A HIGH THROUGHPUT VLSI ARCHITECTURE DESIGN FOR H.264 CONTEXT-BASED ADAPTIVE BINARY ARITHMETIC DECODING WITH LOOK AHEAD PARSING

Yao-Chang Yang[1], Chien-Chang Lin[1], Hsui-Cheng Chang[1], Ching-Lung Su[2], and Jiun-In Guo[1]

[1]Dept. of Computer Science and Information Engineering, National Chung Cheng University, Chia-Yi 621, Taiwan, ROC
[2]Department of Electronics Engineering, National Yunlin University of Science Technology, Yun-Lin 640, Taiwan, ROC
e-mail: {yangyc, lcc, chc92, jiguo}@cs.ccu.edu.tw[1], kevinsu@yuntech.edu.tw[2]

ABSTRACT

In this paper we present a high throughput VLSI architecture design for Context-based Adaptive Binary Arithmetic Decoding (CABAD) in MPEG-4 AVC/H.264. To speed-up the inherent sequential operations in CABAD, we break down the processing bottleneck by proposing a look-ahead codeword parsing technique on the segmenting context tables with cache registers, which averagely reduces up to 53% of cycle count. Based on a 0.18μm CMOS technology, the proposed design outperforms the existing design by both reducing 40% of hardware cost and achieving about 1.6 times data throughput at the same time.

1. INTRODUCTION

ISO/IEC Moving Picture Experts Group (MPEG) and ITU-T Video Coding Experts Group (VCEG) jointly develop the latest video standard, H.264/AVC [1] for next generation multimedia coding applications. Compared with the previous MPEG standards, H.264 provides over two times higher compression ratio with higher video coding quality. However, the computational complexity of H.264 video coding is much higher than that in the previous MPEG standards, which induces the necessity of achieving real-time processing of H.264 video coding through dedicated hardware design.

There are two techniques adopted in H.264 entropy coding. One is Context-based Adaptive Variable Length Coding (CAVLC) for baseline profile video. The other is Context-based Adaptive Binary Arithmetic Coding (CABAC) for main profile video. Compared to the CAVLC, adopting CABAC can save about 9% to 14% of bit-rates at the cost of higher computational complexity [2].

The inherent sequential data dependency in the CABAD severely limits the data throughput rate, which imposes difficulties in achieving real-time H.264 video decoding on the HD format videos like HD1080@30fps. It normally takes 3 clock cycles to decode one-bin of CABAD codeword, which has imposed a severe processing bottleneck in the existing CABAD design [3]. According to our observation, the data dependency mentioned above can be first released by adopting the design concept of pipelining in decoding CABAD codeword, which can improve the data throughput rate to be decoding one-bin of CABAD codeword in two cycles. In addition, we can further improve the data throughput by partitioning the large context table into multiple segmented context tables and adding cache registers for storing temporary context values. With this technique, we can achieve the data throughput of decoding one-bin CABAD codeword within less than two cycles.

Although there is over two times of speed-up in decoding CABAD codeword by adopting the techniques mentioned above, it is still possible to improve the data throughput. According to our analysis, the current data dependency in the CABAD exists in renormalizing the probability model of the CABAD coded bit-stream after decoding each bin of CABAD codeword. However, the re-normalization is not taken place in some cases. By exploiting this fact, we propose a look-ahead codeword parsing scheme to detect if the re-normalization on the probability model occurs in CABAD. If the look-ahead conditions fit, we can decode two bits of CABAD codeword in each cycle, which furthermore releases the processing bottleneck of CABAD. On the other hand, if the look-ahead condition fails, it still takes one cycle to decode one-bit of CABAD codeword without additional overhead.

Based on these design techniques, we have designed the proposed high throughput CABAC decoder for H.264. The proposed design can operate at 120MHz at the cost of about 37K gates with 4,032 bits of memory (or about 83K gates in total). This performance can meet the real-time processing requirement in H.264 video decoding on HD1080 video. As compared to the existing design [3], the proposed design both owns 40% reduction of hardware cost and possesses over 1.6 times data throughput improvement.

The rest of this paper is organized as follows. In Section II, we introduce the CABAD flow. Then, we present the proposed design and the adopted design techniques in Section III. In Section IV, we show the experimental results and performance comparison of the proposed design with others. Finally, we conclude this paper in Section V.

2. CABAD FLOW

The CABAD flow consists of three fundamental parts. One is context and probability modeling. Another is binary arithmetic decoding. The other is binarization. The H.264 standard defines an extensive set of context information associated with syntax elements (SEs). In the first step, context modeling is to select the context index according to which SE is to be decoded and referring syntax information from top, left, or the current macroblock (MB). Second, the CABAD is to decode one bin and renormalize probability model. At last, binarization step is in charge of checking if the successive decoded bin (or bit) is in bin string. If not, the decoder will keep on decoding next bin. If yes, the decoder is prepared to decode the first bin of next SE.

Fig. 1 shows the data flow of CABAD. At first, CABAD needs to re-initialize both the context tables in the beginning of each slice and probability model from bit-stream. Then the SEs of each MB can be decoded one by one.

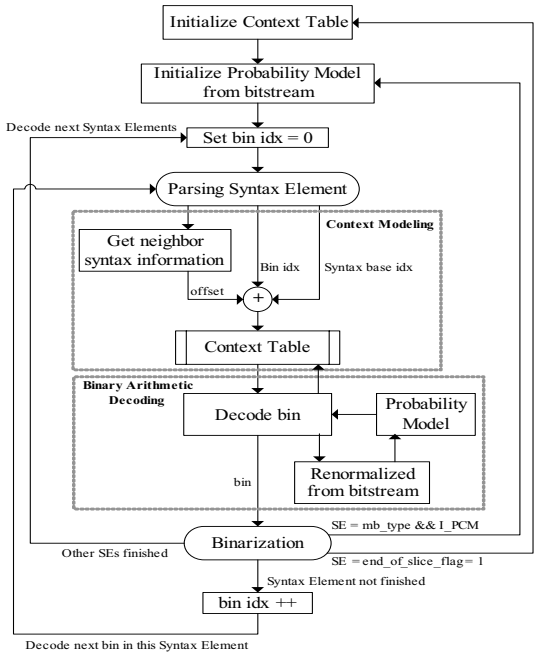

Fig. 1: Data flow in CABAD

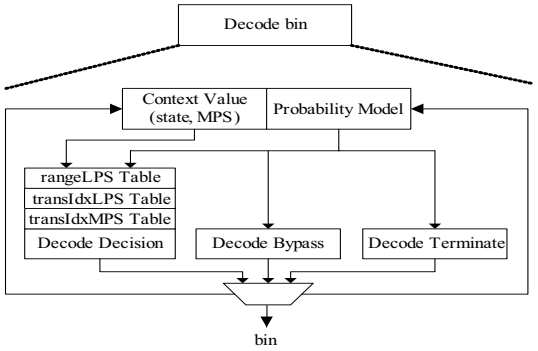

Fig. 2: Decoding engines of *Decode bin*

The *Decode bin* stage consists of three decoding engines, including *decode decision*, *decode bypass*, and *decode terminate*. Only one of these engines works at a time to generate a bin value. Besides the context tables, the H.264/AVC standard also defines three other tables for *decode decision* to update the probability model, including *rangeLPS* table, *transIdxLPS* table, and *transIdxMPS* table. The concept of *Decode bin* is shown in Fig. 2.

3. PROPOSED CABAD DESIGN

The proposed VLSI architecture for H.264 CABAC decoder is shown in Fig. 3. We partition this architecture into three parts, where each part respectively handles the decoding operations on different levels including MB information, slice data and MB layer level, and residual data level. In the following, the proposed design is described in more details.

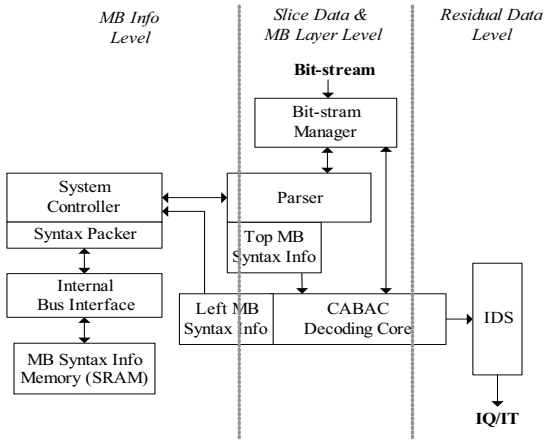

Fig. 3: Architecture of the proposed CABAD design

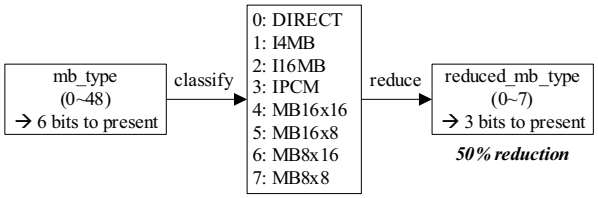

Fig. 4: Proposed syntax element reduction

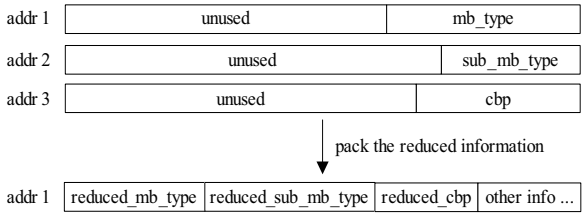

Fig. 5: Proposed syntax information packing

3.1. MB information Level

The CABAD process needs to refer the left and the top MB syntax information to decide the context index when decoding a SE's bin. We propose an efficient method to reduce the required memory size, bandwidth, and access times for storing SEs of neighboring MBs. According to the referring characteristics, we find that it is not necessary to use the complete syntax information. Let us take the mb_type information as an example. The range of mb_type is from 0 to 48 specified in standard. When referring syntax element, as mb_type of I slice, we just need to know it is Intra16x16 instead of what kind of Intra16x16, such as I_16x16_0_0_0, or I16x16_1_0_0, or etc. So, we can classify it into 8 parts as shown in Fig. 4 for both reducing 50% of line buffer for storing the information and simplify the hardware complexity (e.g. using a 3-bit comparator instead of a 6-bit one).

Besides, we pack some syntax information together and then access them through internal bus instead of accessing them individually, as shown in Fig. 5. Thus, we can reduce 68% of memory bandwidth in average.

3.2. Slice Data and MB Layer Level

This is the major part in the proposed CABAC decoder. The Parser primarily plays three roles, including accessing the MB information, controlling the entire decoding, and decoding the slice headers. The CABAD core is the most important part in the proposed design. We propose four techniques to improve the performance of the CABAD. They are adopting pipeline scheduling, using segmented context tables, adding cache registers, and doing look-ahead codeword parsing.

Fig. 6 shows the original CABAD process. It takes three cycles to generate a bin, which imposes a critical processing bottleneck. Thus, a pipeline scheduling on the CABAD is adopted, as shown in Fig. 7, which reduces 33% of cycle count at least in the CABAD compared with that in Fig. 6.

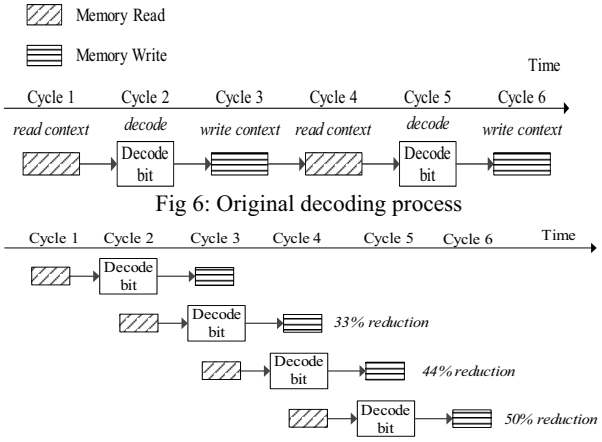

Fig 6: Original decoding process

Fig. 7: Proposed pipelined decoding process

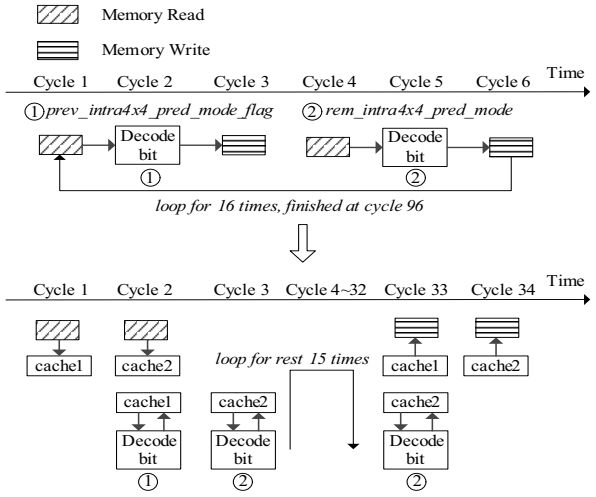

Fig. 8: Cycle-count reduction by combining the pipeline scheduling and cache registers

When an internal loop occurs, there might be a succession of context memory access using the same couple context values. Therefore, we exploit the design concept of cache memory to add cache registers for storing the context values and write back to context memory only once at the end of the internal loop. For example, there is an internal loop occurs when decoding intra4x4 mode. The CABAC decoder needs to decode prev_intra4x4_pred_mode_flag and rem_intra4x4_pred_mode for sixteen times continuously. Fig. 8 shows the combination of the pipeline scheduling technique and cache registers. When adding cache registers, we can reduce the memory access times into once. Using this technique, we can achieve the data throughput rate of decoding 1-bit per cycle, which reduces about 65% of cycle count in as compared with that in Fig. 7.

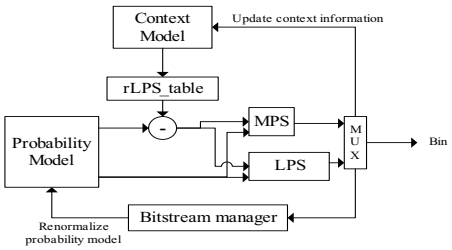

Fig. 9: Original decoding decision flow

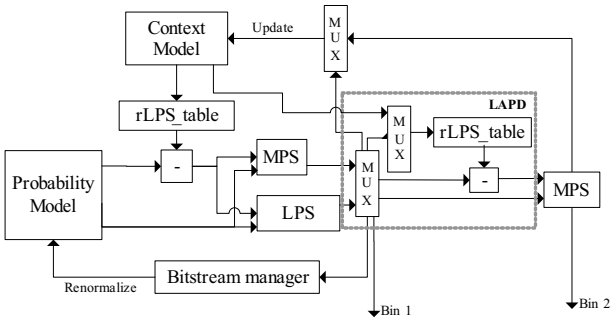

Fig. 10: Proposed decision flow with look-ahead codeword parsing

Moreover, the data throughput of CABAD can be further improved by exploiting the characteristics in re-normalizing the probability model. According to our analysis, the current data dependency exists in re-normalizing the probability model of the CABAD coded bit-stream after decoding each bin. However, the re-normalization is not taken place in some cases. By exploiting this fact, we propose a look-ahead codeword parsing scheme to detect if the re-normalization on the probability model occurs in CABAD. If the look-ahead condition fits, we can decode two bins in each cycle, which furthermore releases the processing bottleneck of CABAD. On the other hand, if the look-ahead condition fails, it still takes one cycle to decode one-bit of CABAD codeword without additional overhead. Fig. 9 shows the original decoding decision flow and Fig. 10 shows the proposed decoding decision flow with look-ahead codeword parsing scheme. The Look Ahead Parsing Detection (LAPD), as Fig. 10 shows, is used to detect if codiRange>=256 (i.e. without re-normalizing probability model and shifting bitstream) and codiRange>=codiOffset (i.e. passing through MPS stage). If these two conditions match, we can generate the second bin. The overhead of the proposed look-ahead codeword parsing includes two look-up tables, one adder, and one multiplexer for purpose of providing the capability of decoding two bits in one cycle.

Since we might be able to decode two bins per cycle, in order to match the feeding throughput, we need a more efficient way to access memory to load multiple context value cycle by cycle. Therefore, we partition one context table into multiple segmented context memories. Thus, combining the segmented context

memories with cache registers, we can read and write memory in a more flexible way, as shown in Fig. 11.

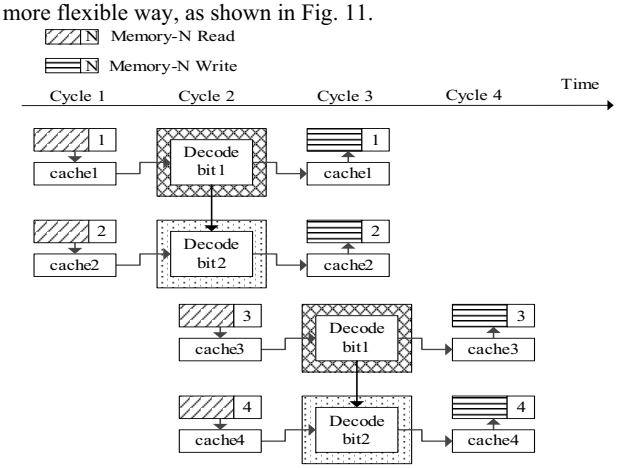

Fig. 11: Optimized decoding process

Fig. 12 shows the architecture of the proposed CABAD core. By exploiting all the proposed design techniques, we can averagely reduce about 53% of cycle count in the CABAD as compared to the original one.

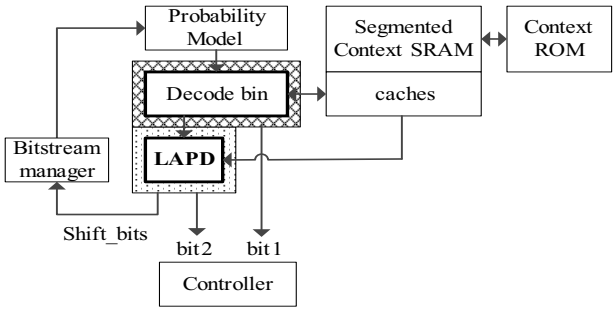

Fig. 12: Architecture of the proposed CABAD core

3.3. Residual Data Level

The residual data decoded by the proposed CABAC decoder are sent to Interleaved Double Stack (IDS) [4], which is proposed in our previous work, located at the right part of the proposed design in Fig 3. The IDS is used to be in charge of reordering the residual data decoded from the proposed CABAC decoder. It can support different scan orders like zig-zag scan order for frame coding and field scan orders for filed coding.

4. PERFROMANCE COMPARSION

In this section, we discuss about the performance evaluation and compare the proposed design with the existing one. We have implemented the proposed design in Verilog HDL with nLint checking [5], simulated with ModelSim and ncVerilog, and synthesized it using Synopsys DC tool according to a TSMC 0.18μm CMOS technology. The proposed design operates at 120MHz with the cost of 83,157 gates in total, including IDS and all the context memories. We have integrated the proposed design in a H.264 BP/MP video decoder system for system and FPGA verification, which passes over hundred testing sequences including the conformance sequences from JVT [6] and those generated by H.264 reference software encoder JM93 [7].

Table 1 shows the comparison results of the proposed design with the existing one [3]. According to our system verification, it respectively takes about 463 cycles, 308 cycles and 254 cycles to decode one MB in I slices (with qp=36), P and B slice (with qp=26), which achieves the real-time CABAD on HD1080i videos. This performance is better than the design [3] in terms of about 1.6 times data throughput for the H.264 IBBP configuration sequences.

Table 1. Comparison of the proposed design with the design [3]

	ISCAS2005 [3]	Proposed design
Technology	TSMC 0.13μm	TSMC 0.18μm
Gate count	138,226	83,157
Max speed	200 MHz	120 MHz
Target spec.	HD720@30fps	HD1080i@30fps
Average cycles/MB	I slice: 1,661	462.44 (qp=36)
	P slice: 576	307.73 (qp=26)
	B slice: 328	253.56 (qp=26)

5. CONCLUSION

In this paper we have proposed a high throughput VLSI architecture design for H.264 CABAD. By exploiting four design techniques such as adopting pipeline scheduling, using segmented context tables, adding cache registers, and doing look-ahead codeword parsing, we have successfully broken down the processing bottleneck of CABAC decoding to averagely reduce up to 53% of cycle count in comparison with non-improved one. Based on a 0.18μm CMOS technology, the proposed design outperforms the existing design [3] by both reducing 40% of hardware cost and achieving over 1.6 times data throughput at the same time.

6. REFERENCE

[1] Joint Video Team (JVT) of ISO/IEC MPEG&ITU-T VCEG, "ISO/IEC 14496-10", 2003.

[2] D. Marpe, H. Schwarz, and T. Wiegand, "Context-based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 560-576, July 2003.

[3] J. W. Chen, C. R. Chang, and Y. L. Lin, " A Hardware Accelerator for Context-Based Adaptive Binary Arithmetic Decoding in H.264/AVC,"*Proc. ISCAS,* pp.4525 – 4528, 23-26 May 2005.

[4] 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," *Proc. ISCAS,* pp.6110 – 6113, 23-26 May 2005.

[5] Michael Keating and Pierre Bricaud, "*Reuse Methodology Manual*" 3rd edition by *Kluwer Academic Publishers,* 2002.

[6] http://ftp3.itu.ch/av-arch/jvt-site/draft_conformance/

[7] Joint Video Team (JVT) reference software JM9.3