Hindawi Publishing Corporation
EURASIP Journal on Applied Signal Processing
Volume 2006, Article ID 42568, Pages 1–14
DOI 10.1155/ASP/2006/42568

High Efficiency EBCOT with Parallel Coding Architecture for JPEG2000

Jen-Shiun Chiang, Chun-Hau Chang, Chang-Yo Hsieh, and Chih-Hsien Hsia

Department of Electrical Engineering, College of Engineering, Tamkang University, Tamsui, Taipei 25137, Taiwan

Received 8 October 2004; Revised 13 October 2005; Accepted 29 January 2006

Recommended for Publication by Jar-Ferr Kevin Yang

This work presents a parallel context-modeling coding architecture and a matching arithmetic coder (MQ-coder) for the embedded block coding (EBCOT) unit of the JPEG2000 encoder. Tier-1 of the EBCOT consumes most of the computation time in a JPEG2000 encoding system. The proposed parallel architecture can increase the throughput rate of the context modeling. To match the high throughput rate of the parallel context-modeling architecture, an efficient pipelined architecture for context-based adaptive arithmetic encoder is proposed. This encoder of JPEG2000 can work at 180 MHz to encode one symbol each cycle. Compared with the previous context-modeling architectures, our parallel architectures can improve the throughput rate up to 25%.

1. INTRODUCTION

The newest international standard of JPEG2000 (ISO/IEC 15444-1) [1–4] was proposed in December 2000. It has better quality at low bit rate and higher compression ratio than the widely used still image compression standard JPEG. The decompressed image is more refined and smoother [2]. Furthermore, JPEG2000 has more novel functions such as progressive image transmission by quality or resolution, lossy and lossless compressions, region-of-interest encoding, and good error resilience. Based on these advantages, JPEG2000 can be used in many applications such as digital photography, printing, mobile applications, medical imagery, and Internet transmissions.

The architecture of JPEG2000 consists of discrete wavelet transform (DWT), scalar quantization, context-modeling arithmetic coding, and postcompression rate allocation [1–4]. The block diagram of the JPEG2000 encoder is shown in Figure 1. It handles both lossless and lossy compressions using the same transform-based framework, and adopts the idea of the embedded block coding with optimized truncation (EBCOT) [5–7]. Although the EBCOT algorithm offers many benefits for JPEG2000, the EBCOT entropy coder consumes most of the time (typically more than 50%) in software-based implementations [8]. In EBCOT, each subband is divided into rectangular blocks (called code blocks), and the coding of the code blocks proceeds by bit-planes. To achieve efficient embedding, the EBCOT block coding algorithm further adopts the fractional bit-plane coding ideas, and each bit-plane is coded by three coding passes. However, each sample in a bit-plane is coded in only one of the three coding passes and should be skipped in the other two passes. Obviously, considerable computation time is wasted in the straightforward implementations due to the multipass characteristics of the fractional bit-plane coding of the EBCOT.

Recently, many hardware architectures have been analyzed and designed for EBCOT to improve the coding speed [9–11]. A speedup method, sample skipping (SS) [9], was proposed to realize the EBCOT in hardware to accelerate the encoding process. Since the coding proceeds column by column, a clock cycle is still wasted whenever the entire column is empty. In order to solve the empty column problems of SS, a method called group-of-column skipping (GOCS) [10] was proposed. However GOCS is restricted by its predefined group arrangement and it requires an additional memory block. An enhanced method of GOCS called multiple-column skipping (MCOLS) [11] was also proposed. MCOLS performs tests through multiple columns concurrently to determine whether the column can be skipped. The MCOLS method has to modify the memory arrangements to supply status information for determining the next column to

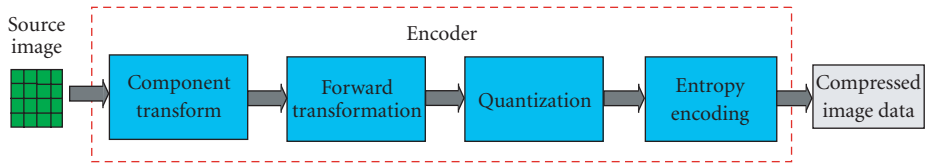

Figure 1: JPEG2000 encoder block diagram.

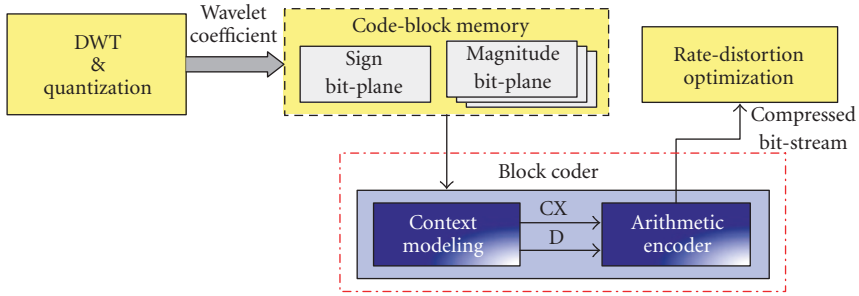

Figure 2: Block diagram of the embedded block coder.

be coded, and it limits the number of simultaneously combined columns. Besides the intensive computation, EBCOT needs massive memory locations. In conventional architectures, the block coder requires at least 20 Kbit memory.

Chiang et al. proposed another approach to increase the speed of computation and reduce the memory requirement for EBCOT [12]. They use pass-parallel context modeling (PPCM) technique for the EBCOT entropy encoder. The PPCM can merge the multipass coding into a single pass, and it can also reduce memory requirement by 4 Kbits and requires less internal memory accesses than the conventional architecture.

In order to increase the throughput of the arithmetic coder (MQ-coder), people like to design MQ-coder by pipelined techniques [13]. However, the pipelined approach needs a high-performance EBCOT encoder, otherwise the efficiency of the MQ-coder may be reduced. This paper proposes a parallel context-modeling scheme based on the PPCM technique to generate several CX-D data each cycle, and a matched pipelined MQ-coder is designed to accomplish a high-performance Tier-1 coder. Since the EBCOT encoder takes most of the computation time, our proposed parallel context-modeling architecture can further be applied to the multirate approach [14] to reduce the power consumption.

The rest of this paper is organized as follows. Section 2 describes the embedded block coding algorithm. Section 3 introduces the speedup scheme of our proposed context modeling. Section 4 describes the pipelined arithmetic encoder architecture. The experimental results and performance comparisons are shown in Section 5. Finally, the conclusion of this paper is given in Section 6.

2. BLOCK CODING ALGORITHM

In this section, we will focus on the concept of EBCOT. EBCOT consists of two major parts: context modeling and arithmetic encoder (Tier-1), and rate-distortion optimization (Tier-2). Figure 2 shows the block diagram of the embedded block coder. As introduced in the previous section, the EBCOT block coder of Tier-1 consumes most of the time in the JPEG2000 encoding flow. At the beginning, the discrete wavelet transform and scalar quantization are applied to the input image data. After that, the quantized transform coefficients are coded by the context modeling and the adaptive binary arithmetic coder to generate the compressed bitstream. Finally, the bit stream is truncated by a postcompression rate-distortion optimization algorithm to achieve the target bit-rate. The key algorithms about the context modeling and arithmetic encoder are described in the following sections.

2.1. *Context modeling*

The encoding method in the context modeling is bit-plane coding. In this module, each wavelet coefficient is divided into one-sign bit-plane and several magnitude bit-planes. Each bit-plane is coded by three coding passes to generate a context-decision (CX-D) pair.

The concept of bit-plane coding is to encode the data according to the contribution for data recovery. The most important data for data recovery is encoded firstly. Figure 3 is an example of bit-plane coding. All data can be divided into one-sign bit-plane and several magnitude bit-planes. Since the most significant bit (MSB) is more important than least

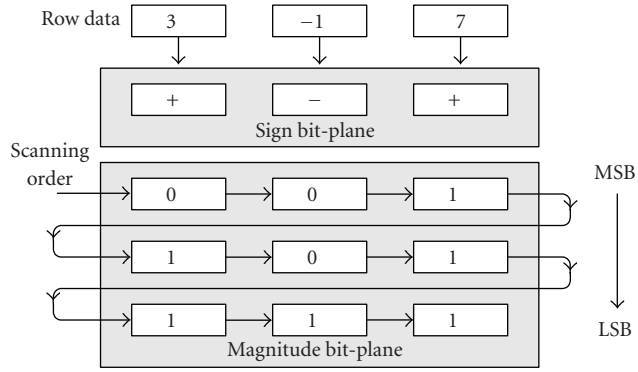

Figure 3: An example of the bit-plane coding.

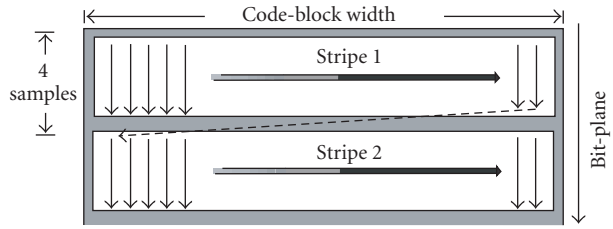

Figure 4: The scanning order of a bit-plane.

significant bits (LSBs), the scanning order is from MSB to LSB. During the process of the bit-plane coding, every four rows form a stripe. A bit-plane is divided into several stripes, and each bit-plane of the code block is scanned in a particular order. In each stripe, data are scanned from left to right. The scanning order is stripe by stripe from top to bottom until all bit-planes are scanned. The scanning order of each bit-plane is shown in Figure 4. In order to improve the embedding of the compressed bit-stream, a fractional bit-plane coding is adopted. Under this fractional bit-plane coding method, each bit-plane is encoded by three passes. These three passes are significance propagation (Pass 1), magnitude refinement (Pass 2), and cleanup (Pass 3). For the EBCOT algorithm, each bit in the code block has an associated binary state variable called "significant state." Symbols "0" and "1" represent insignificant and significant states, respectively. The significant state is set to significant after the first 1 is met. The pass type is determined according to these "significant" states. The conditions for each pass are described as follows.

Pass 1. The coded sample is insignificant and at least one of the neighbor samples is significant.

Pass 2. The relative sample of the previous bit-plane is set significant.

Pass 3. Those samples have not been coded by Pass 1 or Pass 2 in the current bit-plane.

These three passes are composed of four coding primitives: zero coding (ZC), sign coding (SC), magnitude refinement coding (MR), and run-length coding (RLC). These primitives are determined according to the neighbor states. Figure 5 depicts the different neighborhood states used for each type of coding primitives. There are total 19 contexts defined in the JPEG2000 standard. The MQ-coder encodes every sample in each bit-plane according to these data pairs. The details about these primitives are described as follows.

ZC is used in Passes 1 and 3. The samples that are insignificant must be coded in ZC.

SC is used in Passes 1 and 3. The sample set to significant just now must be coded by this operation.

MR is only used in Pass 2. The samples that have been significant in the previous bit-plane must be coded by this operation.

RLC is only used in Pass 3. This operation is used when four consecutive samples in the same stripe column are uncoded and all neighboring states of these samples are insignificant.

During the process of coding, we need two types of memory to store the bit-plane data and the neighboring states, respectively. For the bit-plane data, the memory requirement is two; while for the state variable, the memory requirement is three. The functions about the memory requirements are described as follows.

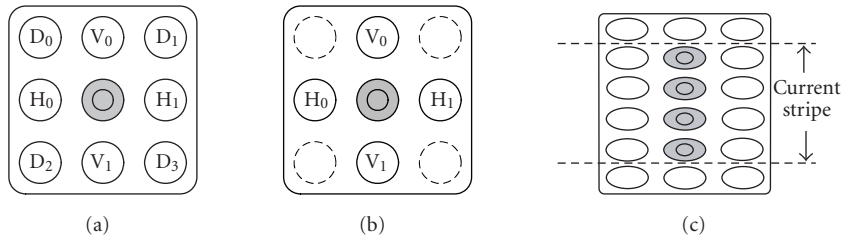

Figure 5: The neighbor states referred to by different primitives (a) ZC and MR, (b) SC, and (c) RLC.

Table 1: The number of "wasted samples" for each pass.

Image	Total sample	Wasted samples		
		Pass 1	Pass 2	Pass 3
Boat	1646592	1255591	1323978	713615
Pepper	1589248	1211206	1329045	638245
Zelda	1343488	998658	1135374	552944
Average	1526442	1155152	1262799	634935

Bit-plane data

$X[n]$ is used to store the sign bit-plane data of each code block.

$V_p[n]$ is used to store the magnitude bit-plane of each code block.

State variable

$\sigma[n]$ is used to store the significant state of each sample in a code block.

$\Pi[n]$ is used to record whether or not the sample has been coded by one of the three coding passes.

$\gamma[n]$ is used to record whether or not the sample has been processed by MR operation.

Each memory is 4 Kbits in size to support the maximum block size, and therefore the total internal memory is 20 Kbits.

2.2. Adaptive context-based arithmetic encoder

The compression technique adopted in JPEG2000 standard is a statistical binary arithmetic coding, which is also called MQ-coder. The MQ-coder utilizes the probability (CX) to compress the decision (D).

In the MQ-Coder, symbols in a code stream are classified as either most-probable symbol (MPS) or least-probable symbol (LPS). The basic operation of the MQ-coder is to divide the interval recursively according to the probability of the input symbols. Figure 6 shows the interval calculation of MPS and LPS for JPEG2000. We can find out whether MPS or LPS is coded, and the new interval will be shorter than the original one. In order to solve the finite-precision problems when the length of the probability interval falls below a certain minimum size, the interval must be renormalized to become greater than the minimum bound.

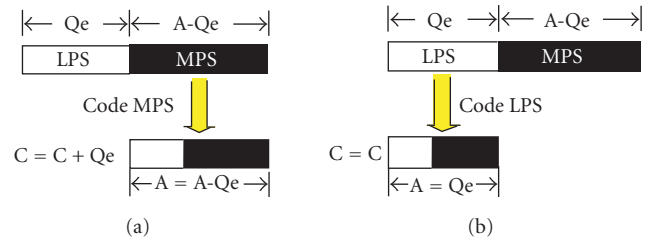

Figure 6: Interval calculation for code MPS and code LPS.

3. SPEEDUP ALGORITHM FOR CONTEXT MODELING

As introduced in the previous section, the block coding algorithm adopts the fractional bit-plane coding idea, in which three individual coding passes are involved for each bit-plane. In a JPEG2000 system, each sample in the bit-plane is coded by one pass and skips the other two. These skipped samples are called "wasted samples." Table 1 shows the number of "wasted samples" obtained from the coding of three 512×512 gray-scale images. For the "Boat" image, 1 646 592 samples need to be coded, but only 391 001 (1 646 592–1 255 591) samples are encoded by Pass 1. The EBCOT algorithm consumes a great number of times due to the tedious coding process. Besides, the multipass bit-plane coding also increases the frequency of memory access state variables that may cause much dynamic power of internal memory. From these observations, we use two speedup methods to reduce the execution time. The first method is to process three coding passes of the same bit-plane in parallel. The second method is to encode several samples concurrently. The techniques about these two methods are discussed in the following sections.

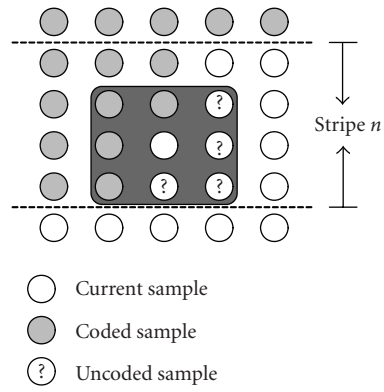

Figure 7: An example of the location of the predicted sample.

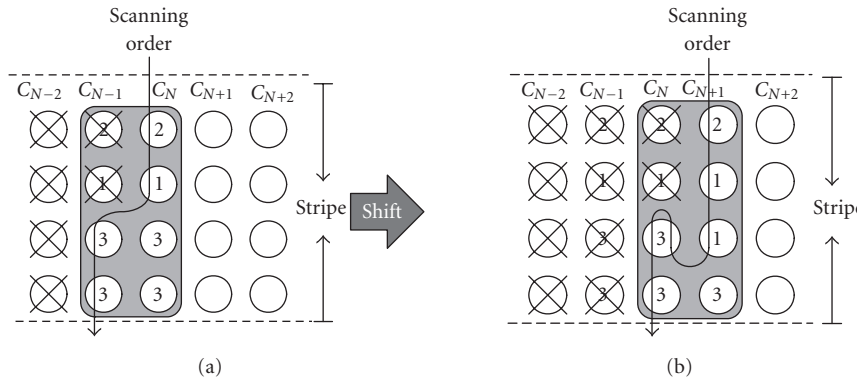

Figure 8: The scanning order of the pass-parallel algorithm.

3.1. Pass-parallel algorithm

Because of the inefficiency of the context-modeling of EBCOT, the pass-parallel method, pass-parallel context modeling (PPCM) [12, 15], can increase the efficiency by merging the three coding passes to a single one. If we want to process the three passes in parallel, there are two problems that must be solved. First, the scanning order of the original EBCOT is Passes 1, 2, and then Pass 3, and this scanning order may become disordered in the parallel-coding process [15]. Since the significant state may be set to one in Passes 1 and 3, the sample belonging to Pass 3 may become significant earlier than the other two prior coding passes and this situation may confuse the subsequent coding for samples belonging to Passes 1 and 2. Second, in parallel coding, those uncoded samples may become significant in Pass 1, and we have to predict the significant states of these uncoded samples correctly while Passes 2 and 3 are executed. Figure 7 gives an example about the location needed to be predicted.

In order to solve these problems, some algorithmic modifications are required. Here the Causal mode is adopted to eliminate the significance dependent on the next stripe. In order to prevent the samples belonging to Pass 3 to be coded prior to the other two coding passes, the coding operations of Pass 3 are delayed by one stripe column. Figure 8 shows an example of the scanning order of the pass-parallel algorithm. The numbers shown in Figure 8 are the pass numbers. At the first-column scanning, samples belonging to Passes 1 and 2 are scanned, but samples belonging to Pass 3 are skipped. At the second-column scanning, the scanning procedure goes to the next column and scans from top to bottom, and then goes to the previous column to scan the samples belonging to Pass 3. The samples belonging to Pass 3 in the current column should be skipped for the next-column scanning. Therefore in Figure 8(a), the scanning order starts from the first two samples of the current column C_N (Passes 2 and 1), respectively, and then goes to the previous column C_{N-1} to finish the scanning of the unscanned samples (Passes 3 and 3), respectively. Then the scanning procedure goes to the next column as shown in Figure 8(b). In the same manner, the scanning order starts from the first 3 samples in the current column C_{N+1} (Passes 2, 1, and 1), respectively, and then scans the last two samples in the previous column C_N (Passes 3 and 3), respectively.

Moreover, in PPCM two significant state variables σ_0 and σ_1 are used to represent the significant states of Passes 1 and 3, respectively. Besides, both the significant states are set to "1" immediately after the first MR primitive is applied. Since

Table 2: The state information of the two significant states in pass-parallel algorithm for current sample.

Significant states		Description	
σ_0	σ_1	Significance for current sample	First refinement for current sample
0	0	Insignificant	False
0	1	Significant	True
1	0	Significant	True
1	1	Significant	False

Table 3: The significant states of the pass-parallel algorithm for three coding passes. The symbol "||" means the logic operation of OR.

Samples	Significant states		
	Pass 1	Pass 2	Pass 3
Coded sample	$\sigma_0[n]$	$\sigma_0[n]$	$\sigma_0[n] \| \sigma_1[n]$
Uncoded sample	$\sigma_0[n] \| \sigma_1[n]$	$\sigma_0[n] \| \sigma_1[n] \| V_p$	$\sigma_0[n] \| \sigma_1[n]$

the refinement state variable $\gamma[n]$ can be replaced by the logic operation

$$\gamma[n] = \sigma_0[n] \oplus \sigma_1[n], \tag{1}$$

the memory requirement is not increased even if two significant states are introduced. The significant state and refinement state can be calculated as shown in Table 2.

Because two significant states are used, the significant state $\sigma[n]$ of the original one must be modified. We divide the significant states into two parts, coded sample and uncoded sample. For samples belonging to Pass 1, the significant states of the coded samples are equal to $\sigma_0[n]$; the significant states of the uncoded samples are

$$\sigma_{P1}[n] = \sigma_0[n] \| \sigma_1[n]. \tag{2}$$

For samples belonging to Pass 2, the significant states of the coded samples are equal to $\sigma_0[n]$. By utilizing the property that a sample will become significant if and only if its magnitude bit is "1," the significant states of the uncoded samples are determined by

$$\sigma_{P2}[n] = \sigma_0[n] || \sigma_1[n] || V_p[n]. \tag{3}$$

For samples belonging to Pass 3, the significant states of all neighbors are determined by

$$\sigma_{P3}[n] = \sigma_0[n] \| \sigma_1[n]. \tag{4}$$

The significant state for Passes 1, 2, and 3 can be calculated as shown in Table 3.

The pass-parallel algorithm needs four blocks of memory. These four blocks are classified as $X[n]$ (records all signs of samples in a bit-plane), $V_p[n]$ (records all magnitudes of samples in a bit-pane), $\sigma_0[n]$ (records the significance of Pass 1), and $\sigma_1[n]$ (records the significance of Pass 3). Each size of these memory blocks is 4 Kbits, and the total size of the memory requirement is 16 Kbits. Four Kbit memory is saved compared to the conventional one [1, 2].

3.2. Parallel coding

As introduced in the previous section, the pass-parallel algorithm can process three passes in parallel, and therefore no samples will be skipped. However, the operation speed can be increased further. In order to increase the computation efficiency further, we propose a parallel-coding architecture to process several samples concurrently. In the parallel architecture, the encoder will generate several CX-D pairs a time, however we have only one MQ-coder, and the MQ-coder can only encode a CX-D pair a time. Therefore a parallel-in-serial-out (PISO) buffer is needed for the MQ-coder to store the CX-D data generated by the parallel encoder temporarily. Before discussing the suitable size of the PISO buffer, we must determine the number of samples to be coded concurrently in a stripe column. For the parallel processing, as shown in Figure 9, we can either process two samples (Group 2) or four samples (Group 4) concurrently. Since two samples (Group 2) or four samples (Group 4) are encoded concurrently, the system must detect how many CX-D pairs are generated in a clock cycle. Group 2 or Group 4 generates different numbers of CX-D pair in a clock cycle, and the quantity of the generated CX-D pairs is called the output number of CX-D pair. A Group 4 process cycle may have output numbers from 1 to 10, and a Group 2 process cycle may have output numbers from 1 to 6. In Group 4, if the four encoded samples belong to Pass 3 and the magnitudes are all 1, respectively, under the run-length coding condition, it will generate 10 CX-D pairs. The 10 CX-D pairs include one RLC datum, 2 UNIFORM data, 3 ZC data, and 4 SC data. In the similar manner, Group 2 will generate 6 CX-D pairs at most, and the 6 CX-D pairs include one RLC datum, 2 UNIFORM data, one ZC datum, and 2 SC data.

A statistical analysis is used to determine which one, Group 2 or Group 4, can have the optimal hardware efficiency. Let us analyze Group 4 first. If the EBCOT encoder processes four samples concurrently, the output number of CX-D pair can be from 1 to 10. Six test images with two

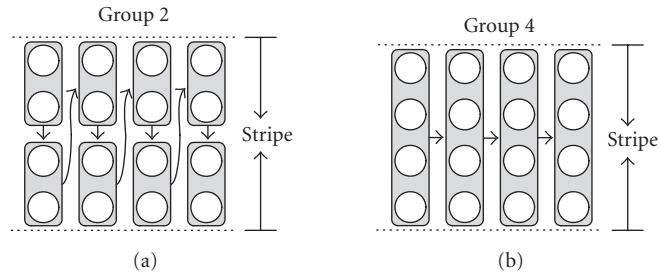

Figure 9: The parallel coding of Group 2 and Group 4.

Table 4: The probability of output numbers for processing 4 samples.

Image size	Test image	Output number									
		1	2	3	4	5	6	7	8	9	10
515 ∗ 512	Lena	184643	21206	26551	74479	65142	34910	11221	1672	71	11
		43.972%	5.050%	6.323%	17.737%	15.513%	8.314%	2.672%	0.398%	0.017%	0.003%
	Jet	207898	25514	30803	79569	67169	34227	10565	1568	66	14
		45.453%	5.578%	6.734%	17.396%	14.685%	7.483%	2.310%	0.343%	0.014%	0.003%
	Baboon	138568	19739	27996	147592	96889	38805	9297	1105	56	10
		28.865%	4.112%	5.832%	30.745%	20.183%	8.083%	1.934%	0.230%	0.012%	0.002%
2048 ∗ 2560	Bike	4306661	571842	713555	2125985	1584469	722201	193532	27016	1106	160
		42.030%	5.581%	6.964%	20.748%	15.463%	7.048%	1.889%	0.264%	0.011%	0.002%
	Cafe	3900550	631006	756489	2859833	1812041	726830	176424	22719	1041	179
		35.827%	5.796%	6.948%	26.268%	16.644%	6.676%	1.620%	0.209%	0.010%	0.002%
	Woman	3436608	422442	559734	2061204	1588258	757961	208006	26929	1144	196
		37.921%	4.661%	6.176%	22.744%	17.526%	8.364%	2.295%	0.297%	0.013%	0.002%
Average		39.011%	5.130%	6.494%	22.606%	16.669%	7.661%	2.120%	0.290%	0.013%	0.002%

different sizes are used to find the probability of each output number (from 1 to 10). Table 4 shows the simulation result. In order to increase the operation speed, the MQ-coder proposed in this paper is a pipelined approach. For the pipelined approach, the frequency of the MQ-coder can operate about twice faster than the original context modeling. From the simulation results of Table 4, around 44.141% (39.011% + 5.130%) possibilities can be processed by the MQ-coder immediately, however more than half of the possibilities cannot be processed immediately and a large size of PISO buffer is needed. Therefore the size of PISO buffer must be very large. Besides the size of the PISO buffer, there is another problem to be considered. Since the output number of CX-D is not constant, the encoder must determine the output state at the current clock cycle before the CX-D pairs are put into the PISO buffer. For four samples coded concurrently, there are 1024 possibilities and it must be determined within one clock cycle, and it is a long clock cycle.

On the other hand, let us analyze the effect of Group 2. Table 5 shows the simulation results of Group 2 with the same image of Group 4. Around 74.202% (30.444% + 43.758%) possibilities of the data can be processed immediately. The size of the PISO buffer is much smaller than that of Group 4. The output number of CX-D pairs is from 1 to 6, and there are only 64 possibilities. Compared to 1024 possibilities of Group 4, the clock cycle time can be much shorter in the Group 2 approach. By the above analyses, Group 2 is better for the hardware integration between the context modeling and the MQ-coder for parallel processing.

In fact, even though the MQ-coder is faster than the context modeling, the valid data still can be overwritten in the limited size of the buffer. Therefore a "stop" signal is needed to halt the operation of the context modeling. Figure 10 is the block diagram of our proposed block coder. The size of the PISO buffer decides the stop time of the context modeling. According to Figure 10, we use the six images with different buffer sizes to analyze the stop times. Table 6 shows the stop times and gate counts for different buffer sizes. Each buffer is with a 9-bit register. Since the maximum number of output is 6, the buffer size to simulate starts from 6.

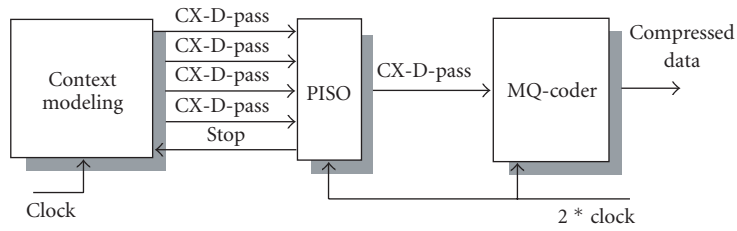

Figure 10: Proposed architecture of Tier-1.

Table 5: The probability of output numbers for processing 2 samples.

Image size	Test image	Output number 1	2	3	4	5	6
512 ∗ 512	Lena	213596	232360	133997	37356	373	145
		34.572%	37.609%	21.688%	6.046%	0.060%	0.023%
	Jet	241125	251715	135527	36001	431	126
		36.263%	37.856%	20.382%	5.414%	0.065%	0.019%
	Baboon	169038	409818	170974	34891	464	120
		21.525%	52.186%	21.772%	4.443%	0.059%	0.015%
2048 ∗ 2560	Bike	5083006	6465530	3002632	714731	7583	2351
		33.275%	42.325%	19.656%	4.679%	0.050%	0.015%
	Cafe	4728482	8226924	3224414	704373	6855	2181
		27.990%	48.700%	19.087%	4.170%	0.041%	0.013%
	Woman	4049392	6117527	3030085	737384	7939	2395
		29.039%	43.870%	21.729%	5.288%	0.057%	0.017%
Average		30.444%	43.758%	20.719%	5.007%	0.055%	0.017%

From Table 6, increasing the buffer size may reduce the stop times. However, the larger the buffer size is, the less the effect reaches. For example, the buffer size changes from 6 to 7 and it has 70.7%, ((3931 − 1150)/3931), improvement. When the buffer size changes from 14 to 15, there is only 11%, ((71 − 63)/71), improvement. Considering the hardware cost and efficiency, we select the buffer size to be 10.

In order to code two samples concurrently, the significant states of Table 3 must be modified. Figure 11 shows the parallel-coding status. There are two parts (Part I and Part II) in the parallel-coding status. At the beginning, both samples A and B are coded concurrently and then both samples C and D are coded subsequently. Let us use Part I as an example to explain the modification of the significant state. The neighbor states of A and B are included in the coding window (shaded area). The area circled by the dotted line is the neighbor states of sample A. The significant states referred to by sample A are the same as that we introduced in Table 3. The area circled by the solid line is the neighbor states of sample B. Since A and B are coded concurrently, the neighbor significance of A that sample B refers to must be predicted. If sample B is coded by Pass 1 or Pass 2, significance $\sigma[A]$ is predicted as (5). If sample B is coded by Pass 3, $\sigma[A]$ is predicted as (6),

$$\sigma[A] = \sigma_0[A] \| Sp,$$

$$Sp = \begin{cases} V_p[A], & \text{pass type of A is 1,} \\ & \text{where } V_p[A] \text{ is the magnitude of sample A,} \\ 1, & \text{pass type of A is 2,} \\ 0, & \text{pass type of A is 3,} \end{cases} \tag{5}$$

$$\sigma[A] = \sigma_0[A] \| V_p[A]. \tag{6}$$

The detail operations of the proposed parallel context modeling are described in Figure 12, and the block diagram of the proposed parallel context-modeling architecture is described in Figure 13.

4. ARITHMETIC ENCODER DESIGN

For a general EBCOT, the order of the CX-D pairs sent into the MQ-coder is Passes 1, 2, and 3, respectively. If the pass-parallel method is applied, the system needs a very large buffer to store the CX-D pairs belonging to Passes 2 and 3. The data dependency on coding order can be cut off if RESET

Table 6: The gate counts and the stop times for different buffer sizes.

Buffer size	Gate count	Stop number						Average
		Lena	Jet	Baboon	Boat	Pepper	Zelda	
6	764	3502	4397	2591	4221	4252	4624	3931
7	850	979	1409	604	1215	1326	1369	1150
8	970	413	647	182	537	599	527	484
9	1105	235	362	74	309	317	240	256
10	1185	162	221	41	224	199	124	161
11	1241	129	154	26	190	163	73	122
12	1323	106	112	17	172	126	48	96
13	1476	93	93	12	162	111	31	83
14	1586	77	78	9	152	93	20	71
15	1767	66	72	5	145	80	12	63

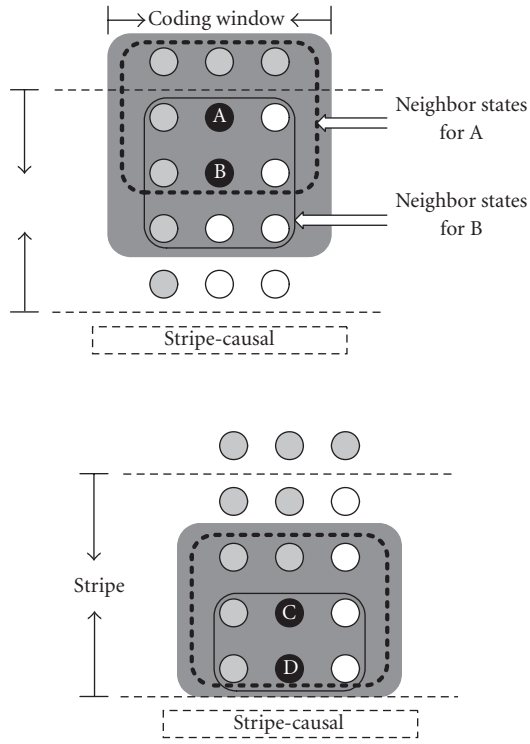

Figure 11: The parallel-coding status: (a) Part I and (b) Part II.

and RESTART modes are used. By RESET and RESTART modes we can use three MQ-coders to replace the large buffer. Since the CX-D data pairs of each coding pass generated by the context modeling are interleaved rather than concurrent, as shown in Figure 14, instead of using three MQ-coders, a low-hardware-cost pass switching arithmetic encoder (PSAE) was proposed in our previous work [12]. It uses three sets of context registers and coding state registers to achieve resource sharing of the MQ-coder for interleaved data.

Based on this concept, a pipelined MQ-coder is proposed, as shown in Figure 15. There are four stages in our design. The operation for each stage is described as follows. In Stage 1, in order to process the CX data belonging to different passes, respectively, it must increase the number of the context registers in the "CX table." However, there are only 14 contexts generated in Pass 1, 3 contexts in Pass 2, and 16 contexts in Pass 3. At the beginning, CX and "pass" are sent to the CX table to select an index and the MPS symbol. The MPS symbol is used to determine whether LPS or MPS is coded. The index is used to find the probability (Qe) of the current symbol and two new indexes (NLPS and NMPS). The correct updated index of current CX is not known until Stage 2 is finished. Therefore, the predicting scheme must be

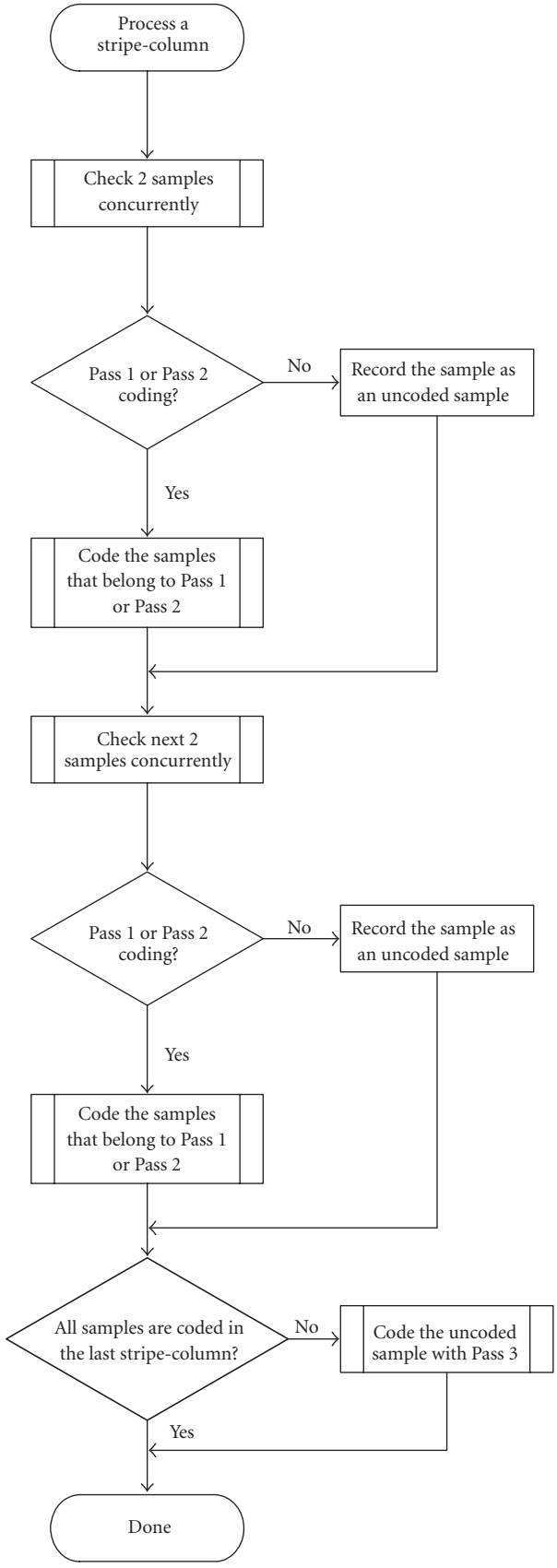

Figure 12: The detail operations of the proposed parallel context modeling.

Table 7: The chip features of parallel coding architecture.

Process technology	TSMC 0.35 um 1P4M	
Chip size	$2.44 \times 2.45\,\text{mm}^2$	
Frequency	Context modeling: 90 MHz Others: 180 MHz	
Power consumption	92 mW	
Synopsis reports for area	Component	Gate counts
	Context-modeling	8871
	MQ-coder	13611
	Memory	15247
	PISO buffer	1152
	Total	38881

used to select the correct index when the next CX and "pass" are the same as the current CX and "pass."

In Stage 2, the new interval (A) is calculated. After calculating the interval, the shift number of A is obtained according to the leading zeros of A. In order to increase the clock rate, the 28-bit lower bound (C) is divided into 16 bits and 12 bits. The operation of the low 16 bits is calculated in Stage 2 and the other is in Stage 3. This technique has been adopted in [10]. In Stage 3, the Byteout procedure and final calculation of C are executed. Note that the sizes of the coding state registers (A, C, CT, B) in Stage 2 and Stage 3 must be triple of the original ones. In Stage 4, since the output from the Byteout procedure is 0, 1, or 2 bytes, a FIFO is needed to make the last bit string in order. For a compression system, a large amount of input data are compressed into a smaller amount of output data. The probability of a 2-byte output is low. Therefore, a large size of the FIFO is not needed, and in general five bytes are enough. The maximum frequency of this MQ-coder can reach 180 MHz.

5. EXPERIMENTAL RESULTS

Based on the proposed techniques and architecture, we designed a test chip for the context modeling and MQ-coder. The chip features are summarized in Table 7.

5.1. Execution time

In order to increase the performance of the context modeling, both pass-parallel and coding-parallel (CP) are used in our proposed architecture. The execution time of our proposed architecture is compared with sample skipping (SS) and pass-parallel methods. The results of MCOLS are not compared here, since the number of the columns in a group and the number of simultaneously examined columns for MCOLS may affect the coding speed by an extra cost. We use six images of size 512×512 to simulate, and the experimental results are shown in Tables 8 and 9.

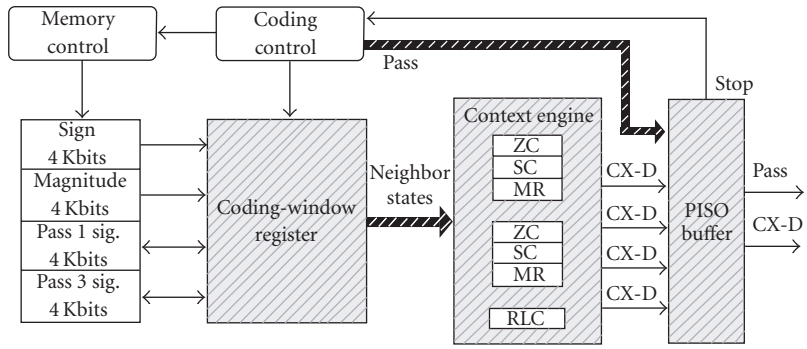

Figure 13: The block diagram of the proposed parallel context modeling.

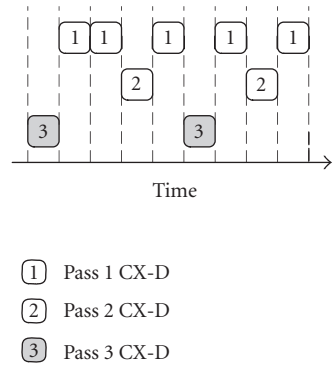

Figure 14: Parts of the CX-D pairs of image "Baboon."

Table 8: Execution time comparison.

Gray-scale test image	Execution time (clock cycles)			
	SS	SS + GOCS	Pass-parallel	Pass-parallel + CP (proposed)
Lena	1998998	1750322	1431739	1083918
Jet	1865079	1665888	1309989	979650
Baboon	2367662	2106508	1748425	1383739
Boat	1778578	1523707	1359648	1017169
Pepper	1758413	1531577	1277950	945675
Zelda	1573185	1353883	1142081	816326
Average	1890319	1655314	1378305	1037740

Table 9: Performance comparison.

Gray-scale test image	Reduced percentage Proposed architecture/other works			Execution performance (megasamples/second)
	SS	SS + GOCS	Pass-parallel	Pass-parallel + CP (proposed)
Lena	45.77%	38.07%	24.29%	123.67
Jet	47.47%	41.19%	20.86%	148.24
Baboon	41.55%	34.31%	25.22%	112.24
Boat	42.81%	33.24%	25.19%	145.90
Pepper	46.22%	38.25%	26.00%	151.27
Zelda	48.11%	39.70%	28.52%	148.40
Average	45.1%	37.30%	25.01%	138.29

Table 10: The differences of the PSNR between default mode and particular mode.

Test image	Compression ratio	PSNR (dB)					
		Code block = 64 × 64			Code block = 32 × 32		
		Default	Pass-parallel	Difference	Default	Pass-parallel	Difference
Lena (512 × 512)	8 : 1	41.4971	41.3599	0.1372	41.3813	41.1244	0.2589
	16 : 1	37.9316	37.7575	0.1741	37.8227	37.5185	0.3042
	32 : 1	34.4089	34.1876	0.2213	34.2363	34.0203	0.2160
	64 : 1	31.0082	31.0045	0.0037	30.9883	30.8024	0.1859
	100 : 1	29.1360	29.0952	0.0408	28.9808	28.8722	0.1086
	200 : 1	26.3491	26.2493	0.0998	26.1488	26.0265	0.1223
Baboon (512 × 512)	8 : 1	30.6148	30.5189	0.0959	30.5301	30.3007	0.2294
	16 : 1	26.6769	26.5923	0.0846	26.5890	26.4380	0.1510
	32 : 1	23.9612	23.8928	0.0684	23.9091	23.8432	0.0659
	64 : 1	22.1755	22.1680	0.0075	22.1494	22.1253	0.0241
	100 : 1	21.5192	21.3797	0.1395	21.5139	21.4880	0.0259
	200 : 1	20.2068	20.0599	0.1469	20.6728	20.6008	0.0720
Jet (512 × 512)	8 : 1	40.8825	40.6983	0.1842	40.8825	40.6983	0.1842
	16 : 1	36.7044	36.5029	0.2015	36.7044	36.5029	0.2015
	32 : 1	32.8160	32.5639	0.2521	32.8160	32.5639	0.2521
	64 : 1	29.3317	29.0414	0.2903	29.3317	29.0414	0.2903
	100 : 1	27.3779	27.0489	0.3290	27.3779	27.0489	0.3290
	200 : 1	24.8169	24.7857	0.0312	24.7587	24.6169	0.1418
Bike (2048 × 2560)	8 : 1	37.3174	37.1299	0.1875	37.1106	36.7397	0.3709
	16 : 1	32.9317	32.7512	0.1805	32.6763	32.3803	0.2960
	32 : 1	29.0480	28.9138	0.1342	28.8052	28.6223	0.1829
	64 : 1	25.8353	25.7501	0.0852	25.6222	25.5111	0.1111
	100 : 1	24.0939	24.0128	0.0811	23.9551	23.8677	0.0874
	200 : 1	21.8572	21.7936	0.0636	21.7363	21.6562	0.0801
Café (2048 × 2560)	8 : 1	31.5286	31.3333	0.1953	31.2841	30.8391	0.4450
	16 : 1	26.3112	26.1728	0.1384	26.1002	25.8465	0.2537
	32 : 1	22.7029	22.6191	0.0838	22.5484	22.4209	0.1275
	64 : 1	20.3345	20.2952	0.0393	20.2456	20.1643	0.0813
	100 : 1	19.2259	19.1954	0.0305	19.1502	19.1009	0.0493
	200 : 1	17.8733	17.8378	0.0355	17.8379	17.7816	0.0563
Woman (2048 × 2560)	8 : 1	37.5158	37.3455	0.1723	37.5158	37.0353	0.4805
	16 : 1	32.8517	32.7111	0.1406	32.6878	32.4885	0.1993
	32 : 1	29.2674	29.1784	0.0890	29.1473	28.9764	0.1709
	64 : 1	26.7952	26.7476	0.0476	26.7014	26.6232	0.0782
	100 : 1	26.5695	25.5301	0.0394	25.5144	25.4447	0.0697
	200 : 1	24.2154	24.1874	0.0280	24.1837	24.1340	0.0497

5.2. Compression performance

In our design, since three particular modes (RESET, RESTART, and Causal) are used, the image quality may be affected by these modes. In order to find the affection, three images with size 512 × 512 (Lena, Baboon, Jet) and three images with size 2048 × 2560 (Bike, Cafe, Woman) are used to evaluate the effects. Table 10 shows the differences of PSNR

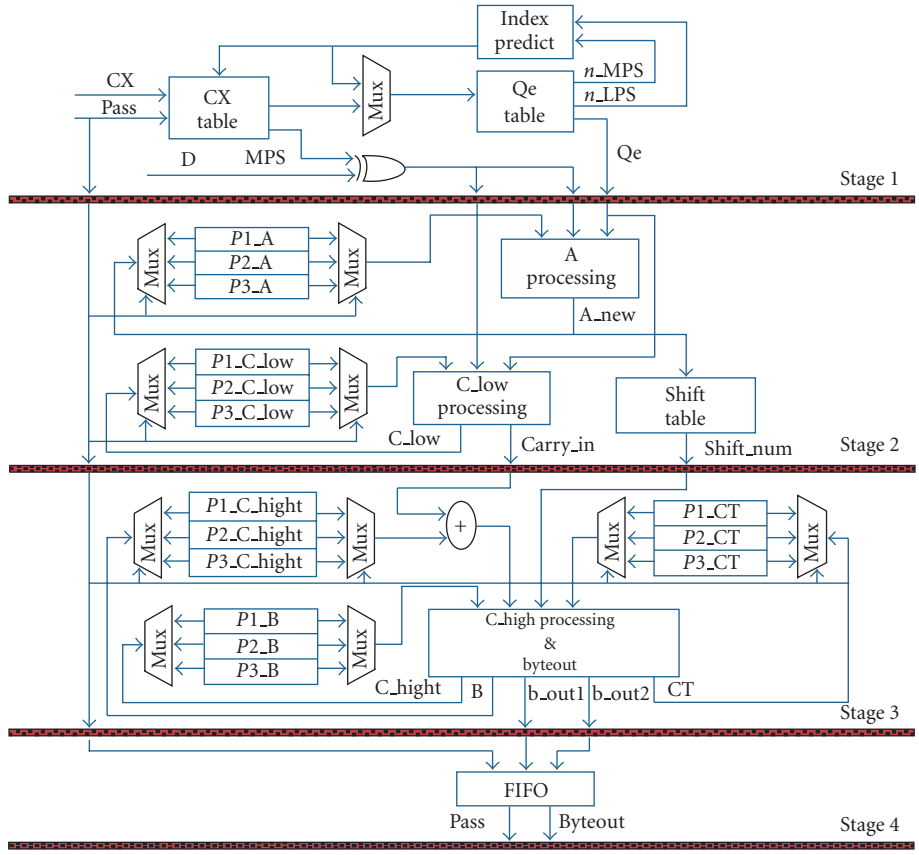

Figure 15: The proposed architecture of MQ-coder.

between the default mode and particular mode in different compression ratios. The combination of these modes introduces degradation on image quality as small as 0.0037 to 0.4805 dB that depends on the content of the encoded image.

6. CONCLUSION

There are two parts in our design: context-modeling and arithmetic encoder. For context-modeling, the pass-parallel method is adopted to merge the three-pass coding into a single-pass coding and reduce the frequency of memory access. Moreover, the coding-parallel method is used to reduce the execution time further. The throughput rate of the context-modeling is 45% better than SS method and is 25% better than the pass-parallel method. For the arithmetic encoder, we use a 4-stage pipelined architecture to reduce the clock cycle time. In order to process the interleaved data, a three-input multiplexer is used to replace three MQ-coders, and therefore the hardware cost can be reduced dramatically.

REFERENCES

[1] Information technology—JPEG 2000 image coding system—Part 1: Core coding system, ISO/IEC 15444-1, December 2000.

[2] D. Taubman and M. Marcellin, *JPEG2000: Image Compression Fundamentals, Standards, and Practice*, Kluwer Academic, Dordrecht, The Netherlands, 2001.

[3] C. Christopoulos, A. Skodras, and T. Ebrahimi, "JPEG2000 still image coding system: an overview," *IEEE Transactions on Consumer Electronics*, vol. 46, no. 4, pp. 1103–1127, 2000.

[4] C. Christopoulos, T. Ebrahimi, and A. N. Skodras, "JPEG2000: the new still picture compression standard," in *Proceedings of the ACM Multimedia 2000 Workshops*, pp. 45–49, Los Angeles, Calif, USA, October-November 2000.

[5] D. Taubman, "High performance scalable image compression with EBCOT," *IEEE Transactions on Image Processing*, vol. 9, no. 7, pp. 1158–1170, 2000.

[6] D. Taubman, E. Ordentlich, M. Weinberger, G. Seroussi, I. Ueno, and F. Ono, "Embedded block coding in JPEG2000," Tech. Rep. HPL-2001-35, HP Labs, Palo Alto, Calif, USA, February 2001.

[7] D. Taubman, E. Ordentlich, M. Weinberger, G. Seroussi, I. Ueno, and F. Ono, "Embedded block coding in JPEG2000," in *Proceedings of IEEE International Conference on Image Processing*, vol. 2, pp. 33–36, Vancouver, BC, Canada, September 2000.

[8] M. D. Adams and F. Kossentini, "JasPer: a software-based JPEG-2000 codec implementation," in *Proceedings of IEEE International Conference on Image Processing*, vol. 2, pp. 53–56, Vancouver, BC, Canada, September 2000.

[9] D. Santa-Cruz and T. Ebrahimi, "An analytical study of JPEG 2000 functionalities," in *Proceedings of IEEE International Conference on Image Processing*, vol. 2, pp. 49–52, Vancouver, BC, Canada, September 2000.

[10] C.-J. Lian, K.-F. Chen, H.-H. Chen, and L.-G. Chen, "Analysis and architecture design of block-coding engine for EBCOT in JPEG 2000," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 3, pp. 219–230, 2003.

[11] H.-H. Chen, C.-J. Lian, T.-H. Chang, and L.-G. Chen, "Analysis of EBCOT decoding algorithm and its VLSI implementation for JPEG 2000," in *Proceedings of IEEE International Symposium on Circuits and Systems*, vol. IV, pp. 329–332, Scottsdale, Ariz, USA, May 2002.

[12] J.-S. Chiang, Y.-S. Lin, and C.-Y. Hsieh, "Efficient pass-parallel architecture for EBCOT in JPEG2000," in *Proceedings of IEEE International Symposium on Circuits and Systems*, vol. I, pp. 773–776, Scottsdale, Ariz, USA, May 2002.

[13] K.-K. Ong, W.-H. Chang, Y.-C. Tseng, Y.-S. Lee, and C.-Y. Lee, "A high throughput context-based adaptive arithmetic codec for JPEG2000," in *Proceedings of IEEE International Symposium on Circuits and Systems*, vol. 4, pp. 133–136, Scottsdale, Ariz, USA, May 2002.

[14] A.-Y. Wu, K. J. R. Liu, Z. Zhang, K. Nakajima, and A. Raghupathy, "Low-power design methodology for DSP systems using multirate approach," in *Proceedings of IEEE International Symposium on Circuits and Systems*, vol. 4, pp. 292–295, Atlanta, Ga, USA, May 1996.

[15] 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 *Proceedings of IEEE International Symposium on Circuits and Systems*, vol. 3, pp. 865–868, Vancouver, BC, Canada, May 2004.

Jen-Shiun Chiang received the B.S. degree in electronics engineering from Tamkang University, Taipei, Taiwan, in 1983, the M.S. degree in electrical engineering from University of Idaho, Idaho, USA, in 1988, and the Ph.D. degree in the electrical engineering from Texas A&M University, College Station; Tex, USA, in 1992. He joined the faculty members of the Department of Electrical Engineering at Tamkang University in 1992 as an Associate Professor. Currently, he is a Professor at the department. His research interest includes digital signal processing for VLSI architecture, architecture for image data compression, SOC design, analog-to-digital data conversion, and low-power circuit design.

Chun-Hau Chang was born in Taipei city, Taiwan, in 1978. He received the B.Eng. degree in electronics engineering from Southern Taiwan University of Technology, Tainan, Taiwan, the M.S. degree in electrical engineering from Tamkang University, Taipei, Taiwan, in 2002 and 2004, respectively. His research interests are efficient implementations of multimedia hardware, high-performance parallel signal processing, and digital image signal processing.

Chang-Yo Hsieh received the B.S. degree in electrical engineering from Tamkang University, Taipei, Taiwan, in 2003. He is currently pursuing the M.S. degree in the Department of Electrical Engineering, Tamkang University. His research focuses on the image/video signal processing, and digital signal processing for VLSI design.

Chih-Hsien Hsia was born in Taipei city, Taiwan, in 1979. He received the B.S. degree in electronics engineering from Northern Taiwan Institute of Science and Technology, Taipei, Taiwan, in 2003. He received the M.S. degree in electrical engineering from Tamkang University, Taipei, Taiwan, in 2005. Currently he is pursuing the Ph.D. degree at the Department of Electrical Engineering, Tamkang University, Taipei, Taiwan. His research interests include DSP/IC design, VLSI architecture for image/video data compression, multimedia system design, multiresolution signal processing algorithms, and subband coding.