

High Efficiency Concurrent Embedded Block Coding Architecture for JPEG 2000

Tsung-Da Lin, Wei-Bin Yang and Chang-Yu Hsieh*

*Department of Electrical Engineering, Tamkang University,
Tamsui, Taiwan 251, R.O.C.*

Abstract

Embedded block coding with optimized truncation (EBCOT) is the most important part of JPEG 2000. Due to the bit level operation and the three-pass scanning technique, the EBCOT may take more than 50% operation time in the JPEG 2000. This paper presents a high efficiency concurrent EBCOT (HECEBC) entropy encoder hardware architecture. The proposed HECEBC can concurrently process the four samples in a stripe column. Furthermore this architecture can be extended to process several stripe columns concurrently for the JPEG 2000 to accomplish high resolution applications in real time. Besides, the HECEBC uses the technique of concentrated context window to stabilize the Context-Decision (CX-D) output to relax the load in between the arithmetic encoder (AE) and the parallel-in-serial-out (PISO) buffer to triple the EBC performance.

Key Words: EBCOT, JPEG 2000, Pass Detection, PPEBC, Significant State

1. Introduction

JPEG has good compression performance for natural images, and it was widely used in the image compression systems in the past decade. However, in the low bit rate image compression JPEG may generate very severe blocking effect and usually annoys people. JPEG 2000 [1–5] is the newest image compression standard proposed by ISO/IEC JTC1/SC29/WG1 [1]. It outperforms JPEG in many features. For low bit rate image compression it can have better quality than that of JPEG. Besides, JPEG 2000 provides many features such as progressive quality and resolution image transmission, region of interest (ROI), loseless and lossy compression, good error resilience, ..., etc. Due to these good features, JPEG 2000 can be applied to networking transmissions, digital still camera, surveillance, digital theater, ..., etc. JPEG 2000 can provide such high compression quality and good features, however the complexity of the algorithm is much higher than that of JPEG.

Figure 1 shows the processing block diagram of JPEG 2000. After the color transform the image is processed by the function blocks of 2D-DWT (2D-discrete wavelet transform) and uniform quantization, respectively, and finally the data are fed to the function block of EBCOT (embedded block coding with optimized truncation) [6–9] to generate the bit stream. Due to the different scanning order of the function blocks, JPEG 2000 needs tile memory blocks in between function blocks of color transform and 2D-DWT, and code-block memory in between function blocks of uniform quantization and EBCOT. The size of the memory depends on the processing capacity of the function blocks. Among the function blocks, 2D-DWT and EBCOT have very high complexity, and they take most of the efforts in JPEG 2000 [10].

The operation of EBCOT consists of two steps, embedded block coding (EBC) and rate-distortion optimization (RDO). EBC tries to encode the quantized DWT data by context-based arithmetic coding approach; on the other hand RDO tries to decide the best truncation point of the bit stream encoded by the EBC according to the bit-rate. The data structure of JPEG 2000 is shown in

*Corresponding author. E-mail: cyhsieh@ee.tku.edu.tw

Figure 2. After quantization, the sub-band of the 2D-DWT is divided into several code-blocks that can be processed by the EBC. Generally, the largest code-block is $64 \times 64 \times 10$ bits in size (5 KB). Each code-block is then decomposed into several bit-planes (BP). The EBC skips those insignificant bit-planes and then encodes the code-block from the most significant bit-plane, bit-plane by bit-plane. According to different significances, each pixel on the bit-plane is encoded by one of the three passes sequentially as shown in Figure 3. Because the bit-plane number of the insignificant bit-plane depends on the data content of the code-block, the data rate is not constant and it needs internal memory for data buffering. The EBC is in bit-level operation, and therefore the computation complexity is much higher than other function blocks.

In JPEG 2000, EBC must process a large quantity of data. In order to increase the efficiency of EBC, most of the researches of EBC focus on the parallel architecture to increase the operation speed. Chiang et al. [11] proposed the pass parallel approach for EBC, and it needs only 16 K bits internal memory. This architecture can merge the three coding passes to a single coding pass to increase the coding efficiency. The CX-D pair generated by the pass parallel EBC may not follow the sequence order of the original coding pass order, and they proposed the pass switch arithmetic encoder (PSAE) to process the disordered CX-D pair. The pass parallel EBC is not a parallel architecture and it can only process one bit data each clock cycle. Fang et al. [12] proposed a word-level parallel EBC architecture. It can concurrently process all the bit-planes to increase the EBC computation

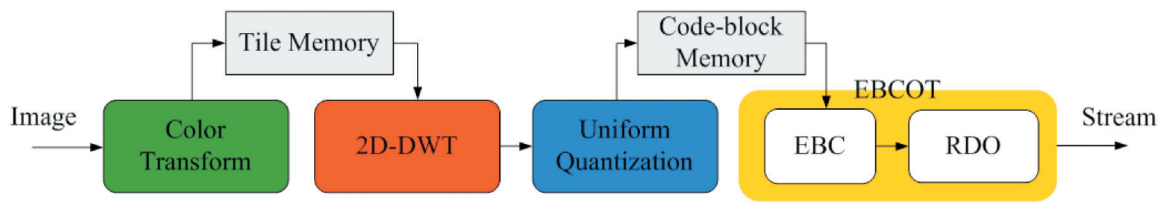


Figure 1. JPEG 2000 system block diagram.

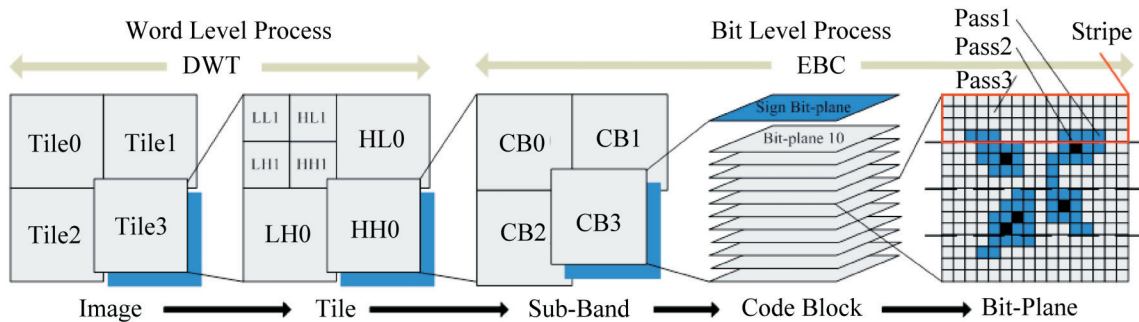


Figure 2. JPEG 2000 data structure.

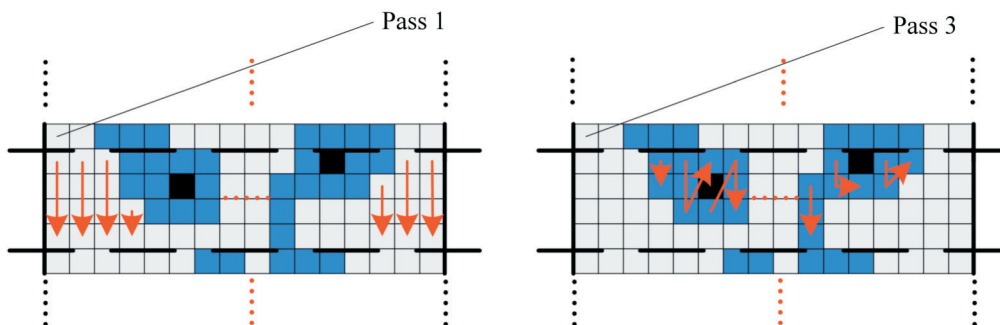


Figure 3. Column stripe scanning order.

speed and also it can reduce the code-block memory. It does not need any internal memory, however it must process all the bit-planes concurrently in a 10-bit computation capacity for each bit-plane, even though the bit-plane is an insignificant bit-plane. Li et al. [13] proposed a three-level parallel architecture. This architecture adopts four word-level circuits to increase the throughput. It uses several shared RLC (run-length coding) results to reduce the clock cycles and power dissipation. Chen et al. [14] proposed a scalable EBC architecture that uses pre-RDO to reduce the number of code-blocks. It uses the scheme of bit-plane parallel EBC to skip the insignificant bit-plane to increase the computation speed of the word-level EBC. It needs 12 Kb internal memory. In this work we propose a high efficiency concurrent EBC, HECEBC, architecture for JPEG 2000. HECEBC can concurrently process four samples in a stripe column. This architecture can be extended to process several stripe columns concurrently for the JPEG 2000 to process high resolution application in real time.

The rest of this paper is organized as follows. Section 2 briefly describes the operation of pass parallel embedded block coding. The proposed HECEBC architecture is illustrated in section 3. The experimental results and comparisons are given in section 4. Finally section 5 concludes this work.

2. Overview of Pass Parallel Embedded Block Coding

In the JPEG 2000 standard, the symbols of each bit-plane are classified into three passes (Pass 1, Pass 2, and Pass 3, respectively), and each bit-plane is processed in three times by EBC. In each pass the EBC processes its

own pass and skips those passes not belonged to its own. The encoding procedures are shown in Figure 4(a) and the EBC block diagram is shown in Figure 5. Since the pass of each symbol is not predictable in each bit-plane, the scanning order is not fixed and makes the encoding difficultly. To overcome the difficulties of the three-pass scheme of the standard EBC, Chiang et al. [11] proposed pass parallel EBC (PPEBC) to merge the three-pass scheme to a single pass operation to improve the pass encoding performance, and the procedures are shown in Figure 4(b). The PPEBC algorithm consists of two steps: context modeling (CM) and arithmetic encoding (AE). The coefficients of the DWT must pass through the code block memory to be decomposed into bit-planes, and then the bit-plane data are fed to the context modeling function block to generate context and decision pair (CX-D pair). The arithmetic encoder further uses the CX-D pair to generate the compressed bit stream.

In the PPEBC operation, it needs to calculate four state variables: magnitude data (Vp), sign data (χ), significant state (σ), and refinement state (γ). The definitions of these four state variables are listed in Table 1, where K represents the number of the significant bit-plane; k represents the number of the current bit-plane, and $k = 0$ is the LSB.

According to the significance of each sample, EBC

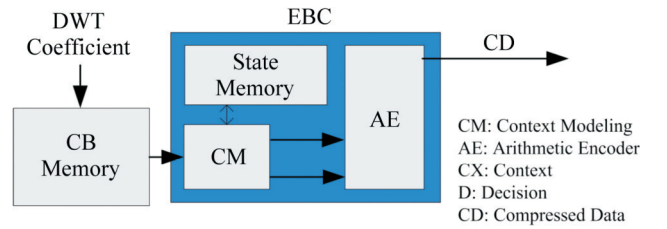
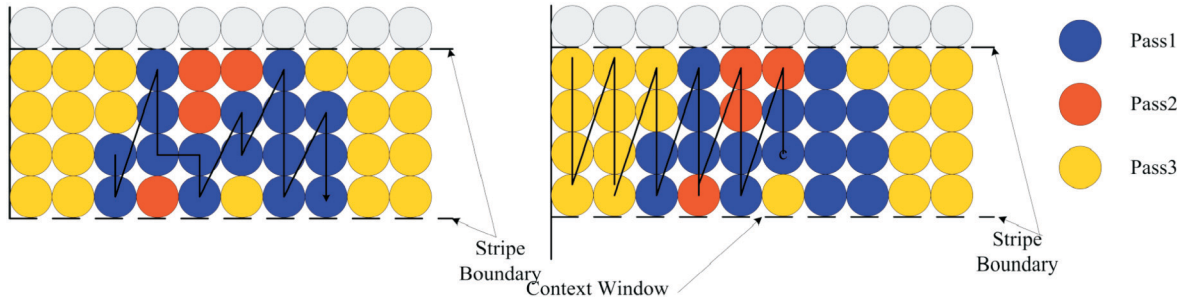


Figure 5. EBC block diagram.



(a) The encoding procedures of EBC

(b) The encoding procedures of the PPEBC

Figure 4. The encoding procedures.

Table 1. State variables used in the context modeling

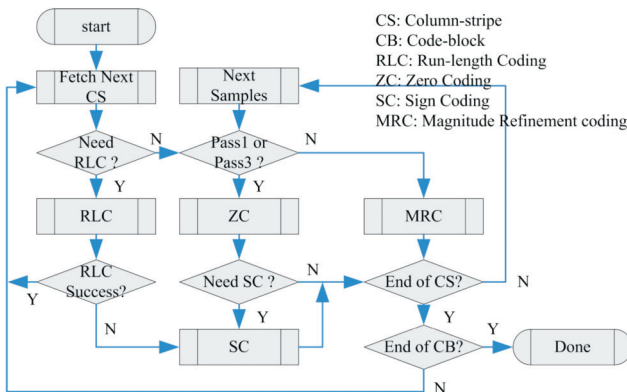
Category	Variables	Description	Formula
Bit-plane Data	$Vp[k]$	Magnitude Data	$(Coef \& 2^k)$
	$\chi[k]$	Sign Data	$Sgn(Coef)$
Coding State Variable	$\sigma[k]$	Significant State	$\prod_{k+1}^{K-1} Vp$
	$\gamma[k]$	Refinement State	$\sigma[k+1]$

classifies it into three categories (passes), and the context modeling encodes the sample in three different coding passes upon its significance category. The pass category can be decided by equation (1). In (1) P_c is the pass classification of sample c ; s is the 8 surrounding neighbors of sample c ; ϕ_1^s is the significant state of the surrounding neighbors in coding Pass 1, and ϕ_p^c is the significant state of c in coding Pass P .

$$P_c = \begin{cases} 1, & \sigma_c = 0 \& \sum \phi_1^s \neq 0 \\ 2, & \sigma_c = 1 \\ 3, & otherwise \end{cases} \quad (1)$$

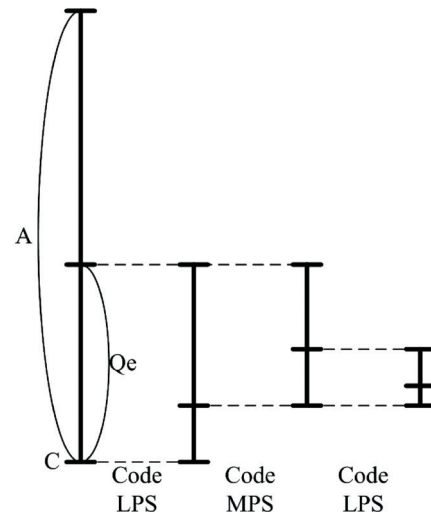
$$\phi_p^c = \begin{cases} \sigma \parallel (Vp \& p \neq 1 \& \sum \phi_1^s \neq 0), & c \in \text{uncoded samples} \\ \sigma \parallel [Vp \& (p = 3 \parallel \& \sum \phi_1^s \neq 0)], & c \in \text{coded samples} \end{cases} \quad (2)$$

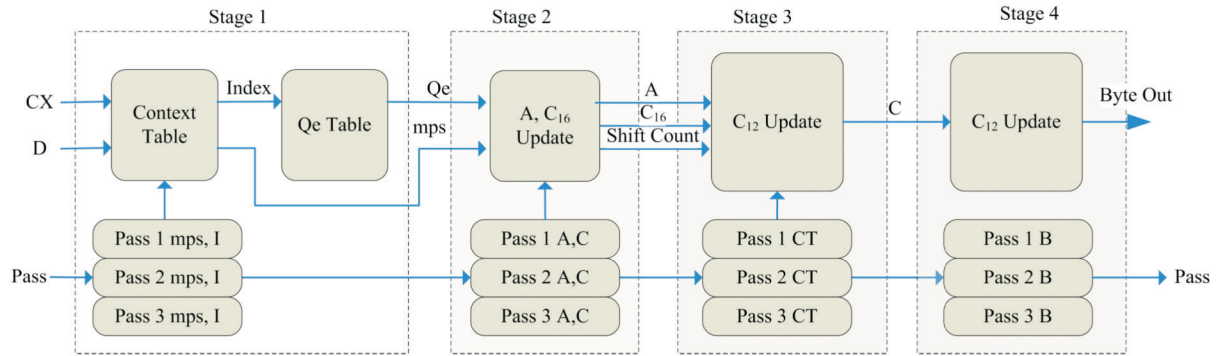
We can use state variables found in Table 1 together with equations (1) and (2) and apply the computation flow shown in Figure 6 to accomplish the context modeling computation. The details of run-length coding (RLC), zero coding (ZC), sign coding (SC), and magnitude refinement coding (MRC) can be referred to [1].

**Figure 6.** Context modeling flow chart.

The other step of PPEBC is the context-based arithmetic coding; the operating diagram is shown in Figure 7. It uses the probability of the encoded data to find the location C of the probability accumulation line and then obtains the entropy coding. The PPEBC must operate arithmetic encoding once for each of the three coding passes. In order to process the alternatively generated CX-D pairs, PPEBC uses pass switch arithmetic encoder (PSAE) [11] for arithmetic encoding. The block diagram of PSAE is shown in Figure 8. There are three register banks for Pass 1, Pass 2, and Pass 3, respectively, in the PSAE to process the CX-D pairs generated by each pass concurrently. The context-based arithmetic encoding has to accumulate the probability for each Qe of the context. Qe uses the 6-bit state variable, I , to express the possibility of the 46 types of Qe . Besides, each context must record its own most possible symbol (MPS), and therefore the arithmetic encoder for each bit-plane needs $7 \times (14 + 3 + 16) + 3 \times (16 + 8 + 28 + 4) = 399$ bits.

In the original JPEG 2000 standard, the context modeling EBC executes Pass 1, Pass 2, and Pass 3 sequen-

**Figure 7.** JPEG 2000 arithmetic encoding procedure.



tially to find its own sample for the context modeling. If the sample is not belonged to that pass, it will skip to the next sample. By this approach 2/3 of the operations in context modeling are operated for checking invalid samples. The encoding efficiency is low, and most of the memory access time to read data from the memory is wasted. On the other hand PPEBC can overcome the shortcoming of the conventional EBC to increase the memory access efficiency to improve the EBC performance.

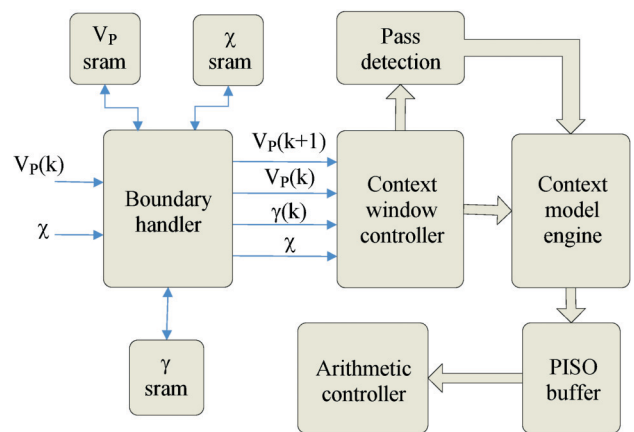
3. High Efficiency Concurrent Embedded Block Coding Architecture and Operation

PPEBC can eliminate all the invalid operations to increase the EBC encoding efficiency. However, the EBC is in bit level operation and it cannot satisfy the real time applications of image compression. In order to increase the EBC operation efficiency for real time application, the high efficiency concurrent embedded block coding (HECEBC) architecture is proposed in this work. The HECEBC architecture uses several parallel circuits to process data concurrently to increase the data throughput. Figure 9 shows the block diagram of the proposed HECEBC architecture. In Figure 9, the data from the external memory are fed to the left hand side of the HECEBC. The HECEBC reads 4 samples at the same column of the magnitude bit-plane in each clock cycle. When reading the first magnitude bit-plane, the HECEBC also reads the corresponding sign data from the sign bit-plane. Then it calculates the state variables for each pass and processes the edge handling. All the calculated state variables are stored in the context window constituted by shift registers for the following pass detection and calcu-

lations for the encoded circuits. The detail operations of HECEBC are described in the following paragraphs, and we will discuss the operation procedures of pass detection, Pass 1, Pass 2, and Pass 3.

In equation (1), the pass detection needs the information of the significant state to detect the pass classification for the current sample. The Pass 1 coding procedure requires the significant states of the surrounding neighbor samples for itself pass detection, zero coding, and sign coding. The neighbor samples separate coded samples and uncoded samples, as shown in Figure 10. In Figure 11(a), the significant state of the coded samples and uncoded samples are represented by ϕ_1^s and σ_s respectively for the current sample. ϕ_1^s and σ_s can be derived from (3) by the coded sample and uncoded sample respectively.

$$\phi_1^c = \begin{cases} \sigma_c, & c \in \text{uncoded sample} \\ \sigma_c \parallel [Vp \& (\sum \phi_1^s \neq 0)], & c \in \text{coded sample} \end{cases} \quad (3)$$



Equation (3) indicates that we need to know the significant state information, ϕ_1^s , of the surrounding neighbors when encoding Pass 1 to calculate ϕ_1^c . Let us use Figure 11 to illustrate the relationship between ϕ_1^s and ϕ_1^c . Figure 11(b) shows when the neighbors of that sample are uncoded and according to (3) $\phi_1^s = \sigma_s$ for sample 1. Figure 11(c) shows the significant state information of the surrounding neighbors when EBC is encoding sample 2. Since the first point is a coded sample, its significant state becomes:

$$\begin{aligned}\phi_1^1 &= \sigma_1 \parallel \{Vp \& [(\phi_1^2 + \phi_1^5 + \phi_1^6) \neq 0]\} \\ &= \sigma_1 \parallel \{Vp \& [(\sigma_2 + \sigma_5 + \sigma_6) \neq 0]\}\end{aligned}\quad (4)$$

Because only the coded samples need to calculate ϕ_1^s , we can store ϕ_1^s of the coded sample to registers and then we can process the pass detection operation in any samples as shown in Figure 11(a).

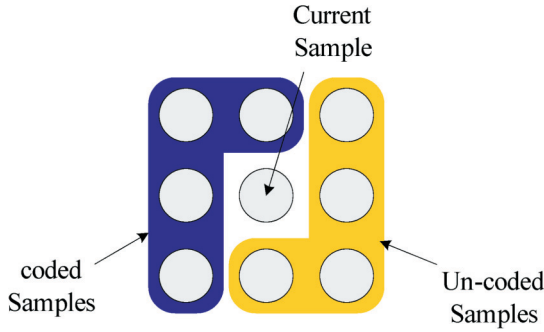


Figure 10. Coded samples and uncoded samples.

When calculating the new ϕ_1^s , we need ϕ_1^s of the coded samples. Therefore we can use the parallel circuits shown in Figure 12(a) to calculate several ϕ_1^s 's of the same column concurrently. Figure 12(a) can be extended to process the pass detections for several columns concurrently, and the architecture block diagram is shown in Figure 12(b).

Vp , χ , and ϕ_1^s are used in Pass 1 coding procedure. Same as the pass detection procedure, ϕ_1^s can be implemented by Figure 12. Vp and χ can be obtained from the external memory and internal state memory. If the current sample detected by pass detection belongs to Pass 2, the current sample is processed by the Pass 2 coding procedure. In the Pass 2 coding procedure, ϕ_2^c , and Vp are needed for MRC. ϕ_2^c can be derived from (2) as follows:

$$\phi_2^c = \sigma_c \parallel (Vp \& \sum \phi_1^s \neq 0) \quad (5)$$

From (5) we can find the significant states of the 8 surrounding neighbors for the Pass 2 coding procedure. Vp and γ can be obtained from the external memory and internal state memory.

If the current sample belongs to Pass 3, the Pass 3 coding procedure is activated for context modeling operation. The Pass 3 coding procedure needs Vp , χ , and ϕ_3^c . Vp and χ are obtained by the same approach as that of the Pass 1 coding procedure. ϕ_3^c is derived from (2) as:

$$\phi_3^c = \begin{cases} \sigma_c \parallel (Vp \& \sum \phi_1^s \neq 0), & c \in \text{uncoded samples} \\ \sigma_c \parallel Vp, & c \in \text{coded samples} \end{cases} \quad (6)$$

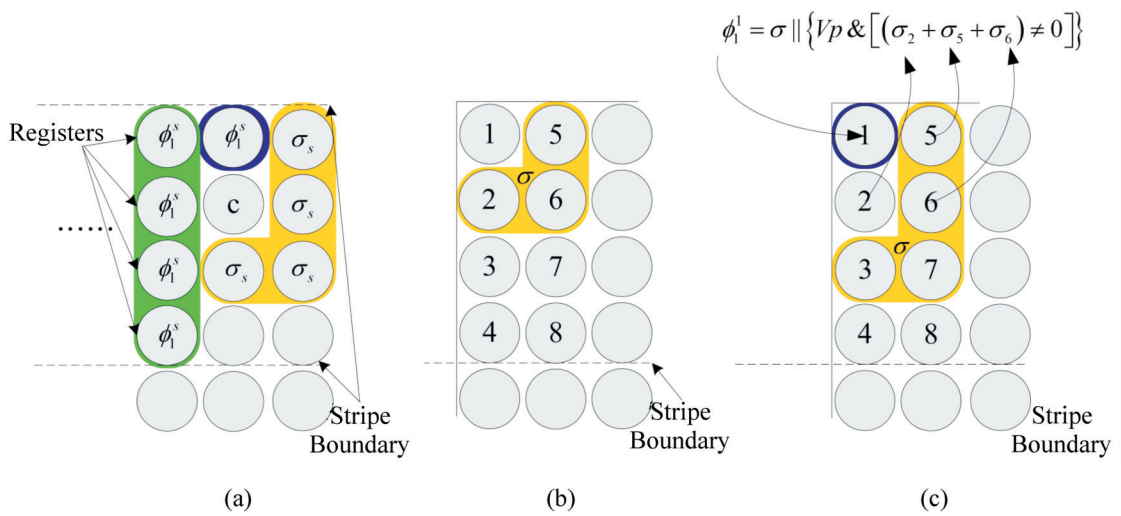


Figure 11. Pass detection.

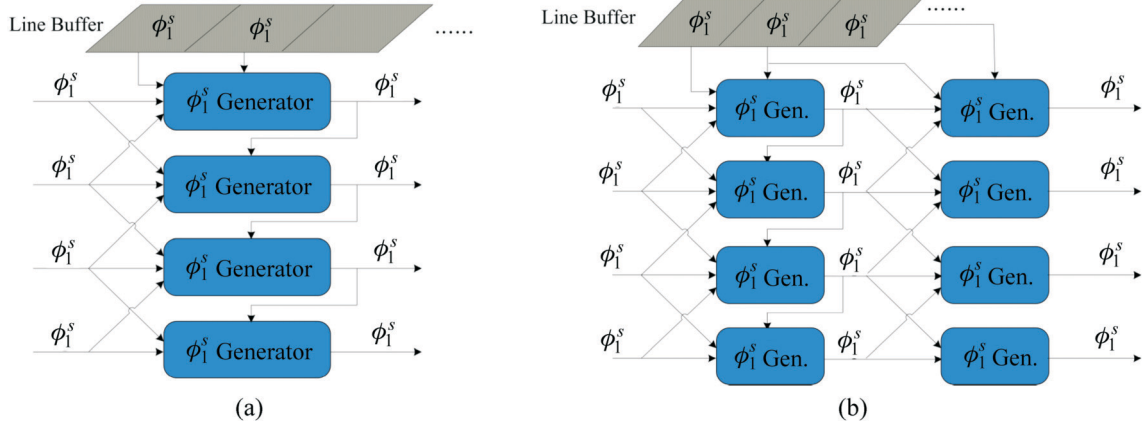


Figure 12. Architecture block diagram of the pass detection.

Referred to [11], ϕ_2^c is called virtual significant state, σ_v . According to equations (3), (4), (5), and (6) the significant states in each pass are summarized in Table 3. Actually the virtual significant state σ_v can be calculated by σ such that the pass coding procedures do not need to encode by following the coding sequences of Pass1, Pass 2, and Pass 3.

If we can predict (calculate) more virtual significant states, the context modeling can be extended to any numbers of sample of parallel processing. Figure 13 shows the architecture of two-column parallel processing. Here the shift register width of the context window must be expanded from 4 bits to 8 bits, and one more σ_v generator circuit is needed.

In the context modeling extension approach, the in-

ternal memory of σ , Vp , and χ must be modified. In the non-extension context modeling operation, we expect the system can access memory read and write within one

Table 3. The significant state selections in different coding passes

usage	Significant State Selection (ϕ_p^c)		
	Coded samples	Current samples	Uncoded samples
Pass detection	σ_v	σ	σ
Pass 1 coding	σ_v	σ	σ
Pass 2 coding	σ_v	σ_v	σ_v
Pass 3 coding	$\sigma_c \parallel Vp$	σ_v	σ_v

“ $\parallel$ ” is OR logic operation.

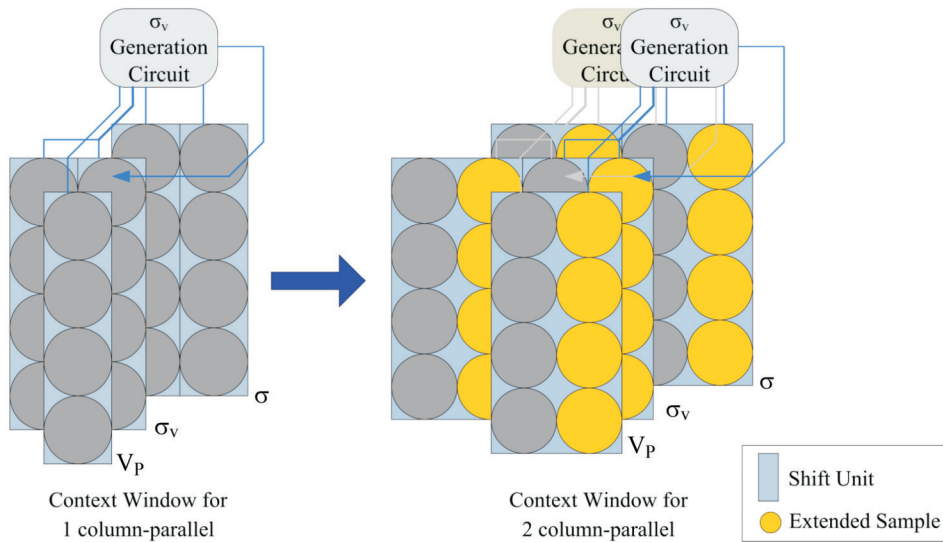


Figure 13. Context window extension.

clock cycle, and therefore the width of the data bus is twice of that of the stripe column, namely 8 bits. In the extension approach, memory access for read and write in each clock cycle becomes double, and the data bus should be expanded but the memory has to be rearranged as shown in Figure 14. In the context modeling extension scheme, it can allow two stripe context encoders to encode two stripe columns concurrently to double the throughput. In a similar manner, the architecture can be extended further to any size of parallel processing to increase the throughput for specific applications.

HECEBC can double the context modeling performance by increasing the virtual significant state operation circuits. Particularly the extension architecture neither needs to increase the size of the internal memory, nor increases the memory access time of the external memory. This architecture can overcome the slow speed calculation problem of the EBCOT.

4. Experimental Results and Comparisons

The HECEBC can increase the throughput of EBC by increasing the virtual significant state circuits. However, the increasing of the virtual significant circuits may increase the critical path to reduce the clock frequency of the system. Table 4 shows the performance comparisons of 4, 8, and 12 samples parallel processing for HECEBC. According to Table 4, the maximum operating frequency indeed reduces for the 12 samples processing, but the performance increases.

Based on the proposed concurrent context modeling architecture and PSAE, an experimental HECEBC with 4-sample processing was designed and synthesized by Artison TSMC 0.18 μm standard cell library. Furthermore, we use Artison Memory Compiler to generate the area of the internal memory. The area of the internal memory can help us to estimate the reasonable hardware cost of the previous research works. The context modeling can operate at 75 MHz, and the PSAE operates at 300 MHz. The detail specifications are listed in Table 5.

Table 6 shows the comparisons of computation time and throughput rate of the performance for several previous works [3,10–13,15,16]. In Table 6 we categorize the encoding architectures into bit-plane based and word based architectures. The proposed architecture has better performance than the previous bit-plane based architectures [3,10,11,15]. The performance of the proposed architecture has triple efficiency than that of the previous

Table 5. The specification of the experimental EBC circuit

Process Tech.	TSMC 0.18 μm	
Frequency	Context-Modeling	75 MHz
	Others	300 MHz
Power Consumption	169 mW	
Synopsys Report for Area	Component	Gate Count
	Context-Modeling	7320.27
	MQ-Coder	14385.01
	SRAM	17648.99
	Total	39354.27

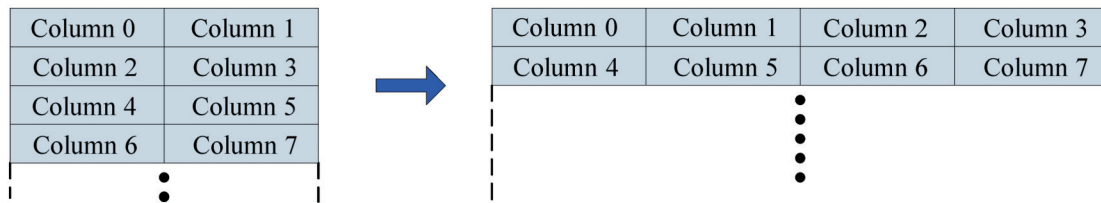


Figure 14. The reconfiguration of the memory for two columns operation.

Table 4. Throughput vs. maximum operation frequency of 4, 8, and 12 samples processing

	Throughput (Sample/cycle)	Area (Gate Count)	Max. Frequency (MHz)	Speed (Mega-Samples/s)
4 SP-HECEBC	4	19374.1	103.3	413.2
8 SP-HECEBC	8	21705.2	91.4	731.2
12 SP-HECEBC	12	23144.7	77.0	924.0

works. Although the performance of the proposed architecture is less than the word based architectures [12,13,16], the hardware cost of each word based architectures is greater than the proposed architecture, as shown in Table 7.

Table 7 shows the comparisons of logic area, memory requirement, and estimated memory area of several previous works [3,10–13,15,16]. The total area of the proposed architecture is less than that of each previous work, because the 4K-bit internal memory saving compares with each bit-plane based architecture. The advantages of the word based architecture [12,13,16] are almost free internal memory and high encoding performance. However, the hardware cost of the word based architecture is larger than the proposed architecture.

Tables 6 and 7 conclude two short conclusions. First, although the performance of the word based architecture is double than that of the proposed HECEBC, its hardware cost is doubled. Second, the proposed ar-

chitecture has good speed performance with suitable hardware cost.

5. Conclusion

In this paper we propose a high efficiency concurrent EBC, HECEBC, architecture for JPEG 2000. HECEBC can concurrently process four samples in a stripe column. This architecture can be extended to process several stripe columns concurrently for the JPEG 2000 to process high resolution application in real time. In the conventional context modeling operation, the generation of the CX-D pairs is not stable and it usually needs a large parallel-in-serial-out (PISO) buffer to relax the load of the arithmetic encoder. The proposed HECEBC uses the technique of the concentrated context widow. This approach can stabilize the CX-D pair generation and further to relax the load of the arithmetic encoder to triple the EBC performance.

Table 6. Performance comparisons

	Coding Type	Computation Time (cycles)	Throughput Rate (Mega-Samples/s)
[3]	Bit-plane Based	$(3n - 2)N^2$	N/A
[10]		$1.3nN^2$	47.3 @ 50 MHz
[11]		nN^2	59.6 @ 50 MHz
[15]		$1.3nN^2$	93.9 @ 100 MHz
[13]	Word Based	$1.5N^2$	434.2 @ 81 MHz
[14]		$0.91N^2$	659.3 @ 75 MHz
[16]		N^2	559.7 @ 55 MHz
This work	Bit-plane Based	$\frac{1}{4} nN^2$	293.6 @75MHz

n: Number of Bit-Plane, N: Width of each Bit-Plane.

Table 7. Hardware cost comparisons

	Logic Area (Gate Count)	Memory Size (Bits)	Estimated Memory Area (Gate Count)	Power Consumption (mW)
[3]	N/A	N/A	N/A	N/A
[10]	19000	$4N^2$	23532	N/A
[11]	23927	$4N^2$	23532	92
[15]	21589	$4N^2$	23532	131
[12]	91758	12N	1103	N/A
[13]	84405	4.5K	6618	780
[16]	152136	N/A	N/A	N/A
This work	21705	$3N^2$	17649	169

N: Width of each Bit-Plane.

References

- [1] *JPEG 2000 Part I: Final Draft International Standard (ISO/IEC JTC1/SC29/WG1 N1855*, Aug. (2000).
- [2] Taubman, D. and Marchellin, M., *JPEG 2000: Image Compression Fundamentals, Standards and Practice*, Norwell, MA: Kluwer Academic (2002).
- [3] *JPEG 2000 Verification Model 7.0 (Technical Description)*, ISO/IEC JTC1/SC29/WG1 N1684 (2000).
- [4] *JPEG 2000 Requirements and Profiles*, ISO/IEC JTC1/SC29/WG1 N1271 (1999).
- [5] Christopoulos, C., Skodras, A. and Ebrahimi, T., "The JPEG2000 Still Image Coding System: An Overview," *IEEE Transactions on Consumer Electronics*, Vol. 46, pp. 1103–1127 (2000).
- [6] Taubman, D., "High Performance Scalable Image Compression with EBCOT," *IEEE Transactions on Image Processing*, Vol. 9, pp. 1158–1170 (2000).
- [7] Taubman, D., Ordentlich, E., Weinberger, M., Seroussi, G., Ueno, I. and Ono, F., "Embedded Block Coding in JPEG2000," *HPL-2001-35*, HP Labs, Palo Alto, Feb. (2001).
- [8] Taubman, D., Ordentlich, E., Weinberger, M., Seroussi, G., Ueno, I. and Ono, F., "Embedded Block Coding in JPEG2000," *Proc. IEEE Int. Conf. on Image Processing*, Vol. 2, pp. 33–36 (2000).
- [9] Adams, M. D. and Kossentini, F., "Jasper: A Software-Based JPEG-2000 Codec Implementation," *Proc. IEEE Int. Conf. Image Processing*, Vol. 2, pp. 53–56 (2000).
- [10] Lian, C. J., Chen, K. F., Chen, H. H. and Chen, L. G., "Analysis and Architecture Design of Block-Coding Engine for EBCOT in JPEG2000," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 13, pp. 219–230 (2003).
- [11] Chiang, J. S., Lin, Y. S. and Hsieh, C. Y., "Efficient Pass-Parallel Architecture for EBCOT in JPEG 2000," *IEEE International Symposium on Circuits and Systems*, Vol. 1, pp. 773–776 (2002).
- [12] Fang, H. C., Chang, Y. W., Wang, T. C., Lian, C. J. and Chen, L. G., "Parallel Embedded Block Coding Architecture for JPEG2000," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 15, pp. 1086–1097 (2005).
- [13] Li, Y. and Bayoumi, M., "A Three-Level Parallel High-Speed Low-Power Architecture for EBCOT of JPEG2000," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 16, pp. 1153–1163 (2006).
- [14] Chen, C. C., Chang, Y. W., Fang, H. C. and Chen, L. G., "Analysis of Scalable Architecture for the Embedded Block Coding in JPEG2000," *IEEE International Symposium on Circuits and Systems*, pp. 2609–2612 (2006).
- [15] Hsiao, Y. T., Lin, H. D. and Jen, C. W., "High-Speed Memory-Saving Architecture for the Embedded Block Coding in JPEG2000," *IEEE International Symposium on Circuits and Systems*, Vol. 5, pp. 133–136 (2002).
- [16] Zhang, Y. Z., Wu, C., Wang, W. T. and Chen, L. B., "Performance Analysis and Architecture Design for Parallel EBCOT Encoder of JPEG2000," *IEEE Transaction on Circuits and Systems for Video Technology*, Vol. 17, pp. 1336–1347 (2007).

Manuscript Received: Aug. 27, 2008

Accepted: Jun. 3, 2009