High Performance Two-Symbol Arithmetic Encoder in JPEG 2000

Yu-Wei Chang, Hung-Chi Fang, and Liang-Gee Chen

Abstract—*In JPEG 2000, the Arithmetic Encoder (AE) is the throughput bottleneck. An AE capable of encoding multiple symbols per cycle is an efficient way to increase the throughput for high-resolution image. In this paper, a two-symbol AE architecture is proposed, which can encode two symbols per cycle. This architecture exploits highly parallelism to shorten the critical path for two-symbol encoding. The proposed architecture can achieve 180 M symbols/sec with 0.35 um CMOS technology and the gate count is 7.7 K.*[1]

Index Terms—JPEG 2000, arithmetic encoder, embedded block coding with optimized truncation (EBCOT), MQ-coder, JBIG2, multi-symbol AE.

I. INTRODUCTION

JPEG 2000 [1], which is a new still image coding standard, is well known for its excellent compression performance and numerous features, such as region of interest, various kinds of scalabilities, error resilience, etc. The block diagram of JPEG 2000 is shown in Fig. 1. JPEG 2000 adopts the Discrete Wavelet Transform (DWT) as its transform algorithm and the Embedded Block Coding with Optimized Truncation (EBCOT) as its entropy coding algorithm. The EBCOT is a two-tiered algorithm, in which tier-1 is the Embedded Block Coding (EBC) and tier-2 is the rate-distortion optimization. After the DWT, each subband is divided into code-blocks, which are independently coded by the EBC to form embedded bit streams. The EBC is the most complex part in JPEG 2000 [2]. It can be further divided into the Context Formation (CF) and the Arithmetic Encoder (AE). The CF generates the symbol, a binary Decision (D), and a ConteXt (CX) to adapt the probability of the symbol. The AE encodes the symbols into embedded bit streams by context-based adaptive probability updating for each CX-D pair [1].

The AE is the throughput bottleneck of JPEG 2000. Due to the serial processing nature, it is difficult to exploit the parallelism of the AE [2]. Moreover, the CF producing multiple CX-D pairs per cycle is proposed [2][3][4] but the AE can only encode one CX-D pair per cycle. In previous works, the FIFO (First-In First-Out) is inserted between the CF and the AE to reduce the bubbles. However, the FIFO can only slightly alleviate this problem. An alternative to solve this problem is to increase throughput of the AE by multi-symbol coding. However, the strong correlation between

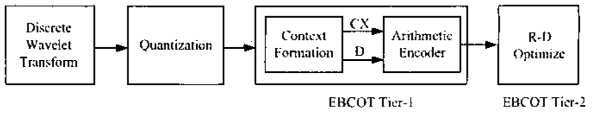

Fig. 1. Block diagram of the JPEG 2000 encoder.

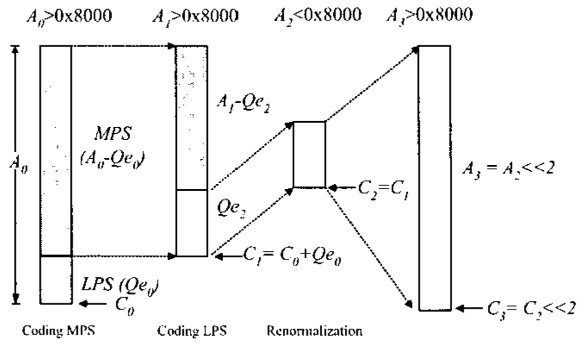

Fig. 2. Arithmetic encoding procedure.

successive symbols makes multi-symbol coding difficult. Moreover, the byte-out procedure and the renormalization operation require iterations that introduce long latency. The second symbol cannot be encoded until the iterations of the first symbol are finished. Therefore, multi-symbol AE by simply cascading single-symbol AE will lead to long critical path. In [5], the throughput of AE is improved with multi-symbol coding under a criterion, i.e. when renormalization iteration steps are less than 2 in successive symbols coding.

In this paper, a two-symbol AE architecture is proposed to solve the above problems, which can always encode two symbols without any constraints. This architecture can efficiently unfold the iterations of the byte-out and renormalization procedures, and therefore the critical path is minimized.

II. ARITHMETIC ENCODING ALGORITHM IN JPEG 2000

Arithmetic coding is a recursive interval subdivision coding procedure as shown in Fig. 2. The entire bit stream is the pointer to the final of interval. For each CX-D pair, the interval, A, is partitioned into two sub-intervals, the Most Probable Symbol (MPS) interval and the Least Probable Symbol (LPS) interval, by the LPS probability, Qe. The value of Qe is adapted by the associated CX of the D. If the D is the MPS, Qe is added to bit stream pointer, C, which points to the lower bound of A. On the other hand, if the D is the LPS, C remains the same while A is updated as Qe. This coding convention requires that the D should be recognized as the MPS or the LPS, rather than simply 0 or 1.

The recursive coding operations for A and C are performed

[1]Yu-Wei Chang is with Graduate Institute of Electrical Engineering, College of Electrical Engineering and Computer, National Taiwan University. Taipei, Taiwan. (e-mail: wayne@video.ee.ntu.edu.tw)
Hung-Chi Fang is with Graduate Institute of Electrical Engineering, College of Electrical Engineering and Computer, National Taiwan University. Taipei, Taiwan. (e-mail: honchi@video.ee.ntu.edu.tw)
Liang-Gee Chen is with Graduate Institute of Electrical Engineering, College of Electrical Engineering and Computer, National Taiwan University. Taipei, Taiwan. (e-mail: lgchen@video.ee.ntu.edu.tw)

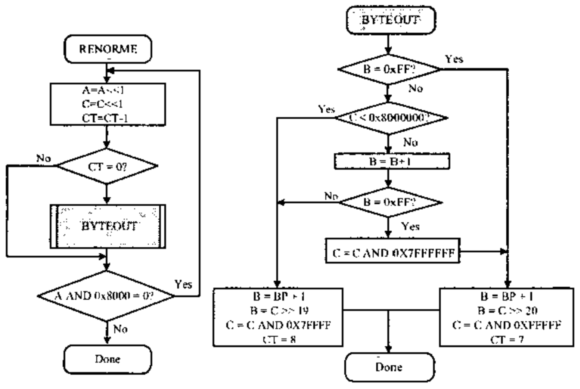

Fig. 3. The flowchart of RENORME and the BYTEOUT procedures.

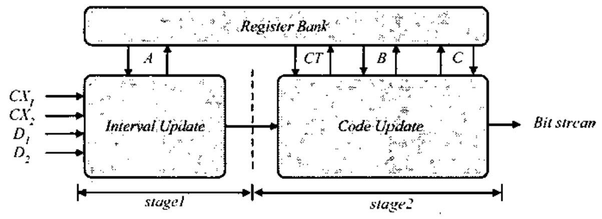

Fig. 4. The block diagram of two-symbol AE architecture. There are two pipeline stages: interval update and code update.

with fixed precision integer arithmetic MQ-coder. A 16-bit integer is used for A, in which 0x0800 is equivalent to 0.75. The C is 28 bits, in which LSB 16 bits, C_{16}, is used to represent the lower bound of interval and MSB 12 bits, C_{12}, is the buffer for the overflow bits. Moreover, the MSB of C_{12} is carry bit. In order to keep enough precision, the renormalization procedure (RENORME) is incurred whenever the A falls below 0x8000. In RENORME, A and C are shifted left until A is larger than 0x8000. During the RENORME, a byte-out procedure (BYTEOUT) is incurred to output one byte to B, which is output byte buffer, whenever C_T, available space (not include the carry bit) in C_{12}, is zero. The flowchart of RENORME and BYTEOUT is shown in Fig. 3 [1]. The theoretical range of the number of renormalization loops is [0,15] and the BYTEOUT is triggered twice at most, and therefore, two bytes at most will be outputted during the RENORME. To distinguish bit streams from the markers (started with 0xFF), bit-stuffing should be handled. As shown in Fig. 3, bit-stuffing is incurred whenever B is 0xFE and the carry bit is 1, or B is 0xFF. With bit-stuffing, the updated value of C_T is 7 since the stuffing bit consumes one bit space.

III. Proposed Architecture

In this section, the high performance two-symbol Arithmetic Encoder (AE) architecture is proposed. It has two-stage pipeline stages including interval update and code update, as shown in Fig. 4.

A. Two-stage Pipelined Architecture

The first stage, *Interval Update* module, updates A with

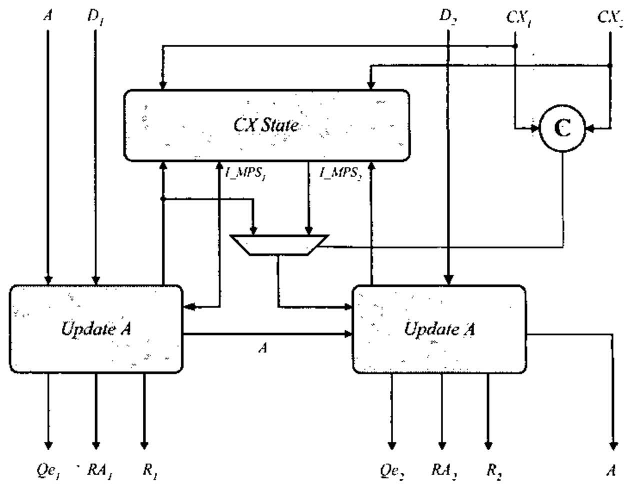

Fig. 5. The circuit of the Interval Update module. This module update interval register A for two CX-D pairs and the probability state for each CX.

subinterval division and renormalization according to the two input CX-D pairs. The second stage, *Code Update* module, updates C according to the results of the first stage. The second stage performs the RENORME and BYTEOUT to generate embedded bit streams. As shown in Fig. 3, the shift amount of C depends on the shift mount of A in the RENORME. The delay time is too long if the renormalization of C is performed after the shift numbers of A is determined. Therefore, separating C from A with one stage pipelining is a good choice to shorten the critical path.

B. Interval Update Stage

The block diagram of the first stage, *Interval Update* module, is shown in Fig. 5. This module contains the interval register A and the context state register bank, *CX State*, which includes the indication of MPS symbol, *MPS*, and the index of probability table, I, for each context. Moreover, this module generates the estimated probability of LPS, Qe, the sub-interval range selection, R, and renormalization amount, RA, for the second stage. Note that, if two contexts are the same, the up-to-date I and *MPS* for the first symbol will be chosen as the input of *Update A* module for the second symbol, as shown by the multiplexer in Fig. 5.

As shown in Fig. 2, A will be updated as A-Qe or Qe depending on whether the MPS or the LPS is coded. By detecting numbers of leading zero of the updated A, RA can be obtained. Then, the renormalization of A is completed by shifting A with RA. Although this method seems reasonable, the delay time is too long to complete interval update procedure. To reduce the delay, we proposed a new technique to generate RA in parallel with updating A. The diagram of *Update A* module is shown in Fig. 6. Note that there are only two possible values, A-Qe and Qe, for the updated A. In the first case, A-Qe is always larger than 0x29FF since A must be larger than 0x8000 and Qe must be smaller than 0x5601. Therefore, there are only three possible values, i.e. 0, 1, and 2, for RA. In this case, RA can be obtained easily within short delay time. For the second case, A is updated as Qe, which is

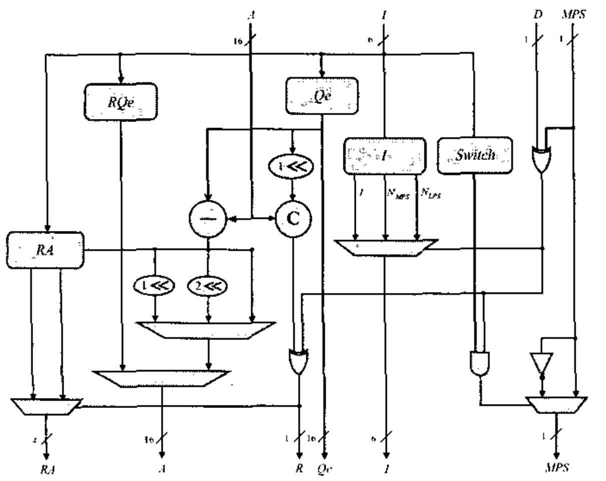
Fig. 6. The circuit diagram of Update A module. This module updates interval register and renormalizes A.

obtained by table lookup. Therefore, *RA* can be obtained by the pre-defined *RA* table, which stores the numbers of leading zero of each entry of *Qe* table. Besides the tables defined in standard [1], i.e. *Qe* table, *I* table and *Switch* table, we also form a new table, *RQe*. The *RQe* table stores the result of renormalization for each *Qe* entry in the probability table. By use of the *RQe* table, no shifter is required when *Qe* is selected as next *A*. As shown in Fig. 6, the updated and renormalized A is chosen from two paths. One is from *RQe* and the other path is from renormalized *A-Qe*. The updated procedure and renormalization procedure of *A* are in parallel processing, and therefore the delay time is much shorter than the conventional architecture.

C. Code Update Stage

As shown in Fig. 3, the BYTEOUT will occur whenever the C_T, the available space in C_{12}, is zero. At the same time, the byte stored in *B* will be outputted and most significant byte in C_{12} will be moved to *B*. During the BYTEOUT, the carry propagation and bit-stuffing problem should also be handled due to finite length of bit stream buffer. In conventional architecture [2][3][4], the RENORME and BYTEOUT are achieved by cascading several shifters and conditional selection circuits for generating bit streams. If the code update procedure for the second symbol is performed after the RENORME and the BYTEOUT of the first symbol, the delay time is too long due to sequential processing.

To shorten the delay of code updating between two symbols, we proposed several techniques to parallelize the RENORME and the BYTEOUT. The proposed circuit for *Code Update* module is shown in Fig. 7. This module updates *C*, C_T, and *B* and generate embedded bits stream BO_1 BO_2, BO_3, and BO_4 with the corresponding output enable control signal, OE_1, OE_2, OE_3, and OE_4. The *Update C* module for the first symbol updates *C* according to the range selection, *R*. The *Mask Generator* module produces 28-bit wide *MASK* in parallel with *Update C* module. With bit-wise *AND* operation with *MASK* and *C*, the bits in *C* that will be outputted during BYTEOUT, which is realized with *Byte Output* module, will

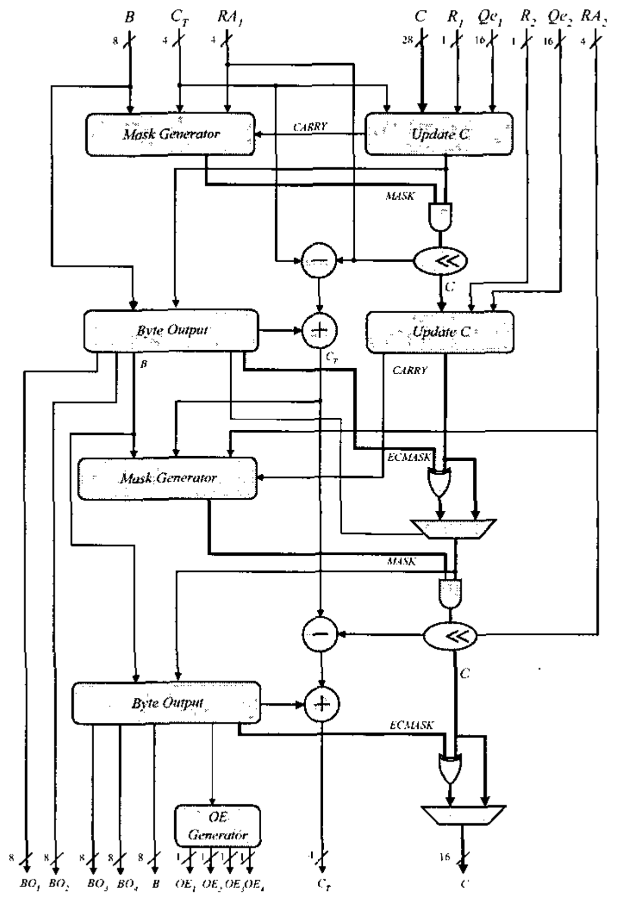
Fig. 7. The circuit diagram of the second stage of AE. This module updates and renormalizes code register C. This module also generates embedded bit streams with corresponding output enable signals.

become zero. Then, the one-step renormalization of *C* is achieved with a shifter, as shown in Fig. 7. The renormalized *C* is connected to *Update C* module for the second symbol. Note that the Update C module for the second symbol is in parallel processing with *Byte Output* module for the first symbol, and therefore, the critical path is shortened.

The circuit diagrams of *Update C* and *Mask Generator* module are shown in Fig. 8 and Fig. 9 respectively. The *Update C* module is actually a 28-bit adder, which is implemented with one 16-bit adder and one 12-bit incrementer. The MSB of *C*, i.e. *CARRY* bit, is obtained by shifting *C* with C_T and then is chosen according to *R*. The *CARRY* is connected to *Mask Generator* module for mask selection. In Fig. 9, the *MASK* comes from two branches. The left branch is the case with bit-stuffing whereas the right branch is the case without bit-stuffing. The M_1 and M_2 masks are chosen when only one byte is needed to output whereas the M_3 mask is chosen when two bytes are needed to output. The M_1 mask means no byte output is needed. In this module, firstly, the *RA* is compared with C_T+7, C_T+8, and C_T respectively to select candidates of *MASK*. Then, the candidates of *MASK* are shifted with *CT*. The function of this shifter is to align *MASK* with un-renormalized *C*. Finally, *B* is compared with 0xFF

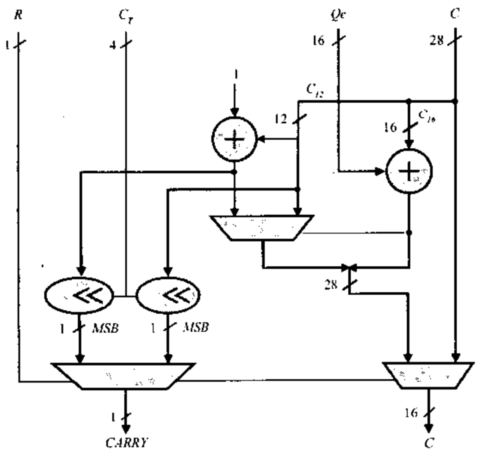

Fig. 8. The circuit diagram of Update C module

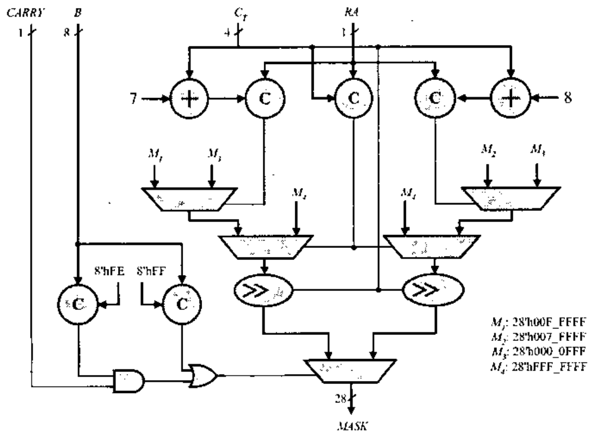

Fig. 9. The circuit diagram of Mask Generator module

and 0xFE. The results of comparison are combined with *CARRY* to check if bit-stuffing is required and to select final *MASK*. Actually, *MASK* chosen from M_2 may be incorrect in one case, i.e., when bit-stuffing is required for outputting the second byte since the actual value of the first output byte is unavailable. However, *Byte Output* module will handle bit-stuffing problem and generates Error Compensated Mask, *ECMASK*, to correct *C* to avoid error propagation after the processing of *Update C* module for the second symbol.

IV. Implementation And Comparison

The proposed architecture is described by the Verilog HDL and is synthesized with 0.35 μm cell library. The implementation results are summarized in Table I. In [15], the multi-symbol AE is designed under the constraint that the shift amount of *A* must be smaller than two. Thus, it cannot always encode two symbols per cycle.

To show the effectiveness of the proposed techniques, both proposed AE and conventional AE are described by Verilog HDL and then synthesized. The results of area for both architectures are almost the same while the critical path are quite different. The histogram of delay paths is shown in Fig. 10. By use of proposed techniques, the critical path is reduced from 17 ns to 11 ns, which is 35% improvement. Note that the distribution of path delays is centralized at 11 ns in the proposed AE due to highly parallel processing, while only few numbers of path delays are critical (17 ns) in conventional AE architecture.

TABLE I
ARITHMETIC ENCODER PERFORMANCE AND COMPARISON

	Proposed
Technology	0.35 μm
Pipeline stage	2
Critical path	11 ns
Gates count	7.7 k
Throughput	2 symbol/cycle

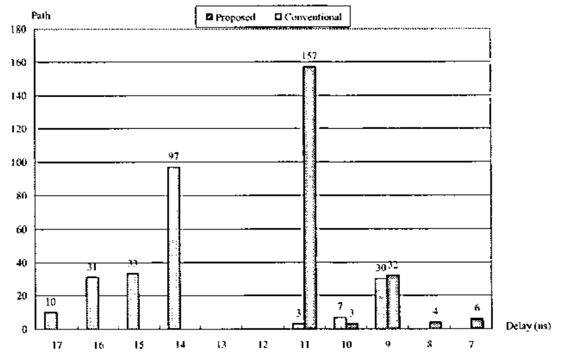

Fig. 10. Path delay of the proposed and conventional AE architecture.

V. Conclusion

In this paper, a two-symbol AE architecture is proposed, which can encode two symbols per cycle. This architecture can minimize the delay for two-symbol encoding by several parallel processing techniques. The proposed architecture is highly optimized for both area and timing. It can achieve 180 *M* symbols/sec using 0.35 *um* CMOS technology and the gate count is 7.7 *K*.

References

[1] ISO/IEC JTC1/SC29/WG1 N1855, JPEG 2000 Part1:Final Draft International Standard (ISO/IEC FDIS15444-1), Aug. 2000.

[2] 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 Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 3, pp.219-230, Mar. 2003.

[3] H. C. Fang, T. C. Wang, C. J. Lian, T. H. Chang and L. G. Chen, "High speed and memory efficient EBCOT architecture for JPEG 2000," *in Proceedings of IEEE International Symposium on Circuits and Systems*, vol. 2, pp25-28, May 2003.

[4] J. S. Chiang, Y. S. Lin, and C. Y. Hsieh, "Efficient pass-parallel architecture for EBCOT in JPEG2000," *in Proceedings of IEEE International Symposium on Circuits and Systems*, vol. 1, pp.773-774, May. 2002.

[5] Schumacher, P.R, "An efficient JPEG2000 tier-1 coder hardware implementation for real-time video processing," *IEEE Transactions on Consumer Electronics*, vol 49, pp780-786, 2003.