Concurrent Bit-Plane Coding Architecture for EBCOT in JPEG2000

Jen-Shiun Chiang, Chang-Yo Hsieh, Jin-Chan Liu, and Cheng-Chih Chien
Department of Electrical Engineering, VLSI Lab.
Tamkang University
Tamsui, Taipei, Taiwan
E-mail: {chiang, cyhsieh, jcliu, chien}@ee.tku.edu.tw

***Abstract*—This work presents a concurrent bit-plane coding architecture for EBCOT of JPEG2000. The architecture uses two bit-planes at the same time to encode data and this scheme can reduce the requirement of internal memory efficiently. Compared with the conventional approach, our concurrent architecture can save 8K-bit internal memory. In our proposed architecture, it can process data as long as the data of the two bit-planes are available, and at the same time the system can keep reading data from the external memory. This approach can increase the computation efficiency and avoid the waiting time for reading external data. It can also reduce the access times of the internal memory. Compared with the conventional context modeling architecture, the proposed concurrent bit-plane coding architecture can reduce the computation time by more than 50%.**

I. INTRODUCTION

JPEG2000 is the latest still image compression standard developed by ISO/IEC JTC1/SC29/WG1 [1]. It is the advanced version of JPEG and improves a lot of features in low bit-rate image compression. Furthermore it has more novel features such as lossy and lossless compression, progressive image transmission by quality or resolution, region of interest coding, and good error resilience [2].

JPEG2000 is consisted of discrete wavelet transform (DWT), scalar quantization, context modeling, binary arithmetic coding, and post-compression rate allocation. The main scheme of the context modeling is implemented by the embedded block coding with optimized truncation (EBCOT) technique, which provides a rich set of features, such as scalable resolution and SNR with a random access property [3]. However, the EBCOT takes more than 50% of encoding time in a software-based JPEG2000 implementation system [4] [5]. Since the EBCOT consumes a lot of computation time, many researches focus on this module to improve the efficiency. Among them, sample skipping (SS) and group-of-column skipping (GOCS) methods focus on skipping the many unnecessary coded samples and columns [5]. Taubman et al. proposed a concurrent symbol processing at fractional bit-plane coding [9]. On the other hand, EBCOT requires 20K-bit internal memory [8], and this amount of memory requirements is quite large when considering ASIC implementation. Therefore the pass-parallel architecture is proposed to improve the computation performance and can reduce 4K-bit internal memory requirements [6] [7]. In order to further improve the performance efficiency, we propose a concurrent bit-plane coding architecture for EBCOT. This work adopts the technique of parallelism and can process two bit-planes a time. The proposed concurrent bit-plane coding architecture can not only increase the computation performances, but also it can save 8K-bit internal memory. The memory access times are reduced and it can reduce the computation time by more than 50%.

II. EBCOT ALGORITHM AND SPEEDUP METHODS

The block diagram of the JPEG2000 encoder is shown in Figure 1. The discrete wavelet transform and the scalar quantization are first applied for the input image data. The quantized transform coefficients are then entropy coded by using context modeling and adaptive binary arithmetic coding. Finally, the compressed data is organized into a feature-rich code-stream by using post-compression rate-distortion optimization algorithm [2]. The key algorithms of the entropy coding involved in this paper are described in the following subsections.

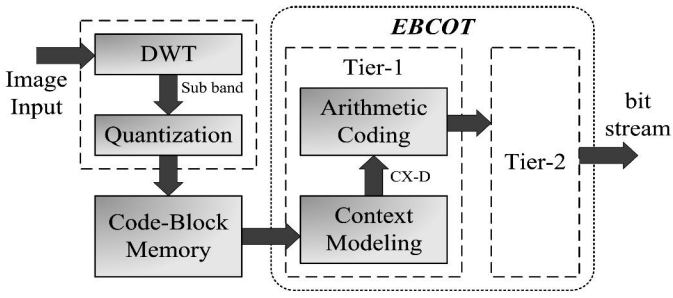

Figure 1. Block diagram of the JPEG2000 encoder

A. Context Modeling

After the transformation and quantization steps are performed, each sub-band is partitioned into rectangular blocks (called code-blocks), typically 64×64 or 32×32 in dimension. In the context modeling module, all quantized transform coefficients of the code-blocks are expressed in sign-magnitude representation and divided into one sign bit-plane and several magnitude bit-planes. Each bit-plane in a code-block can be divided into several stripes, and each stripe is composed of four row samples. In order to improve the embedding of the compressed bit-stream, each bit-plane is coded in three coding passes. Each sample in a bit-plane is coded in only one of the three coding passes and skipped in the other two passes. The three coding passes are: significant propagation (Pass 1), magnitude refinement (Pass 2), and cleanup (Pass 3). The block coding algorithm is composed of four coding primitives, and they are zero coding (ZC), sign coding (SC), magnitude refinement (MR), and run-length coding (RLC). The more detail about the context modeling algorithm can be found in [1] and [8].

B. Adaptive Binary Arithmetic Coding

The MQ coder is an adaptive binary arithmetic coder with renormalization-driven probability estimation. To reduce complexity, there are only 18 contexts used in JPEG2000, and each coding context is represented by 5 bits of the state information. Since the core of the MQ coder is adaptive in nature, the content of the selected context is updated based on the probability estimation process defined in JPEG2000 whenever a renormalization occurs. A byte of compressed data is removed and outputted from the high order bits of the code register C periodically to keep C from overflowing. When all of the symbols have been encoded, the FLUSH procedure is executed to terminate the encoding operations and generate the required terminating marker. Several bytes are also generated in the FLUSH procedure.

C. Speedup Method

In the EBCOT coder, each sample in a bit-plane is coded in only one of three coding passes and skipped in the other two passes. Therefore, clock cycles may be wasted in processing sample locations that do not belong to the current coding pass [5] [6]. Obviously, a large number of clock cycles may be wasted if the straightforward method is used. Many literatures focus on this topic, such as the sample skipping (SS) and group-of-column skipping (GOCS) [5] approaches try to skip the unnecessary code samples and columns, and the concurrent symbol processing [9] uses the idea of parallel encoding of a column in each coding pass. The key idea of the SS method is to skip unnecessary code samples in a single column. The SS is more efficient compared with the straightforward method, but a clock cycle is still wasted when a stripe column is "empty", which means that none of the samples of the stripe column belongs to the current coding pass. Therefore, the second speedup method, GOCS, is designed to further improve the processing speed. It skips a group of "empty columns" simultaneously at the cost of an extra GOCS memory. Besides, the number of columns in a group is a compromise between processing speed and area cost. The third speedup method, concurrent symbol processing, uses stripe-column concurrent encoding at each coding pass. This approach can increase the speed of producing CX-D pairs in a clock cycle. Because this method has more generated numbers of CX-D pairs in a clock cycle, it needs buffers between the context modeling and arithmetic coder. On the other hand, the concurrent symbol processing approach encodes data in each coding pass, and therefore it may encounter wasted computation time in many unnecessary coding stripe columns.

D. Pass Parallel Context Modeling

The pass-parallel architecture was used in PPCM and PCCM approaches [6] [7]. Because the inefficiency of the context modeling of EBCOT, the PPCM can increase the efficiency by merging the three coding passes to a single one. Moreover, the PCCM can encode a column in each stripe concurrently. PPCM and PCCM require four blocks of memory and each block takes 4K-bit. These four blocks are classified as χ (records all signs of samples in a bit-plane), ν_P (records all magnitudes of samples in a bit-plane), σ_0 (records the significance of Pass 1), and σ_1 (records the significance of Pass 3) respectively. The refinement memory can be replaced by $\sigma_0 \oplus \sigma_1$, where $\oplus$ is the logical exclusive-or operation. Therefore, the memory requirement of PPCM and PCCM are 4K bits less than that of the conventional design. Since the PPCM merges the three coding passes to a single pass, it encounters two problems. One is that the coded sample belonged to Pass 3 may become significant earlier than Pass 1. The other is how to predict neighbor significances of the coded samples that are belonged to Pass 1, Pass 2, and Pass 3 respectively. The PPCM proposed two methods to solve the first problem. Firstly it uses two memory blocks σ_0 and σ_1 to record the significances of Pass 1 and Pass 3, and then it delays the Pass 3 coding one stripe column. For the second approach, it uses Table I to predict the neighbor significances. Besides, it uses "stripe causal" mode of JPEG2000 [1] to break the correlation between the current stripe and next stripe. By using these techniques, all samples in each column can be encoded one by one efficiently.

III. PROPOSED ARCHITECTURE

Based on the pass-parallel architecture, we propose a concurrent bit-plane coding architecture that can reduce two 4K-bit coding state memories (bit-plane relationship variable $\pi[k]$ and refinement state variable $\gamma[k]$). The block diagram

TABLE I. THE PREDICTED TECHNIQUE FOR THREE PASS TYPES

Pass Type	Significant Prediction	
	Coded samples	*Un-coded samples*
Pass 1	$\sigma_0[k]$	$\sigma_0[k] \parallel \sigma_1[k]$
Pass 2	$\sigma_0[k]$	$\sigma_0[k] \parallel \sigma_1[k] \parallel \nu_P[k]$
Pass 3	$\sigma_0[k] \parallel \sigma_1[k]$	$\sigma_0[k] \parallel \sigma_1[k]$

"||": OR logic operation, k: location of the coded sample

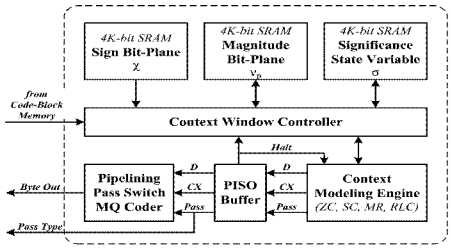

Figure 2. The block diagram of the proposed context modeling

of the proposed context modeling is shown in Figure 2. The proposed architecture requires three 4K-bit internal memories $\chi[k]$, $\nu_P[k]$, and $\sigma[k]$. The context modeling engine performs coding operations (ZC, SC, MR, and RLC) according to the information provided by the context window controller. During the encoding process, the context window controller reads the current magnitude bit-plane data from the internal memory and the next magnitude bit-plane data from the external code-block memory at the double-deck bit-plane data. In order to improve the efficiency of the whole system, we use the pipelined pass switching arithmetic encoder (PSAE) [6] [7] in our encoder. The parallel context modeling may generate a large amount of CX-D pairs at the same time [7]. In order to prevent the overflow of a large amount of CX-D pairs, a group of parallel-in-serial-out (PISO) buffers should be included in front of the PSAE, and the PISO buffer may send out "halt" signal for context modeling when overflow occurs in the system.

A. Concurrent Bit-Plane Coding

In order to increase the system computation efficiency and save internal memory requirement, we propose a new context modeling architecture that can encode two bit-planes concurrently. In the two bit-planes, the proposed architecture encodes the first bit-plane with the coding procedures belonged to Pass 1 and Pass 3 and at the same time encodes the second bit-plane with the coding procedure belonged to Pass 2. In context modeling, when Pass 1 and Pass 3 encode ZC and SC, the next encoding of the bit-plane must belong to the first refinement, and the first refinement memory can be omitted. Totally the new architecture needs only 12K-bit internal memories. Compared with the conventional approach, the concurrent bit-plane coding architecture can save 8K-bit memories. The concurrent bit-plane coding procedure is showed in Figure 3. In Figure 3, Pass 1 and Pass 3 are both encoded into the 5^{th} bit-plane in a code-block at present, and at the same time Pass 2 is encoded into the 4^{th} bit-plane at the same column. As Pass 1 and Pass 3 finish the encoding in the 5^{th} bit-plane and Pass 2 of the 4^{th} bit-plane finishes the encoding, the encoding of Pass 1 and Pass 3 goes to the 4^{th} bit-plane and the encoding of Pass 2 goes to the 3^{rd} bit-plane to process the encoding. The concurrent bit-plane encoder repeats this procedure until the encoding of Pass 1 and Pass 3 finishes the 0^{th} bit-plane.

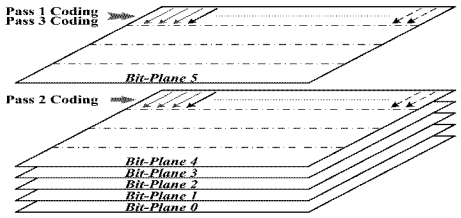

Figure 3. Concurrent bit-plane coding

The proposed architecture is based on techniques of pass-parallel and concurrent bit-plane coding, and the orders of the three fractional bit-plane coding are no longer one by one but are mixed instead. Therefore the prediction method in the significance parameter must be revised to keep encoding the result correctly. In order to maintain the correctness of the code in the pass-parallel architecture, it adopts the approach of delaying one column for the operation of Pass 3 [6] [7]. For concurrent bit-plane coding the architecture of PPCM must be revised. Here we encode three fractional bit-planes coding at the same time. To keep the correctness the significance state value prediction method of the neighbor samples must be revised in the code procedure of Pass 3. Here we introduce the idea of "virtual significance" $\sigma_v[k]$. $\sigma_v[k]$ is the significance state of the code procedure of Pass 3 after the encoding procedures of Pass 1 and Pass 2. Fig. 4 shows the calculation circuit of $\sigma_v[k]$.

The coefficient of Pass 1 can be calculated according to its code of the conditions mentioned in [1], and can be expressed as follows:

$$C_{P1}[X] = !\sigma[X] \;\&\&\; \sigma_N[X] \tag{1}$$

Where $\sigma_N[X]$ is calculated by the neighbors significance state value for location X, which is shown in Figure 4. The coefficients for Pass 2 and Pass 3 can be calculated by equations (2) and (3) respectively.

$$C_{P2}[X] = \sigma[X] \,||\, \chi[X] \tag{2}$$

$$C_{P3}[X] = !C_{P1}[X] \;\&\&\; !\sigma_v[X] \tag{3}$$

After we finish the three-coding-pass detection, each coefficient can be encoded then. However, each coding pass needs using of neighbors significance values for ZC, SC, MR, and RLC. In order to predict the significance state value of the neighbor samples correctly, it must divide the neighbor

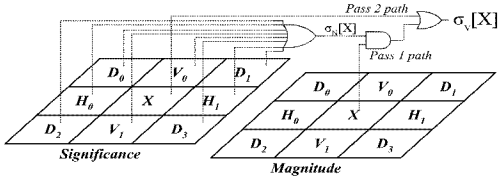

Figure 4. Virtual significance predicts of location X ($\sigma_N[k]$ neighbors significance for location X)

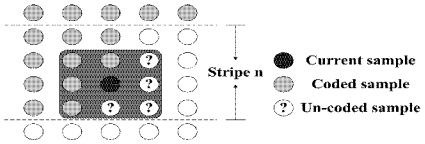

Figure 5. An example of the location of the predicted sample

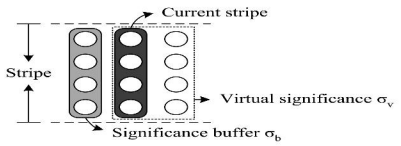

Figure 6. Virtual significance σ_v and signifcance buffer σ_b for Pass 3

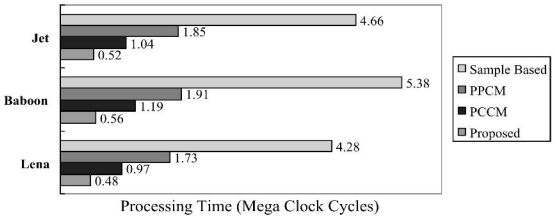

Figure 7. Processing time compared for proposed and convention scheme

samples of the context window into coded samples or un-coded samples. As shown in Figure 5 the coded samples are coded before the current sample, and the un-coded samples are coded after the current sample.

The three encoded procedures need neighbor samples significance state values while encoding, which have different prediction methods according to the coded and un-coded samples. For Pass 1 encoding, the coded or un-coded samples use $\sigma[k]$ to finish the encoding. The significance state value $\sigma[k]$ obtained after Pass 3 cannot be used directly. Because it may cause the significance state value mistake of the neighbor samples of Pass 1, and will generate wrong encoding result at Pass 1 and Pass 2 coding. In order to solve this problem, a 4-bit significance state buffer $\sigma_b[k]$ is used. The significance state value obtained after Pass 3 encoding must be stored in buffer $\sigma_b[k]$ first, as shown in Figure 6. The value of $\sigma_b[k]$ is logic "OR" operated with the significance state value obtained after Pass 1 coding and the result is written back to the significance state memory $\sigma[k]$. Therefore the significance state value of the un-coded samples of Pass 3 is $\sigma_v[k]$, and the significance state value of the coded samples are $\sigma_b[k]$.

The significance state value can be calculated by equations (4) and (5) for Pass 2 coding. Equation (4) is the significance state value of the un-coded sample, and (5) is the significance state value of the coded sample. Finally, the significance state values of the three coding pass prediction is summarized in Table II.

$$\sigma_{2u}[k] = \sigma[k] \parallel \nu_P[k] \tag{4}$$

$$\sigma_{2c}[k] = \sigma[k] \parallel \sigma_b[k] \tag{5}$$

TABLE II. THE PREDICTED TECHNIQUE FOR THE THREE PASS TYPES

Pass Type	Significant Prediction		
	Current sample	*Coded samples*	*Un-coded samples*
Pass 1	$\sigma[k]$	$\sigma[k]$	$\sigma[k]$
Pass 2	$\sigma[k] \parallel \nu_P[k]$	$\sigma[k] \parallel \sigma_b[k]$	$\sigma[k] \parallel \nu_P[k]$
Pass 3	$\sigma_v[k]$	$\sigma_b[k]$	$\sigma_v[k]$

"||": OR logic operation. k: location of the coded sample

B. *Pipelined Pass Switch Arithmetic Encoder*

In order to increase the performance of the MQ coder, we use a pipelined architecture to divide all the coding procedures into four stages. This architecture is proposed in [7]. In pass-parallel technique the CX-D data streams sent into the MQ coder from our concurrent bit-plane coding architecture are interleaved. Therefore more context registers and coding state registers are used. Based on these phenomena the pipelined MQ coder proposed in [7] is modified to fit for this concurrent bit-plane coding architecture.

IV. EXPERIMENTAL RESULTS

We use three different images, Jet, Baboon, and Lena, which are all with size 512×512 as the test images. These three images are processed respectively by using the proposed concurrent bit-plane coding architecture, PCCM [7], PPCM [6], and sample based architecture [3]. The processing times in mega clock cycles are shown in Figure 7. The proposed architecture reduces the computation time by more than 50%.

V. CONCLUSION

In this paper we propose a concurrent bit-plane coding architecture to save 8K-bit internal memory for EBCOT in JPEG2000. Moreover, the non-delay pass-parallel architecture is proposed, and it can increase computation efficiently. Compared with conventional architectures, the proposed concurrent bit-plane coding architecture can reduce the computation time by more than 50%.

REFERENCES

[1] JPEG 2000 Part I Final Committee Draft Version 1.0, *ISO/IEC JTC1/SC29/WG1 N1646R*, Mar. 2000.

[2] A. Skodras, C. Christopoulos, and T. Ebrahimi, "The JPEG 2000 still image compression standard," *IEEE Signal Processing Mag.*, vol. 18, pp. 36-58, Sept. 2001.

[3] D. Taubman, "High performance scalable image compression with EBCOT," *IEEE Trans. Image Processing*, vol. 9, no. 7, pp. 1158-1170, July 2000.

[4] M. D. Adams and F. Kossentini, "JasPer: a software-based JPEG-2000 codec implementtion," *IEEE Int. Conf. Image Processing*, vol. 2 pp. 53-56, Sep. 2000.

[5] C. J. Lian, K. F. Chen, H. H. Chen, and L. G. Chen, "Analysis and architecture design of block-coding engine for EBCOT in JPEG2000," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 13, pp.219-230, March 2003.

[6] J. S. Chiang, Y. S. Lin, and C. Y. Hsieh, "Efficient pass-parallel architecture for EBCOT in JPEG2000," *IEEE Int. Symp. Circuits and System*, vol. I, pp. 773-776, May 2002.

[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 Int. Symp. Circuits and System*, vol. III, pp. 865-868, May 2004.

[8] D. Taubman, E. Ordentlich, M. Weinberqer, G. Seroussi, I. Ueno, and F. Ono, "Embedded block coding in JPEG2000," *IEEE Int. Conf. Image Processing*, vol. 2, pp. 33-36, Sept. 2000.

[9] A. K. Gupta, S. Nooshabadi, and D. Taubman, "Efficient VLSI architecture for buffer used in EBCOT of JPEG2000 encoder," *IEEE Int. Symp. Circuits and System*, pp. 4361-4364, May 2005.