A Conflict-free Multi-Symbol Arithmetic Encoder for H.264/AVC

De-Yuan Shen, Che-Wei Chang, Yu-Nan Pan and Tsung-Han Tsai
Dept. of Electrical Engineering, National Central University,
Taoyuan 32001, Taiwan
E-mail: {dyshen, cwc, kobbe, han}@dsp.ee.ncu.edu.tw

***Abstract*—This paper proposed a conflict-free multi-symbol arithmetic encoder (AE) for H.264/AVC. The throughput of a multi-symbol AE is usually limited by concurrent access of context memory which degrades the performance of the whole H.264/AVC encoder. With proposed Hybrid Context Memory architectures (HCM), about 1/10 contexts under critical conflict are implemented by Critical Register Array to solve the insufficient ports bottleneck. The number of encoding symbol per cycle is improved by 15% and 55.5% in 2-symbol and 4-symbol version, respectively. The implementation results also show that the hardware design can achieve a high encoding rate and throughput comparing to the previous works.**

I. INTRODUCTION

H.264/Advanced Video Coding (AVC) is the latest video coding standard developed by the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG) [1]. Two entropy coding techniques, context-based adaptive binary arithmetic coding (CABAC) and context-based adaptive variable length coding (CAVLC) are used in the standard. CABAC adopted in main and high profiles, offers comparison results that are 9%-14% better than those obtained with the baseline CAVLC [2] but at the expense of increased complexity in the entropy coder.

Fig. 1 shows a CABAC block diagram. First, a syntax element and neighboring side information are processed to generate its binary symbols and associated context indexes by Binarizer and Context Address Generator, respectively. Then, the pairs of symbol and context index are consecutively loaded into a binary arithmetic encoder (AE). The encoding process of AE consists of three functional elements: probability state accessing, tag (*low*) and interval (*range*) maintaining. Most Probable Symbol (*MPS*) and range of Least Probable Symbol (*rangeLPS*) can be obtained after probability state of a symbol is accessed from context memory. If the symbol is matched to the *MPS*, *range* is subtracted by *rangeLPS*; otherwise, *range* is set equal to *rangeLPS*. *low* is always updated to the lower bound of next *range*. Moreover, a renormalization process that enlarges the *range* and generates the coded bits for output is evoked if *range* is considered small. Besides regular coding, bypass coding is applied for the symbols with equal probability to speed up encoding process.

Several researches have been proposed to reduce the architecture complexity and improve the throughput of AE.

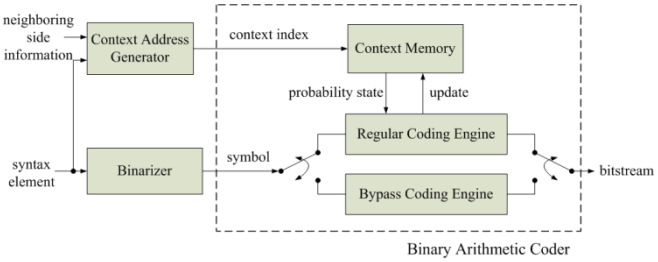

Fig. 1 The CABAC block diagram.

Basically, they can be classified by its parallelism as one-symbol or multi-symbol. A multi-symbol AE is more suitable under high throughput and low power consideration. It encodes multiple symbols at a cycle so that the given throughput can be achieved with relatively low operating frequency, and vice versa. Lo *et al.* [5] encode 2 residual symbols in parallel to enhance the throughput. Moreover, Chen *et al.* [3] proposed the forwarding and pre-read/write techniques to increase the throughput of SRAM-based AE. Ideally, the throughput should be increased proportionally to the parallelism of the design. For example, 2 symbol/cycle is generated with a 2-symbol AE and 4 symbol/cycle is generated with a 4-symbol AE. However, it does not usually reach the case. The bottleneck lays in conflict of the context memory which would degrade half of the throughput.

This work proposed a conflict-free multi-symbol AE that aims to deal with the throughput loss suffering from memory conflict. Contexts are divided into two groups based on the rate of conflict happens. Then, Hybrid Context Memory (HCM), i.e. a multi-bank dual-port SRAM and Critical Register Array, individually implements the two groups of contexts. The simulation results show that up to 55.5% throughput improvement is reached.

The remainder of the work is organized as follows. In Section 2, the proposed multi-symbol AE and its' design bottleneck are described. In Section 3, the proposed HCM architecture is introduced. Section 4 shows the implementation results. Finally, we conclude the work in Section 5.

II. PROPOSED MULTI-SYMBOL AE

The block diagram of proposed multi-symbol AE is shown in Fig. 2. For each one-symbol AE coding engine, it is divided into a seven-stage pipeline architecture, i.e. *AG context*, *read*

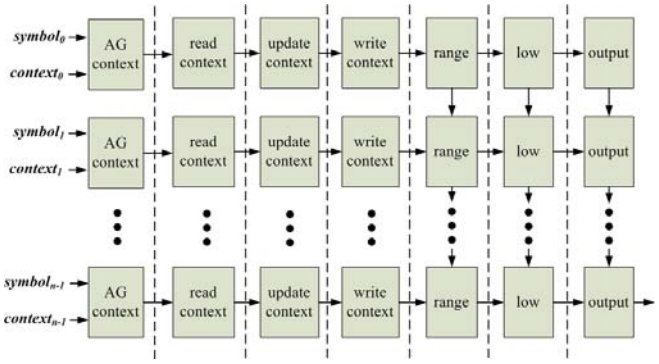

Fig. 2 The proposed multi-symbol AE block diagram.

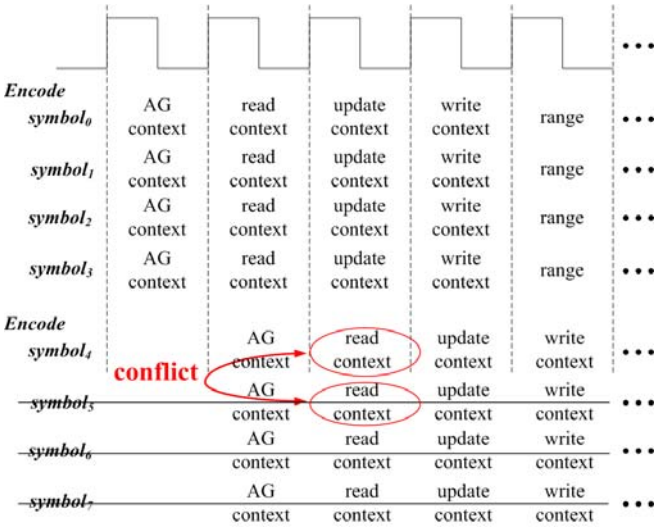

Fig. 3 The conflict in the multi-symbol AE.

context, *update context*, *write context*, *range*, *low* and *output*. In *AG*, *read*, *update* and *write context* stage, it reads probability state of a context, updates it; and then writes back to the context memory. *range* and *low* stage maintain the updated value of range and low, respectively. Finally, *output* stage resolves outstanding bits and packs output bits into coded bitstreams. The AEs are cascaded in order to encode *n* symbols in parallel. The probability states of each encoding symbol are read/update/write concurrently. Moreover, the updated value of range and low are propagated to next symbol so that the updated value can be used in the same cycle.

As we can see, *n* times read and *n* times write would simultaneously operate in a cycle. That means, unless, *2n* ports are required for the context memory. A multi-bank dual-port memory, then, is adopted in the proposed design. To meet a better performance, the number of bank in the memory is chosen equal to *n*. Because read/write ports utilization would be low if the number is larger than *n*; on the contrary, access conflict would frequently occurred if it is less than *n*.

A *conflict* occurs when more than one context has to read from or write to the same bank concurrently. For example, as shown in Fig.3, if contexts of $symbol_4$ and $symbol_5$ are stored

TABLE I
THROUGHPUT LOSS OF THE MULTI-SYMBOL AE.

parallelism	test sequences	throughput	throughput loss (%)
2-symbol	352x288	1.71	14.5%
	720x480	1.68	16%
	1920x1088	1.63	18.5%
4-symbol	352x288	2.45	38.8%
	720x480	2.43	39.3%
	1920x1088	2.27	43.3%

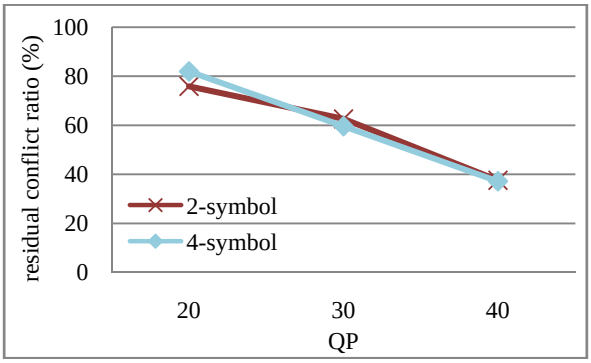

Fig. 4 The conflict rate for context of residual symbols.

in the same bank in the 4-symbol AE, a conflict may arises. The $symbol_5$ would not be encoded at current cycle because its probability state is not ready. Due to the data dependency, the symbols after $symbol_5$ would not be coded as well. A bubble cycle would arise and only one symbol is coded in the cycle. Obviously, the throughput is degraded a lot in this case. The simulation results show that the conflict brought up to 43% throughput loss as shown in Table I. With the serious throughput loss, the hardware utilization is also considered low. Therefore, to design a conflict-free context memory architecture is quite necessary for multi-symbol AE.

III. PROPOSED CONFLICT-FREE HYBRID CONTEXT MEMORY ARCHITECTURE

The Hybrid Context Memory architecture (HCM) is proposed in our design. Contexts are divided into two groups, i.e. *critical context* and *normal context*, depending on the rate of conflict happens. For the critical ones, conflict constantly happens because they easily occur conflict with the contexts of neighboring symbols. To solve this problem, critical contexts are better to be implemented by a memory component with sufficient read/write ports. Thus, we use register array, i.e. Critical Register Array (CRA), to implements the critical contexts since it does not have limited port characteristic. On the contrary, the remaining contexts are simply implemented by multi-bank dual-port SRAM since *2n* ports may be quite enough. It should be noted that the area cost may be increased by adopting the register array and several comparators. Therefore, only partial but most critical of contexts are selected to critical group.

In fact, as shown in Fig. 4, we observed that most conflicts are occurred during the encoding the residual data especially

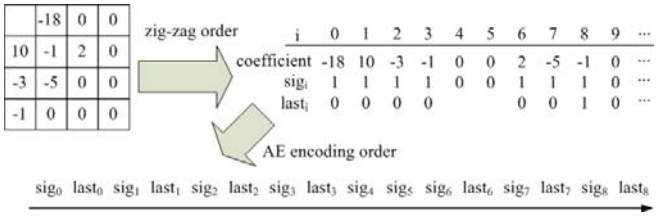

Fig. 5 Example of encoding symbols for a 4x4 luma block.

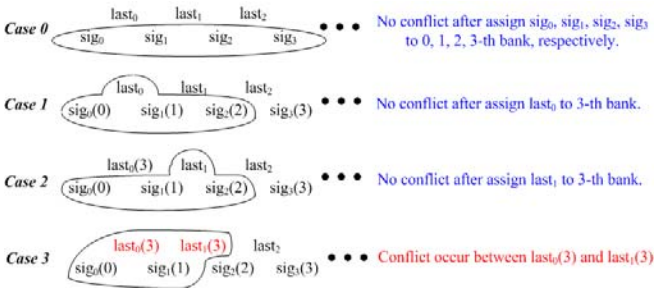

Fig. 6 The conflict between sig and last symbols.

when video quality is high (larger amount of residual data). The residual syntax elements include coded block pattern(*CBP*), significant coefficient flag (*sig*), last significant coefficient flag (*last*), coded block flag, coefficient level and coefficient sign flag.

Fig. 5 shows an example of a 4x4 block of luma transform coefficients. Coefficients are scanned in zig-zag order and a *sig* bit is used to determine whether its value is zero. If a *sig* bit is not zero, it is follow by a *last* bit to determine whether the last nonzero coefficient in the block is reached. After *sig* and *last* were coded, coefficients level and sign flag start to be coded in the backward direction.

The contexts between *sig* and *last* symbols contribute most throughput loss. It is because they are interlacedly located in encoding process. As shown in Fig. 6, a series of *sig* and *last* symbols are coded by a 4-symbol AE. A *last* symbol is coded depended on the value of prior *sig* symbol. For example, in case 0, sig_0, sig_1 and sig_2 are zero and no *last* symbol is coded. Moreover, we assign sig_0, sig_1, sig_2, sig_3, $last_0$ and $last_1$ to proper banks in order to avoid conflict in case 0~2. However, eventually it causes conflict in case 3. Obviously, it is impossible to assign them into four bank memory without any throughput loss. Since the conflict is evitable, it frequently happens during encoding series of *sig* and *last* symbols. According to simulation results, the context of last symbols in Luma4x4, Chroma DC and Chroma AC block are assigned as critical context. Besides, *CBP* also introduces lots of throughput loss. It indicates which 8x8 blocks in the macroblock contain nonzero coded transform coefficients. It tends to generates conflict between itself and neighboring syntax elements.

Therefore, there are 44 contexts stored in CRA as shown in Table II. The remaining contexts are uniformly distributed to each bank of dual-port memory by assigning them with modular order (*context index % numbers of bank*). 44 of 496

TABLE II
THE CRITICAL CONTEXTS DEFINED IN THE DESIGN.

Syntax element	number of context
Luma block $last_0$~$last_{15}$	15
Chroma DC block $last_0$~$last_3$	3
Chroma AC block $last_0$~$last_{14}$	14
Coded block pattern	12
Total	44

CRA and it is considered as a fair tradeoff between area cost and throughput.

The architecture and dataflow of HCM for 4-symbol AE is shown in Fig. 7 (a), (b). CRA stores the probability state of critical contexts, and the remaining contexts are stored in the 4-bank dual-port SRAM. For each context read operation, the probability state can be read from the dual-port memory path or CRA path. If the context is a critical context, the data from CRA is chosen; otherwise, the one from dual-port path with corresponding bank is chosen. In addition, each updated probability state is written to either dual-port memory or CRA depending on whether its associated context belongs to critical context.

IV. IMPLEMENTATION RESULTS

The proposed multi-symbol AE hardware architecture is written in Verilog HDL and synthesized using Synopsys Design Compiler targeted towards as Artisan TSMC 0.18 um cell library. We choose and implemented 2-symbol and 4-symbol versions. Several video sequences from 352x288 (*foreman, akiyo, mobile*), 720x480 (*crew, habour, sailormen*) to 1920x1088 (*sunflower, rush_hour, station2*), with low to high motion and different quantization parameter (20, 30, 40) are used for simulation. Table III and IV show the comparisons with other multi-symbol AE designs. With HCM, a higher symbol per cycle and throughput is achieved. In 2-symbol and 4-symbol, the symbol per cycle is increased to 1.92 and 3.70 which are close to the ideal value 2 and 4. Notably, there is still throughput loss in the multi-bank dual-port SRAM. Comparing to the case without optimization (referring to Table I), it achieves 15% and 55.5% improvement of throughput. The improvement is also significant comparing to other designs (4.3%~37.1%). Moreover, since the symbol per cycle is higher in the proposed design, we can achieve a lower operating frequency in a given real-time application. The hardware can achieve 639M/685M throughput operating at 185M/333MHz with 12.6K/34.6K gates in 2-symbol/4-symbol versions. Our design exhibits better performance than Lo‘s work [5] with higher symbol per cycle and throughput. The maximum operating frequency is higher because the architecture is well divided into balanced pipeline stages. Compared with 1.84/3.32 symbol per cycle in Chen’s work [3], a 48M higher encoding rate is reached with limited area overhead.

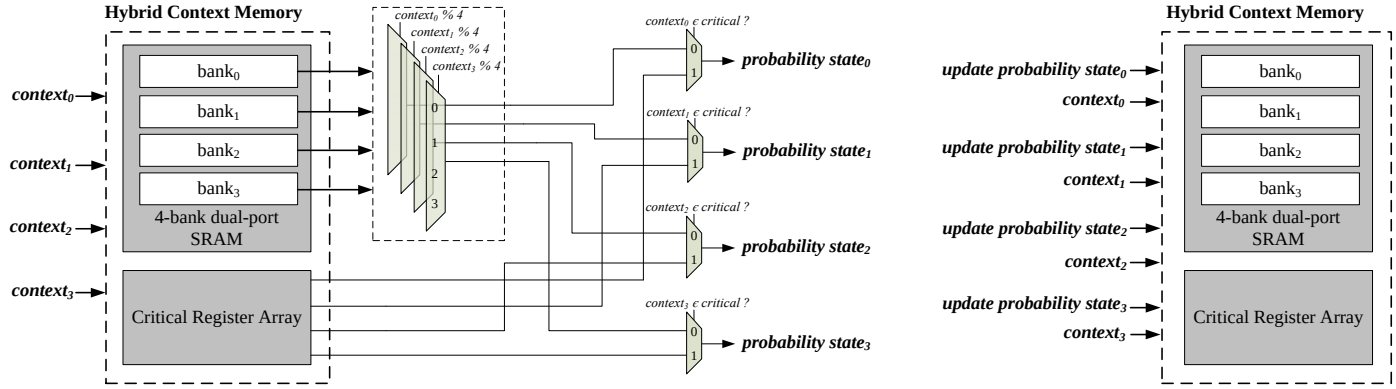

Fig. 7 Architecture and dataflow of HCM. (a) read context, (b) write context.

TABLE III

SYMBOL PER CYCLE COMPARE TO PREVIOUS WORKS.

Parallelism	Design	symbol/cycle	imrpovement (%)
2-symbol	[5]	1.4 ~ 1.70*	37.1%~12.9%
	[4]	1.57	22.3%
	[3]	1.84	4.3%
	proposed	**1.92**	-
4-symbol	[3]	3.32	11.4%
	proposed	**3.70**	-

TABLE IV

HARDWARE PERFORMANCE COMPARE TO PREVIOUS WORKS.

(ALL IMPLEMENTED IN 0.18 UM TECHNOLOGY)

Parallelism	Design	Area	Max operating frequency	Throughput (symbol/sec)
2-symbol	[5]	0.451 mm^2	135MHz	189M ~ 230M
	[3]	18.9K gates	345MHz	635M
	proposed	**12.6K gates**	**333MHz**	**639M**
4-symbol	[3]	32.1K gates	192MHz	637M
	proposed	**34.9K gates**	**185MHz**	**685M**

V. CONCLUSIONS

The conflict-free Hybrid Context Memory for AE is proposed in this work. With individually implementing contexts under different memory component, the memory access conflict can be avoided. The implementation results show that symbol per cycle is improved by 15% and 55.5% in 2-symbol and 4-symbol version, respectively. For a given real-time application, the proposed design is able to operate in a corresponding low frequency or less cycle since the symbol per cycle is higher. The hardware can achieve 639M/685M throughput which is the highest comparing to the previous works. Also, the proposed design is reconfigurable so that it can be redesign with different parallelism scheme to meet the system requirement.

REFERENCES

[1] Joint Video Team, Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification, ITU-T Rec. H.264 and ISO/IEC 14496-10 AVC, 2003.

[2] D. Marpe, H. Schwartz and T.Wiegand, "Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC video compression standard," *IEEE Trans. Circuits System Video Technology*, vol. 13, no. 7, pp. 620–636, Jul. 2003.

[3] Yu-Jen Chen, Chen-Han Tsai and Liang-Gee Chen, **"**Architecture design of area-efficient SRAM-based multi-symbol arithmetic encoder in H.264/AVC", *IEEE International Symposium on Circuits and Systems*, Greece, May 2006.

[4] Sze, V., Chandrakasan, A.P., Budagavi, M. and Minhua Zhou, **"**Parallel CABAC for low power video coding", *IEEE International Conference on Image Processing*, San Diego, U.S.A, Oct. 2008.

[5] Chia-Cheng Lo, Ying-Jhong Zeng and Ming-Der Shieh , "Design and test of a high-throughput CABAC encoder", *IEEE Region 10 Conference*, Oct. 2007.