A NOVEL TRACE-PIPELINED BINARY ARITHMETIC CODER ARCHITECTURE FOR JPEG2000

Minsoo Rhu and In-Cheol Park

Korea Advanced Institute of Science and Technology (KAIST)
{minsoo.rhu@gmail.com, icpark@ee.kaist.ac.kr}

ABSTRACT

Embedded block coding with optimized truncation (EBCOT) employed in the JPEG2000 standard accounts for the majority of the processing time, because the EBCOT is full of bit operations that cannot be implemented efficiently in software. The block coder consists of a bit-plane coder (BPC) followed by a binary arithmetic coder (BAC), where the most up-to-date BPC architectures are capable of producing symbols at a much higher rate than the conventional BACs can handle. This paper proposes a novel pipelined BAC architecture that can encode input symbols at a much higher rate than the conventional BAC architectures. The proposed architecture can significantly reduce the critical path delay and can achieve a throughput of 400 M symbols/sec. The critical path delay synthesized with 0.18-μm CMOS technology is 2.42 ns, which is almost half of the delay taken in conventional BAC architectures.

***Index Terms*—EBCOT, JPEG2000, VLSI architecture, arithmetic coding, trace scheduling**

1. INTRODUCTION

JPEG2000 [1] is the latest international image compression standard developed to address the needs of many applications. In 2004, Digital Cinema Initiatives (DCI), which is a joint venture of seven major Hollywood studios, selected JPEG2000 as the compression format to be used for digital distribution of motion pictures. Thus, JPEG2000 is not only being utilized for still image compression, but also opening up a potential field of video compression. [2] As a frame consists of 4096×2160 pixels in the DCI specification, a high performance JPEG2000 encoding system is in demand to accommodate the need of video applications.

The JPEG2000 adopts the discrete wavelet transform as its primary transforming algorithm. The overall block diagram of JPEG2000 is shown in Fig. 1. The transformed coefficients are split into codeblocks and are sequentially processed by an entropy coding algorithm known as embedded block coding with optimized truncation (EBCOT). The EBCOT is the most complicated part that accounts for the majority of the computation time in the JPEG2000 encoding system. [3] This is due to the fact that EBCOT is inherently a bit-level operation which cannot be executed efficiently in software. As shown in Fig. 1, while the DWT is a word-level processing scheme, EBCOT is inherently a bit-level processing algorithm, causing the computational complexity in JPEG2000. Therefore, developing a high-performance EBCOT unit is crucial in developing an efficient JPEG2000 application systems.

The EBCOT is basically a two-tiered algorithm consisting of Tier-1 and Tier-2 called the embedded block coding and the rate-distortion optimization, respectively. Tier-1 is again divided into two sub-modules: bit-plane coder (BPC) and binary arithmetic coder (BAC). The BPC utilizes the neighboring information of the current bit to construct the context information consisting of *context* (CX) and *decision* (D), which will be entropy coded by the following BAC. The major timing limitation is caused by the BAC, because it is inherently dependent on control statements and arithmetic operations. As a result, the BAC becomes a throughput bottleneck of the entire JPEG2000 encoding system, but its serial processing nature makes it difficult to exploit parallelism. Several pipelining techniques have been proposed in previous researches, but the processing speed of the BPCs suggested so far still exceeds that of current BAC architectures. Therefore, improving the BAC throughput would potentially increase the overall performance of JPEG2000 systems.

In this paper, a novel pipelined BAC architecture is proposed to reduce the critical path delay, and thus to increase the throughput. In Section 2, we briefly explain the arithmetic encoding algorithm of JPEG2000. The proposed method to enhance BAC performance is discussed in Section 3. Experimental results are given in Section 4 along with comparison to conventional BAC architectures. Finally, concluding remarks are made in Section 5.

2. BINARY ARITHMETIC CODING IN JPEG2000

The BAC adopted in JPEG2000 originated from the MQ coder, which uses context-based probability estimation. Fundamentally, the goal of the arithmetic encoding procedure initiated by the BAC is to compute the final offset value by dividing the probability interval recursively, as shown in Fig. 2. The BPC outputs, D and CX, are forwarded to the BAC, where D represents either the more probable symbol (MPS) or the less probable symbol (LPS) and CX represents the significance information for one bit and its neighbors. Depending on the bit location in the bit plane, the bit and its neighbors may have various significance states. As a result, 19 context types are used in the BAC to designate various combinations of the significance states. For a D/CX pair, the probability interval is partitioned into two sub-intervals, the MPS and the LPS sub-intervals, by applying the probability of the LPS, *Qe*. All the values of *Qe* are stored in a lookup table. The *Qe* value is accessed with an index determined by the CX, making *Qe* adaptively chosen according to the CX.

During the encoding procedure, the probability interval, *A*, and the code register, *C*, are updated as shown in Fig. 2. The interval *A*

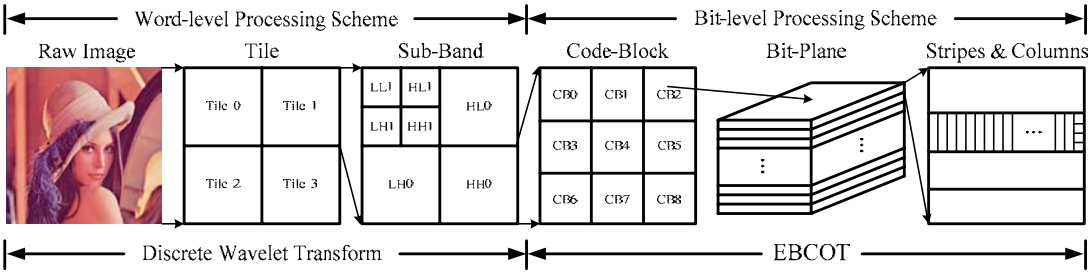

Fig. 1. Block diagram of a JPEG2000 encoder system.

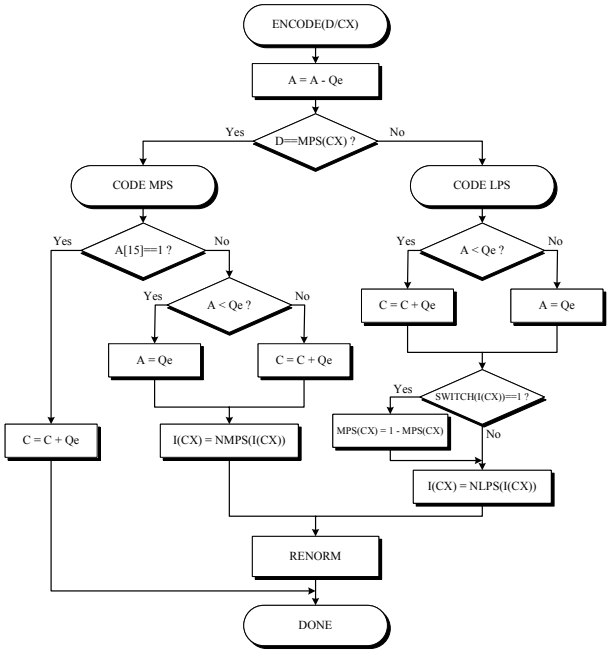

Fig. 2. Flow chart of Binary Arithmetic Coding.

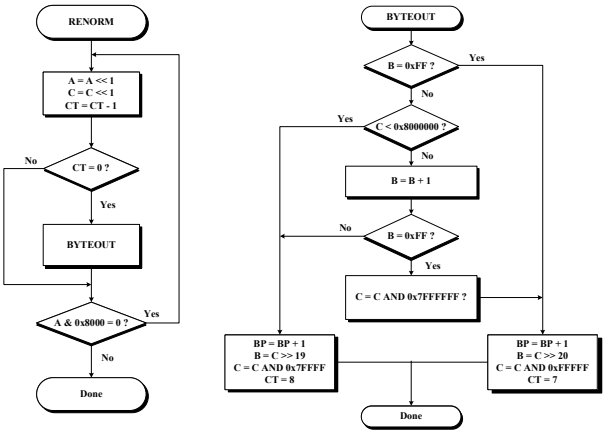

Fig. 3. Flow chart of RENORM and BYTEOUT procedures.

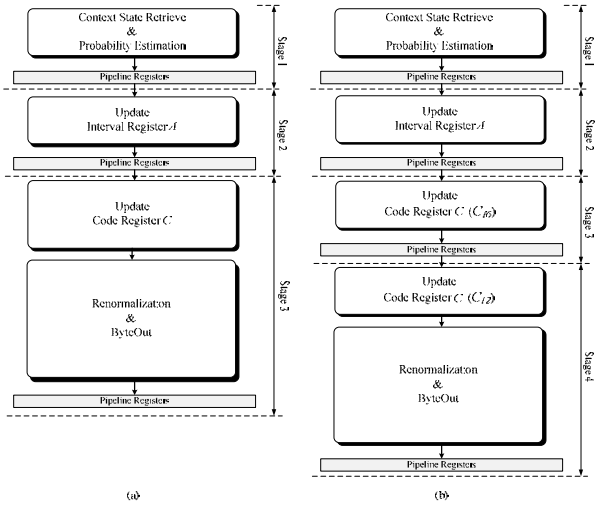

Fig. 4. Conventional pipelined architectures
(a) 3-stage pipeline and (b) 4-stage pipeline.

must be kept in the range of (0x8000, 0xFFFF). Since both *A* and *C* are finite-precision values, the *renormalization* (RENORM) procedure is invoked whenever *A* falls below 0x8000. As shown in Fig. 3, RENORM continuously shifts *A* and C, one bit at a time, until *A* becomes larger than 0x8000. During RENORM, procedure BYTEOUT is invoked whenever the counter register (*CT*) value, representing the number of available bits in the upper 12bits of *C*, counts down to zero. During the BYTEOUT procedure, a compressed byte is extracted from the high-order bits of *C* and replaces the byte value originally kept in register *B*. The byte value being replaced is the entropy-coded output of the BAC. In encoding a single bit, the maximum shift amount of 15 occurs when the renormalization is invoked with $A = Qe = 0x0001$. As the *CT* value is reset to either 7 or 8 in Fig. 3, the BYTEOUT procedure can be called twice at most when encoding a single D/CX pair, resulting in the emission of two bytes, *BO_1* and *BO_2*.

As the arithmetic coding is composed of conditional statements that decide the subsequent branches, the direct hardware implementation of Fig. 3 is quite inefficient. Therefore, many pipelined architectures have been suggested [3]-[10] to minimize the effect of the condition-driven operations and to increase parallelism, which can be divided into two categories as shown in Fig. 4. The 3-stage pipelined architecture calculates the updated *C* and derives the output bytes all at once in stage 3, while the 4-stage pipelined architecture divides the stage into two stages, stage 3 and stage 4, in order to reduce the critical path delay by dividing the update of the code register into two parts, the upper 12bits (C_{12}) and the lower 16bits of *C* (C_{16}). However, no matter what the

TABLE I
NUMBER OF BYTEOUTS WHILE ENCODING A SINGLE D/CX PAIR (LOSSLESS COMPRESSION)

Test Image (Size)	Total number of D/CX pairs being encoded	0-BYTEOUT (%)	1-BYTEOUT (%)	2-BYTEOUTs (%)
Parrot (128x128)	302071	280504 (92.8)	21561 (7.1)	6 (0.001)
Cameraman-Gray (256x256)	540194	497582 (92.1)	42610 (7.8)	2 (0.0003)
Lena (512x512)	1835638	1674756 (91.2)	160880 (8.7)	2 (0.0001)
Butterfly (512x512)	2479782	2247321 (90.6)	232457 (9.3)	4 (0.0001)
Room (768x768)	5372003	4848500 (90.2)	523499 (9.7)	4 (0.00007)
Harbor (768x768)	4990085	4545841 (91.1)	444226 (8.9)	18 (0.00036)
Globe (4096x2048)	60967208	56019826 (91.8)	4946979 (8.1)	403 (0.0006)
Space-craft (4096x4096)	99865481	91976576 (92.1)	7888543 (7.8)	362 (0.0003)

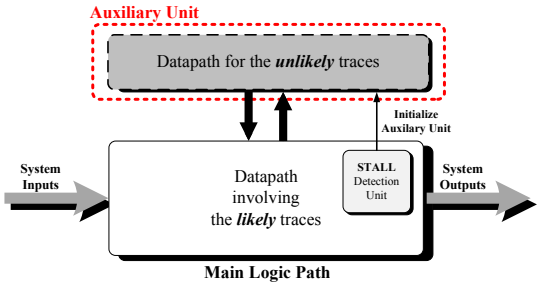

Fig. 5. Applying the trace scheduling concept to optimize a system.

previous pipelined architectures emphasized on, the critical path is established in the derivation of C along with the BYTEOUT procedure that extracts the output bit stream. Specifically, the serial path established in determining the first and second output bytes (BO_1, BO_2) complicates the design and lengthens the derivation of the updated values of C and B to be used in encoding the next D/CX pair. Although the exact location of where the critical path is established might vary, the longest path delay generally includes the derivation of BO_1 and BO_2. The critical path delay of the 4-stage pipelined architecture suggested in [10] was 5.37ns, whereas 5.43ns was the path delay of the 3-stage pipelined architecture in [9], which means that an alternative approach rather than merely increasing the number of pipeline stages could be more effective.

3. PROPOSED ARCHITECTURE

In the following sections, we suggest novel optimization schemes, named as trace pipelining and renormalization look-ahead, which are highly effective in shortening the critical path delay and increasing overall throughput.

3.1 Trace Pipelining

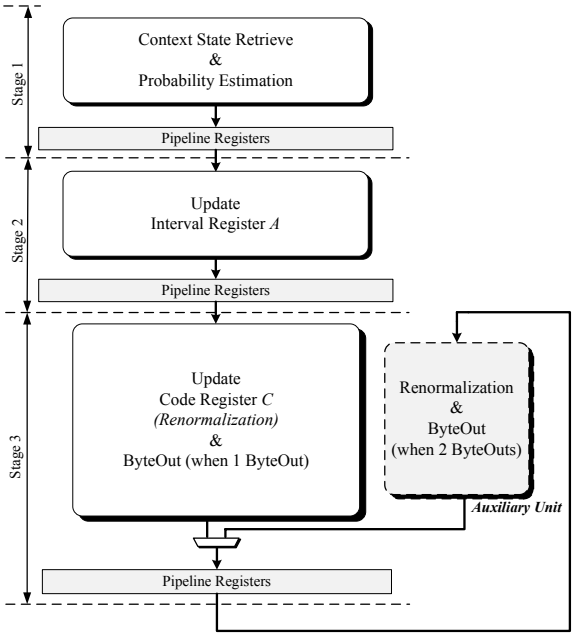

Fig. 6. Applying trace pipelining to the pipelined BAC architecture

Trace Scheduling is an optimization technique adopted in high-level language compilers, which separates out *unlikely* codes and adds handlers for exits from the *likely* trace. The main assumption is that the likely trace is so much more probable than the alternatives that the cost of the bookkeeping code is not a deciding factor: if moving an instruction makes the likely trace execute faster, the instruction is moved. The approach is also effective in the hardware implementation if a certain path is far more likely to be taken than the other paths. Fig. 5 shows how trace scheduling can be applied to optimize a hardware system, where the main logic path is built under the assumption that only the *likely* cases need to be considered. As the *unlikely* cases are completely discarded from this datapath, the main logic path can be built with much more simple and compact logics. The stall detection unit

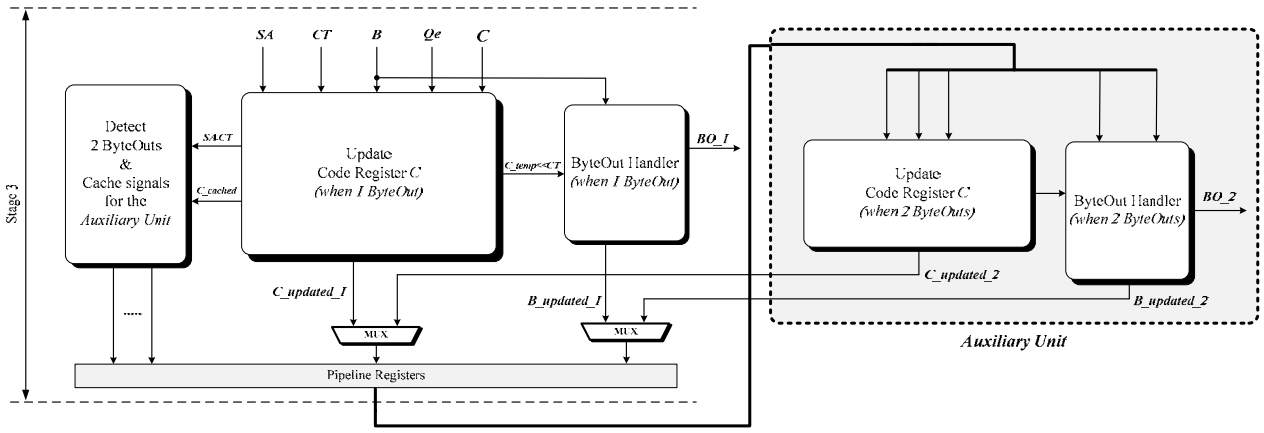

Fig. 7. Detailed structure of stage 3 and the auxiliary unit.

checks for the *unlikely* cases, and stalls the main logic path while the auxiliary unit derives the correct system outputs. Since we can assume that the probability of the *likely* cases far outweighs that of the *unlikely* cases, the performance of the overall system remains intact.

Accordingly, when a system shows a skewed-distribution on its possible paths, the concept mentioned above can be applied to optimize the critical path of its pipelined architecture. In developing a pipelined architecture, we can apply the optimization scheme to each of the pipeline stages if any one of those stages shows a skewed-distribution on its possible paths. We name this as *trace pipelining*, which is applicable to optimize the BAC in JPEG2000. According to our analysis, the probability of two-byte emission is less than 0.003% when encoding a single D/CX pair. We have tested several images on various sizes, and counted the number of BYTEOUT procedures triggered while encoding a single input symbol in order to clearly visualize the skewed distribution of BYTEOUTs. As shown in Table I, the probability of an input D/CX pair to trigger a BYTEOUT procedure is less than 10% on average. The figures become much more skewed when we consider the possibility of a two consecutive BYTEOUT procedure being triggered, which is less than 0.003% on average. As mentioned before, updating *C* and *B* and deriving the output bit-stream are mainly contributing to the critical path delay. If a pipelined architecture is designed with assuming that only one byte can be emitted at most, the critical path can be significantly shortened compared to the previous pipelined architectures.

Therefore, the proposed pipelined architecture is based on the trace pipelining concept to make the common cases of zero-byte or single-byte emission run faster, while ruling out the unlikely case of two-byte emission and compensating them later. Fig. 6 and Fig. 7 show how the proposed trace pipelining can be applied to optimize the critical path of a pipelined BAC, which contains an auxiliary unit that would be exploited only when two bytes should be emitted. Stage 3, which is in charge of updating *C* and *B*, is basically designed under the assumption that only the first byte, *BO_1*, can be emitted at most. Fig. 8 shows how the value of code register *C* is updated in stage 3 for the common cases of zero or single-byte emission. The overall shift amount required for RENORM (*SA*) is forwarded from stage 2. The updated value of *C* after zero or single-byte emission is denoted as *C_updated_1* and *MASK* refers to either 0x7FFF or 0xFFFFF according to

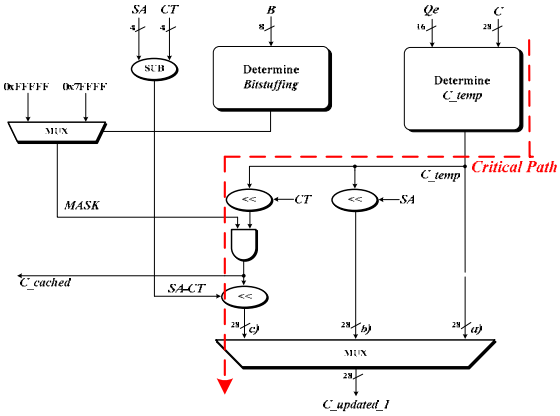

Fig. 8. The circuit diagram of the Update Code Register *C* module.

procedure BYTEOUT in Fig. 3. The path that was needed to deal with the second byte emission is moved to the auxiliary unit, and a detector that checks the rare case is placed in stage 3. As the auxiliary unit is not participated in the normal pipeline operations, the critical path length is significantly shortened. If the detector senses the occurrence of the rare case, it initiates the auxiliary unit and stalls the pipeline until the auxiliary unit completes all the works related to the emission of the second byte, *BO_2*. The signals to be forwarded to the auxiliary unit, such as *C_cached* and *SA-CT*, are stored in the pipeline registers and are utilized when two successive bytes should be emitted while encoding a single D/CX pair. *C_cached*, which is a temporary value of *C* to be cached for the auxiliary unit, is chosen as the value of *C_temp* shifted by *CT* and masked with *MASK*. It is utilized to calculate *C_updated_2* which is the updated value of *C* after both *BO_1* and *BO_2* are emitted. *C_temp* is determined as either *C* or *C* + *Qe* according to Fig. 2 and is also a temporary value of *C* to be used to calculate *C_updated_1* and *C_updated_2*. Fig. 9 shows how the signals forwarded from stage 3 are utilized in the auxiliary unit to derive *C_updated_2*.

As the number of bytes being emitted for a D/CX pair is almost always zero or one, we can ignore the additional cycles caused by

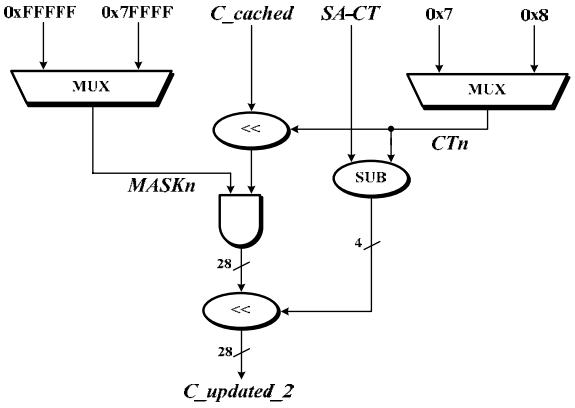

Fig. 9. Derivation of *C_updated_2* in the auxiliary unit.

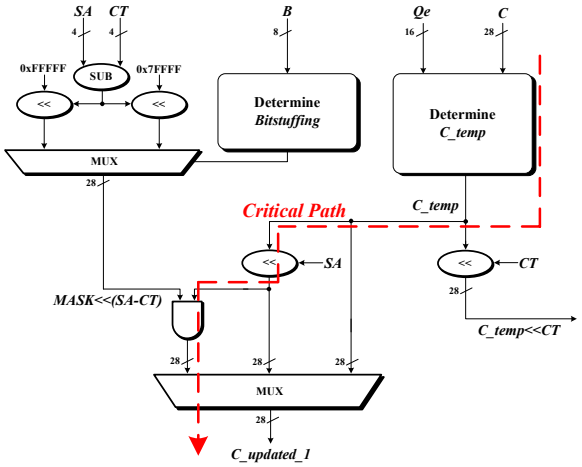

Fig. 10. Update Code Register C module optimized by Eq. (3).

stalling the pipeline only for the rare case, thereby maintaining the system throughput to nearly one D/CX per cycle.

3.2 Renormalization Look-ahead Scheme

Although applying the proposed trace pipelining can significantly shorten the lengthy critical path appearing in conventional pipelined architectures, there are still some areas entitled for further improvement. For instance, two consecutive shifting operations required to derive *C_updated_1* and *C_updated_2* in Fig. 8 and Fig. 9 contribute the critical path, slackening the extraction of *BO_1* and *BO_2*. If we can decrease the number of consecutive shifting operations in deriving the updated value of *C*, a faster extraction of the output bit-stream would be feasible, eventually resulting in a shorter critical path delay. We suggest a method called *renormalization look-ahead* (RLA), which is efficient in optimizing the BAC design. As the proposed method is powerful in deriving an equation that can shorten the critical path, it enables a high-performance implementation of BAC. The concept is similar to that of carry look-ahead addition in that it calculates recursive operations all at once. The fundamental principle behind RLA is based on the distributive law in mathematics.

Since shiting *x* left by *n*-bits is equivalent to multiplying *x* by 2^n, we can apply the distributive law to the shift operation required for renormalizing *A* and *C* in order to reduce the number of consecutive shift operations needed to update *C*. As most of the arithmetic operations and multiplexing performed in the BAC are dependent upon internal conditions as shown in Fig. 3, a fast derivation of *C_updated_1* and *C_updated_2* will allow a quick extraction of *BO_1* and *BO_2*, significantly shortening the critical path delay in stage 3 and the auxiliary unit. For example, the longest path delay in deriving *C_updated_1* is established when procedure BYTEOUT is called during RENORM. For this case, *C_updated_1* is expressed as follows.

$$[(C\_temp<<CT)\ \&\ MASK]<<(SA\text{-}CT) \qquad (1)$$

Due to the multiplicative nature of shifting operation and the distributive law, Eq. (1) can be simplified as follows.

$$[C\_temp<<(CT+SA\text{-}CT)]\ \&\ [MASK<<(SA\text{-}CT)] \qquad (2)$$

$$(C\_temp<<SA)\ \&\ [MASK<<(SA\text{-}CT)] \qquad (3)$$

Since the value of *SA* and *CT* are available at the very beginning of stage 3, the shifting operation of *C_temp* and *MASK* in (3) can be initiated simultaneously and independently, thereby canceling the cascaded shifting operation in Eq. (1). As shown in Fig. 8, *C_updated_1* is determined by one of three cases: *a)* no renormalization occurs, *b)* renormalization occurs but no bytes are emitted and *c)* renormalization occurs and one byte is emitted. Since the longest critical path delay is established for case c), utilizing Eq. (3) will be effective in minimizing the critical path delay of *C_updated_1*. Fig. 10 shows how *C_updated_1* can be implemented using Eq. (3). The two cascaded shift operations, which were a performance bottleneck in Fig. 8, are eliminated in Fig. 10 by adopting Eq. (3). In a similar fashion, *C_updated_2* is optimized as follows.

$$(C\_temp<<SA)\ \&\ [MASKn<<(SA\text{-}CT\text{-}CTn)] \qquad (4)$$

Here, *CTn* refers to the updated value of *CT* after the *BO_1* is extracted during procedure RENORM and is either 7 or 8 as shown in Fig. 3, and *MASKn* functions exactly the same as *MASK* in the first BYTEOUT procedure. Compared to Fig. 9, Eq. (4) can be implemented with a less number of cascaded shifters as in Fig. 10, enabling a faster derivation of *C_updated_2*.

3.3 Hardware Reduction in the Auxiliary Unit

As mentioned earlier, the possibility of two-byte emission is less than 0.003%. Therefore, the throughput of BAC would not be deteriorated even if we spend more cycles to handle the two-byte emission case. Since stage 1, 2 and 3 are in the IDLE state while the auxiliary unit is operating, we can exploit the inactive hardware units in order to minimize the hardware cost of the auxiliary unit. Therefore, in the actual implementation of the proposed BAC, the cost of the auxiliary unit is minimized by sharing the inactive units in the regular pipeline stages.

4. PERFORMANCE RESULTS

The proposed architecture is described in a Verilog hardware description language and synthesized using Synopsys Design Compiler with 0.18-μm cell library. Since the previously suggested

TABLE II
EFFECTS OF THE PROPOSED METHODS IN 0.18- μM CELL LIBRARY

Architecture	Critical Path (ns)	CLK Freq. (MHz)	Throughput (M symbols/s)	Gate count	FOM
Reference	4.43	225	225	8.3 K	27.11
Proposed	2.42	413	413	9.1 K	45.38

TABLE III
COMPARISON TO PREVIOUS ARCHITECTURES

Architecture	Critical Path (ns)	Throughput (M symbols/s)	Gate count	FOM	Technology (μm)
Proposed	2.42	413	9.1 K	45.38	0.18
[4]	9.09	110	9.8 K	11.22	0.18
[5]	4.85	206	7 K	29.43	0.18
[6]	4.61	216	8.7 K	24.83	0.25
[7]	5.4	185	NA	NA	NA
[8]	4.82	207	6.9 K	30.00	0.35
[9]	5.43	184	8.5 K	21.65	0.35
[10]	5.37	186	6.6 K	28.18	0.35

pipelined architectures [4]-[10] were synthesized in diverse technologies, a reference version is designed to fairly compare the proposed architecture with conventional pipelined architectures. The reference version is an implementation of the conventional pipelined BAC that employs no optimization techniques proposed in this paper. The overall implementation results of the proposed architecture are summarized in Table II and comparisons to conventional architectures are shown in Table III, where the processing throughput per gate count (M symbols / sec / K gates) is selected as a figure of merit (FOM). The reference version shows a somewhat improvement compared to [6] and [8], and the improvement seems to be coming from the better technology applied to it. Yet, the extent of its improvement is quite insignificant since the longest logic paths of the conventional BACs, without the proposed optimization schemes, are always longer than the proposed BAC. Compared to the reference version and [6], the throughput of the proposed architecture is 1.8 and 1.9 times higher, respectively, but the overall gate count is increased only by 5%. The increase is mainly caused by the parallel processing nature of the proposed architecture, which is tolerable considering the almost double increase in throughput. As a whole, the proposed BAC shows the highest FOM compared to the previous architectures summarized in Table III.

5. CONCLUSIONS

In this paper, we have presented two methods, named as trace pipelining and renormalization look-ahead scheme, developed to optimize the binary arithmetic coder of JPEG2000. The trace pipelining technique has been applied to make the common case of single-byte emission run faster and to rule out the unlikely cases, and eventually to reduce the critical path delay. In addition, the renormalization look-ahead scheme has been suggested to reduce the number of cascaded shifting operations. Compared to the conventional BAC architectures resorting to intuitive techniques, the proposed methods are not only efficient but also systematic. Experimental results show that the pipelined BAC architecture based on the proposed methods achieves a much higher throughput than the conventional BAC architectures, because the proposed trace pipelining and renormalization look-ahead shorten the critical path delay to almost half.

6. REFERENCES

[1] ISO/IEC JTCI/SC29/WGI N1855, JPEG2000 Part1:Final Draft International Standard (ISO/IEC FDIS FDIS15444-1), Aug. 2000.

[2] A. Bilgin and M.W. Marcellin, "JPEG2000 for Digital Cinema", *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp.3878-3881, May. 2006.

[3] C.J. Lian, K.F. Chen, H.H. Chen, and L.G. Chen, "Analysis and architecture design of block-coding engine for EBCOT in JPEG 2000," *IEEE Trans. Circuits Syst. Video Technol.,* vol.13, No.3, pp.219-230, Mar. 2003.

[4] Y.Z. Zhang, C. Xu, W.T. Wang and L.B. Chen, "Performance Analysis and Architecture Design for Parallel EBCOT Encoder of JPEG2000," *IEEE Trans. Circuits Syst. Video Technol.,* vol.17, No.10, pp.1336-1347, Oct. 2007.

[5] K. Zhu, F. Wang, X. Zhou, and Q. Zhang, "An efficient accelerating architecture for tier-1 coding in JPEG2000," *International Conference on Solid-State and Integrated Circuits Technology*, vol. 3, pp.1653-1656, Oct. 2004.

[6] K.Z. Mei, N.N. Zheng, Y.H. Liu and J.Yao, "Design of high speed Arithmetic Encoder," in *Proc. IEEE Int. Conf. Solid-State Integr. Circuits Technol.(ICSICT'04),* Oct. 2004, pp. 1617-1620.

[7] J.S. Chiang, C.H. Chang, Y.S. Lin, C.Y. Hsieh and C.H. Hsia "High-speed EBCOT with dual context-modeling coding architecture for JPEG2000," *IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 3, pp.865-868, May. 2004.

[8] K.K. Ong, W.H. Chang, Y.C. Tseng, Y.S. Lee and C.Y. Lee, "A high throughput context-based adaptive arithmetic codec JPEG2000," *IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 1, pp.872-875, Sep. 2002.

[9] M. Tauri, M.Oshita, T.Onoye and I.Shirakawa, "High-Speed implementation of JBIG Arithmetic Coder," in *TENCON'99*, Region-10, Vol. 2, p. 1291-1294, 1999.

[10] Y. Li and M. Bayoumi, "A three-Level Parallel High-Speed Low-Power Architecture for EBCOT of JPEG2000," *IEEE Trans. Circuits Syst. Video Technol.,* vol.16, No.9, pp.1153-1163, Sep. 2006.