Design of a High Speed CAVLC Encoder and Decoder with Parallel Data Path

G Abhilash
M.Tech Student, CVSR College of Engineering,
Department of Electronics and Communication Engineering,
Hyderabad, Andhra Pradesh, India.

R Ramprakash
Assistant Professor, CVSR College of Engineering,
Department of Electronics and Communication Engineering,
Hyderabad, Andhra Pradesh, India.

Abstract- Variable length coding (VLC) is very suitable for regular data and efficient to compress data without any loss. VLC uses shorter bits of codewords instead of data occurring frequently, but uses the longer bits of codewords instead of data occurring infrequently. It is used in MPEG – 1/2/4 and H.26X (video and image compression standards). The entropy decoder in MPEG-4 AVC/H.264 baseline standard adopts Content Adaptive Variable Length Decoder (CAVLD). Because of symbol-to-symbol dependency, a traditional CAVLC decoder consumes lots of clock cycles in decoding and brings down the performance. We discover the decoding of two parameters spending almost eighty percent of computing time through profiling the computation of sub-modules and analyzing the encoding rules, which are non-zero coefficient (Level) and run_before. Thus this paper proposes a fast algorithm adapted for run_before decoder and the parallel architecture for level decoder, to improve the decoding performance. According to the features of these two methods, we name these two new methods as MLD (Multiple Level Decoding) and NZS (Non-Zero Skipping for run_before decoding). By performing parallel operation on level decoder, MLD can decode two levels in one cycle at most situations, and NZS can produce several values of run_before in the same cycle. These two methods have the advantages of low complexity and regularity design.

Keywords-- CAVLC Encoder, Decoder, H.264.

I. INTRODUCTION

In coding theory, variable-length code can map source symbols to a binary code with variable number of bits and it is a technique of lossless data compression. Variable-length code can allow sources to be compressed and decompressed with zero error and still be read back symbol by symbol. With the right coding strategy an i.i.d. source may be compressed almost arbitrarily close to its entropy. This is in contrast to fixed-length coding methods, for which data compression is only possible for large blocks of data, and any compression beyond the logarithm of the total number of possibilities comes with a finite probability of failure. Some examples of well-known variable-length coding strategies are Huffman coding, Lempel-Ziv coding and arithmetic coding.

Context-Adaptive Variable Length Coding (CAVLC) is a form of entropy coding used in H.264/MPEG-4 AVC video encoding. It is an inherently lossless compression technique, like almost all entropy-coders. In H.264/MPEG-4 AVC, it is used to encode residual blocks of transform coefficients in zigzag order. It is an alternative to Context-based adaptive binary arithmetic coding (CABAC), a more space-efficient entropy encoding scheme which requires considerably more processing to decode. CAVLC is supported in all H.264 profiles, unlike CABAC which is not supported in Baseline and Extended profiles.

The rest of the paper is organized as follows. Proposed embedding and extraction algorithms are explained in section II. Experimental results are presented in section III. Concluding remarks are given in section IV.

II. PROPOSED ALGORITHM

There are five elements in CAVLC codeword and strong dependency is existed between these elements. According to the architecture of previous multi-symbol design, the decoding process for Coeff_token and sign of T1s can be merged in the same process, and then a typical CAVLC decoding flow can be simplified to four steps as shown in

Figure 1. Furthermore, we are going to improve both level decoder and run_before decoder to achieve the requirement of multi-symbol decoding. In this paper, our proposed level decoder can produce two symbols and several symbols in run_before decoder in the most cases.
For this example in Figure 1, it needs one cycle for Coeff_token & T1s sign decoder and Total_zeros decoder respectively. Level decoder needs five cycles, and run_before decoder needs 2 cycles. Totally the decoding process needs 9 cycles. Compared to a typical CAVLC decoder whose decoding cycle is 22, we can reduce 59% of cycles. The more detailed descriptions are shown as follows.

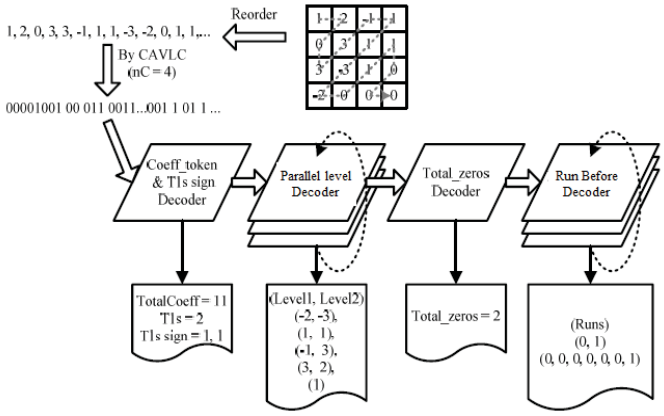

Figure 1. Simplified CAVLC Algorithm

2.1. Combined Coeff_token & T1s sign Decoder

Coeff_token decoding is a process of look-up table whose algorithm is the same as typical one. Reviewing the VLC tables of coeff_token, it is obvious to see that many redundant data can be eliminated. For the purpose of saving the hardware cost, we have to classify and merge these tables. The architecture of look-up table will be described in detail in next chapter. According to Yu as soon as the Coeff_token is decoded, the length of sign bits for trailing ones is predictable. Thus we can decode all the sign bits of trailing ones in the same cycle instead of one cycle for one sign. Furthermore, the logic circuit of T1s sign decoding is very simple such that the combination of these two processes is possible. Because more than one symbol will be decoded, the level buffer of output unit has to support multiple inputs.

2.2. Parallel Level Decoder

Based on the concept of the word-level parallel processing, several level decoders are employed to meet this kind of requirement. This initiates our MLD (Multiple Level Decoding) method. With this method, the proposed level decoder can produce two symbols in every clock cycle at the most cases.
According to the CAVLC decoding algorithm, we know that suffix Length has to be updated in every level decoded. Because of the dependency of each suffix Length, we have to straight pass this information to the second level decoder such that two levels can be decoded simultaneously.
Our proposed design does not support the full case of two-level decoding because it results in a huge shift circuit. Only when the total bits of these two levels are less than 32 bits, two levels decoded can be performed successfully. We will use one cycle for one level when the total bits of two levels are more than 32 bits.
Because this case is seldom occurred, there is no increase on decoding cycles. The diagram of two-level decoding is illustrated in Figure 2.

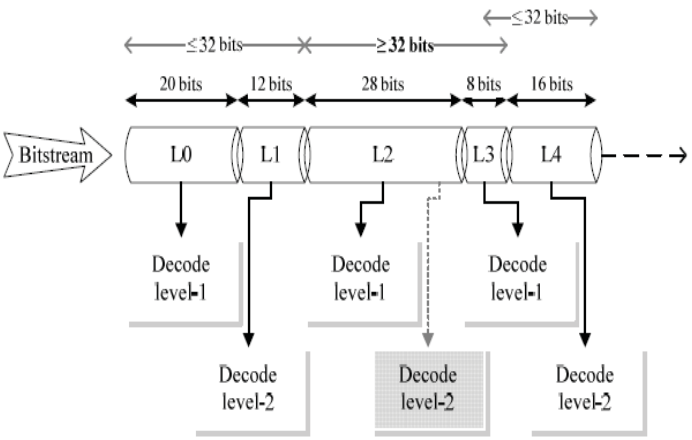

Figure 2. The diagram of Multiple Level decoding for level decoder

2.3 Total_zeros Decoder

This is also a process of table look-up which is similar to the process of TotalCoeff, and we also partition the VLC tables into several sub groups to eliminate the redundant data. According to the parameter TotalCoeff, we can obtain the corresponding value. The implementation is the same as coeff_token decoder.

2.4. Non-Zero Skipped Run_before Decoder

In run_before decoding, each level is represented with a run_before which indicates the number of zeros before leading level. Traditionally run_before decoding methods solved only one run_before at each cycle. There is an example illustrated in TABLE 1, we find four runs decoded by four consecutive ones, and furthermore each value of run is zero. This is an important and good character to speed up the decoding performance, and then we trace the algorithm carefully and analyze every VLC table. It reveals some regularity in run_before table which is marked in italics in TABLE 2. We find that, in different columns, the codeword consists of ones if the value of run_before equals to zero. The codewords will be single one, double ones or triple ones, if the value of run_before equals to zero.

Symbol	Codeword	Value	Output samples
Run_before	01	zerosLeft=2; run=1	3,2,1,-1,-1,0,1
Run_before	**1**	zerosLeft=1; run=**0**	3,2,1,-1,-1,0,1
Run_before	**1**	zerosLeft=1; run=**0**	3,2,1,-1,-1,0,1
Run_before	**1**	zerosLeft=1; run=**0**	3,2,1,-1,-1,0,1
Run_before	**1**	zerosLeft=1; run=**0**	3,2,1,-1,-1,0,1
Run_before	N/A	zerosLeft=1; run=1	0,3,2,1,-1,-1,0,1

Table 1. An example of Run_before decoding

We propose the NZS (Non-Zero Skipping for run_before decoding) method to perform the multi-symbol decoding on run_before. The key idea of NZS is to look forward the bitstream and find out how many run_befores with the values of zero.

Table 2. Table for run_ before

Run_before	1	2	3	4	5	6	> 6
0	**1**	**1**	**11**	**11**	**11**	**11**	**111**
1	0	01	10	10	10	000	110
2		00	1	01	011	001	101
3			0	001	010	011	100
4				000	001	010	011
…	…	…	…	…	…	…	…
13							0000000001
14							00000000001

III. EXPERIMENT AND RESULT

The 4x4 set of coefficients are selected from the internet. ModelSim 10.1d software platform is use to perform the experiment. The PC for experiment is equipped with an Intel P4 2.4GHz Personal laptop and 2GB memory.

The proposed Algorithm uses Parallel level decoder and Non zero skipping decoder to increase the speed of CAVLC Algorithm. By using these two methods we complete decoding process with in only 9 clock cycles. Compared previous researches the proposed algorithm is 3 times faster. Table 3 shows the comparison of average cycles.

Table 3. Comparison of Average Cycles

	Frequency (MHz)	Average Cycle (per MB)
[11]	125	193
[12]	71.43	480
[14]	175	312
[20]	74.25	279
Proposed	160	141

IV. CONCLUSION

By analyzing the CAVLC algorithm and the structure of VLC tables, we propose a new algorithm for run_before decoder and a parallel architecture for level decoder, which are called NZS and MLD respectively. With an aid of these two methods, the decoding cycle can be saved a lot, to speed up the hardware performance. Consequently our design can drive the H.264 decoder to achieve the real-time requirement running at a low operation frequency. The result of evaluation shows that the proposed CAVLC decoder uses the least processing cycle for decoding a macroblock.

REFERENCES

[1] Advanced Video Coding, document JVT-E022, Final Committee Draft, ITU-T Rec.H.264/ISO/IEC 11496-10, Sep. 2002.

[2] V. Lappalainen, A. Hallapuro, and T. D. Hämäläinen, “Complexity of optimized H.26L video decoder,” IEEE Trans. Circuits Syst. Video Technol., vol. 13, no. 7, pp. 717–725, Jul. 2003.

[3] M. Horowitz, A. Joch, F. Kossentini, and A. Hallapuro, “H.264/AVC baseline profile decoder complexity analysis,” IEEE Trans. Circuits Syst. Video Technol., vol. 13, no. 7, pp. 704–716, Jul. 2003.

[4] X. Quan, L. Jilin, W. Shijie, and Z. Jiandong, “H.264/AVC baseline profile decoder optimization on independent platform,” in Proc. WCNM, Sep. 2005, pp. 1253–1256.

[5] X. Qin and X. Yan, “A memory and speed efficient CAVLC decoder,” Proc. SPIE, vol. 5960, pp. 1418–1426, Jul. 2005 [Online]. Available: http://spiedigitallibrary.org/proceedings/resource/2/psisdg/5960/1/596046 1.

[6] Y. C. Chao, S. T. Wei, J. F. Yang, and B. D. Liu, “Combined CAVLC decoder and inverse quantizer for efficient H.264/AVC decoding,” in Proc. IEEE Int. Conf. Asia Pacific Conf. Circuits Syst., Dec. 2006, pp.259–262.

[7] 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.

[8] Y. H. Moon, “A new coeff-token decoding method with efficient memory access in H.264/AVC video coding standard,” IEEE Trans. Circuits Syst. Video Technol., vol. 17, no. 6, pp. 729–736, Jun. 2007.

[9] W. Di, G. Wen, H. Mingzeng, and Z. Ji, “A VLSI architecture design of CAVLC decoder,” in Proc. 5th Int. Conf. ASIC, Oct. 2003, pp. 962–965.

[10] M. A. J. Biswas and S. K. Nandy, “High performance VLSI architecture design for H.264 CAVLC decoder,” in Proc. ASAP, Sep. 2006, pp. 317– 322.

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

[12] 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 Int. Conf. Asia Pacific Conf. Circuits Syst., Dec. 2006, pp. 1240–1243.
[13] T. H. Tsai, D. L. Fang, and Y. N. Pan, "A hybrid CAVLD architecture design with low complexity and low power considerations," in Proc. IEEE Int. Conf. Multimedia Expo, Jul. 2007, pp. 1910–1913.
[14] 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," in Proc. IEEE Int. Conf. Circuits Syst., May 2005, pp. 6110– 6113.
[15] Y. H. Moon, G. Y. Kim, and J. H. Kim, "An efficient decoding of CAVLC in H.264/AVC video coding standard," IEEE Trans. Consumer Electron., vol. 51, no. 3, pp. 933–938, Aug. 2005.
[16] H. Y. Lin, Y. H. Lu, B. D. Liu, and J. F. Ynag, "Low power design of H.264 CAVLC decoder," in Proc. IEEE Int. Conf. Circuits Syst., May 2006, pp. 2689–2692.
[17] I. E. G. Richardson, H.264 and MPEG-4 Video Compression—Video Coding for Next Generation Multimedia. New York: Wiley, 2003, pp. 198–207.
[18] T. L. Chang, Y. M. Tsai, C. D. Chien, C. C. Lin, and J. I. Guo, "A high-performance MPEG4 bitstream processing core," in Proc. ICME, Jun. 2004, pp. 467–470.
[19] K. Sühring, Ed. (2007). JVT Reference Software JM 11.0 [Online]. Available: http://bs.hhi.de/−suehring/tml
[20] T. G. George and N. Malmurugan, "A new fast architecture for HD H.264 CAVLC multi-syntax decoder and its FPGA implementation," in Proc. ICCIMA, Dec. 2007, pp. 118–122.