC, Entropy decoding, Video compression

1. Introduction

The recent evolution of digital media technologies has enabled users to use numerous video applications, even on mobile devices, through diverse network channels. Along with these communication and hardware implementation technologies, various video compression technologies have been developed not only by standard bodies, but also by private companies. The ISO/IEC Moving Picture Expert Group (MPEG) and the ITU-T Video Coding Expert Group (VCEG) have released multiple video compression standards: MPEG-1/2/4 and H.261/262/263/264. Among them, H.264/AVC is considered to be a major video coding standard, owing to its coding efficiency. It was developed by a Joint Video Team, which was a temporary group formed by the MPEG and VCEG bodies [1]. These video coding standards have been widely used for mass production applications such as broadcasting, media players, and mobile video services. Moreover, these video codecs have been vigorously implemented in hardware circuits for huge markets. On the other hand, there are many other video coding technologies, such as the On2 series and VC-1. Even though these technologies were not developed by international standards bodies, they have been widely used in the market, especially for personal computer (PC) applications with software implementation. However, since the advent and widespread use of smartphones, video codecs are no longer just for the PC platform. As smartphones proceed in replacing conventional cellular phones, many video codecs are being considered for mobile devices. Mobile devices have one major limitation: light batteries. Video codecs for mobile devices are commonly implemented in hardwired circuits for lower power consumption. However, it is difficult to implement all video codecs in hardware logic because of development costs, design time, and lack of flexibility. For new video codecs, many developers should be prepared for long development times and high financial costs in developing and debugging hardware logics.

On the other hand, software-based multi-format video decoders have been developed on multi-core platforms. Various parallel implementations are utilized to alleviate the implementation costs of multi-format decoders with hardwired circuits. On the multi-core platform, it is possible to develop multi-format decoders that allow easy performance evaluation with minimum cost and effort compared to other hardwire designs. In addition, the multi-core platform requires less power for fast decoding of multi-format multimedia content due to the low clock speed. Parallel video decoders with multi-core platform can be implemented based on both data-level and functional-level parallelism. However, considering

* Correspondence: dgsim@kw.ac.kr
[1]Department of Computer Engineering, Kwangwoon University, Wolgye-dong, Nowon-gu 447-1, Seoul 139-701, Republic of Korea
Full list of author information is available at the end of the article

resolution, scalability, and performance in parallelism, the macroblock-level parallelism approach, which is a form of data-level parallelization, is widely used for video decoders [2–4]. For macroblock-level parallelism, the 2D wave-front approach is widely known and used. This method can perform parallel decoding of multiple macroblocks without any decoding dependencies. However, this 2D wave-front approach cannot be used for entropy decoding because of bit-by-bit dependency in a slice, even though back-end decoding can be parallelized with the multi-core platform [5]. Because the performance of parallel video decoders is highly influenced by sequential parts such as entropy decoding, the high-performance entropy decoder is an essential prerequisite for parallel video decoders. In addition, entropy decoding is one form of bottleneck that can decrease decoding throughput not only in parallel decoders but also in sequential decoders, especially for high-bitrate streams.

H.264/AVC supports two entropy coding techniques: context-based adaptive variable-length coding (CAVLC) and context-based adaptive binary arithmetic coding (CABAC). CABAC provides a coding gain of approximately 10% ~ 15%, compared with CAVLC. However, it has high computational complexity and requires a large number of serial operations. With software-based implementations, it is difficult to achieve real-time decoding for CABAC for such high bitrate ranges [6–8]. The CAVLC mode is also widely used, not only for mobile devices, but also for many other high-quality applications through baseline and main profiles. Although context-based adaptive variable-length decoding (CAVLD) has less computational complexity than context-based adaptive binary arithmetic decoding (CABAD), the CAVLD is also considered to be one of the most complex variable-length decoding (VLD)-based entropy decoders in the market. We can state that a platform capable of implementing CAVLD in real time can also decode other VLD-based entropy decoders. In this paper, a fast CAVLD implementation is presented with a developed bitstream processing unit (BsPU) based on a table mapping algorithm. Note that the BsPU has multiple designated instructions for entropy decoding. The new table mapping algorithm is applied to several syntax elements such as coeff_token, level, and run_before. In addition, certain designated BsPU instructions are efficiently used to reduce the number of branches in arithmetic computations.

The rest of the paper is organized as follows. In section 2, the conventional entropy decoding algorithms and the BsPU are introduced. In section 3, the proposed fast CAVLD based on a new table mapping algorithm is presented. In section 4, experimental results are shown and discussed. Finally, concluding remarks are given in section 5.

2. H.264/AVC CAVLD and its conventional acceleration algorithms

CAVLD is considered to be a bottleneck in both parallel and sequential decoders of H.264/AVC because of its bit-by-bit dependency. Furthermore, as video resolution is increasing based on market demands, the common bitrate for high-resolution videos such as full high definition (full HD) now ranges from 10 to 20 megabits per second (Mbps) for high-quality applications. For such high-bitrate applications, a portion of the complexity for the entropy decoding would significantly increase. For fast CAVLD of H.264/AVC, there are several algorithms such as table mapping, which can improve the overall latency of entropy decoding without parallelism. In order to develop an efficient table mapping approach for multi-stage pipelined processors, it is necessary to consider not only the memory requirements and the number of memory accesses, but also the number of conditional branches.

2.1 CAVLD of H.264/AVC

CAVLC is one of the entropy coding methods of H.264/AVC to encode quantized transformed coefficients in 4 × 4 (or 2 × 2) blocks. CAVLC employs some characteristics to improve coding efficiency as follows: (1) Run-level coding is more effective to encode strings of zero coefficients after zig-zag scanning. (2) After zig-zag scanning, non-zero coefficients in high frequency are likely to be ±1. CAVLC refers to the number of ±1 coefficients as *TrailingOnes* and encodes it. (3) The number of non-zero coefficients is highly correlated with those of neighboring blocks. CAVLC refers to the number of non-zero coefficients as *TotalCoeff* and uses different variable-length coding (VLC) tables depending on the number of non-zero coefficients of neighboring blocks. (4) The levels of non-zero coefficients tend to be higher near low-frequency components. Hence, CAVLC adaptively selects the VLC table to encode the current level value depending on the associated previous coded level. The zig-zag scanned coefficients are represented by five types of syntax elements. These syntax elements are defined as follows:

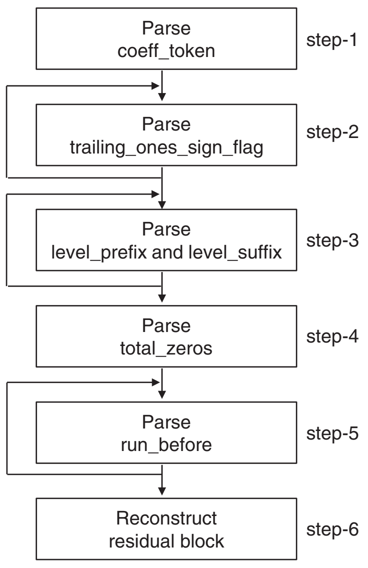
1. coeff_token: This syntax element specifies both *TotalCoeff* and *TrailingOnes*.
2. trailing_ones_sign_flag: This syntax element specifies the sign of a coefficient in *TrailingOnes.*
3. level_prefix and level_suffix: These syntax elements specify the value of a non-zero coefficient level except for *TrailingOnes*.
4. total_zeros: This syntax element specifies the number of zero coefficients preceding the last non-zero coefficient.
5. run_before: This syntax element specifies the number of consecutive zero coefficients preceding each non-zero coefficient including *TrailingOnes*.

Figure 1 shows a flowchart of CAVLD and the decoding procedure with an example bitstream of a 4 × 4 block. The process of CAVLD consists of six steps, as shown in Figure 1. In this example, the average number of non-zero transform coefficients of the top and left blocks is assumed to be five. After the decoding procedure for the 4 × 4 block, zig-zag scanned coefficients, {0, 3, 0, 1, −1, −1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0}, are reconstructed.

2.2 Conventional fast entropy decoding algorithms

Many previous studies on fast implementation of CAVLD have proposed the use of hardwire logic [9–11]. However, hardware implementations have many drawbacks such as the long development period, the large silicon area, and the low reusability of multi-format video decoders, as previously mentioned. This paper focuses on fast entropy decoding algorithms for processor-based implementations. The simplest approach is the table lookup by sequential search (TLSS) as implemented in the joint model (JM) reference software [12]. TLSS finds symbols by comparing all possible codewords with part of the input bitstream. This approach requires a large number of comparisons and therefore cannot be used for real-time applications. Table lookup by binary search (TLBS) has also been proposed to improve the entropy decoding speed. The codewords are rearranged into a binary search tree structure, and the symbols can be extracted from the tree by using binary search. TLBS offers much better performance than TLSS; however, a more efficient implementation is needed for real-time applications.

On the other hand, several conventional algorithms can be used to reduce the number of memory accesses. Moon et al. [13] proposed a fast decoding method for coeff_token and run_before in the CAVLD of H.264/AVC. It is based on arithmetic operations and works by considering the characteristics of the associated codewords. This approach does not require memory access and repeated comparisons; however, many conditional branches might be involved, depending on the codewords used. Another memory-efficient CAVLD for fast entropy decoding [14] was presented to accelerate coeff_token, run_before, and total_zeros based on arithmetic operations. In this study, codewords were categorized, and several arithmetic operations were proposed for each category. This algorithm is known to be more effective than Moon's algorithm; however, it also requires many conditional branches. Frequently occurring conditional branches are not suitable for multi-stage pipelined processors because wrong branch prediction can significantly reduce performance of the processors. Although many multi-stage pipelined processors have their own branch prediction units, the performance of the modules depends on the applications. In particular, the performance of instruction-level parallelism can be further degraded by wrong branch prediction for many digital signal processors (DSP) based on the very long instruction word (VLIW) architecture. For multi-stage pipelined processors, a reduction in the number of conditional branches is also important along with a reduction in the number of memory accesses and the table memory requirement [15]. Many table-based algorithms have been proposed for a number of entropy decoders based on VLD because these approaches are known to


Parse coeff_token
step-1
Parse trailing_ones_sign_flag
step-2
Parse level_prefix and level_suffix
step-3
Parse total_zeros
step-4
Parse run_before
step-5
Reconstruct residual block
step-6

bitstream : 101001110010111101101

Syntax element	Codeword	Value
coeff_token	1010	*TotalCoeff*=5, *TrailingOnes*=3
trailing_ones_sign_flag	0	+
trailing_ones_sign_flag	1	-
trailing_ones_sign_flag	1	-
level	1	+1
level	0010	+3
total_zeros	111	3
run_before	10	1
run_before	1	0
run_before	1	0
run_before	01	1

Reconstructed residual : 0,3,0,1,-1,-1,0,1,0,0,0,0,0,0,0,0

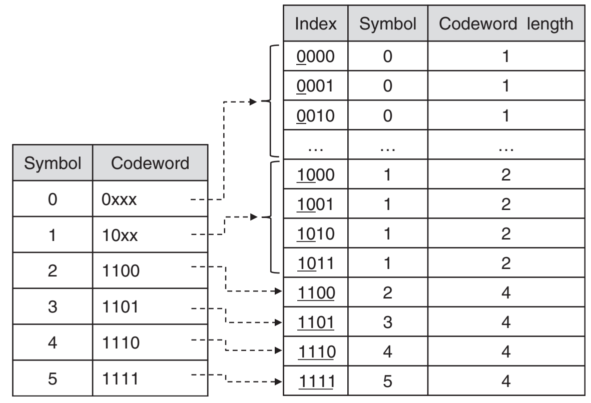
Figure 1 CAVLD flowchart and procedure for an example 4 × 4 block.

be fast and have a large redundant memory allocation [16–19].

As shown in Figure 2, the table mapping algorithm tries to find the correct symbol using memory addressing with the maximum-length syntax element (in bits). Note that the length is that of the longest codeword for the syntax element. However, the table memory requirement increases exponentially by 2^n, where n is the length of the longest codeword. In addition, such high memory requirements can cause frequent cache misses, which have a significant impact on overall performance. Thus, the memory requirement needs to be reduced. One fast entropy decoding algorithm based on table mapping was proposed for coeff_token in CAVLD [20]. This algorithm divides the mapping table for coeff_token depending on the number of leading zeros. Multiple tables are required, but the total memory requirements can be reduced using multiple small tables. However, in practice, this approach is not suitable for pipelined processors because conditional branches might be involved in the choice of one of the four tables before table indexing.

For coeff_token decoding, a multi-level table mapping algorithm was proposed [21]. The first 8-bit codeword is decoded using a table mapping algorithm, and the rest of the bits are then decoded with one branch operation and an additional table mapping algorithm. This algorithm reduces the amount of table memory; however, it frequently suffers from pipeline stalls caused by branch operations.

For the level and run_before syntax elements, the multi-level table mapping algorithms [22] are also used for fast decoding. The level syntax element is represented with the Exp-Golomb code, the codewords of which have leading zeros which are less than 15. The first 8-bit codeword is decoded using table mapping, and the rest of the bits are decoded using arithmetic operations by employing characteristics of Exp-Golomb.



Symbol	Codeword
0	0xxx
1	10xx
2	1100
3	1101
4	1110
5	1111

Index	Symbol	Codeword length
0000	0	1
0001	0	1
0010	0	1
...	...	...
1000	1	2
1001	1	2
1010	1	2
1011	1	2
1100	2	4
1101	3	4
1110	4	4
1111	5	4

Figure 2 Original codeword table and new table for table mapping.

Codewords with more than 15 leading zeros before the separator require exceptional handling. This algorithm also requires several conditional branches. For the run_before syntax, one of multiple tables is selected based on the number of zeros among the rest of coefficients to be decoded. When the number of zeros is greater than six, a larger mapping table is required. In this case, two-stage table mapping should be employed to reduce the amount of memory. For the remaining cases, one table mapping algorithm can perform the entire entropy decoding. This algorithm can significantly reduce the table memory requirement; however, many conditional branches should be involved, and these can reduce overall performance of entropy decoding with a multi-stage pipelined processor.

2. 3 Bitstream processing unit

Entropy decoding is a process whereby a codeword is converted into a number or symbol. The codeword is loaded from external memory into internal memory in the entropy decoding process. In addition, the operations in the entropy decoding are mostly bit-level operations. However, frequent accesses to the external memory can significantly degrade decoding performance. To resolve this problem and achieve real-time decoding with a flexible design, it is desirable to use a designated bitstream decoding processor.

The combination of a bitstream decoding processor and parallel back-end decoding is useful for implementing a real-time multi-format decoder with low power consumption. This paper uses the BsPU that consists of a reconfigurable processor (RP) [23] and multiple instructions for multi-format bitstream decoding. Note that we also developed the BsPU by developing new instructions and hardwire logics on the RP, which is an elaboration of ADRES [24]; however, in this paper we will focus on the fast H.264/AVC CAVLD on the BsPU. The BsPU is based on a VLIW architecture and six additional instructions for bitstream decoding, as shown in Table 1. The Showbits instruction is for outputting as many bits as required. The Getbits instruction is for outputting as many bits as required and moving a pointer to the next bit after the bits used. The Skipbits instruction is for skipping as many bits as required. These three instructions are widely used for bitstream access and can be used to reduce the number of cycles required. The count leading zeros (CLZ) instruction is for determining the number of consecutive zeros in a bitstream and is used in reduced instruction set computer processors [25] and many other DSPs. This instruction would be very useful for decoding some syntax elements coded with Exp-Golomb. The BsPU contains an instruction called I_ONERETURN with two input parameters, A and B. When A is 1, this operation returns B; otherwise it

Table 1 BsPU instructions for multi-format bitstream decoding

Instruction	Description
Showbits(*n*)	The Showbits instruction reads *n* bits from the input value
Getbits(*n*)	The Getbits instruction reads and skips *n* bits from the input value
Skipbits(*n*)	The Skipbits instruction skips *n* bits from the input value
CLZ	The CLZ instruction counts leading zeros
I_ONERETURN(*A*, *B*)	If input *A* is 1, return input *B*; if input *A* is any other value, return 0
I_MIN(*A*, *B*)	I_MIN instruction returns the minimum value between *A* and *B*

returns '0' as an output. This operation can be implemented using multiplication; however, the RP requires 3 cycles for multiplication. The I_ONERETURN instruction requires only 1 cycle and is therefore more effective than multiplication for removing conditional branches.

3. Proposed fast CAVLD for BsPU

In this paper, a new fast CAVLD algorithm which uses dedicated instructions for table mapping is proposed. To reduce the number of branch operations, the various instructions are carefully organized to improve overall performance. As the three syntax elements, coeff_token, level, and run_before occupy a large fraction of computation time, we propose a new table mapping algorithm for them with several instructions in the BsPU. In contrast, the trailing_ones_sign_flag can be easily decoded because the length is known as the *TrailingOnes*. For the total_zeros syntax element, we also employed the table mapping algorithm. However, the decoding algorithm is conceptually the same as the simple table mapping algorithm. It allocates 2^9 entries for a table for total_zeros_syntax element since the longest code word has 9 bits.

3. 1 Proposed fast decoding for the coeff_token syntax element

Four codeword tables are used for the coeff_token syntax element in H.264/AVC CAVLD. One of these four tables is selected depending on the average number of non-zero transform coefficient levels of the top and left blocks, which is denoted *nC*, as shown in Table 2. Three of these ones are variable-length tables, and the last one is a fixed-length table. For the variable-length tables, each codeword consists of three parts: a prefix, the number '1', and a set of remainder bits. The prefix consists of consecutive zeros, and the remainder bits are an arbitrary sub-codeword with a length of less than or equal to 3. In the proposed algorithm, each of the three VLC tables is divided into multiple divisions depending on the number of consecutive zeros in order to achieve fast table mapping with minimum memory size. After implementation of a single table mapping for each table, there will be 2^{16} entries, because the longest codeword bit length is 16. In the proposed algorithm, the length of divided divisions for VLC T0 is 8 (=2^3) to avoid any conditional branches regardless of the number of entries. As a result, the required number of entries is 120 (=15 × 2^3) for the VLC T0, because the number of divisions is 15. On the other hand, the required number of entries for the fixed-length coding (FLC) case is 2^6, and the table can be accessed with a 6-bit address. Figure 3 shows a part of the reorganized tables for coeff_token. *TrailingOnes* and *TotalCoeff* are assigned by the remainder and quotient from division of the 'coeff' in the second table by four, respectively. Figure 4 shows two different methods to decode the syntax element for the VLC and FLC cases, depending on the value of *nC*. A branch operation is needed to distinguish the two cases. This paper proposes a single consolidated decoding algorithm for coeff_token without any branch operations. The following is the pseudo-code for the consolidated decoding algorithm of coeff_token.

```
1) i=coeff_token_table_index[nC];
2) buf = Showbits(32);
3) zero_num = CLZ(buf, temp);
4) flag = (i<3);
5) flag2 = I_ONERETURN(flag, zero_num);
6) coeff_token_vlc_table = coeff_token_nc[i][flag2];
7) Skipbits(flag2+flag);
8) suffix = Showbits (6-I_ONERETURN(flag,3));
9) coeff_token = coeff_token_vlc_table[suffix].coeff
10) Skipbits(coeff_token_vlc_table[suffix].length);
```

In the first step, the table index is obtained, using *nC*. Then, the 32-bit bitstream is read. In the third step, zero_num is set to the number of leading zeros in those 32 bits. In step 4, the flag indicates whether a VLC table

Table 2 Four tables for the coeff_token depending on *nC*

Table number	Selection condition	Table name
0	0 < = *nC* < 2	VLC T0
1	2 < = *nC* < 4	VLC T1
2	4 < = *nC* < 8	VLC T2
3	8 < = *nC*	FLC

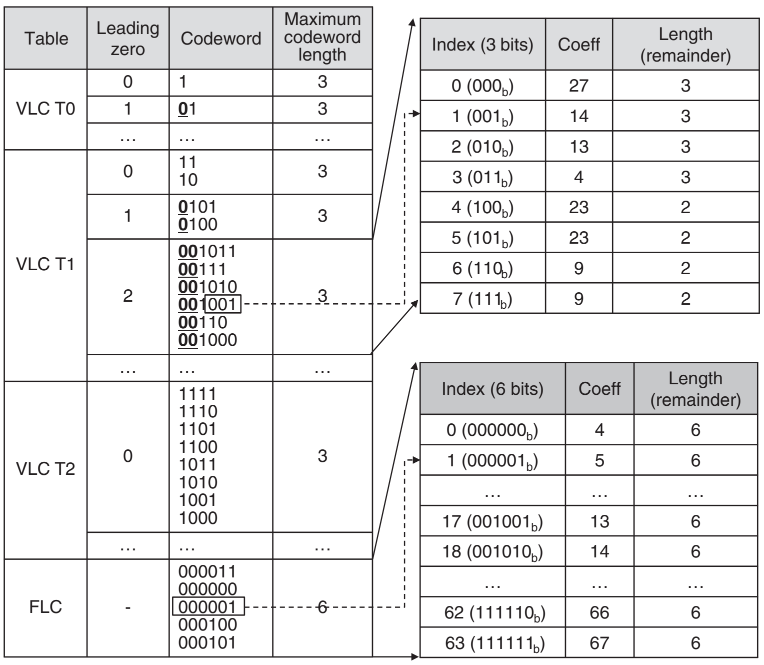


Table	Leading zero	Codeword	Maximum codeword length
VLC T0	0	1	3
	1	**0**1	3
	…	…	…
VLC T1	0	11 10	3
	1	**0**101 **0**100	3
	2	**00**1011 **00**111 **00**1010 **00**1001 **00**110 **00**1000	3
	…	…	…
VLC T2	0	1111 1110 1101 1100 1011 1010 1001 1000	3
	…	…	…
FLC	-	000011 000000 000001 000100 000101	6

Index (3 bits)	Coeff	Length (remainder)
0 (000_b)	27	3
1 (001_b)	14	3
2 (010_b)	13	3
3 (011_b)	4	3
4 (100_b)	23	2
5 (101_b)	23	2
6 (110_b)	9	2
7 (111_b)	9	2

Index (6 bits)	Coeff	Length (remainder)
0 (000000_b)	4	6
1 (000001_b)	5	6
…	…	…
17 (001001_b)	13	6
18 (001010_b)	14	6
…	…	…
62 (111110_b)	66	6
63 (111111_b)	67	6

Figure 3 **Proposed two-stage coeff_token decoding table.**

or the FLC table is needed. The indicator flag2 contains the number of zeros in the VLC case (or is zero in the FLC case). Therefore, this number serves as the division index for the VLC case. In the FLC case, there is only one table and the division index is zero. In the sixth step, the address of the desired division is obtained using the VLC table index. In the FLC case, the start address of the FLC table is returned. In the seventh step, the bits in the prefix and the separator '1' are discarded in the VLC case; however, no bits are discarded in the FLC case. In the eighth step, 3 and 6 bits are loaded for the VLC and FLC case, respectively, in order to specify the address for the mapping table. In the ninth step, the desired value is obtained from the reorganized table using the address. In the last step, the processed bits are discarded to prepare for the next bitstream decoding operation.

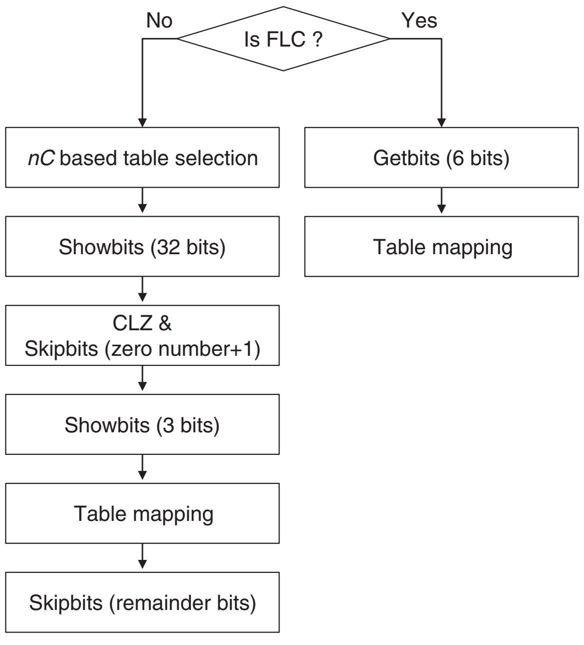


Figure 4 **Two different decoding modes for VLC and FLC.**

3.2 Proposed fast decoding for the level syntax element

H.264/AVC uses seven VLC tables for the level syntax element. For the first level value, the Level_VLC0 or Level_VLC1 table is used depending on the context. If the number of non-zero coefficients is greater than ten and the number of *TrailingOnes* is less than three, the Level_VLC1 table is selected; otherwise, Level_VLC0 is used for the first level value. For the second level value, when the decoded level value is greater than a threshold defined for each step, the next VLC table is used. The thresholds for the seven VLC tables are {0, 3, 6, 12, 24, 48, N/A}.

The VLC table for the level syntax element is essentially based on Exp-Golomb. The number of remainder bits is determined by the number of leading zeros. Then, a known arithmetic operation is used to decode Exp-Golomb-coded syntax elements. Table 3 shows the codewords and corresponding levels for Level_VLC1. As

Table 3 Codewords and corresponding levels for LEVEL_VLC1

Codeword	Level
10	1
11	−1
010	2
011	−2
…	…
0000 0000 0000 0011	−15
0000 0000 0000 0001 xxxx xxxx xxxx	±16…
0000 0000 0000 0000 1xxx xxxx xxxx xx	±2064…

shown in the table, the length of the codeword varies depending on the number of leading zeros until this value is less than 15. Because of the exception, branch operations could be required.

It is inefficient to use the table mapping algorithm alone for the level syntax element because the table memory requirement is huge. This paper proposes a two-stage table mapping algorithm using arithmetic operations with one condition. As shown in Figure 5, if the number of leading zeros is greater than or equal to 16, arithmetic operations are used for level decoding. Otherwise, the two-stage table mapping algorithm is used. As shown in Figure 6, the next table index, the length of remainder bits, and the starting address of the coefficient table are accessed using the number of leading zeros as an index. Addition or comparison is not needed to identify the next table to be used according to the threshold because the next table index is obtained from the first-stage table mapping operation. After the process of leading zeros table mapping in Figure 5, the processed bits are discarded. The coefficient table for the second table mapping is used with a number of bits equal to the number of remainder bits from the first table. H.264/AVC utilizes seven coefficient tables depending on the VLC level number. However, the proposed algorithm consolidates all these coefficient tables into one, as shown in Figure 6. The coefficient level value in the consolidated coefficient table ranges from −2,528 to +2,528. From the starting address of the first table, the coefficient level value can be retrieved using the number of remainder bits as an index. After coefficient table mapping, the remainder bits are discarded.

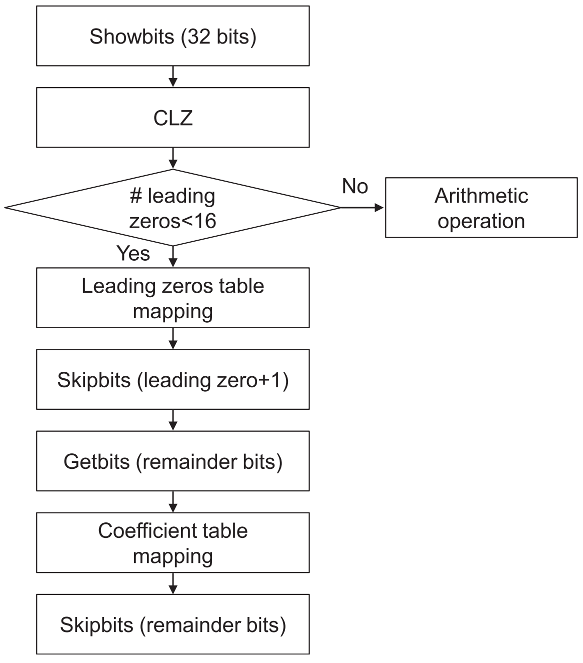

Showbits (32 bits)
CLZ
# leading zeros<16
No
Arithmetic operation
Yes
Leading zeros table mapping
Skipbits (leading zero+1)
Getbits (remainder bits)
Coefficient table mapping
Skipbits (remainder bits)

Figure 5 **Flowchart for decoding operation of the level syntax element.**

3.3 Proposed fast decoding for the run_before syntax element

The run_before syntax element indicates the number of zero coefficients between two consecutive non-zero coefficients. This syntax element has seven tables depending on the number of remaining zero coefficients to be decoded. A table mapping method that does not need branch operations for table selection is proposed, and the consolidated table for the proposed decoding method of the run_before syntax element is as follows: The maximum codeword length from run_before_T0 to run_before_T5 is 3; therefore, six tables with 2^3 entries each are needed. For the last table, run_before_T6, the maximum number of codeword bits is 11; therefore, a mapping table with 2^{11} entries must be used for direct memory access. However, only one codeword in the table consists of 11 bits. Therefore, the codewords can be identified by checking the first 10 bits only, so the proposed mapping table for run_before_T6 consists of 2^{10} entries as shown in Figure 7. The proposed pseudo-code for fast decoding of the run_before syntax element is presented as follows:

```
1) Table_index = I_MIN(zeroleft,7);
2) Bits = Showbits(10);
3) Sub_index = Bits >> I_ONERETURN((zeroleft<7), 7);
4) Run_before = run_vlc_table[Table_index-1][Sub_index].value;
5) Skipbits(run_vlc_table[Table_index-1][Sub_index].length);
```

In step 1, the table number is determined, with zeroleft representing the total number of zero coefficients to be decoded. The run_before_T6 is used in the

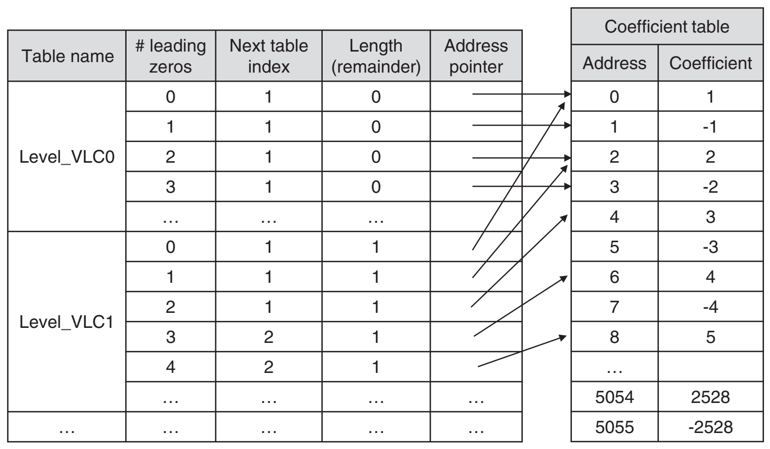


Table name	# leading zeros	Next table index	Length (remainder)	Address pointer
Level_VLC0	0	1	0	
	1	1	0	
	2	1	0	
	3	1	0	
	…	…	…	
Level_VLC1	0	1	1	
	1	1	1	
	2	1	1	
	3	2	1	
	4	2	1	
	…	…	…	…
…	…	…	…	…

Coefficient table	
Address	Coefficient
0	1
1	-1
2	2
3	-2
4	3
5	-3
6	4
7	-4
8	5
…	
5054	2528
5055	-2528

Figure 6 Proposed two-stage decoding table for level syntax element.

case that zeroleft is greater than 6. Otherwise, the table number is selected based on the zeroleft value. In general, a branch operation is needed to decide whether zeroleft is greater than 6. However, in the proposed design, the I_MIN instruction in Table 1 is used to eliminate the branch operation. In step 2, 10 bits are loaded for consecutive decoding. In step 3, the I_ONERETURN instruction is used to carry out table mapping with all 10 bits for run_before_T6 and with only 3 bits for the remaining tables. In step 4, the run_before value is retrieved by using table mapping with the table number and the associated index. In step 5, the processed bits are discarded.

4 Experimental results and discussion

To evaluate the performance of the proposed algorithm, the number of cycles for H.264/AVC CAVLD is investigated with a RP simulator. We compared the proposed algorithms with Kim's method [14], Iqbal's method [20], the multi-level table mapping (MTM) method [21], and open-source software (FFmpeg) [22] in the sense of CAVLD cycles and required memory for various video sequences.

4. 1 Experimental conditions and environment

The performance of bitstream decoding is related to not only the characteristics of video sequences but also their

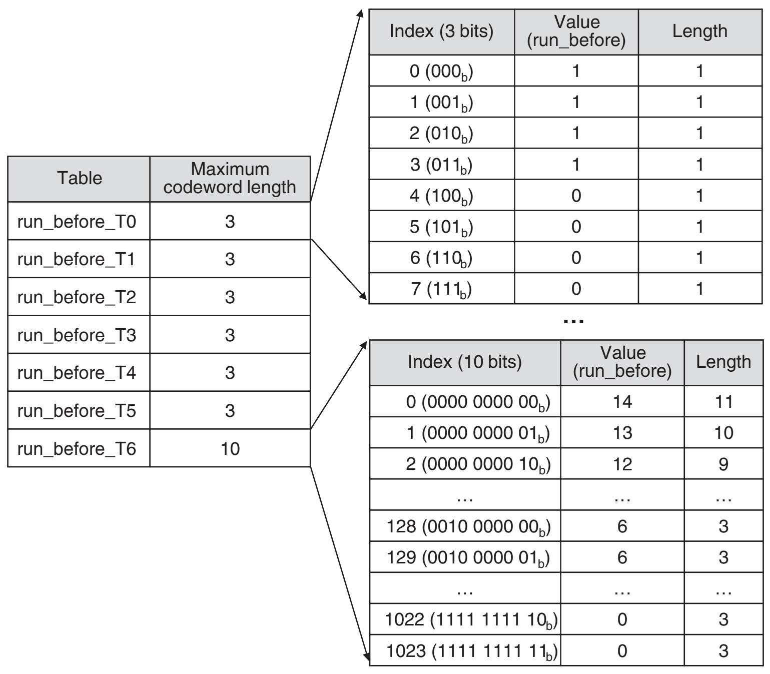


Table	Maximum codeword length
run_before_T0	3
run_before_T1	3
run_before_T2	3
run_before_T3	3
run_before_T4	3
run_before_T5	3
run_before_T6	10

Index (3 bits)	Value (run_before)	Length
0 (000_b)	1	1
1 (001_b)	1	1
2 (010_b)	1	1
3 (011_b)	1	1
4 (100_b)	0	1
5 (101_b)	0	1
6 (110_b)	0	1
7 (111_b)	0	1

…

Index (10 bits)	Value (run_before)	Length
0 ($0000\ 0000\ 00_b$)	14	11
1 ($0000\ 0000\ 01_b$)	13	10
2 ($0000\ 0000\ 10_b$)	12	9
…	…	…
128 ($0010\ 0000\ 00_b$)	6	3
129 ($0010\ 0000\ 01_b$)	6	3
…	…	…
1022 ($1111\ 1111\ 10_b$)	0	3
1023 ($1111\ 1111\ 11_b$)	0	3

Figure 7 Proposed decoding table for run-before syntax element.

Table 4 Test sequences and their bitrates

Resolution	Sequences	QP	Bitrate (Mbps)		FPS (frame/s)
			IPPP	IBBBP	
HD	Parkrun	22	54.2	52.1	30
		28	22.8	20.5	30
		34	8.5	8.0	30
	Shields	22	25.4	23.2	30
		28	4.0	3.3	30
		34	1.2	1.1	30
	Mobcal	22	29.5	24.6	30
		28	5.2	3.6	30
		34	1.3	1.0	30
Full HD	Blue_sky	22	13.8	11.9	30
		28	5.0	4.2	30
		34	2.2	2.0	30
	Riverbed	22	62.2	62.8	30
		28	31.4	31.9	30
		34	15.0	14.7	30
	Station2	22	7.3	7.4	30
		28	2.3	2.5	30
		34	1.2	1.3	30

bitrates. For this reason, we used several sequences (720p and full HD) for the evaluation. Table 4 shows test sequences, their resolutions, and bitrates. For 720p, ‘Parkrun’, ‘Shields’, and ‘Mobcal’ sequences were used. In addition, ‘Blue_sky’, ‘Riverbed’, and ‘Station2’ sequences were used for the full HD test. The encoding conditions are shown in Table 5. We used JM 17.2 reference software to encode the test sequences with multiple quantization parameter (QP) values. By using multiple QP values, we can evaluate the performance of CAVLD algorithms for various bitrates. In addition, we used both IPPP and IBBBP coding structures for thorough analysis because the number of zero value coefficients could vary depending on P and B slices.

As mentioned before, we developed a new table mapping algorithm for CAVLD and designated instructions for multi-format bitstream decoding. The added instructions, Showbits, GetBits, and Skipbits instructions run in 2 cycles, and the other instructions, CLZ and I_ONERETURN, run in 1 cycle. To evaluate the performance of the new table mapping algorithm for the coeff_token, level, and run_before syntax elements, we used a developed CAVLD decoder and compared the decoding cycles of the CAVLD. For example, to evaluate algorithms for coeff_token decoding, we just replaced the proposed algorithm for coeff_token with other conventional algorithms for the syntax element. In that case, the designated instructions are also applied not only for the proposed table mapping algorithm but also for the conventional algorithms.

Table 5 Encoding conditions for test bitstreams

Profile/level	Main/5.0
Intra period	30
Search range	32
Search mode	Fast full search
Coding structure	IPPP/IBBBP
QP	22, 28, 34
Reference frames	5 frames
RD optimization	High complexity mode
Encoder	JM 17.2

4.2 Performance evaluation of the proposed algorithm and several conventional algorithms for CAVLC syntax elements

Table 6 shows the numbers of megacycles per second (MCPS) required by the proposed and conventional algorithms for the coeff_token syntax element and cycle reduction ratio (CRR) of the proposed algorithm, compared with the conventional algorithms. Kim's method does not use any table mapping operations except for arithmetic operations. The rest of the algorithms are based on table mapping. Table 6 shows that the table mapping-based algorithms are generally more effective than the arithmetic-based one. Compared with Kim's method, the proposed algorithm is 17.2% and 17.3% faster for IPPP and IBBBP coding structures, respectively, and it just requires 0.718 kB of memory for the table mapping. As shown in Table 6, the proposed algorithm is around 8% faster than Iqbal's method. The proposed algorithm and Iqbal's algorithm are based on table mapping. The only difference is that the proposed algorithm omits the branch operations. We can state that by omitting branch operations, the proposed algorithm reduces the number of cycles by avoiding pipeline stall in multi-stage pipelined processors. Note that the proposed algorithm is more appropriate on various target systems because it does not cause pipeline stall and it is not affected by the performance of branch prediction. The cycle requirement of the proposed algorithm is almost the same as those of the MTM method. However, the proposed method reduces the memory requirements by approximately 88%. In summary, the proposed algorithm achieves a small number of cycles with a minimum memory requirement for coeff_token decoding.

Table 7 shows the numbers of MCPS and the memory requirements required by the proposed and conventional algorithm for the level syntax, and CRR of the proposed algorithm, compared to the conventional algorithms. The number of cycles used by the proposed algorithm is

Table 6 Comparison of MCPS and memory requirements for coeff_token syntax element

Sequences	QP	MCPS				CRR (%)		
		Kim [14]	Iqbal [20]	MTM [21]	Proposed	Kim vs. proposed	Iqbal vs. proposed	MTM vs. proposed
		IPPP	IPPP	IPPP	IPPP	IPPP	IPPP	IPPP
		IBBBP	IBBBP	IBBBP	IBBBP	IBBBP	IBBBP	IBBBP
Parkrun	22	1,200.8	1,159.5	1,122.1	1,121.6	6.6	3.3	0.0
		579.5	559.2	541.0	540.7	6.7	3.3	0.0
	28	492.3	466.7	444.9	444.4	9.7	4.8	0.1
		249.7	211.8	203.4	203.2	18.6	4.1	0.1
	34	178.2	165.3	155.0	154.8	13.1	6.4	0.1
		84.2	79.0	74.8	74.7	11.3	5.5	0.1
Shields	22	554.4	506.8	469.4	469.1	15.4	7.4	0.1
		261.2	239.1	221.7	221.5	15.2	7.3	0.1
	28	86.2	77.7	70.8	70.7	18.0	9.0	0.1
		35.7	32.3	29.5	29.5	17.2	8.5	0.1
	34	23.4	20.9	18.9	18.9	19.2	9.6	0.0
		10.5	9.4	8.5	8.5	18.8	9.4	−0.1
Mobcal	22	669.1	613.4	569.2	568.9	15.0	7.3	0.1
		282.0	258.1	239.2	239.0	15.2	7.4	0.1
	28	103.5	94.1	86.3	86.3	16.6	8.3	0.0
		36.3	33.2	30.7	30.7	15.4	7.5	0.0
	34	20.3	18.4	16.9	17.0	16.3	7.6	−0.6
		8.9	8.1	7.5	7.5	15.9	7.6	0.0
Blue_sky	22	347.7	311.8	283.5	283.4	18.5	9.1	0.0
		143.8	129.3	117.8	117.8	18.1	8.9	0.0
	28	124.8	110.9	99.4	99.6	20.2	10.2	−0.2
		46.0	41.0	37.0	37.0	19.6	9.8	−0.1
	34	45.2	39.9	35.6	35.8	20.8	10.3	−0.6
		17.5	15.5	13.8	13.9	20.4	10.3	−0.3
Riverbed	22	1,526.9	1,381.1	1,274.1	1,273.4	16.6	7.8	0.1
		763.8	690.9	637.5	637.1	16.6	7.8	0.1
	28	777.5	692.2	623.7	623.2	19.8	10.0	0.1
		390.0	347.4	313.3	313.1	19.7	9.9	0.1
	34	324.5	287.9	256.2	256.0	21.1	11.1	0.1
		160.1	142.1	126.5	126.5	21.0	11.0	0.1
Station2	22	158.4	140.6	125.9	125.9	20.5	10.5	0.0
		73.6	65.4	58.6	58.6	20.4	10.4	0.0
	28	35.5	31.5	28.0	28.0	21.1	11.1	0.0
		17.0	15.1	13.5	13.5	21.0	10.9	0.0
	34	13.2	11.7	10.3	10.4	21.2	11.1	−1.0
		6.3	5.6	5.0	5.0	21.0	10.9	−0.4
Average		-	-	-	-	17.2	8.6	−0.1
		-	-	-	-	17.3	8.4	0.0
Required memory (kB)		0	0.718	5.968	0.718	-	-	-

Table 7 Comparison of MCPS and memory requirements for level syntax element

Sequences	QP	MCPS		CRR (%)
		MTM [21]	Proposed	MTM vs. proposed
		IPPP	IPPP	IPPP
		IBBBP	IBBBP	IBBBP
Parkrun	22	1,146.4	1,121.6	2.2
		552.9	540.7	2.2
	28	465.3	444.4	4.5
		210.1	203.2	3.3
	34	167.1	154.8	7.4
		79.4	74.7	6.0
Shields	22	517.0	469.1	9.3
		243.8	221.5	9.1
	28	79.5	70.7	11.1
		33.0	29.5	10.5
	34	21.6	18.9	12.3
		9.7	8.5	12.1
Mobcal	22	624.7	568.9	8.9
		262.9	239.0	9.1
	28	95.9	86.3	10.0
		33.8	30.7	9.0
	34	18.9	17.0	10.5
		8.3	7.5	10.0
Blue_sky	22	321.3	283.4	11.8
		133.3	117.8	11.6
	28	114.9	99.6	13.3
		42.6	37.0	13.1
	34	41.6	35.8	14.0
		16.1	13.9	13.9
Riverbed	22	1,429.3	1,273.4	10.9
		715.0	637.1	10.9
	28	717.0	623.2	13.1
		360.0	313.1	13.0
	34	296.0	256.0	13.5
		146.5	126.5	13.7
Station2	22	145.2	125.9	13.3
		67.6	58.6	13.3
	28	32.5	28.0	13.7
		15.6	13.5	13.7
	34	12.0	10.4	13.4
		5.8	5.0	13.4
Average		-	-	10.7
		-	-	10.4
Required memory (kB)		3.5	9.937	-

reduced by 10.7% and 10.4% for IPPP and IBBBP coding structures, respectively, compared with the MTM method. For the level syntax element, the proposed algorithm and the MTM method require 9.937 and 3.5 kB, respectively. The proposed algorithm requires approximately 6 kB more than the MTM method for an approximate 11% speedup.

Table 8 shows the numbers of MCPS, the memory requirements of the proposed and conventional algorithms for the run_before syntax, and CRR of the proposed algorithm. The number of cycles used by the proposed algorithm is reduced by approximately 7% and 5%, compared with the MTM method and the arithmetic-based (Kim's) method, respectively. The MTM algorithm uses both table mapping and arithmetic methods, but Kim's method uses only arithmetic operations. The two conventional algorithms use conditional branches for table selection. However, the proposed algorithm involves no branch operations. As for the memory requirements, the proposed algorithm requires 2.109 kB; however, they can easily be loaded into the target system.

In this work, we implemented our own CAVLD to fully exploit the developed instructions along with the proposed table mapping algorithms for coeff_token, level, and run_before syntax elements. To evaluate the high-performance CAVLD algorithm, the developed CAVLD is evaluated by comparing the open H.264/AVC decoder (FFmpeg). Note that JM is not a proper platform for decoding speed evaluations because JM is around two times slower than the FFmpeg software for CAVLD.

Table 9 shows decoding cycles of the proposed CAVLD and FFmpeg one for various QP values. For evaluation of the proposed instructions for the BsPU, we compared decoding cycles between FFmpeg and modified FFmpeg with the proposed instructions. We found that we can save 38% cycles by applying the proposed instructions to the FFmpeg CAVLD. Because Showbits, Getbits, and Skipbits are frequently called for CAVLD, these can be accelerated into 2 cycles with the proposed instructions. For evaluation of the proposed algorithm, the developed CAVLD and modified FFmpeg are compared using the proposed instructions. As a result, we found that the proposed algorithm is 13.5% and 13.0% faster than the modified FFmpeg with the proposed instructions for IPPP and IBBBP coding structures, respectively. We can say that BsPU is effective with the proposed table mapping algorithm that does not cause any branches. In addition, the proposed CAVLD with instructions and table mapping algorithm can save 46.6% and 46.2% cycles for IPPP and IBBBP coding structures, respectively, compared with FFmpeg CAVLD without the proposed

Table 8 Comparison of MCPS and memory requirements for run_before syntax element

Sequences	QP	MCPS			CRR (%)	
		MTM [21]	Kim [14]	Proposed	MTM vs. proposed	Kim vs. proposed
		IPPP	IPPP	IPPP	IPPP	IPPP
		IBBBP	IBBBP	IBBBP	IBBBP	IBBBP
Parkrun	22	1,284.9	1,259.0	1,121.6	12.7	10.9
		614.6	604.3	540.7	12.0	10.5
	28	504.3	492.9	444.4	11.9	9.8
		228.8	225.0	203.2	11.2	9.7
	34	175.0	171.1	154.8	11.6	9.5
		84.3	82.7	74.7	11.4	9.7
Shields	22	529.4	514.6	469.1	11.4	8.8
		249.6	243.0	221.5	11.3	8.8
	28	76.1	74.7	70.7	7.1	5.3
		31.9	31.3	29.5	7.3	5.7
	34	19.9	19.6	18.9	5.1	3.7
		9.0	8.9	8.5	5.1	3.9
Mobcal	22	644.2	626.7	568.9	11.7	9.2
		269.6	262.5	239.0	11.3	8.9
	28	94.0	92.1	86.3	8.2	6.3
		33.5	32.9	30.7	8.3	6.7
	34	18.3	18.0	17.0	7.3	6.0
		8.1	8.0	7.5	7.7	6.4
Blue_sky	22	304.5	297.6	283.4	6.9	4.8
		126.1	123.6	117.8	6.6	4.7
	28	103.5	102.1	99.6	3.8	2.4
		38.6	38.1	37.0	4.2	2.8
	34	36.8	36.4	35.8	2.7	1.6
		14.3	14.2	13.9	3.0	1.9
Riverbed	22	1,361.2	1,339.0	1,273.4	6.4	4.9
		680.8	669.8	637.1	6.4	4.9
	28	650.2	638.2	623.2	4.2	2.4
		326.8	320.7	313.1	4.2	2.4
	34	263.9	260.1	256.0	3.0	1.6
		130.3	128.3	126.5	2.9	1.5
Station2	22	130.8	129.0	125.9	3.8	2.4
		60.9	60.1	58.6	3.7	2.4
	28	28.7	28.5	28.0	2.4	1.7
		13.8	13.7	13.5	2.5	1.8
	34	10.7	10.6	10.4	2.6	1.8
		5.1	5.1	5.0	2.6	1.8
Average		-	-	-	6.8	5.2
		-	-	-	6.8	5.3
Required memory (kB)		0.57	0	2.109	-	-

instructions. At the same time, we can save the required memory from 73.6 to 30.1 kB. The cycle reduction of the proposed CAVLD is persistent regardless of QP values.

5. Conclusions

In this paper, a novel fast CAVLD method is proposed with table mapping and new dedicated instructions for fast entropy decoding of H.264/AVC. The

Table 9 Comparison of MCPS for CAVLD

Sequences	QP	MCPS			CRR (%)		
		FFmpeg [22]	FFmpeg[a]	Proposed	FFmpeg vs. FFmpeg[a]	FFmpeg vs. proposed	FFmpeg[a] vs. proposed
		IPPP	IPPP	IPPP	IPPP	IPPP	IPPP
		IBBBP	IBBBP	IBBBP	IBBBP	IBBBP	IBBBP
Parkrun	22	1,971.0	1,250.6	1,121.6	36.6	43.1	10.3
		944.7	598.9	540.7	36.6	42.8	9.7
	28	802.3	504.5	444.4	37.1	44.6	11.9
		358.2	226.3	203.2	36.8	43.3	10.2
	34	285.4	179.1	154.8	37.2	45.8	13.6
		134.5	84.9	74.7	36.9	44.5	12.1
Shields	22	893.7	554.2	469.1	38.0	47.5	15.4
		420.7	261.0	221.5	38.0	47.3	15.1
	28	134.0	82.4	70.7	38.5	47.2	14.2
		55.3	34.1	29.5	38.3	46.6	13.5
	34	35.7	21.9	18.9	38.7	47.1	13.7
		15.9	9.8	8.5	38.4	46.5	13.1
Mobcal	22	1,082.3	672.2	568.9	37.9	47.4	15.4
		455.4	282.3	239.0	38.0	47.5	15.3
	28	163.1	100.5	86.3	38.3	47.1	14.2
		57.2	35.4	30.7	38.1	46.3	13.2
	34	31.4	19.5	17.0	38.1	46.1	12.9
		13.7	8.5	7.5	37.8	45.5	12.3
Blue_sky	22	538.5	331.0	283.4	38.5	47.4	14.4
		221.4	136.6	117.8	38.3	46.8	13.7
	28	188.4	115.4	99.6	38.7	47.2	13.7
		69.4	42.7	37.0	38.5	46.6	13.3
	34	68.3	41.4	35.8	39.3	47.6	13.6
		26.3	16.1	13.9	39.0	47.2	13.4
Riverbed	22	2,313.6	1,438.6	1,273.4	37.8	45.0	11.5
		1,156.8	719.4	637.1	37.8	44.9	11.4
	28	1,171.3	716.7	623.2	38.8	46.8	13.0
		588.2	359.7	313.1	38.8	46.8	13.0
	34	484.4	297.0	256.0	38.7	47.2	13.8
		240.0	146.5	126.5	38.9	47.3	13.7
Station2	22	242.7	146.9	125.9	39.5	48.1	14.3
		112.9	68.3	58.6	39.5	48.1	14.1
	28	53.3	32.5	28.0	39.0	47.4	13.8
		25.5	15.6	13.5	38.9	47.3	13.7
	34	19.6	12.1	10.4	38.5	47.0	13.9
		9.4	5.8	5.0	38.5	47.0	13.8
Average		-	-	-	38.3	46.6	13.5
		-	-	-	38.2	46.2	13.0
Required memory (kB)		73.6	73.6	30.1	-	-	-

[a]Modified FFmpeg with proposed intrinsics.

proposed algorithm reduces the number of cycles to about 8% ~ 17% from coeff_token, 11% from level, and 5% ~ 7% from run_before, respectively. In addition, the proposed CAVLD is enhanced with the proposed several instructions for the BsPU along with the table mapping. By comparing the developed CAVLD with the FFmpeg one, we achieved a 47% reduction in the number of cycles, compared with FFmpeg CAVLD. In addition, the proposed algorithm reduces the memory requirements for table mapping by approximately 59% compared with FFmpeg on the RP that is commercially used in a smartphone. In the future, we will focus on the development of CABAD on the BsPU.

Competing interests
The authors declare that they have no competing interests.

Acknowledgments
This research was partly supported by Samsung Electronics, the MKE (The Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program (NIPA-2013-H0301-13-1006) supervised by the NIPA (National IT Industry Promotion Agency), and Kwangwoon University during the sabbatical year of 2011.

Author details
[1]Department of Computer Engineering, Kwangwoon University, Wolgye-dong, Nowon-gu 447-1, Seoul 139-701, Republic of Korea. [2]Samsung Advanced Institute of Technology, Nongseo-dong, Giheung-gu, Yongin-si, Gyeonggi-do 446-712, Republic of Korea.

Received: 31 August 2012 Accepted: 3 April 2013
Published: 26 April 2013

References
1. ITU-T, *Advanced video coding for generic audiovisual services," Recommendation ITU-T H.264* (ITU, Geneva, 2012)
2. WC Ku, SH Chou, JC Chu, CL Liu, TF Chen, JI Guo, JS Wang, VisoMT: a collaborative multithreading multicore processor for multimedia applications with a fast data switching mechanism. IEEE Trans Circuit Syst Video Tech **19**(11), 1633–1645 (2009)
3. YK Chen, X Tian, S Ge, M Girkar, Towards efficient multi-level threading of H.264 encoder on Intel hyper-threading architectures, in *Parallel and Distributed Processing Symposium* (Santa Fe, New Mexico, 2004), pp. 63–72
4. C Meenderinck, A Azevedo, B Juurlink, MA Mesa, A Ramirez, Parallel scalability of video decoders. J Signal Process Syst **57**(2), 173–194 (2008)
5. P Li, B. Veeravalli, AA Kassim, Design and implementation of parallel video encoding strategies using divisible load analysis. IEEE Trans Circ Syst Video Tech **15**(9), 1098–1112 (2005)
6. W Yu, Y He, A high performance CABAC decoding architecture. IEEE Trans Consum Electron **51**(4), 1352–1359 (2005)
7. Y Yi, IC Park, High-speed H.264/AVC CABAC decoding. IEEE Trans Circ Syst Video Tech **17**(4), 490–494 (2007)
8. RR Osorio, JD Bruguera, High-throughput architecture for H.264/AVC CABAC compression system. IEEE Trans Circ Syst Video Tech **16**(11), 1376–1384 (2006)
9. H Lin, Y Lu, B Liu, J Yang, A highly efficient VLSI architecture for H.264/AVC CAVLC decoder. IEEE Trans Multimed **10**(1), 31–42 (2008)
10. GG 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. IET Image Process **4**(2), 81–91 (2010)
11. TH Tsai, TL Fang, YN Pan, A novel design of CAVLC decoder with low power and high throughput considerations. IEEE Trans Circ Syst Video Technol **21**(3), 311–319 (2011)
12. JM software, http://iphome.hhi.de/suehring/tml/download. Accessed 4 March 2013
13. YH Moon, GY Kim, JH Kim, An efficient decoding of CAVLC in H.264/AVC video coding standard. IEEE Trans Consum Electron **51**(3), 933–938 (2005)
14. YH Kim, YJ Yoo, J Shin, B Choi, J Paik, Memory-efficient H.264/AVC CAVLC for fast decoding. IEEE Trans Consum Electron **52**(3), 943–952 (2005)
15. F Rivera, M Sanchez-Elez, M Fernandez, N Bagherzadeh, An approach to execute conditional branches onto SIMD multi-context reconfigurable architectures, in *IEEE Proceedings of the DSD* (Porto), pp. 396–402. 30 August - 3 September 2005
16. JS Lee, JH Jeong, TG Chang, An efficient method of Huffman decoding for MPEG-2 AAC and its performance analysis. IEEE Trans Speech Audio Process **13**(6), 1206–1209 (2005)
17. R Hashemian, Memory efficient and high-speed search Huffman coding. IEEE Trans Comm **43**(10), 2576–2581 (1995)
18. SB Choi, MH Lee, High speed pattern matching for a fast Huffman decoder. IEEE Trans Consum Electron **41**(1), 97–103 (1995)
19. SW Wang, JL Wu, SC Chuang, CC Hsiao, YS Tung, Memory efficient hierarchical lookup tables for mass arbitrary-side growing Huffman trees decoding. IEEE Trans Circ Syst Video Tech **18**(10), 1335–1346 (2008)
20. N Iqbal, J Henkel, Efficient constant-time entropy decoding for H.264, in *Design, Automation and Test in Europe Conference and Exhibition* (Nice), pp. 1440–1445. 20–24 April 2009
21. H Gue, X Xia, W Sun, J Zhou, S Yu, An memory-efficient variable length decoding scheme for embedded MPEG-4 video decoders, in *ICSP 2006 Proceedings of IEEE* (Beijing, China, 2006). 16–20 Nov 2006
22. FFMPEG, 2013. http://ffmpeg.org/index.html
23. WJ Lee, SO Woo, KT Kwon, SJ Son, KJ Min, GJ Jang, CH Lee, SY Jung, CM Park, SH Lee, A scalable GPU architecture based on dynamically reconfigurable embedded processor, in *High Performance Graphics* (Vancouver). 5–7 August 2011
24. B Mei, A Lambrechts, D Verkest, JY Mignolet, R Lauwereins, Architecture exploration for a reconfigurable architecture template. IEEE Des Test Comput **22**(2), 90–101 (2005)
25. S Furber, *ARM System-On-Chip Architecture*, 2nd edn. (Addison-Wesley, Boston), pp. 39–42

doi:10.1186/1687-5281-2013-23
Cite this article as: Seo *et al.*: Fast CAVLD of H.264/AVC on bitstream decoding processor. *EURASIP Journal on Image and Video Processing* 2013 2013:23.