A HIGH-PERFORMANCE HARDWIRED CABAC DECODER

Jian-Wen Chen *Youn-Long Lin*

Department of Computer Science
National Tsing Hua University
Hsin-Chu, Taiwan 300
d948317@oz.nthu.edu.tw, ylin@cs.nthu.edu.tw

ABSTRACT

We present a high-performance hardwired context-based adaptive binary arithmetic decoder (CABAD) for H.264/AVC. Based on an analysis of decoding time for different types of syntax elements, we propose three parallel processing techniques. Our decoder takes 309 clock cycles to decode a typical I-type macroblock. It needs to run at only 45MHz for 1080HD application. Therefore, our architecture is suitable for low power mobile applications.

Index Terms— H.264/AVC, CABAC Decoder

1. INTRODUCTION

H.264/AVC is the latest video coding standard jointly developed by the ITU-T Video Coding Experts Group and ISO/IEC Moving Picture Experts Group [1]. It has several new features including multiple reference frame and variable block size motion estimation, integer DCT, in-loop deblocking filter, and context-based adaptive binary arithmetic coding (CABAC) [2][3][4]. In comparison with MPEG-4, it can achieve up to 50% bit-rate saving under the same video quality constraint.

CABAC is one of two entropy coding methods in H.264/AVC. Compared with the other method named Context-based adaptive variable length coding (CAVLC), it saves more than 7% of bit-rate at the expense of higher computation complexity. Profiling results show that it consumes about 10% of total decoding time. Therefore, accelerating the CABAC decoding with hardwired implementation is desirable for high-performance or low-power applications.

Although there are several works proposing fast multiple-bin-per-cycle arithmetic encoding/decoding architectures, none of them takes system design issues into consideration. Due to data and control dependency, it is non-trivial to keep the arithmetic decoding engine highly utilized.

We first analyze the numbers of cycles needed for our previous decoder [6] to decode each type of syntax elements in a macroblock. We use a test sequence “Mobile” in CIF resolution under QP = 28. The analysis results are shown in Table 1. On the average, it takes 782 clock cycles to decode an I-MB, which has about 652 bins. The purpose of this work is to substantially reduce the number of clock cycles.

Table 1. Decoding cycles of each type of syntax elements

Types of Syntax element		Coded_block_flag	Coefficient	Sig. & Last_sig.	Others	Total
I	# Bin	22	319	259	52	652
	# Cycle	127	320	260	75	782
	B./C.	0.17	0.99	0.99	0.69	0.83
P	# Bin	10	44	76	35	165
	# Cycle	63	44	76	106	289
	B./C.	0.16	1	1	0.33	0.57
B	# Bin	5	31	16	26	78
	# Cycle	29	31	16	101	177
	B./C.	0.17	1	1	0.16	0.44

We propose a parallel decoding method to reduce clock cycles for decoding Coded_block_flag syntax elements (SEs), a two-bin-per-cycle method for decoding Coefficient SEs, and a context table re-arrange method for decoding Sig.&Last_sig._pair SEs. The resultant decoder saves the cycle counts per MB by more than 30% (from 782 to 527).

The rest of this paper is organized as following. In Section 2, we present our CABAC decoder architecture. In Section 3, we propose three methods for reducing clock cycles. Our experimental result is shown in Section 4. Finally, we draw conclusions in Section 5.

2. PROPOSED ARCHITECTURE

2.1. H.264/AVC entropy decoder architecture

Figure 1 depicts the block diagram of our H.264/AVC entropy decoder architecture, which is part of a complete hardwired decoder. Our proposed architecture consists of a Variable Length Coding (VLC) parameter decoder, a CABAC decoder, an AHB master wrapper, and a 64-word FIFO. At the beginning, the entropy decoder reads bit stream data from SDRAM and writes to the FIFO through the AHB bus. The VLC decoder decodes slice-level information into the parameter memory and parses data to the CABAC decoder. The CABAC decoder processes

stream data according to the coding information in the parameter memory. It outputs the decoded results to the Mb_info memory and the Coefficient memory for further processing such as IQ/IDCT, compensation, and filtering.

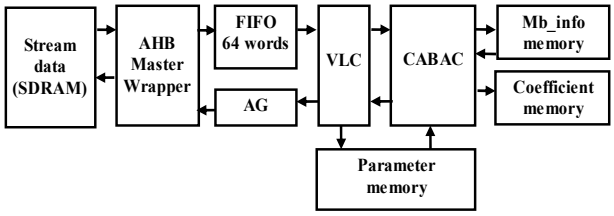

Figure 1. H.264/AVC entropy decoder architecture

2.2. CABAC decoder architecture

Figure 2 depicts our proposed CABAC decoder architecture. At the beginning of slice-level decoding, the decoder gets parameters from the parameter memory and uses the *Build table module* to build a new context table by reading in the initial ROM. For the macroblock layer, the *Se select module* determines the type of the syntax element to be decoded and receives neighboring data for context modeling from the *Get neighbor module*. Then, the *Context model module* calculates context as the index to the context table. The *Se decode module* determines how to decode each bin of the syntax element. Finally, the *Arithmetic engine (AE) module* performs arithmetic decoding and consumes the bit stream.

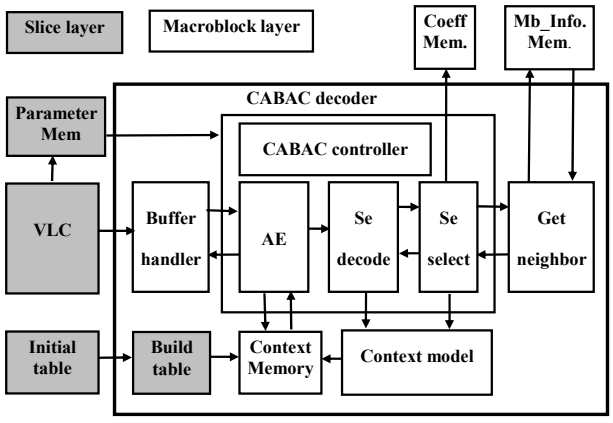

Figure 2. Proposed CABAC decoder architecture

We implement the initial table using a 1484x16-bit ROM and the context table a 399x7-bit two-port SRAM. Additionally, we use two single-port Coefficient memories (533x9-bit) and one two-port Mb_info memory [6]. The Coefficient memory is read by the IDCT module in a ping-pong fashion. The Mb_info memory could be read by intra prediction, motion compensation, and CABAC.

Figure 3 shows the timing diagram and control signals. In the slice layer, the CABAC decoder reads parameters or builds context table according to *start_rd_pa* and *init_slice* signals from the top-level main controller. In the macroblock layer, it starts to decode one macroblock according to the *init_cabac* signal. When it finishes decoding a macroblock, it will notify the main controller by asserting *end_slice* and *end_cabac* signals.

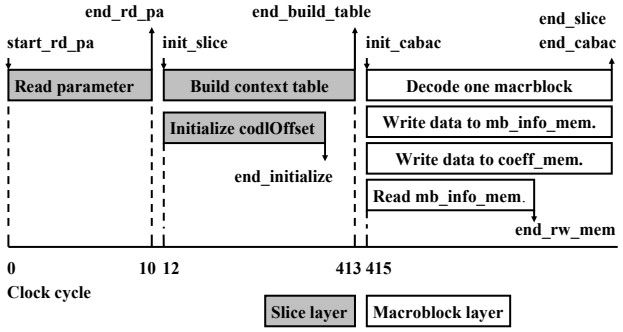

Figure 3. Timing diagram of the CABAC decoder

3. PERFORMANCE ENHANCEMENT METHODS

We propose three methods to reduce clock cycles for decoding syntax elements of types Coded_block_flag, Coefficient, and Sig. & Last_sig. pair, respectively.

3.1. The parallel decoding method

According to analysis in Table 1, our original CABAC decoder consumes around 16% of total cycles to decode Coded_block_flag syntax elements for which each bin needs three additional cycles to get neighboring data and one clock cycle to generate context. We propose parallelizing the tasks of decoding coefficients and getting neighboring data.

Figure 4 shows the flow chart of our parallel method. First, the decoder gets the neighboring data and generates the context of a Coded_block_flag SE. Second, it decodes the Coded_block_flag SE and calculates the address of the neighboring data for the next Coded_block_flag SE. Then, it decodes a Sig. & Last_sig pair SE and gets the left neighboring data at the same time. Third, the CABAC decoder decodes a Coefficient SE and gets the top neighboring data simultaneously. Finally, it goes to decode the next Coded_block_flag SE.

By using the parallel decoding method, we save around 8% of total cycles.

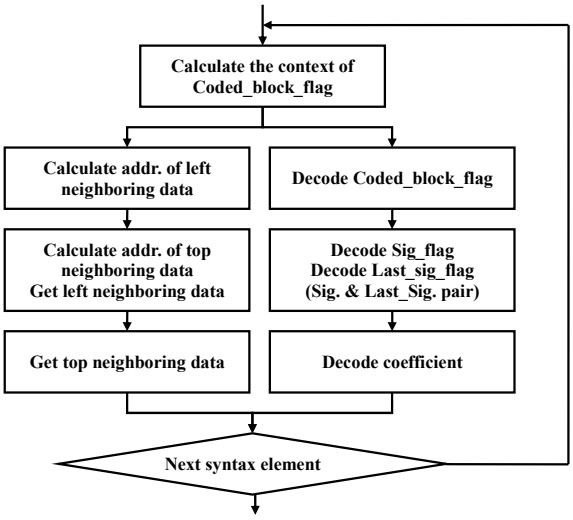

Figure 4. The flow chart of the parallel decoding method

3.2. The two-bin-per-cycle decoding method

The CABAC decoder uses 41% of total cycles to decode Coefficient SEs. Therefore, we proposed a two-bin-per-cycle method as depicted in Figure 5 to decode two bins in one clock cycle.

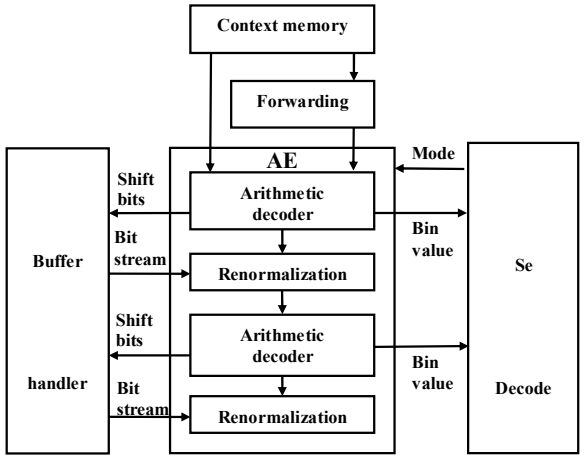

Figure 5. Block diagram of the two-bin decoding method

The arithmetic engine combines two arithmetic decoder similar to He's architecture [5]. It can decode two regular bins, two bypass bins, or one regular and one bypass bin in a single cycle.

In addition, we employ a forwarding logic to avoid reading the un-updated context data when the decoder encounters consecutive bins with the same context. The *Buffer handler module* provides bit stream for renormalization. It stalls the arithmetic decoding process when the stream buffer is empty. The *Se decode module* determines modes and reads bin value to generate syntax element.

Taking into account control overhead and stall due to buffer emptiness, the proposed two-bin-per-cycle method contributes 13% reduction of total cycles.

3.3. The context table rearrangement method

According to analysis shown in Table 1, there are on the average 6 Significant_flags and 4 Last_significant_flags in a 4x4 block. Their decoding accounts for 31.7% of the total cycles. Figure 6 shows the decoding order of Significant_flag and Last_significant_flag. We propose dividing the context table into two tables as shown in Figure 7.

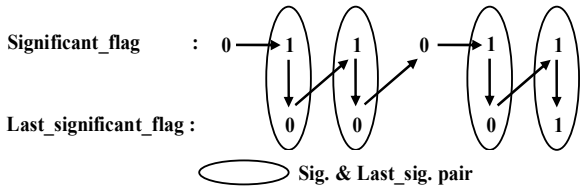

Figure 6. The decoding order of Significant_flag and Last_significant_flag.

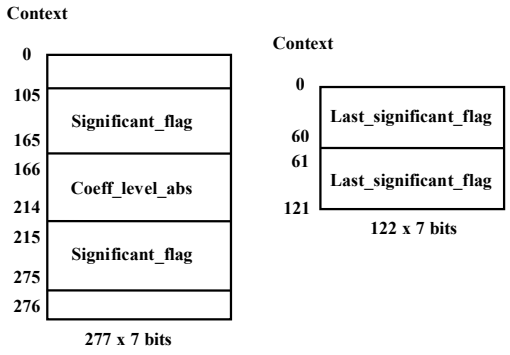

Figure 7. The organization of proposed context tables

Our decoder can concurrently read two context data of Sig. & Last_sig. pair from the new divided context tables. Therefore, the arithmetic engine could decode a Sig. & Last_sig pair in one cycle. In comparison with He's work [5], which groups 15 context data of Sig. and Last_sig. into a very wide entry, our method has better coding efficiency.

By the rearrangement of context table, our CABAC decoder saves 12% of total cycles after taking into consideration stall due to buffer emptiness.

4. EXPERIMENTAL RESULTS

We have implemented the proposed architecture in Verilog RTL and synthesized it targeted towards a TSMC 0.13μm CMOS cell library as shown in Table 2. At the slow-slow corner, the arithmetic engine can run at 188 MHz. Due to the combination of arithmetic engine and forwarding circuit, the whole CABAC decoder can be clocked at 137 MHz. We have also integrated the design into a pure hardwired H.264/AVC main profile decoder and successfully demonstrated an FPGA prototype.

Table 2. The synthesis result

Targeted module	Gate counts	Frequency
Complete CABAC Decoder	40,762	137 MHz
Arithmetic engine	11,475	188 MHz

Table 3 shows the average clock cycles for different sequences. We use five video sequences with different characteristics under QP = 28 to verify our proposed architecture. For the MB with most bins (I-MB in Mobile), our decoder achieves throughput of about 1.24 bins per cycle, which is about 50% better than the original 0.83 bin per cycle in Table 1.

Table 3. Performance of the proposed architecture

Type of MB		Mobile	Foreman	Tempete	News	Carphone	Avg
I MB	bin	652	176	450	202	186	333
	cycle	527	195	402	216	206	309
	b./c.	1.24	0.9	1.12	0.94	0.9	1.08
P MB	bin	165	55	143	28	58	90
	cycle	225	104	181	93	112	143
	b./c.	0.73	0.53	0.8	0.3	0.52	0.63
B MB	bin	78	30	78	22	49	52
	cycle	149	108	165	102	126	130
	b./c.	0.52	0.28	0.47	0.22	0.39	0.4
Avg	bin	298	87	224	84	98	158
	cycle	301	136	249	137	148	194
	b./c.	0.99	0.64	0.9	0.61	0.66	0.8

Based on the above results, we estimate the needed speed of our circuit for different resolutions in Table 4. For HDTV (1920x1088, denoted 1080HD) application, it needs only 45 MHz.

Table 4. The working frequency for different resolutions

Resolution	CIF	D1	720p HD	1080 HD
Working frequency at 30 fps	3 MHz	8 MHz	20 MHz	45 MHz

In Table 5, we compare our design with He's work [5]. Our architecture takes only 40% of cycle counts. Running at top speed, our design can deliver 1.35X higher throughput.

Table 5. In comparison with other's work

	He [5]	Our design
Average cycles/MB	500	194
Frequency	150 MHz (0.18 μm)	137 MHz (0.13 μm)
Throughput	314,572 MB/s	740,489 MB/s

5. CONCLUSION

We have proposed a high-performance CABAC decoder architecture. We employ three novel techniques to speed-up the decoding process of three most frequent types of syntax elements. Experimental results over a wide range of video sequences and MB types show that our decoder can on the average process an MB in fewer than 200 clock cycles. This is more than 2X better than state-of-the-art.

In the future, we would like to further improve its performance and integrate it into a super HDTV system.

6. 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)

[2] D. Marpe, H. Schwarz, and T. Wiegand, "Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard," IEEE Transactions on Circuits and Systems for Video Technology, pp: 620-636, July 2003.

[3] M. Mrak, D. Marpe, and T. Wiegand, "A context modeling algorithm and its application in video compression," IEEE 2003 International Conference on Image Processing, pp. III - 845-8, Sept. 2003.

[4] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," IEEE Transactions on Circuits and Systems for Video Technology, pp. 560-576, July 2003.

[5] W. Yu, and Y. He, "A high performance CABAC decoding architecture," IEEE Transactions on Consumer Electronics, pp. 1352-1359, Nov. 2003.

[6] J.W. Chen, C.R. Chang, and Y.L. Lin, "A hardware accelerator for context-based adaptive binary arithmetic decoding in H.264/AVC," IEEE International Symposium on Circuits and Systems, pp. 4525-4528, May 2005.

[7] C.H. Kim, and I-C. Park, "High Speed Decoding of Context-based Adaptive Binary Arithmetic Codes Using Most Probable Symbol Prediction" IEEE International Symposium on Circuits and Systems, pp. 1707-1710, May 2006.