Word-Level Parallel Architecture of JPEG 2000 Embedded Block Coding Decoder

Yu-Wei Chang, Hung-Chi Fang, Chun-Chia Chen, Chung-Jr Lian, and Liang-Gee Chen, *Fellow, IEEE*

***Abstract*—This paper presents a word-level decoding architecture of embedded block coding in JPEG 2000. This architecture decodes one coefficient per cycle based on the proposed word-level decoding algorithm. This algorithm eliminates state variable memories by decoding all bit-planes in parallel. The proposed column-switching scan order overcomes intra bit-plane dependency and inter bit-plane dependency to enable parallel processing. Implementation results show that the proposed architecture is capable of decoding 54 MSamples/s at 54 MHz, which can support HDTV 720p (1280 × 720, 4:2:2) decoding at 30 frames/s.**

***Index Terms*—Embedded block coding with optimized truncation, image compression, JPEG 2000.**

I. Introduction

JPEG 2000 [1]–[4] uses two key components, discrete wavelet transform (DWT) and embedded block coding with optimized truncation (EBCOT), to achieve excellent coding efficiency and numerous features [5], such as region of interest (ROI) and various scalabilities. The scalabilities come from the multiple decomposition of the DWT and the embedded block coding (EBC) of the EBCOT.

The complexity of JPEG 2000 coding system is much higher than that of JPEG [6]. The EBC occupies 53% of total computation [7], which is the most critical part in JPEG 2000 coding system. Therefore, hardware implementation of the EBC is a must for real-time applications. Many EBC architectures [7]–[13] were proposed. All of them are bit-plane sequential architectures, which encode or decode a code-block from a higher bit-plane to a lower bit-plane sequentially. Besides, all of them require on-chip SRAM to store state variables, which indicate the coding states during bit-plane coding. The sequential processing makes high performance JPEG 2000 system for coding HD motion pictures impossible. To solve this problem, a word-level EBC architecture [14], [15] was proposed to encode one DWT coefficient per cycle. It increases the throughput of JPEG 2000 encoder dramatically and eliminates state variable memories by encoding all bit-planes in parallel. The parallel processing is enabled by looking one column of coefficients ahead to generate required state variables. Based on the word-level architecture, Lai [16] and Li [17] proposed coefficient parallel architectures to further improve the encoding throughput. However, all of these architectures can not be used to decode all bit-planes in parallel since the technique of looking ahead can not be used due to the unavailable values of the look-ahead coefficients. Therefore, the existing solution for the EBC decoder is bit-plane sequential architectures. The sequential architectures make high throughput JPEG 2000 decoder impossible.

The most critical problem to design a parallel decoding architecture is the data dependency in the EBC algorithm. The current sample can not be decoded without finish of decoding the previous sample. Neither looking ahead techniques [15]–[17] nor pass-parallel technique [9] can be used to enable parallel decoding. The most contribution of this paper [18] is that the first word-level EBC architecture for JPEG 2000 decoder is realized. This architecture can decode one coefficient per cycle regardless of coding bit-rate. Therefore, it is possible to decode nearly-lossless moving pictures while maintaining high throughput. The word-level architecture decodes all bit-planes in parallel based on the proposed word-level decoding algorithm. The proposed column-switching scan order, which is not different from that in our previous work [15], overcomes data dependency problems. Besides, the checking conditions for decoding algorithm and resulting parallel architectures are also different.

This paper is organized as follows. Section II reviews the EBC algorithm and the previous EBC architectures. Section III describes the proposed word-level EBC algorithm. The parallel EBC architecture based on the word-level algorithm is presented at Section IV. Implementation results and comparisons are shown in Section V. Finally, Section VI concludes this paper.

II. Preliminary

A. Embedded Block Coding Algorithm in JPEG 2000 Decoder

Embedded Block Coding (EBC) in JPEG 2000 decoder is composed of the Context Formation (CF) and the Arithmetic Decoder (AD), as shown in Fig. 1. The AD decodes a binary-valued decision D to reconstruct a corresponding sample bit by receiving a context (CX) and embedded bit streams.

1) Context Formation: The basic coding unit of the EBC is a code-block with typical size of 64×64 or 32×32. As shown in Fig. 2, the order of bit-plane decoding in a code-block is from the Most Significant Bit (MSB) bit-plane to the least significant bit (LSB) bit-plane. A $W \times W$ bit-plane is further divided into stripes and the size of each stripe is $4 \times W$. The scan order is

Manuscript received June 12, 2006; revised January 23, 2007. This work was supported in part by the National Science Council, R.O.C., under Grant 95-2752-E-002-008-PAE and in part by the MediaTek Fellowship. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Lap-Pui Chau.

The authors are with the DSP/IC Design Lab, Graduate Institute of Electronics Engineering and Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan, R.O.C. (e-mail: wayne@video.ee.ntu.edu.tw; honchi@video.ee.ntu.edu.tw; chunchia@video.ee.ntu.edu.tw; cjlian@video.ee.ntu.edu.tw; lgchen@video.ee.ntu.edu.tw).,

Digital Object Identifier 10.1109/TMM.2007.902822

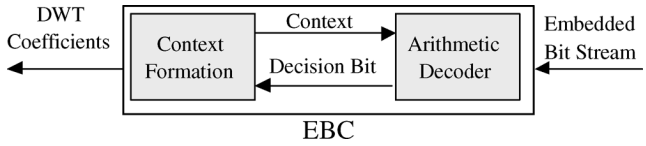


Fig. 1. Diagram of the EBCOT algorithm. It contains context formation (CF) and the arithmetic decoder (AD). The AD decodes a binary-valued decision D to reconstruct a corresponding sample bit by receiving a context (CX) and embedded bit streams.

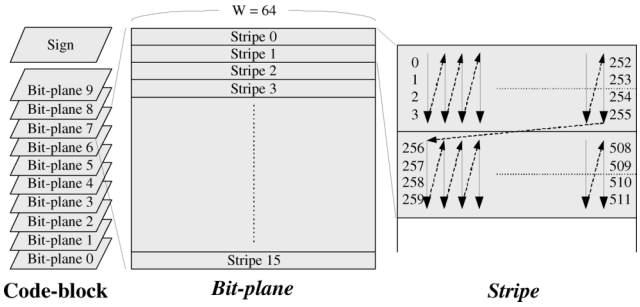


Fig. 2. Diagram of code-block and stripes. A 64×64 code-block is divided into sixteen 4×64 stripes. The numbers in the stripes represent the scan order.

stripe by stripe and column by column in a stripe. Each bit-plane is sequentially scanned by three coding passes: the significant propagation pass (Pass 1), the magnitude refinement pass (Pass 2), and the cleanup pass (Pass 3). The MSB bit-plane, which is an exception, requires only the Pass 3.

In each scan, a context window, as shown in Fig. 3, is involved while modeling the CX of a sample bit in a bit-plane. The sample bit to be decoded lies in the center of the context window and is denoted as C. The eight-connected neighbors of C are further divided into horizontal (H), vertical (V), and diagonal (D) groups according to their relative position to C, i.e., $H = \{h0, h1\}$, $V = \{v0, v1\}$, and $D = \{d0, d1, d2, d3\}$. For the CF, a binary state variable called significant state is defined for a coefficient to indicate whether or not a nonzero magnitude bit has been decoded in previous bit-planes or passes. Then, the coding pass of C is determined by the significant states of C itself and its neighbors. If C has been significant, it belongs to the Pass 2. If C is not significant but at least one of its neighbors is significant, i.e., the significant neighbors have significant contributions to C, it belongs to the Pass 1; otherwise, it belongs to the Pass 3.

There are five state variables used for the CF algorithm, the magnitude bit-plane, the sign bit-plane, the significant state, the visited state, and the first refinement state, and each of them costs 4 kilobits (kb) for the maximum 64×64 code-block size. The magnitude bit-plane and sign bit-plane store the magnitude and the sign of the decoded sample bits, respectively. The visited state indicates whether this sample was decoded or not, and the first refinement state indicates whether this sample, if it belongs to the Pass 2, is decoded by Pass 2 at the first time or not. Total memory requirement for state variables is 20 kb.

For each scanned sample bit, one or more contexts are generated to adapt the probability models of the AD. Total 19 CXs used by the EBC are classified into five categories, magnitude

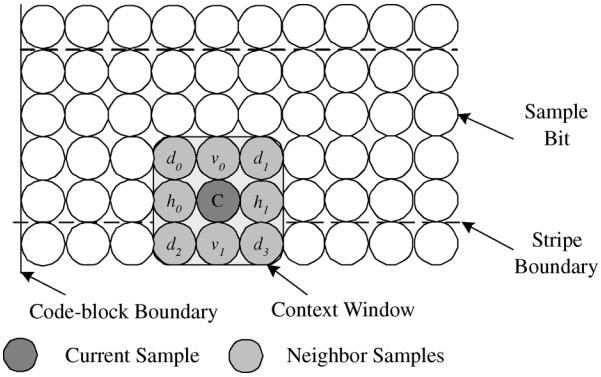


Fig. 3. Context window for context formation. The sample coefficient to be coded is referred as C. The eight neighbors of C are grouped as H, V and D.

coding, sign coding, magnitude refinement coding, run-length coding (RLC) and uniform coding. The CXs for magnitude coding and sign coding are used for the first nonzero sample bit of a coefficient. Two CXs are generated for this sample bit since the magnitude and sign value should be decoded. The CXs for magnitude refinement coding are used for the significant sample bit, i.e., these CXs are used only for the Pass 2. To increase coding efficiency, the RLC CX is used for a special case that all surrounding sample bits of a column are insignificant, and all sample bits in this column are not decoded. If at least one sample bit in this column is nonzero, the decoded bit by use of the RLC CX is also nonzero. Two more uniform CXs plus one sign CX are used to decode the position of the first nonzero sample bit in this column and the sign value of this sample. The RLC CX and uniform CX are used for the Pass 3 only since the condition for RLC are never satisfied during Pass 1 coding. Detailed information for the context mapping is described in [19].

2) Arithmetic Decoder: The AD is an adaptive, multiplication-free, binary MQ coder. It is derived from the Q coder [20] and enhanced by a conditional exchange procedure derived from the MELCODE [21] and the state-transition table known as JPEG-FA [22]. The probability tables are predetermined and provided by the standard.

B. Causal Context and Coding Pass Termination

JPEG 2000 defines the causal context mode which is that the samples in the next stripe are considered as insignificant samples. As shown in Fig. 3, the d_2, v_1, and d_3 are located in the next stripe, and therefore, these samples are considered as insignificant samples no matter what values they are. This mode breaks the dependency from the next stripe.

There are two termination modes for the embedded bit-streams and two reset modes for the arithmetic coder. For the two termination modes, one is to terminate bit-streams at the end of each coding pass, and the other is to terminate them at the end of each code-block. For the two reset modes, one is to initialize the adaptive probability of the arithmetic coder at the start of each coding pass, and the other is to initialize it only at the start of each code-block. By combining the termination and initialization at each coding pass, the dependency between embedded bit-streams are broken.

For simplicity, when causal context mode, pass termination, and pass initialization are enabled, it is called parallel coding mode, otherwise, it is called sequential coding mode. The average peak signal-to-noise ratio (PSNR) loss of the parallel coding mode compared to the sequential coding mode is about 0.25 *dB* in medium to high bit-rate and 0.1 *dB* in low bit-rate [15]. In [19], it shows that the loss is about 0.15 *dB* for 64×64 code-block and 0.35 *dB* for 32×32 code-block at medium bit-rate.

C. Previous EBC Architectures

In this section, we introduce some typical EBC architectures. Lian *et al.* [7] proposed an EBC architecture, which realized the sequential coding mode. Three scans are needed to encode one bit-plane, except for the MSB bit-plane, which needs only one scan. Three speed-up techniques were proposed to reduce the processing time. With these techniques, the number of processing cycles are reduced to 38% in average. Therefore, it requires $1.3 \times NW^2$ clock cycles to encode a code-block with size of $W \times W$ and N nonzero bit-planes. The three scans are required due to several reasons. First, the causal context mode is not enabled in the sequential coding mode, and the sample bits which belong to Pass 1 in the first row of the next stripe would have significant contributions to the samples in the last row of the current stripe. Therefore, Pass 1 should be scanned first. Second, the adaptive probability of the arithmetic coder is not initialized at the start of each coding pass. Three scans should be operated sequentially since the initial probability of the current coding pass should be the final probability of the previous coding pass. The memory requirement for this architecture is 21 kb, in which 20 kb and 1 kb are for state variables and the proposed techniques, respectively. This architecture can realize decoding function if the AE is replaced with AD [23].

Hsiao *et al.* [8] reduces the memory requirement of the state variables in the EBC by exploring the dependency among these state variables. This technique reduces the memory requirement by 20% at the cost of a few logic gates. An architecture for the AE was also proposed in [8], using three pipeline stages in normal operation and four pipeline stages when the byteout is triggered. However, when the AE is in the byteout stage, it cannot process the context decision (CXD) pair from the block coder. At this cycle, the block coder must be stalled.

Chiang *et al.* [9] proposed a pass-parallel EBC architecture, which realizes parallel coding mode. This architecture uses two context windows concurrently, one for the Pass 1 and the Pass 2, and the other for the Pass 3, to process a bit-plane within one scan. Therefore, the processing cycles for a code-block are NW^2. The memory requirement of the state variables is reduced by 4 kb due to two parallel context windows. To handle three independent embedded bit-streams for three coding passes, the pass-switching AE (PSAE), which is composed of one processing element (PE) and three suits of coding status registers, was proposed. The pass-parallel architecture is enabled due to the parallel coding mode. Although this architecture is designed for the encoder, it also can be used for the decoder if the AE is replaced with AD.

All of the above architectures have limited performance since all bit-planes are processed sequentially. To overcome this obstacle, Fang *et al.* [14], [15] proposed a word-level EBC architecture, which encodes all bit-planes in parallel. The memory requirement for the state variables is eliminated due to the parallel algorithm. This EBC architectures contains parallel CF (PCF), reconfigurable FIFO (first-in-first-out), and folded AE (FAE). The PCF scans one DWT coefficient per cycle and generates CX and decision (CXD) pairs. The FIFO receives the CXD pairs and sends to the FAE. The FAE is an extended architecture of the PSAE by folding two bit-planes into one AE module. When the FIFO is full, the PCF must be stalled until the FIFO is available. Because of the limited throughput of FAE and the limited length of FIFO, the effective throughput of the EBC is $1.5\,\mathrm{NW}^2$, which is 33% degradation from the ideal value NW^2. The word-level architecture is enabled by looking one column of coefficients ahead from the current context window. Although this architecture is a parallel architecture, it can not be used to decode all bit-planes in parallel due to two critical reasons. First, the technique of looking one column of coefficients ahead can not be used since these coefficients are unavailable during the decoding process. Second, unlike only forward path between CF and AE in the encoder, there is a feedback path between CF and AD in the decoder. The next CX would depend on the previous D. Therefore, the FIFO architecture between CF and AD would degrade performance dramatically since the CF must be stalled frequently.

To further increase the performance for the EBC, the coefficients parallel architectures [16], [17] were proposed to encode four coefficients in a column. However, these architectures can not be used to decode multiple coefficients due to the same reasons described in the previous paragraph.

As can be seen, there is a huge gap between the encoding throughput and decoding throughput for the EBC. In this paper, we propose a word-level EBC architecture to decode a DWT coefficient per cycle.

III. Parallel EBC Decoding Algorithm

In this section, we propose a word-level EBC algorithm for decoding. By use of this algorithm, the EBC decodes one coefficient per cycle regardless of numbers of bit-planes. All state variables are generated on-the-fly by using parallel algorithm. Moreover, the throughput is significantly increased due to parallel processing. For the proposed algorithm, causal context and pass termination, which are defined as parallel mode in the JPEG 20000 standard, are used.

A. Column-Switching Scan Order

There are two data dependency problems for the EBC decoding algorithm. One is intra bit-plane dependency and the other is inter bit-plane dependency. As shown in Fig. 3, the coding pass and the CX of C depend on the decoded values of the eight surrounding neighbors in the same bit-plane, which is called intra bit-plane dependency, and depend on the decoded values of eight surrounding neighbors in the upper bit-planes, which is called inter bit-plane dependency.

We propose a column-switching scan order to solve above two dependency problems. An example of the scan order in a bit-plane, k, is illustrated with Fig. 4. The numbers in the circle present the decoding orders. There are two subscans, Pass 1 decoding scan in a column and non-Pass 1 (Pass 2 and Pass 3)

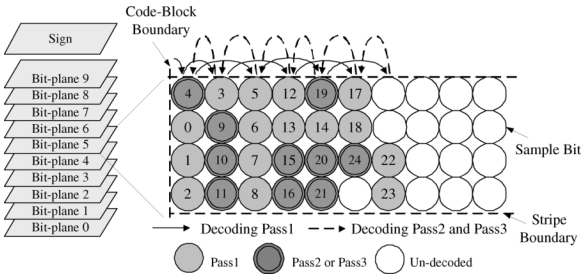


Fig. 4. Proposed column-switch scan order in a bit-plane. The numbers is the precessing cycle. There are two subscans, Pass 1 scan and non-Pass 1 scan. The Pass 1 scan is one column ahead of the non-Pass 1 subscan.

decoding scan in a column. The sample bits are decoded one column by one column in a column-switching manner. Note that the Pass 1 decoding scan precedes the non-Pass 1 decoding scan by one column to solve intra bit-plane dependency. The reason of one column ahead for Pass 1 scan is that nonzero samples which belong to Pass 1 have contributions to the samples which belong to Pass 2 or Pass 3.

In each subscan, only the samples to be decoded are visited and each visited sample requires one processing cycle. It is easy to check which sample should be decoded in the respective subscan. For the Pass 1 subscan, if none of the samples in a column is decoded, the first step is to check whether all the surrounding samples of this column are insignificant. If so, this column is skipped and the non-Pass 1 subscan is started; otherwise, it is to determine the location of the first Pass 1 sample and then to decode it. If at least one of sample is decoded at this column, it is to determine whether there is any Pass 1 sample to be decoded. If not, the non-Pass 1 subscan is started, otherwise, the Pass 1 subscan is continued. For the non-Pass 1 subscan, if the first non-decoded sample is significant, it belongs to Pass 2, otherwise, it belongs to Pass 3. All the conditional decisions for coding pass are finished in one cycle. Therefore, exact one sample is decoded in a cycle and no cycle is wasted for just making decisions but not decoding sample. Since exact one cycle to decode a sample is guaranteed, the numbers of processing cycles needed to decode a bit-plane are equal to the numbers of sample bits in this bit-plane.

For the inter bit-plane dependency problem, it is solved by four columns scan latency between two successive bit-planes, i.e., the bit-plane $(k-1)$ starts to scan when the bit-plane k starts to scan the 4th column. Fig. 5 illustrates a critical example of the nearest distance of two context windows between two successive bit-planes. The number in a circle indicates the order of the decoding cycle except -1 indicates the initial condition. The bit-plane $(k-1)$ starts to scan at the moment that the bit-plane k starts to scan the 4th column at the 14th decoding cycle. The column -1 in the bit-plane $(k-1)$ is initialized with two Pass 1 samples since there are two Pass 1 samples at the 3rd column in the bit-plane k. The nearest distance of two context windows is happened at 36th and 37th decoding cycle. The 7th column is overlapped by two context windows, and all sample bits of this column in the bit-plane k are available since they have been visited by two subscans. Therefore, inter bit-plane dependency problem is solved.

The moving speed of context window in the bit-plane k is slowed since there are four Pass 1 samples at the ninth column while the moving speed of the context window in the bit-plane $(k-1)$ is accelerated since no Pass 1 sample at the fifth column. After the finish of the scan of the 8th column and the sixth column in the bit-plane k and the bit-planes $(k-1)$, respectively, the moving speed of the context window in the bit-plane k is accelerated while that in the bit-plane $(k-1)$ is slowed. This phenomenon is called moving jitter. If there is enough column spacing between two successive context windows, the moving jitter does not cause data conflict. Therefore, three columns spacing (four columns latency) between two successive context windows is used to solve moving jitter problem. In three columns latency, two are considered for that the moving speed of context windows in bit-plane $(k-1)$ is one column faster and that in bit-plane k is one column slower. The remained one is to make sure that at most one column is overlapped by two context windows. Therefore, three columns are the minimum latency between two successive bit-planes.

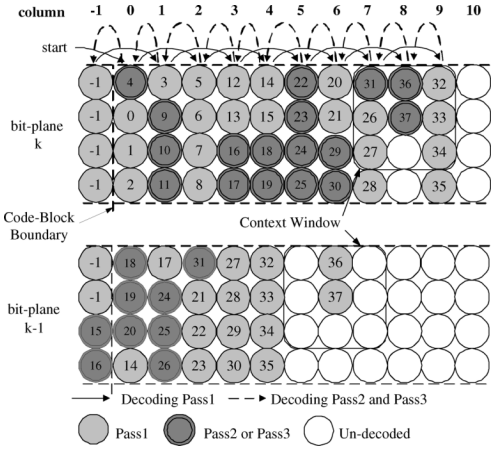


Fig. 5. Example of column-switch scan order between two successive bit-planes. The numbers present the precessing cycle. The bit-plane $(k-1)$ starts to scan at the moment that the bit-plane k starts to scan the 4th column at the 14th decoding cycle. There are three columns spacing (four column latency) between two successive bit-planes.

To enable parallel decoding, all bit-planes in a code-block are scanned with the column-switching manner described above. Although all context windows in all bit-planes scan different sample bit for different coefficients, the overall throughput is one coefficient decoding per cycle since one sample bit is decoded per cycle in each bit-plane. Since there are three columns latency between two successive two bit-planes, the latency to decode a complete coefficient is $4 \times N$ columns, where N is the number of bit-planes in a code-block.

B. Coding Pass Classification

In this section, the coding pass classification algorithm is presented. Let d_c^k denote the binary value of the central sample bit (C) in the bit-plane k, and d_s^k denote the binary value of decoded bit of either one of the eight surrounding samples in the bit-plane k. The s is used to represent any neighbor of C, i.e., $s = \{h0, h1, v0, v1, d0, d1, d2, d3\}$ as shown in Fig. 2. The ranges of k are from $N-1$ to 0, in which zero represents the LSB bit-plane. In the following discussions, a superscript indicates the bit-plane number, and a suffix indicates the location in the context window.

The coding pass of C in the bit-plane k, p_c^k, is determined by the significant contributions of its eight neighbors at bit-plane k. The contribution of s to the k-th bit-plane of C is represented by ϕ_s^k. Note that when C is located on the last row in a column, ϕ_{d2}^k, ϕ_{d3}^k, and ϕ_{v1}^k are set to zero since the causal context mode is used. For s that is decoded before C, its contribution is determined by

$$\phi_s^k = \begin{cases} 1, & \hat{d}_s^k = 1 \\ 1, & (\hat{d}_s^k = 0)\&(d_s^k = 1)\&(p_s^k = 1) \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where $\hat{d}_s^k$ is

$$\hat{d}_s^k = \begin{cases} 0, & k = N-1 \\ d_s^{k+1} | \hat{d}_s^{k+1}, & 0 \le k < N-1 \end{cases}. \tag{2}$$

Note that p_s^k is available since s is decoded before C. For s that is decoded after C, its contribution is

$$\phi_s^k = \begin{cases} 1, & \hat{d}_s^k = 1 \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

According to the scan order described in Section III-A, whether s is decoded before C or not is determined by its location. If s is located at the previous stripe, i.e, d_0, d_1 and v_0 are in the previous stripe when C is in the first row of current-scanned stripe, it is decoded before C. For the case that C is not in the first row of current stripe, whether s is decoded before C or not depends on whether s is decoded or not during the column-switching scan order.

The coding pass p_c^k is determined by

$$p_c^k = \begin{cases} 2, & \hat{d}_c^k = 1 \\ 1, & (\hat{d}_c^k = 0)\&(\sum \phi_s^k \ge 1) \\ 3, & \text{otherwise} \end{cases} \tag{4}$$

where the range of $\sum \phi_s^k$ is from 0 to 8.

C. Context Formation

In this section, we propose a parallel CF algorithm, which calculates state variables on-the-fly. Therefore, no state variable memories are required. In the following sections, we elaborate the algorithms to obtain the CXs for magnitude coding, sign coding and magnitude refinement coding separately.

1) Magnitude Coding: The context mapping for the magnitude coding requires the group HVD. Let H^k, V^k, and D^k denotes the group H, V and D, respectively, in the k-th bit-plane of s relative to C. The group contributions are determined by

$$H^k = \sigma_{h0}^k + \sigma_{h1}^k = \sum_{i=0}^{1} \sigma_{hi}^k \tag{5}$$

$$V^k = \sigma_{v0}^k + \sigma_{v1}^k = \sum_{i=0}^{1} \sigma_{vi}^k \tag{6}$$

and

$$D^k = \sigma_{d0}^k + \sigma_{d1}^k + \sigma_{d2}^k + \sigma_{d3}^k = \sum_{i=0}^{3} \sigma_{di}^k. \tag{7}$$

TABLE I
TRUTH H^χ (V^χ) FOR SIGN CODING

σ_{h0}^k (ϕ_{v0}^k)	χ_{h0} (χ_{v0})	ϕ_{h1}^k (ϕ_{v1}^k)	χ_{h1} (χ_{v1})	H^χ (V^χ)
1	0	1	0	1
1	0	0	X	1
0	X	1	0	1
1	1	1	0	0
1	0	1	1	0
0	X	0	X	0
1	1	1	1	-1
1	1	0	X	-1
0	X	1	1	-1

X Means "Don't Care".

For s is decoded after C, the σ_s^k is defined as

$$\sigma_s^k = \begin{cases} 1, & \hat{d}_s^k = 1 \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

Otherwise, for s is decoded before C, the σ_s^k is defined as

$$\sigma_s^k = \begin{cases} 1, & \hat{d}_s^k = 1 \\ 0, & (\hat{d}_s^k = 0)\&(p_c^k \ne 3)\&(p_s^k \ne 1) \\ 1, & \text{otherwise.} \end{cases} \tag{9}$$

The context mapping are according to the value of H^k, V^k, and D^k.

2) Sign Coding: The context mapping for the sign coding requires the group data, H^χ, V^χ, and D^χ. Let χ_s denotes the sign of s, where "1" stands for a negative coefficient and truth table for H^χ and V^χ is shown in Table I.

3) Magnitude Refinement Coding: The context for magnitude refinement coding are determined by group contributions, H^k, V^k, and D^k, and an indicator, γ_c^k to indicate whether first refinement or not for C. The r_c^k is calculated by

$$\gamma_c^k = \begin{cases} 1, & (\hat{d}_c^{k+1} = 0)\&(d_c^{k+1} = 1) \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

D. Arithmetic Decoder

In the parallel mode, the probability tables are reset on each coding pass, and the embedded bit stream of each pass is terminated to separate it from other coding passes. Termination on each pass prevents error from propagating across passes and makes parallel EBC decoding possible.

IV. WORD-LEVEL EBC ARCHITECTURE

In this section, a word-level EBC architecture for decoder is proposed based on the word-level algorithm. The proposed architecture is shown in Fig. 6. It decodes ten magnitude bit-planes as well as sign bit-plane in parallel. There are three major functional blocks, parallel context formation (PCF), arithmetic decoder (AD), and magnitude register bank (MRB). The state register bank (SRB) stores the coding states of the AD, and the four-symbol arithmetic decoder (FAD) decodes four symbols, which are the maximum numbers of contexts generated by a CF, in a processing cycle. There is no pipeline stage between FAD and PCF. Therefore, one coefficient decoding per cycle is achieved. The dataflow in each CF is the column-switching scan order, as described in Section III-A, and the dataflow of

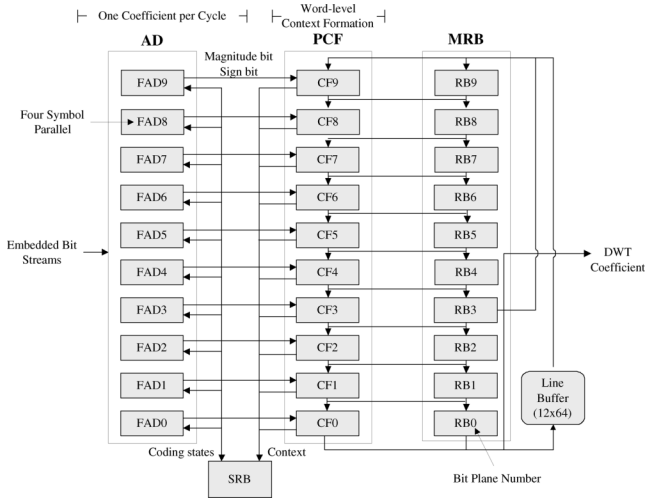


Fig. 6. Embedded block coding decoder architecture. four-symbol arithmetic decoder (FAD) process all generated contexts from parallel context formation (PCF) to guarantee 10-bit coefficient decoding per cycle. The state register bank stores the coding states of FAD. The decoded sample bits from PCF are merged into magnitude register bank (MRB). The line buffer memory stores the coefficients of the last row in the previous stripe.

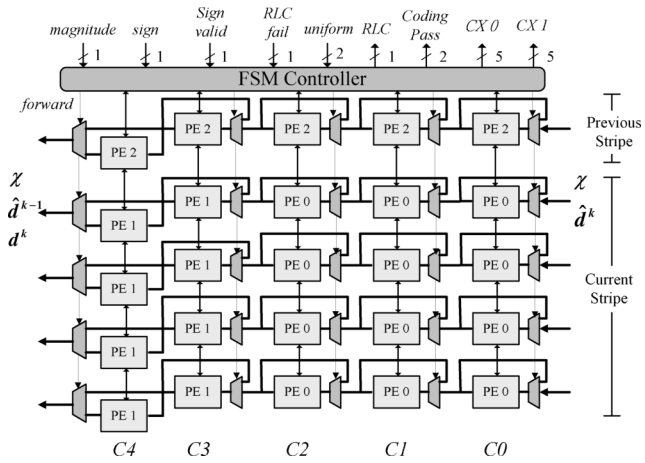


Fig. 7. Context formation architecture. The architecture of each PE is shown in Fig. 8. Each CF contains five columns of PE ($C0 \sim C4$). The *C0*, *C1*, *C2* and *C3* can be regarded as four columns latency between two successive bit-planes. The *C4* is used as a temporal buffer to buffer the output of *C3*.

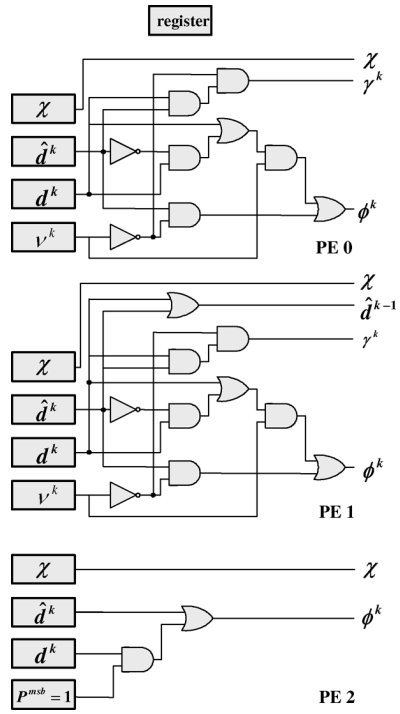


Fig. 8. Elementary processing element architecture in Fig. 7. Each PE generates the corresponding variables defined in Section III.

two successive CFs are separated by four columns. Therefore, the latency to decode a column is 40 columns (40×4 cycles). The 12×64-bit line buffer is used to buffer the decoded coefficients of the last row in the previous stripe. Each 12-bit word in the line buffer contains 11 bits for a coefficient and 1 bit for indicating whether the coding pass of the first significant bit of this coefficient is Pass 1 or not. There are two conditions for that the first significant bit of the last row of the previous stripe has significant contribution to the samples in the first row of the current stripe. One is that it has nonzero value and its coding pass is Pass 1, and the other is that it has nonzero value and its coding pass is Pass 3 as well as the coding pass of the sample in the first row of the current stripe is also Pass 3. Therefore, one bit is sufficient to indicate whether the coding pass of the first significant bit in the previous stripe is Pass 1 or not. The partial decoded coefficients from CF3 are feedbacked to CF9 to serve as the coefficients of the last row in the previous stripe for a code-block size 32×32 since the latency to decode a complete column is larger than 32 columns. The detailed architecture of each functional block will be elaborated in the following sections.

A. Context Formation

The detailed CF architecture is shown in Fig. 7, and the architecture of each PE is shown in Fig. 8. Each CF contains five columns of PE ($C0 \sim C4$). The *C0*, *C1*, *C2*, and *C3* can be regarded as four columns latency between two successive bit-planes. The *C4* is used as a temporal buffer to buffer the output of *C3*. The reason will be explained latter. Each PE generates the corresponding variables defined in Section III. The ν^k in each PE indicates whether this sample is decoded. The $p^{msb} = 1$ in PE2 is used to indicate whether $(\hat{d}_s^k = 0)\&(d_s^k = 0)\&(p_s^k = 1)$ defined in (1) is satisfied or not since the condition, $(\hat{d}_s^k = 0)\&(d_s^k = 0)$, is happened only at the MSB bit-plane of s. A special code, $(\hat{d}^k, d^k, \nu^k) = (1, 1, 0)$, is used to represent γ^k to save one bit register since when $\hat{d}^k = 1$ and $\nu^k = 0$, d^k is meaningless and it can be used to present the γ^k. The finite state machine (FSM) controller receives all variables calculated from each PE, and it generates corresponding coding pass and contexts for the sample bit to be decoded. The FSM controller also receives the decoded magnitude bit and sign bit from AD then writes into the corresponding PE that scans the current-decoded sample. The χ and the $\hat{d}^k$ output from *C3* or *C4* are merged into the dataflow of the CF of the lower bit-plane while the d^k is written into the MRB as shown in Fig. 6.

The forward control signal is activated by the FSM controller whenever four sample bits in a column are decoded. When the forward signal is activated, all the data stored in the register of each PE are shifted by one column left, and the CF in the lower

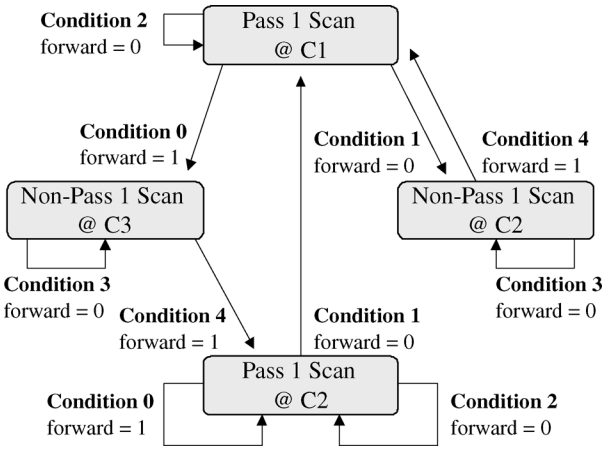


Fig. 9. Finite state machine (FSM) transition diagram. The transition conditions are described in Table II.

TABLE II
TRANSITION CONDITIONS OF FSM CONTROLLER IN FIG. 9

Condition	Description
0	Four Pass 1 samples are decoded
1	There is no more Pass 1 sample
2	There still has Pass 1 sample
3	There still has Pass 2 or Pass 3 sample
4	There is no more Pass 2 or Pass 3 sample

bit-plane fetches a column from either the *C3* or the *C4* of the CF in the upper bit-plane. At the same time, the column PE, *C4*, is used as temporal buffer to buffer the data of *C3* until the forward signal of the CF in the next lower bit-plane is activated. The column-switching scan order, which is described in Section III-A, is realized by FSM controller, and the state-transition diagram is shown in Fig. 9, in which each transition condition is described in Table II. The column-switching scan order realized on the proposed architecture could be seen from the another point of view: the context window moves forward and backward between *C1*, *C2*, and *C3*, while an empty bit-plane of code-block is shifted into the CF from right to left.

B. Four-Symbol Arithmetic Decoder

Fig. 10 shows the FAD architecture which is capable of processing four symbols per cycle. The FAD contains two one-symbol arithmetic decoder (AD) and two uniform decoder (UD). The output of two multiplexors are controlled by the decoding result of the AD0. By the multiplexed controls, the FAD could be operated at one-symbol, two-symbol or four-symbol mode. For the one-symbol mode, only the AD0 is activated when the coding pass is either the Pass 1 and the Pass 3 with zero-valued magnitude bit, or the Pass 2 for the refinement decoding. For the two-symbol mode, both the AD0 and the AD1 are activated for the magnitude and sign decoding in Pass 1 and Pass 3. In four-symbol mode, two more UD are used for the uniform decoding when RLC condition is satisfied but nonzero decoded magnitude bit. In JPEG 2000 standard, there is no adaptability for the uniform coding since the probability

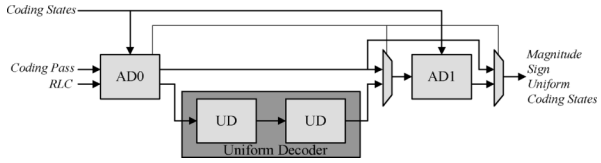


Fig. 10. Architecture of four-symbol arithmetic decoder (FAD). The output of two multiplexors are controlled by the decoding result of the AD0. By the multiplexed controls, the FAD could be operated at one-symbol, two-symbol, or four-symbol mode.

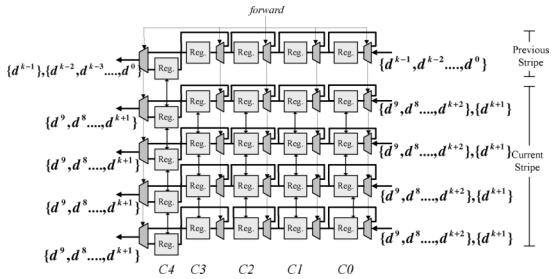


Fig. 11. Architecture of the register bank (RB) in bit-plane k. The RB receives a column of decoded samples (d^{k+1}) from the CF in the upper bit-plane and merges them with the column of partial-decoded coefficients ($d^9, d^8 \ldots d^{k+2}$) from the RB in the upper bit-plane.

TABLE III
HARDWARE REQUIREMENT OF THE PROPOSED ARCHITECTURE

	PCF	MRB	FAD	Control
Datapath (gates)	2431×10	11885	9732×10	4898
SRAM (bits)	12×64	-	-	-

of appearance of the position for the first nonzero bit in a column is random. Therefore, the UD module is specially designed to shorten its critical path by removing the circuits for the adaptive functions. The critical path of two UD is equal to that of one AD. The output of the FAD is the magnitude bit, sign bit, uniform bit and updated coding states. The updated coding states are written back to SRB, as shown in Fig. 6 and the others are fed back to the CF.

C. Magnitude Register Bank

As shown in Fig. 6, the MRB is used to buffer the decoded samples from the CF. The architecture of register bank (RB) for a bit-plane is shown in Fig. 11. Each RB contains five columns of registers. The forward signal passed from the FSM controller of the CF controls the dataflow. The RB receives a column of decoded samples (d^{k+1}) from the CF in the upper bit-plane and merges them with the a column of partial-decoded coefficients ($d^9, d^8 \ldots d^{k+2}$) from the RB in the upper bit-plane. The registers in RB can be classified into two parts, in which one part is for storing partial-decoded coefficients in the current stripe and the other part is for storing the coefficients of the last row of the previous stripe. For the bit-plane k, the former requires $5 \times 4 \times (9 - k)$ bits and the latter requires $5 \times k$ bits.

TABLE IV
COMPARISON OF THE PARALLEL ARCHITECTURE AND OTHER WORKS

	Process (μm)	Throughput (MSamples/sec)	Frequency (MHz)	Cycles for a Code-Block	Gates (NAND-2)	Memory (Bits)	Encode/ Decode	Coding Mode	PI (N=6 W=64)
[7]	0.35	4.6	50	$1.3NW^2$	19000	$5W^2$	Yes/Yes	sequential	0.027
[8]	0.35	11.7	142	$1.3NW^2$	21589	$4W^2$	Yes/Yes	sequential	0.024
[9]	0.35	NA	NA	NW^2	23927	$4W^2$	Yes/Yes	parallel	0.029
[15]	0.18	54	81	$1.5W^2$	91758	$12W$	Yes/No	parallel	0.029
ours	0.18	54	54	W^2	138414	$12W$	No/Yes	parallel	0.028

V. EXPERIMENTAL RESULTS

A. Implementation

The word-level architecture is described by the Verilog Hardware Description Language (HDL) and is logic-synthesized. This architecture supports 11 bits bit-width decoding, in which 1 bit for sign and 10 bits for magnitude. The selection of numbers of bit-width is decoder issue. The numbers of bit-width influence the quality of decoding images. For example, the bit-width of the DWT coefficients would grow up to 12 bits in the worst case for the high-high band using 9–7 filter in the encoder. If total 12 bits are losslessly encoded but only 11 bits are decoded in the decoder, the decoded image quality is about $45 \sim 50\,\mathrm{dB}$. In this implementation, we implement 11 bit-plane coders to decode high quality image.

The implementation results are shown in Table III. The logic gates are measured with numbers of NAND-2 gates and the datapath includes the registers. The supported code-block size is 64×64 or 32×32. It is capable of processing 54 Msamples/s at 54 MHz and can support HDTV 720p (1280×720, 4:2:2) resolution pictures decoding losslessly at 30 fps (Frames Per Second) in real time. The proposed word-level decoding architecture combined with level-switched scheduling is realized in [24] to achieve pixel-pipelined dataflow. This codec is implemented on a 20.1 mm^2 die with 0.18 μm CMOS technology to achieve 124 MSamples/s at 42 MHz. There are three block coders and each block coder provides 42 MSamples/s. The degradation of frequency from the synthesized result 54 MHz comes from that the level-switched scheduling [24] is applied, and therefore the critical path is increased. Each block coder occupies about 4 mm^2 die area, but this area also contain the additional logic gates and memory that enable level-switched scheduling and encoding functions as well as the circuits for system integration.

The total 399×10 bits memory requirement for storing coding states of AD are implemented with registers since SRAM implementation occupies larger silicon area. To support parallel decoding with SRAM implementation, multiple physical memories instead of single physical memory are required to provide sufficient internal bandwidth. However, the implementation with multiple physical memories occupies larger area and consumes more power than that with registers. Therefore, we use registers to implement the coding states of AD. The 399 bits for a bit-plane contains two parts, one for indexes of probability table for contexts and the other for arithmetic register. There are 14, 3, and 16 contexts used for Pass 1, Pass 2, and Pass 3, respectively, and each context requires 7 bits. The arithmetic register for one coding pass are 56 bits. Therefore, the memory requirement of the coding states for one bit-plane is 399 bits ($= (14 + 3 + 16) \times 7 + 56 \times 3$), and ten magnitude bit-planes require 3990 bits.

B. Comparison

The comparisons with previous works are summarized in Table IV. The throughput is the number of DWT coefficients that can be processed in a cycle, and the frequency is synthesized frequency. The precessing cycles means the number of cycles required to process a code-block with size $W \times W$, and the number of magnitude bit-planes of the code-block is N. The gates and memory are the required resources to implement various architectures. The coding mode indicates which coding mode is supported for this architecture. Note that the parallel coding mode is a subset of the sequential coding mode. Therefore, the proposed architecture can not be used in a generalized decoder since the word-level architecture can not decode all formats of bit-streams. It is suggested using this architecture for the specific applications requiring high throughput rate such as HD decoding.

Although [7]–[9] implement the encoder architectures, the decoder function can be achieved by replacing the one-symbol AE with one-symbol AD. However, the values of throughput listed in the Table IV are ideal values for the decoder. The throughput would be lower than those of the encoder since some sample bits generate two or four symbols in a cycle. The approach of buffering symbols by FIFO in the encoder design can not be used due to the feedback path between the CF and the AD. If the AD can not process all the generated symbols in a cycle, the CF must be stalled. The ideal throughput can be achieved by replacing one-symbol AE with four-symbol AD at a cost of increase of logic gates. For [15], it implements the word-level parallel architecture for the encoder. However, the word-level parallel for the decoder can not be achieved even if the AE is replaced with AD since the technique of looking ahead can not be used for the decoder design. For the proposed architecture, the encoder function can be realized if the AD is replaced with AE since the proposed scan order can be applied for both encoding flow and decoding flow.

The reported throughput for the sequential architectures [7]–[9] are dependent on the average number of bit-planes assumed. For a kind of architecture, the effective throughput for processing images with average 4 bit-planes would be two times higher than that for processing images with average 8 bit-planes. The numbers of average bit-planes are related to the quantization value as well as target bit-rate. For the word-level architecture, the throughput is constant as long as the average number of bit-planes is smaller than that the hardware supports.

Even if the number of bit-planes to be processed exceeds that the hardware support, the exceeding LSB bit-planes can be truncated at a cost of the scarification of image quality. The most important advantage of the word-level architectures is that they can guarantee constant throughput no matter how many numbers of bit-planes are processed. The drawback is that the hardware efficiency compared to the sequential architectures is quiet low at very low bit-rate since only one or two bit-planes are processed in average. Therefore, which architecture is adopted depends on the target bit-rate and applications. Since the throughput of the sequential architectures is dependent on the average number of bit-planes, it is hard to make a comparison for various architectures in terms of throughput. However, we can use mathematical form to present the processing cycles required for processing a code-block, as listed in the Table IV, and using it to judge the area efficiency for various works at a specific number of bit-planes. We define a performance index (PI), which is defined as the numbers of samples are processed at one cycle and one unit area, i.e., $(2^{12} \times (W \times W))/(ProcessingCycles(N, W) \times Gates)$ where N is number of bit-planes and W is code-block size, is used to make a comparison among various works. The 2^12 factor is used to enlarge the value of PI. At the case $N = 6$ and $W = 64$ in which the average compression ratio (CR) is about 2, all works have almost the same PI, 0.027. It means all works have the same area efficiency at the case $N = 6$. Note that the PIs of [7]–[9], [15] are calculated according to the encoding speed and gates for the encoder and the gates do not include the cost of memories. Experimental results for testing various images show that the average numbers of bit-planes are about 3.4 at CR 10. At this case, the PIs of [7]–[9] are all 0.047 while the PI of this work is still 0.028. Although the PI of this work is lower than sequential architectures by 1.67 times, the sequential architectures can not achieve high throughput decoding even they have higher PI. Besides, if the target CR is 10, the numbers of bit-plane coders in the word-level architecture can be reduced appropriately to increase the area-efficiency since several LSB bit-planes are empty. Adopting ten bit-plane coders in this work is just a case for implementation. You can choose six bit-plane coders for the CR 10 since several LSB bit-planes are empty, or more than ten bit-plane coders for the CR 2 to support lossless decoding while achieving high throughput at the same time. How many bit-planes used can be dependent on which application is focused.

Finally, we compare the system integration for various EBC architectures. When the bit-plane sequential coding architectures [7]–[9] are integrated into system, either off-chip code-block memory or on-chip code-block memory is required since DWT is a word-level operation while the EBC is bit-plane level operation. The code-block memory is used to buffer the DWT coefficients. If the code-block memory is implemented with off-chip memory, the power consumption for accessing off-chip memory is large. If the code-block memory is implemented with on-chip memory to reduce the access power of the memory, it costs NW^2 bits to store the DWT coefficients and the area is increased. For the data organization in the memory, several columns of a bit-plane are usually grouped as a word to avoid redundant accesses for the memory since DWT has word-level dataflow while the sequential EBC has bit-plane dataflow. In this data organization, multiple memory words are required to form a DWT coefficient in the decoder. The special memory control and additional data buffer for the dataflow conversion are required. For the word-level architectures, both the DWT and the EBC have word-level dataflow. They can be integrated into system with few efforts on the data organization and memory control. Besides, some system-level schedulings [25], [26], [24] can be applied to reduce the memory requirement between the DWT and EBC.

VI. Conclusion

This paper presents a word-level decoding architecture of EBC in JPEG 2000 decoder. This architecture is based on the proposed word-level decoding algorithm. This algorithm overcomes intra bit-plane dependency and inter bit-plane dependency by the proposed column-switching scan order. It also eliminates state variable memories used in the conventional decoding architecture. Implementation results shows that the word-level architecture can support HDTV 720p (1280×720, 4:2:2) decoding losslessly at 30 fps.

References

[1] JPEG 2000 Requirements and Profiles 1999, ISO/IEC JTC1/SC29/WG1 N1271.

[2] JPEG 2000 Verification Model 7.0 (Technical Description) 2000, ISO/IEC JTC1/SC29/WG1 N1684.

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

[4] D. Taubman and M. Marchellin, *JPEG2000: Image Compression Fundamentals, Standards and Practice*. Norwell, MA: Kluwer, 2002.

[5] A. Skodras, C. Christopoulos, and T. Ebrahimi, "The JPEG 2000 still image compression standard," *IEEE Signal Process. Mag.*, vol. 18, no. 5, pp. 36–58, Sep. 2001.

[6] W. Pennebaker and J. Mitchell, Eds., *JPEG: Still Image Data Compression Standard*. New York: Van Nostrand Reinhold, 1992.

[7] 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.

[8] Y.-T. Hsiao, H.-D. Lin, and C.-W. Jen, "High-speed memory saving architecture for the embedded block coding in JPEG 2000," in *Proc. IEEE Int. Symp. Circuits. Syst.*, Scottsdale, AZ, May 2002, vol. 5, pp. 133–136.

[9] J.-S. Chiang, Y.-S. Lin, and C.-Y. Hsieh, "Efficient pass-parallel architecture for EBCOT in JPEG 2000," in *Proc. IEEE Int. Symp. Circuits and Syst.*, Scottsdale, AZ, May 2002, vol. 1, pp. 773–776.

[10] K. Andra, C. Chakrabarti, and T. Acharya, "A high-performance JPEG 2000 architecture," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 3, pp. 209–218, Mar. 2003.

[11] G. Pastuszak, "A high-performance architecture for embedded block coding in JPEG 2000," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, no. 9, pp. 1182–1191, Sep. 2005.

[12] 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," in *Proc. IEEE Int. Symp. Circuits. Syst.*, Vancouver, BC, Canada, May 2004, vol. 3, pp. 865–868.

[13] A. K. Gupta, M. Dyer, A. Hirsch, S. Nooshabadi, and D. Taubman, "Design of a single chip block coder for the ebcot engine in JPEG2000," in *Proc. IEEE Int. Midwest Symp. Circuits and Systems*, Aug. 2005, pp. 63–66.

[14] H.-C. Fang, T.-C. Wang, C.-J. Lian, T.-H. Chang, and L.-G. Chen, "High speed memory efficient Ebcot architecture for JPEG2000," in *Proc. IEEE Int. Symp. Circuits. Syst.*, Bangkok, Thailand, May 2003, vol. 2, pp. 736–739.

[15] H.-C. Fang, Y.-W. Chang, T.-C. Wang, C.-J. Lian, and L.-G. Chen, "Parallel EBCOT architecture for JPEG 2000," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, no. 9, pp. 1086–1097, Sep. 2005.

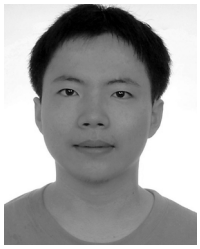
[16] Y.-K. Lai, L.-F. Chen, and T.-L. Huang, "A high throughput and memory efficient ebcot architecture for JPEG2000 in digital camera applications," in *IEEE Int. Conf. Consumer Electronics Dig. Technical Papers*, Jan. 2005, pp. 449–450.
[17] Y. Li, M. Elgamel, and M. Bayoumi, "A partial parallel algorithm and architecture for arithmetic encoder in JPEG2000," in *Proc. IEEE Int. Symp. Circuits and Syst.*, Kobe, Japan, May 2005, vol. 5, pp. 5198–5201.
[18] Y.-W. Chang, H.-C. Fang, C.-C. Chen, and L.-G. Chen, "Design and implementation of word-level embedded block coding architecture in JPEG 2000 decoder," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing*, Toulouse, France, May 2006, vol. 2, pp. 449–452.
[19] D. Taubman, "High performance scalable image compression with EBCOT," *IEEE Trans. Image Process.*, vol. 9, no. 7, pp. 1158–1170, Jul. 2000.
[20] J.-L. Mitchell and W.-B. Pennebaker, "Software implementation of the Q-coder," *IBM J. Res. Develop.*, vol. 32, no. 6, pp. 753–774, Nov. 1988.
[21] F. Ono, S. Kino, M. Yoshida, and T. Kimura, "Bi-level image coding with MELCODE-comparison of block type code and arithmetic type code," in *Proc. IEEE Global Telecommunications Conf.*, 1989, pp. 255–260.
[22] W. Pennebaker and J. Mitchell, *JPEG: Still Image Data Compression Standard*. New York: Springer, 1992.
[23] H.-H. Chen, C.-J. Liang, T.-H. Chaing, and L.-G. Chen, "Analysis of Ebcot decoding algorithm and its vlsi implementation for jpeg 2000," in *Proc. IEEE Int. Symp. Circuits. Syst.*, Scottsdale, AZ, May 2000, vol. 4, pp. 329–3332.
[24] Y.-W. Chang, H.-C. Fang, C.-C. Cheng, C.-C. Chen, C.-J. Lian, and L.-G. Chen, "124 Msmples/s pixel-pipelined motion-jpeg 2000 codec without tile memory," in *IEEE Int. Solid-State Circuits Conf. Dig. Technical Papers*, San Francisco, CA, Feb. 2006, pp. 404–405.
[25] B.-F. Wu and C.-F. Lin, "A low memory qcb-based dwt for jpeg2000 coprocessor supporting large tile size," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing*, Philadelphia, PA, Mar. 2005, vol. 5, pp. 9–12.
[26] H.-C. Fang, Y.-W. Chang, C.-C. Cheng, C.-C. Chen, and L.-G. Chen, "Memory efficient JPEG 2000 architecture with stripe pipeline scheme," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing*, Philadelphia, PA, Mar. 2005, vol. 5, pp. 1–4.

Yu-Wei Chang was born in Taipei, Taiwan, R.O.C., in 1980. He received the B.S. degree in electrical engineering from National Taiwan University (NTU), Taipei, Taiwan, R.O.C., in 2003, and the Ph.D. degree in 2007, also from NTU.

He is a currently a senior engineering in NovaTek, Inc., Hsinchu, Taiwan. His research interests include algorithm and architecture for image and video coding system.

Hung-Chi Fang was born in I-Lan, Taiwan, R.O.C., in 1979. He received the B.S. degree in electrical engineering in 2001 and the Ph.D. degree in 2005, both from National Taiwan University, Taiwan, R.O.C.

During 2005, he was a Visiting Student at Princeton University, Princeton, NJ, supported by the Graduate Students Study Abroad Program of the National Science Council ofTaiwan. He is currently a Senior Engineering with MediaTek, Inc., Hsinchu, Taiwan. His research interests are VLSI design and implementation for signal processing systems, image processing systems, and video compression systems.



Chun-Chia Chen was born in Changhwa, Taiwan, R.O.C., in 1982. He received the B.S. degree in electrical engineering in 2004 and the M.S. degree in 2006 from the Graduate Institute of Electronics Engineering, both at National Taiwan University, Taipei, Taiwan, R.O.C.

He is currently a Senior Engineering at MediaTek, Inc., Hsinchu, Taiwan. His research interests include algorithms and architectures for JPEG 2000 and JBIG2.

Chung-Jr Lian received the B.S., M.S., and Ph.D. degrees in electrical engineering from National Taiwan University (NTU), Taipei, Taiwan, R.O.C., in 1997, 1999 and 2003, respectively.

He is now a Postdoctoral Research Fellow at NTU in the DSP/IC Design Lab. His major research interests include image and video coding (JPEG, JPEG 2000, MPEG, H.264/AVC, etc.) and image and video codec VLSI architecture design.

Liang-Gee Chen (S'84–M'86–SM'94–F'01) was born in Yun-Lin, Taiwan, R.O.C., in 1956. He received the B.S., M.S., and Ph.D. degrees in electrical engineering from National Cheng Kung University, Tainan, Taiwan, in 1979, 1981, and 1986, respectively.

He was an Instructor (1981–1986) and an Associate Professor (1986–1988) in the Department of Electrical Engineering, National Cheng Kung University. In the military service during 1987 to 1988, he was an Associate Professor in the Institute of Resource Management, Defense Management College. In 1988, he joined the Department of Electrical Engineering, National Taiwan University (NTU), Taipei, Taiwan. During 1993 to 1994, he was a Visiting Consultant in the DSP Research Department, AT&T Bell Labs, Murray Hill, NJ. In 1997, he was a Vvisiting Scholar of the Department of Electrical Engineering, University of Washington, Seattle. During 2001 to 2004, he was the first Director of the Graduate Institute of Electronics Engineering (GIEE) at NTU. Currently, he is a Professor of the Department of Electrical Engineering and GIEE in NTU, Taipei. He is also the Director of the Electronics Research and Service Organization of the Industrial Technology Research Institute, Hsinchu, Taiwan. His current research interests are DSP architecture design, video processor design, and video coding systems.

Dr. Chen has served as an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY since 1996, as Associate Editor of the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION SYSTEMS since 1999, and as Associate Editor of the IEEE Transactions on CIRCUITS AND SYSTEMS II since 2000. He has been the Associate Editor of the *Journal of Circuits, Systems, and Signal Processing* since 1999, and a Guest Editor for the *Journal of Video Signal Processing Systems*. He is also the Associate Editor of the PROCEEDINGS OF THE IEEE. He was the General Chairman of the 7th VLSI Design/CAD Symposium in 1995 and of the 1999 IEEE Workshop on Signal Processing Systems: Design and Implementation. He is the Past-Chair of Taipei Chapter of IEEE Circuits and Systems (CAS) Society, and is a member of the IEEE CAS Technical Committee of VLSI Systems and Applications, the Technical Committee of Visual Signal Processing and Communications, and the IEEE Signal Processing (SP) Technical Committee of Design and Implementation of SP Systems. He is the Chair-Elect of the IEEE CAS Technical Committee on Multimedia Systems and Applications. During 2001–2002, he served as a Distinguished Lecturer of the IEEE CAS Society. He received the Best Paper Award from the R.O.C. Computer Society in 1990 and 1994. Annually from 1991 to 1999, he received Long-Term (Acer) Paper Awards. In 1992, he received the Best Paper Award of the 1992 Asia-Pacific Conference on circuits and systems in the VLSI design track. In 1993, he received the Annual Paper Award of the Chinese Engineer Society. In 1996 and 2000, he received the Outstanding Research Award from the National Science Council, and in 2000, the Dragon Excellence Award from Acer. He is a member of Phi Tan Phi.