A 385 MHz 13.54 K Gates Delay Balanced Two-Level CAVLC Decoder for Ultra HD H.264/AVC Video

Yuan-Hsin Liao, Gwo-Long Li, and Tian-Sheuan Chang, *Senior Member, IEEE*

***Abstract*—To satisfy the heavy performance requirement in real-time high-resolution H.264/AVC, very large-scale integrated implementation of the entropy decoder is necessary since it dominates the overall decoder throughput. In this paper, we propose a high-throughput delay balanced two-level context-based adaptive variable length coding (CAVLC) decoder with 21% shorter critical path delay in comparison to the traditional two-level decoder design. Furthermore, redundant decoding processes are removed by a skipping mechanism. The proposed CAVLC decoder only needs 127.13 cycles per macroblock on average to support level 5.1 decoding with 13.54 k gate counts under 90-nm CMOS technology.**

***Index Terms*—Context-adaptive variable length decoder (CAVLD), H.264.**

I. Introduction

H.264 IS THE state-of-the-art video coding standard to provide better compression efficiency compared to earlier MPEG-4 and H.263 standards due to the adoption of advanced coding techniques. The H.264/AVC coding standard specifies two entropy coding tools called context-based adaptive variable length coding (CAVLC) and context-based adaptive binary arithmetic coding (CABAC) [1], [2] with context-based adaptive modeling. However, since the bitstream boundary between successive codeword is only known after the codeword length detection of the former one, the decoding procedure is inherently sequential and is hard to be accelerated. In addition, as available network bandwidth becomes higher and high-definition (HD) television gains popularity, the demand for better visual quality grows fast. That means video application system is expected to support HD or larger resolution encoding and decoding. The larger resolution trend and the heavy decoding requirement lead to the result that more data has to be processed in the same time for video decoders, and make it more difficult to work in real-time for CPUs. Thus, very large-scale integrated acceleration of entropy decoder is necessary since its throughput dominates the overall decoder system performance.

Manuscript received March 4, 2011; revised August 24, 2011; accepted March 6, 2012. Date of publication June 1, 2012; date of current version October 26, 2012. This paper was recommended by Associate Editor J. Ridge.

Y.-H. Liao is with PixArt Imaging, Inc., Hsinchu 300, Taiwan (e-mail: yhliao@dragons.ee.nctu.edu.tw).

G.-L. Li is with the Industrial Technology Research Institute, Hsinchu 302, Taiwan (e-mail: glli@itri.org.tw).

T.-S. Chang is with the Department of Electronics Engineering, National Chiao Tung University, Hsinchu 300, Taiwan (e-mail: tschang@dragons.ee.nctu.edu.tw).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSVT.2012.2202081

To fit decoding of HD videos, multisymbol decoding is often adopted by various approaches for acceleration, especially for critical parsing stages such as trailing_ones_sign_flag, level, and run_before. However, the main obstacle to parallel decoding is how to break the recursive dependencies between codewords. In the trailing_ones_sign_flag parsing stage, [3] and [4] implemented the parsing procedure in a single cycle since the number of TrailingOnes is already derived in the coeff_token parsing stage. In the level parsing stage, two level decoders were cascaded to produce two level symbols in one cycle [5], but it induced a huge critical path delay. In the run_before parsing stage, since the codewords of variable length coding (VLC) table used for run_before are much less and shorter than others, the data dependency obstacle is much easier to be overcome, and thus several efficient multi-run_before decoding architectures had been proposed to boost the throughput of CAVLC decoder. Thus, in [5], it parsed multiple run_before symbols with zero value in one cycle by counting the bit length of the series of "1" bits of input bitstream since the corresponding codeword was composed of "1" bits when run_before is equal to 0. However, this method is only effective in the high bit-rate coding but inefficient in the low bit-rate coding where the residual blocks are very sparse. Yu *et al.* [6] proposed a combined look-up table for decoding successive two run_before symbols at the same time. Wen *et al.* [7] adopted a bit-position VLC decoding approach that all run_before symbols were decoded using less than three cycles in one block to achieve high throughput at the expense of significant hardware cost raise. Lee *et al.* [8] presented a multisymbol decoder that can decode three run_before symbols in one cycle. Furthermore, a pattern-search method has been reported in [9]. In this method, a block can be reconstructed directly without performing CAVLC decoding procedure if a pattern was matched in a preestablished look-up table. In summary, for the two critical loops, the level parsing process and run_before parsing process, a lot of techniques have been proposed to speed up run_before parsing process, but few have been proposed to improve level parsing performance.

To improve the CAVLC speed for the target applications, this paper proposes a highly efficient two-level decoding architecture. The proposed design is a delay balanced design such that it can operate at higher clock rate. Besides, the design is further accelerated by the proposed skipping techniques to remove redundant decoding process. The final design can satisfy the Level 5.1 decoding requirement in H.264/AVC.

The organization of this paper is as follows. In Section II, we introduce the design principle and coding flow of CAVLC in detail. Section III addresses the proposed CAVLC hardware architecture design. The implementation results are shown in Section IV and the conclusion is given in Section V.

II. Overview of CAVLC

The followings show the decoding flow of CAVLC. In CAVLC, a residual block is represented by five types of syntax elements (SEs). These SEs are defined as follows.

1) *coeff_token*: This SE indicates the total number of nonzero coefficients (TotalCoeffs) including TrialingOnes (a series of ±1). Since the coding units of CAVLC are 4×4 and 2×2 blocks, TotalCoeffs can have any value between 0 and 16 and TrialingOnes can have anything between 0 and 3. The coeff_token decoding needs to look up three variable-length codeword tables and a fixed-length codeword table. The choice of look-up table depends on the total number of nonzero coefficients to the left and on top of the current block nA and nB, respectively.
2) *trailing_ones_sign_flag*: This 1-bit SE indicates the sign of TrialingOnes, and is coded in reverse order.
3) *level*: The SE level represents the value of remaining nonzero coefficient and is also coded in reverse order. Each level is composed of a prefix part (level_prefix) and a suffix part (suffix_part).
4) *total_zeros*: The sum of zero coefficient numbers, except for zeros after the last nonzero coefficient, is represented by this SE. The choice of VLC table depends on the total number of nonzero coefficients of the current block.
5) *run_before*: Number of zeros preceding each nonzero coefficient is encoded as this SE. The VLC table for coding each run_before is chosen according to the number of zeros left (zerosLeft).

Fig. 1 shows the flow diagram of CAVLC decoding. The decoding process consists of six steps: coeff_token parsing, trailing_ones_sign_flag parsing, level parsing, total_zeros parsing, run_before parsing, and residual block reconstruction. Table I shows an example for the decoding procedure of a CAVLC-coded residual block as depicted in Fig. 2 and its corresponding decoded information. The input bitstream provided for the CAVLC decoder is "00001000_11100101_11101101," and after the decoding procedure, the 4 × 4 residual block, "0, 3, 0, 1, −1, −1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0," is reconstructed.

TABLE I
CAVLC Decoding Procedure for 4 × 4 Residual Block Depicted in Fig. 2

Bitstream: 000010001110010111101101			
SE	Codeword	Value	Output Array
coeff_token	100	TotalCoeffs = 5, TrailingOnes = 3	N/A
TrailingOne sign	0	+	$\underline{1}$
TrailingOne sign	1	−	$\underline{-1}, 1$
TrailingOne sign	1	−	$\underline{-1}, -1, 1$
level	1	+1	$\underline{1}, -1, -1, 1$
level	0010	+3	$\underline{3}, 1, -1, -1, 1$
total_zeros	111	3	3, 1, −1, −1, 1
run_before	10	1	$3, 1, -1, -1, \underline{0}, 1$
run_before	1	0	3, 1, −1, −1, 0, 1
run_before	1	0	3, 1, −1, −1, 0, 1
run_before	01	1	$3, \underline{0}, 1, -1, -1, 0, 1$
Reconstructed block: 0, 3, 0, 1, −1, −1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0			

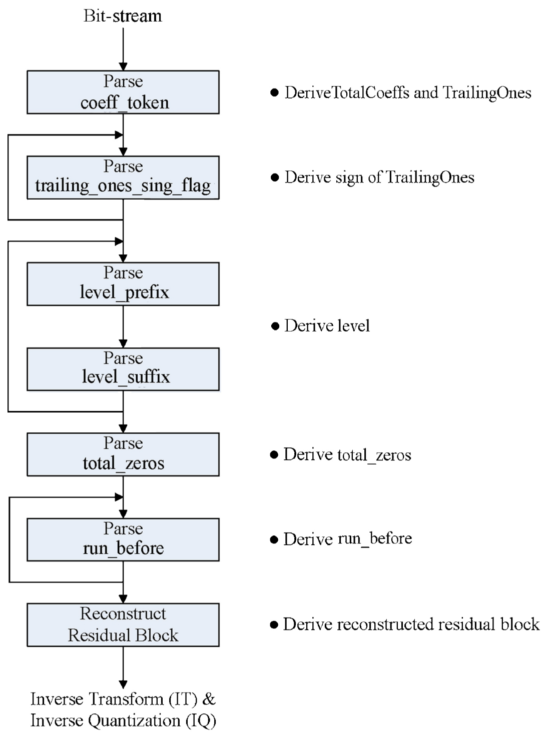

Bit-stream
Parse coeff_token
● DeriveTotalCoeffs and TrailingOnes
Parse trailing_ones_sing_flag
● Derive sign of TrailingOnes
Parse level_prefix
● Derive level
Parse level_suffix
Parse total_zeros
● Derive total_zeros
Parse run_before
● Derive run_before
Reconstruct Residual Block
● Derive reconstructed residual block
Inverse Transform (IT) & Inverse Quantization (IQ)

Fig. 1. CAVLC decoding flow.

III. Proposed CAVLC Decoder

Based on the CAVLC decoding flow, the major throughput obstacle is the level parsing procedure, which needs arithmetic operations and accounts for a critical loop in the whole CAVLC decoding procedure. However, directly cascading level decoders for multilevel decoding will not gain any throughput improvement since the inter-codeword dependency and succession of arithmetic operations lead to an unavoidably long critical path. Moreover, the inter-*level* dependency of suffixLength, which cannot be calculated until the value of current level is determined, makes it unable to exploit pipeline structure. Thus, it seems both direct multisymbol decoding and pipelining scheme do not work for the level decoding process. A good speedup method needs to break the inter-*level* dependency and the inter-codeword dependency. Hence, in the followings, we will first investigate the characteristics of the level decoding flow, and then propose our delay balanced two-level decoding process.

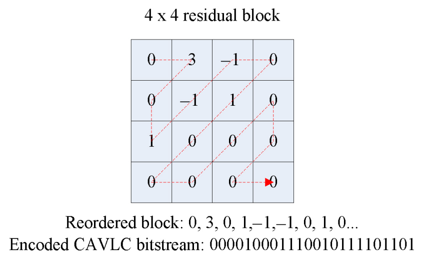

4 x 4 residual block
0 3 –1 0
0 –1 1 0
1 0 0 0
0 0 0 0
Reordered block: 0, 3, 0, 1,–1,–1, 0, 1, 0...
Encoded CAVLC bitstream: 000010001110010111101101

Fig. 2. Transmitted bitstream for a 4 × 4 residual block.

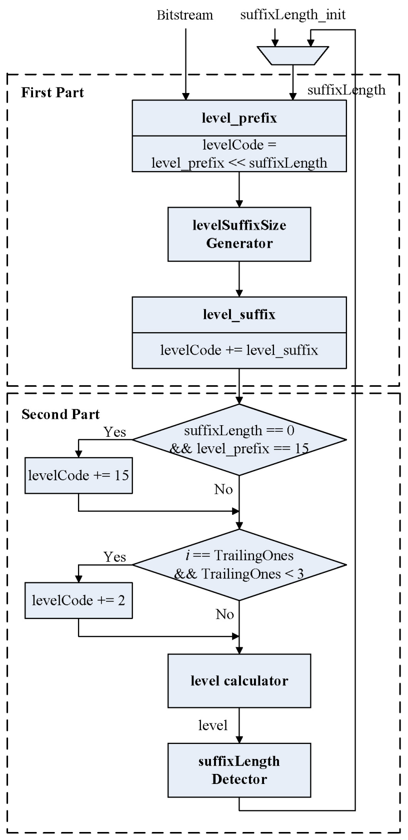

Bitstream
suffixLength_init
suffixLength
First Part
level_prefix
levelCode = level_prefix << suffixLength
levelSuffixSize Generator
level_suffix
levelCode += level_suffix
Second Part
Yes
suffixLength == 0 && level_prefix == 15
levelCode += 15
No
Yes
i == TrailingOnes && TrailingOnes < 3
levelCode += 2
No
level calculator
level
suffixLength Detector

Fig. 3. Original level decoding procedure defined in H.264/AVC standard.

A. Analysis for CAVLC

Fig. 3 shows the flow chart of the level decoding procedure. The decoding procedure consists of two parts: bitstream scanning process and level computating. The bit string of each level is formed with level_prefix and level_suffix as

$$\begin{aligned}\text{level\_bitString} &= [\text{level\_prefix}][\text{level\_suffix}]\\ &= [0\ldots.01][\text{level\_suffix}]\end{aligned} \tag{1}$$

where level_prefix consists of a series of "0" bits followed by a terminating "1" bit. The value of level_prefix is constrained in the range of 0–15. In the bitstream scanning process, after the value of level_prefix is determined by detecting the leading zeros in the bitstream, the parameter levelSuffixSize, which represents the bit length of level_suffix, is calculated as

$$\begin{aligned}&\text{if (level\_prefix == 15)}\\ &\quad \text{levelSuffixSize} = 12\\ &\text{elseif (level\_prefix == 14\&\&suffixLength == 0)}\\ &\quad \text{levelSuffixSize} = 4\\ &\text{else}\\ &\quad \text{levelSuffixSize} = \text{suffixLength}.\end{aligned} \tag{2}$$

TABLE II

Threshold Values for suffixLength Transition

Current suffixLength	Threshold Value to Modify suffixLength
0	0
1	3
2	6
3	12
4	24
5	48
6	N/A

Based on the levelSuffixSize, bits belonging to level_suffix are scanned, and the initial value of levelCode is calculated as

$$\text{levelCode} = (\text{level\_prefix} << \text{suffixLength}) + \text{level\_suffix}. \tag{3}$$

In the second part, levelCode is adjusted in case of special conditions. If level_prefix is equal to 15 and suffixLength is equal to 0, levelCode will be increased by 15, and if the number of TrailingOnes is less than 3, the first levelCode in the level decoding procedure will be increased by 2. Once the final value of levelCode is obtained, the value of level will be determined as if levelCode is even, level = (levelCode + 2)/2, otherwise, level = (-levelCode − 1)/2. Finally, since the absolute value of level tends to be larger in the level decoding procedure, the adaptive probability model shall be changed according to the previous decoded level. As a result, if the absolute value of decoded level is larger than the threshold listed in Table II, suffixLength must be modified to a more suitable value since small suffixLength is for small level and vice versa.

In above analysis, the main barriers to exploit parallel decoding are inter-*level* dependency of suffixLength and the unknown demarcation between successive codewords. Although the codeword length can be derived in the first part of level decoding procedure as follows:

$$\text{CodewordLength} = \text{level\_prefix} + 1 + \text{levelSuffixSize} \tag{4}$$

the updated suffixLength that affects the levelSuffixSize of next level cannot be obtained until the value of current level is determined. However, a modified suffixLength detector (MSD) algorithm [4] was presented to advance the computation of suffixLength prior to the determination of the value of current level. Fig. 4 depicts the MSD decoding procedure in which the input signal of MSD is level_prefix instead of the value of level. From the current decoding information and the level_prefix, the suffixLength provided for next level decoding process can be calculated in the first part. With this efficient algorithm, the level decoding process can be realized as shown in Fig. 5. However, despite the fact that the MSD algorithm shortens the critical path delay of *level* decoding process, multilevel decoding based on cascaded level decoders still

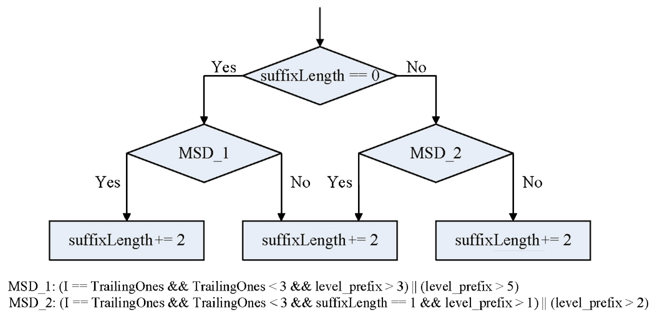

suffixLength == 0
Yes
No
MSD_1
MSD_2
Yes
No
Yes
No
suffixLength += 2
suffixLength += 2
suffixLength += 2
MSD_1: (I == TrailingOnes && TrailingOnes < 3 && level_prefix > 3) || (level_prefix > 5)
MSD_2: (I == TrailingOnes && TrailingOnes < 3 && suffixLength == 1 && level_prefix > 1) || (level_prefix > 2)

Fig. 4. MSD decoding procedure.

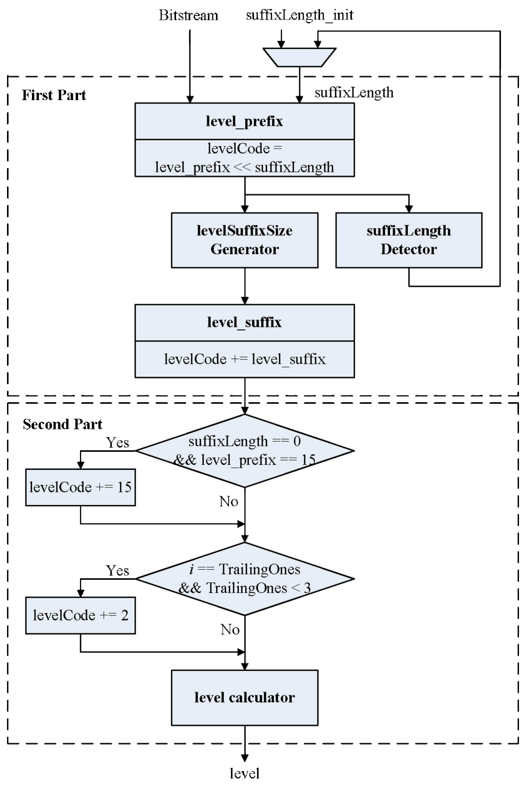

Bitstream
suffixLength_init
First Part
suffixLength
level_prefix
levelCode = level_prefix << suffixLength
levelSuffixSize Generator
suffixLength Detector
level_suffix
levelCode += level_suffix
Second Part
Yes
suffixLength == 0 && level_prefix == 15
levelCode += 15
No
Yes
i == TrailingOnes && TrailingOnes < 3
levelCode += 2
No
level calculator
level

Fig. 5. Modified level decoding procedure with MSD algorithm.

leads to an unavoidably long critical path, and thus remains unsuitable for implementation.

In our approach, to further expedite the throughput of CAVLC decoder instead of straight cascading level decoders, we take advantage of MSD algorithm to exploit a high-performance two-level decoding architecture. In general case, the levelSuffixSize that indicates the codeword length of level_suffix is equal to suffixLength. Consequently, the start point of next level codeword in the bitstream can be decided as soon as the level_prefix decoding has finished. Moreover, the adjustment of levelCode in the second part is only applied to the first level of the residual block. It means that those two special conditional branches can be skipped in the second level decoding. Based on these two features, we propose a delay balanced two-level decoding (DBTLD) architecture that efficiently shortens the critical path in comparison to the traditional design that cascades two level decoders directly.

B. Proposed DBTLD

Fig. 6 shows the block diagram of proposed DBTLD procedure. The second level decoding process is designed for the general case that levelSuffixSize is equal to suffixLength. Since the codeword length of first level can be determined immediately after the level_prefix is decoded, and the examination process of levelCode increment is unnecessary for the second level decoding process, a balanced structure can be obtained.

The first level decoding process is the same as shown in Fig. 5. For bitstream supplying for the second level decoding process, the input bitstream is shifted according to suffixLength and level_prefix_1. Afterward, instead of generating levelSuffixSize_2, the level_suffix_2 is parsed directly by fetching the output of first suffixLength_1 detector (SD_1) that is referred to the MSD algorithm. Finally, without checking the two special cases for increasing levelCode_2, the level mapping process is executed directly. Compared to the conventional approach to cascade two MSD algorithm-based level decoders, the critical path delay of proposed DBTLD engine is improved by 21% (from 3.25 ns to 2.56 ns) according to the implementation result.

It is important to note that the second level is not always valid. If the codeword length exceeds the bitstream width (32 bits), which may occur when level_prefix = 15 or level_prefix = 14 with suffixLentgh = 0, the decoded second level must be flushed.

C. CAVLC Decoding Architecture Design

Based on the DBTLD engine, the CAVLC decoding architecture is designed as shown in Fig. 7. The overall flow follows the decoding process as shown before, and the operations of each block will be described below. First, the bitstream is fetched by the bitstream fetcher unit, which is similar to other VLC decoding unit with nonregistered Barrel Shifter. Then, the coefficient token is decoded by the coeff_token unit. In the TrailingOnes decoding unit, all sign flags are scanned in one cycle. After level decoding procedure is done, all nonzero coefficients are stored in a 16-entry-deep and 13-bit-wide output buffer. Finally, in the run_before decoding unit, the corresponding level is transmitted to its actual position in the output buffer whenever a run_before symbol is decoded. In the output buffer, the concept of prediction-based run_before look-up table combination method [6] is employed here that two run_before symbols are decoded in one cycle. As a result, the overall critical path of our proposed CAVLC decoder starts from the Barrel Shifter, then goes through the DBTLD Decoder, and finally ends at the residual block reconstruction.

Fig. 8 shows the architecture of residual block reconstruction. After TrailingOnes and levels are pushed into the output buffer in order, one or two level symbols are moved to their final locations, respectively, depending on the coeffsLeft and zerosLeft information in each cycle. The movement starts from the last coefficient and ends until no more run_befores are decoded. The parameter coeffsLeft denotes the remaining number of nonzero coefficients needs to be moved, and zerosLeft represents the remaining number of zeros to be decoded.

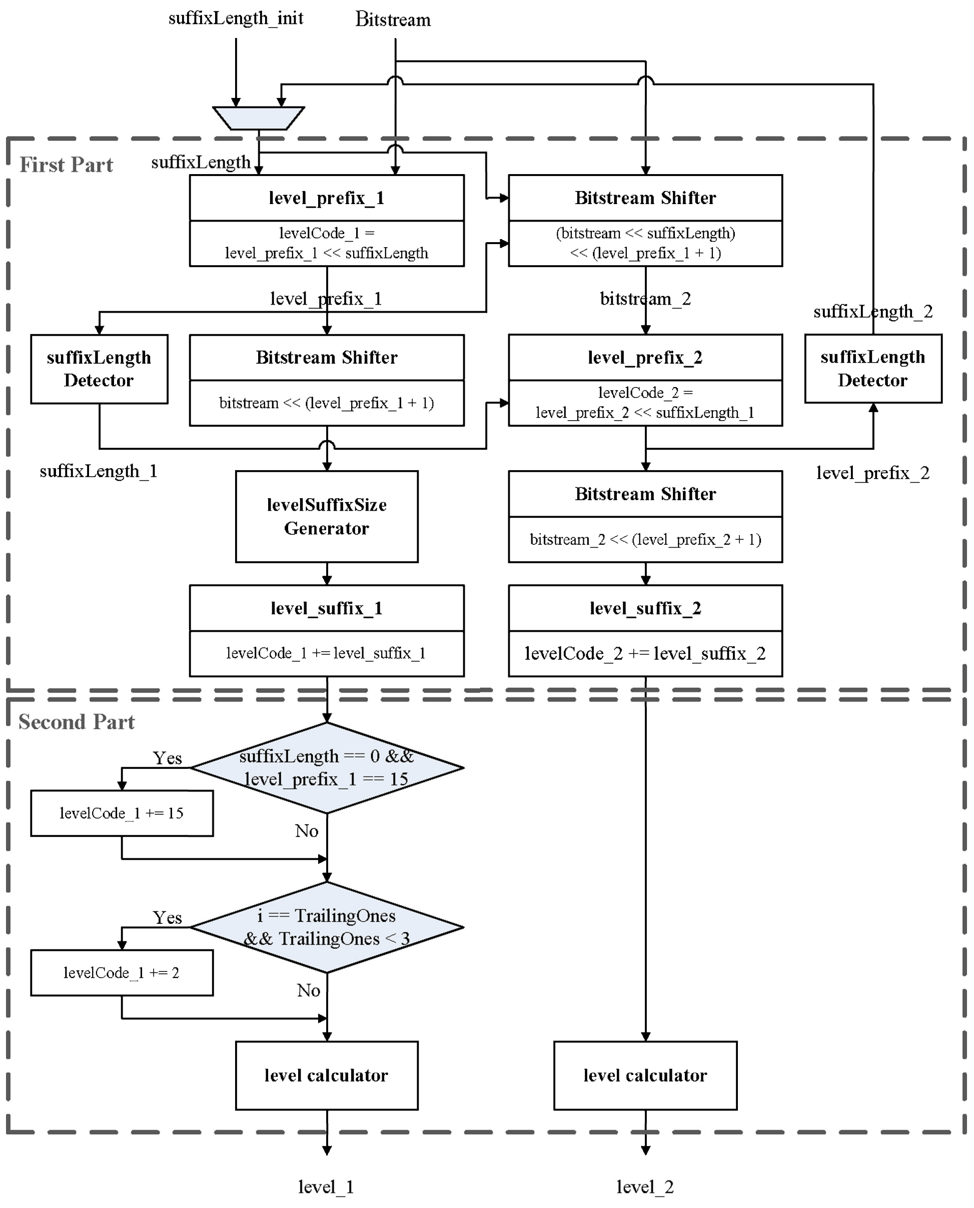

suffixLength_init
Bitstream
First Part
suffixLength
level_prefix_1
levelCode_1 = level_prefix_1 << suffixLength
Bitstream Shifter
(bitstream << suffixLength) << (level_prefix_1 + 1)
level_prefix_1
bitstream_2
suffixLength_2
suffixLength Detector
Bitstream Shifter
bitstream << (level_prefix_1 + 1)
level_prefix_2
levelCode_2 = level_prefix_2 << suffixLength_1
suffixLength Detector
suffixLength_1
levelSuffixSize Generator
Bitstream Shifter
bitstream_2 << (level_prefix_2 + 1)
level_prefix_2
level_suffix_1
levelCode_1 += level_suffix_1
level_suffix_2
levelCode_2 += level_suffix_2
Second Part
Yes
suffixLength == 0 && level_prefix_1 == 15
levelCode_1 += 15
No
Yes
i == TrailingOnes && TrailingOnes < 3
levelCode_1 += 2
No
level calculator
level calculator
level_1
level_2

Fig. 6. Proposed delay balanced two-level decoding procedure.

Table III shows an example for the reconstruction process. In the beginning, all nonzero coefficients are arranged in order, output buffer index 0 to (TotalCoeffs − 1). After total_zeros is decoded, coefficients are moved to the indices that are calculated as (coeffsLeft + zerosLeft − 1) in reverse order, and the value of the original position of the moved coefficient is replaced by 0. In this example, first, the last coefficient 1 is moved to index 8 (6 + 3 − 1), and the coefficient −1 is moved to index 6 (5 + 2 − 1). In the next cycle, only one run_before symbol is valid since no more zeros left to be decoded, and the coefficient −2 is moved to index 4 (4 + 1 − 1).

To further accelerate the decoding procedure, a skipping mechanism is employed to remove redundant decoding processes as follows.

1) *Zero Block Skip*: When TotalCoeffs is equal to 0, the remaining decoding processes are skipped since nonzero coefficients do not exist in the block.

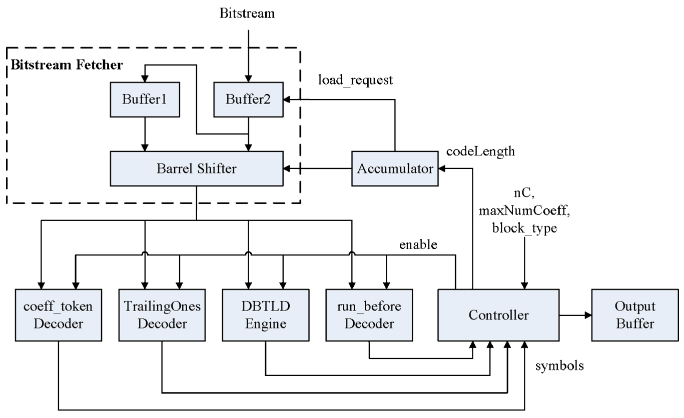


Fig. 7. Proposed CAVLC decoder.

2) *Level Skip*: When TotalCoeffs is equal to TrailingOnes, the level decoding procedure is skipped since there are no nonzero coefficients left to be decoded.

TABLE III
EXAMPLE OF RESIDUAL BLOCK RECONSTRUCTION PROCESS

Decoded Symbol	coeffsLeft	zerosLeft	Output Buffer															
total_zeros = 3	x	x	4	3	2	−2	**−1**	**1**	0	0	0	0	0	0	0	0	0	0
run_before_1 = 1 run_before_2 =	6	3	4	3	2	**−2**	0	0	**−1**	0	**1**	0	0	0	0	0	0	0
run_before_1 = 1 run_before_2 = x	4	1	4	3	2	0	**−2**	0	−1	0	1	0	0	0	0	0	0	0

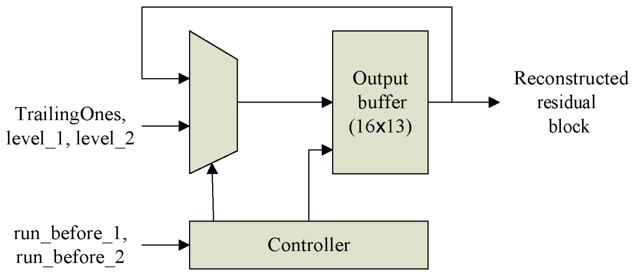


Fig. 8. Residual block reconstruction architecture.

TABLE IV
PERFORMANCE COMPARISONS FOR DIFFERENT CAVLC DECODERS FOR 4 K × 2K SEQUENCES

	Min. Working Frequency	Max. Frame Rate (With 0.18 μm Technology)
Yu [6]	174 MHz	19.1
Tsai [5]	138 MHz	30.8
Proposed	125 MHz	41.1

Min. working frequency = (Max. MB processing rate) * (Average cycle/MB) Max. frame rate = (Max. frequency)/(Average cycle/MB)/(Max. MBs/frame)

3) *Total Zeros Skip*: When TotalCoeffs is equal to maximum number of coefficients (maxNumCoeff), the total_zeros decoding procedure and run_before decoding procedure is bypassed because there are no zero coefficients to be decoded.
4) *Run Skip*: When total_zeros is equal to 0 or TotalCoeffs is equal to 1, run_before decoding procedure is not necessary.

Moreover, in the CAVLC decoding procedure, because coeff_token, trailing_ones_sign_flag, level, total_zeros, and run_before decoding units are not operated simultaneously, only one of them is designated to work in each cycle. As a result, idled units are turned off by functional gating to save power consumption.

IV. IMPLEMENTATION RESULTS

Table IV shows the average decoding performance of the proposed CAVLC decoder. To fairly compare with previous works, we use the same testing environment and technology as others that all the sequences with resolution of 4K × 2K (4096 × 2034) are intra coded.

Compared to Tsai's work [5], the main differences are the implementation strategies for the two critical loops of CAVLC decoding (level and run_before parsing process). For level decoding, we adopt the MSD algorithm in [4] to establish an efficient two-level decoding solution while the literature [5] employed a cascaded architecture with retiming technique to aim at efficient multilevel decoding. For run_before decoding, we adopt our previous work [6] since it provides a stable two-symbol throughput in general case while the performance of [5] depends on the distribution of coefficients in the residual block.

The register transfer level simulation result shows that Lee's design [8], which only focuses on boosting run_before decoding procedure, can achieve higher decoding speed in the low bit-rate coding such as high quantization parameter or simple image since the residual block is very sparse. However, in the high bit-rate coding that really demands high decoding speed, our proposed design prevails over other existing designs since we take both level and run_before decoding procedures into speedup consideration.

Table V shows the synthesis results of the proposed CAVLC decoder and a comparison with other existing works. The proposed CAVLC decoder is synthesized with CMOS 90-nm technology. Our design can enhance the throughput by exploiting multisymbol decoding scheme for both level and run_before symbols while allowing the maximum working frequency to be 385 MHz with 13.54 k gate count and 193 MHz with 14.37 k gate count in CMOS 90-nm and 0.18-μm technology, respectively. Lin's design [3] has minimum hardware cost, but its decoding speed of the two main critical loops, level and run_before, is only one symbol per cycle, which is merely half in comparison to our design. In contrast, for our design, two run_before symbols can be decoded in each cycle by applying the prediction-based run_before look-up table combination method [6]. Furthermore, with the DBTLD engine, not only two *level* symbols can be decoded at the same cycle, but also 21% critical path delay is saved in comparison to the traditional two-level decoder. Table VI shows the maximum frame rate (frames per second) for different Level limits defined in H.264/AVC standard. The result shows that our proposed CAVLC decoder can achieve real-time decoding for all Level conditions. In addition, the maximum frame rate of our design reveals that our proposal can decode more than 82 f/s in real time for 4096×2304 frame resolution and thus suitable for H.264/AVC scalable and multiview extension.

TABLE V

IMPLEMENTATION RESULT COMPARISONS FOR DIFFERENT CAVLC DECODER DESIGNS

Specifications	Proposed		Lin [3]	Yu [6]	Lee [8]	Alle [4]	Tsai [5]
Technology	90 nm	0.18 μm	0.18 μm	0.18 μm	0.13 μm	0.13 μm	0.18 μm
Max. frequency	385 MHz	193 MHz	213 MHz	125 MHz	125 MHz	250 MHz	160 MHz
Area: Logic part (gate count)	13 544	14 373	6771	13 192	15 602	17 202	13 175
Area: Memory part (bits)	W/O		W/O	W/O	W/O	5120	W/O
Average cycle/MB	127.13		N/A	177	148.8	N/A	141

TABLE VI

WORKING FREQUENCY FOR DIFFERENT LEVEL LIMITS

Level	Max. MBs/Frame	Max. MB Processing Rate (MBs/s)	Max. Frame Format	Max. Specified Frame Rate	Working Frequency	Max. Frame Rate of Our Design (f/s)
1	99	1458	QCIF	15.0	0.19 MHz	30589.9
2	396	11 880	CIF	30.0	1.52 MHz	7647.5
3	1620	40 500	625 SD	25.0	5.15 MHz	1869.4
3.1	3600	108 000	720p HD	30.0	13.73 MHz	841.2
3.2	5120	216 000	SXGA	42.2	27.46 MHz	591.5
4	8192	245 760	2K × 1K	30.0	31.25 MHz	369.7
4.2	8704	522 240	2K × 1080	60.0	66.4 MHz	347.9
5	22 080	589 824	3672 × 1536	26.7	74.95 MHz	137.2
5.1	36 864	983 040	4096 × 2304	26.7	125.13 MHz	82.2

V. CONCLUSION

To realize high decoding performance and low hardware cost real-time entropy decoding systems, a high-throughput and fully hardwired entropy decoder for H.264/AVC was proposed. Unlike previous multisymbol CAVLC decoding architectures, which only accelerated the decoding procedure of run_before symbols, our proposed CAVLC decoder can further elevate the throughput by applying the DBTLD architecture that can decode two *level* symbols in one cycle and shorten the critical path delay by 21% in comparison to the conventional approach of cascading two level decoders, and allowed the maximum working frequency to be about 385 MHz. Implementation results showed that our proposed entropy decoder can support up to level 5.1 entropy decoding with only 13.54 k gate counts.

REFERENCES

[1] *Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec.H.264 ISO/IEC 14496-10 AVC)*," document JVT-G050, Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, Mar. 2003.

[2] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 560–576, Jul. 2003.

[3] H.-Y. Lin, Y.-H. Lu, B.-D. Liu, and J.-F. Yang, "A highly efficient VLSI architecture for H.264/AVC CAVLC decoder," *IEEE Trans. Multimedia*, vol. 10, no. 1, pp. 31–42, Jan. 2008.

[4] M. Alle, J. Biswas, and S. K. Namdy, "High performance VLSI architecture design for H.264 CAVLC decoder," in *Proc. IEEE Int. Conf. Applicat.-Specific Syst., Architect. Processors*, Sep. 2006, pp. 317–322.

[5] T.-H. Tsai, T.-L. Fang, and Y.-N. Pan, "A novel design of CAVLC decoder with low power and high throughput considerations," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 31, no. 3, pp. 311–319, Mar. 2011.

[6] G.-S. Yu and T.-S. Chang, "A zero-skipping multi-symbol CAVLC decoder for MPEG-4 AVC/H.264," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2006, pp. 5583–5586.

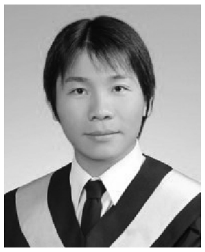
[7] Y.-N. Wen, G.-L. Wu, S.-J. Chen, and Y.-H. Hu, "Multiple-symbol parallel CAVLC decoder for H.264/AVC," in *Proc. IEEE Asia Pacific Conf. Circuit Syst.*, Dec. 2006, pp. 1240–1243.

[8] G.-G. Lee, C.-C. Lo, Y.-C. Chen, H.-Y. Lin, and M.-J. Wang, "High-throughput low-cost VLSI architecture for AVC/H.264 CAVLC decoding," *IET Image Process.*, vol. 4, no. 2, pp. 81–91, 2010.

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

Yuan-Hsin Liao received the B.S. and M.S. degrees in electronics engineering from National Chiao Tung University, Hsinchu, Taiwan, in 2008 and 2010, respectively.

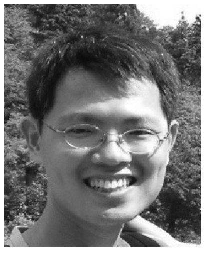
In 2010, he joined the PixArt Imagine, Inc., Hsinchu. His current research interests include video processing, computer vision, and intellectual property and system-on-a-chip design.



Gwo-Long Li received the B.S. degree from the Department of Computer Science and Information Engineering, Shu-Te University, Kaohsiung, Taiwan, in 2004, the M.S. degree from the Department of Electrical Engineering, National Dong Hwa University, Hualien, Taiwan, in 2006, and the Ph.D. degree from the Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan, in 2011.

He is currently an Engineer with the Industrial Technology Research Institute (ITRI), Hsinchu. His current research interests include video signal processing and its very large-scale integrated architecture design.

Dr. Li was the recipient of the Excellent Masters Thesis Award from the Institute of Information and Computer Machinery in 2006.



Tian-Sheuan Chang (S'93–M'06–SM'07) received the B.S., M.S., and Ph.D. degrees in electronic engineering from National Chiao Tung University (NCTU), Hsinchu, Taiwan, in 1993, 1995, and 1999, respectively.

He is currently a Professor with the Department of Electronics Engineering, NCTU. From 2000 to 2004, he was a Deputy Manager with Global Unichip Corporation, Hsinchu. His current research interests include (silicon) intellectual property and system-on-a-chip design, very large-scale integration signal processing, and computer architecture.