MEMORY-LESS BIT-PLANE CODER ARCHITECTURE FOR JPEG2000 WITH CONCURRENT COLUMN-STRIPE CODING

Minsoo Rhu and In-Cheol Park

Korea Advanced Institute of Science and Technology (KAIST)
{minsoo.rhu@gmail.com, icpark@ee.kaist.ac.kr}

ABSTRACT

In implementing an efficient block coder for JPEG2000, the memories required for storing the state variables dominate the hardware cost of a block coder. In this paper, we propose a novel bit-plane coder (BPC) architecture that derives all the state variables on the fly, thereby eliminating the memory requirement. In addition, we present a concurrent column-stripe coding algorithm which merges the scanning of all three coding passes into a single context window to generate all relevant context outputs concurrently in a single clock cycle. Experimental results show that the memory requirement and overall hardware cost of the proposed BPC are much smaller than those of previous architectures. Furthermore, as the column-stripe can be encoded for all three passes in a single clock cycle, a minimum of four context outputs are generated per cycle. Therefore, the following arithmetic coder that encodes the BPC outputs can never be in the idle state, enabling fast computation of overall block coding.

1. INTRODUCTION

JPEG2000 [1] adopts the discrete wavelet transform (DWT) as its primary transforming algorithm, whose transformed coefficients are split into codeblocks and sequentially processed by an entropy coding algorithm known as embedded block coding with optimized truncation (EBCOT). While the DWT is a word-level processing scheme, the EBCOT is inherently a bit-level processing algorithm that leads to its computational complexity. As a result, the block coding unit accounts for the majority of the computation time in the JPEG2000 encoding system, [2] which is why developing a high-performance block coder is crucial in implementing an efficient JPEG2000 encoding system.

The Tier-1 algorithm of EBCOT is divided into two sub-modules: bit-plane coder (BPC) and binary arithmetic coder (BAC). The BPC utilizes the neighboring information of the current bit to construct the context information consisting of context (CX) and decision (D), which will be entropy coded by the following BAC. Concerning the BPC, previous researches [2]-[6] mainly focused on reducing the number of state variables required for block coding and on devising efficient scanning mechanisms in order to reduce the block coding time. The methods presented in [2] reduced the number of clock cycles wasted in checking the pass membership, but there were still some clock cycles wasted with the overhead of additional memory. The memory saving algorithm proposed in [3] and [4] reduced the use of memory by 4K bits, but the remaining 16K bits memory still become a burden when the ASIC implementation is considered. Regarding the efficient-scanning mechanism, the causal relationship among the three coding passes is the most challenging factor. In dealing with the dependency issue, the architectures in [4], [5] and [6] adopt two separate context windows in order to delay the coding operations for Pass 3, thereby eliminating the reciprocal effect between Pass 3 and the other two passes as shown in Fig. 1. This mechanism has the advantage of scanning a whole W×W sized bit-plane in much less clock cycles, but the D/CX pairs of Pass 2 and 3 modeled out of the BPC can be delayed compared to Pass 1.

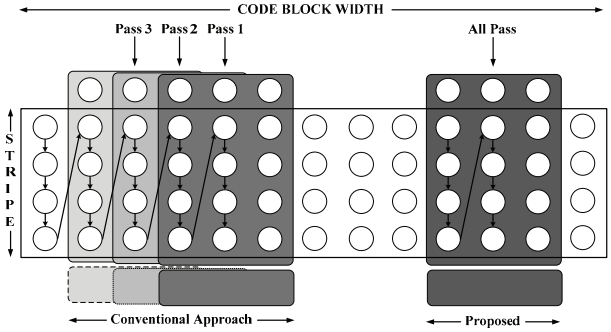

Fig. 1. Column-stripe based context window
(a) Conventional approach, (b) Proposed scanning-mechanism.

In this paper, we propose a novel BPC architecture that addresses the problems mentioned above. First, the proposed BPC derives all the state variables on the fly to reduce the required memory size to zero. Secondly, a novel concurrent column-stripe coding algorithm (CCCA) is presented to model all the D/CX pairs that are supposed to be generated within a column-stripe in a single-cycle. The algorithm adopts a single context window as shown in Fig. 1, so the proposed BPC always generates at least four D/CX pairs in a clock cycle, whereas the conventional approach can have no D/CX pairs generated when all the bits in a column-stripe are to be coded in Pass 2 or 3.

2. MEMORY-LESS BPC ARCHITECTURE

The DWT output is denoted in the signed-magnitude representation consisting of a sign bit and an *m*-bit magnitude. In the EBCOT, each bit-plane of the DWT output is incrementally encoded in the order of significance propagation pass (P1), magnitude refinement pass (P2), and clean-up pass (P3). Each sample in a bit-plane is coded in only one of the three coding passes and skipped in the other two. Therefore, the bit-plane coding algorithm [1] utilizes five state variables to determine the pass membership of each bit and to generate the relevant D/CX pairs. These include two bit-plane data memories and three state memories each of which has the size of (W×W) bits. Their

TABLE I. MEMORY REQUIREMENTS FOR BLOCK CODING ALGORITHM

Category	Name	Description	Size
Bit-Plane Data	υ_p	The p-th magnitude bit-plane.	4K
	χ	The sign bit-plane.	4K
Coding State Variable	σ	The significance state.	4K
	π	The bit-plane relationship variable identifying whether or not each sample has been coded in a previous pass of the same bit-plane	4K
	γ	The refinement state variable. The refinement primitive requires the information concerning whether or not each sample has been previously refined in P2.	4K

notations and functions are summarized in Table I. If a sample bit at location $\mathbf{k} = (k_1, k_2)$ is associated with '$\sigma[\mathbf{k}] = 0$' and if at least one of its eight neighbors is already significant, the sample bit is encoded in P1. In P2, a sample which has been significant in previous bit-planes is encoded. Lastly, samples that have not been encoded in P1 and P2 are encoded in P3. Accordingly, $\sigma[\mathbf{k}]$ and $\pi[\mathbf{k}]$ are used for the pass determination of each bit, and $\gamma[\mathbf{k}]$ is necessary for the magnitude refinement coding of P2 [1]. Note that the significance of a sample bit and its neighbors, which can become significant while encoded either in P1 or P3, are crucial in determining the pass membership of the sample bit. During the coding procedure in these two passes, the sample location becomes significant if the sample bit has a magnitude of '1'.

The BPC architectures in [2]-[6] all adopt a code-block of 64×64, so a total of 20K-bit memory is required when we store all the five variables, which is a huge burden when the ASIC implementation is considered.

2.1 Deriving the State Variables on the fly

Let the magnitude of a sample at location $\mathbf{k}$ be ($\upsilon_{m-1}[\mathbf{k}]$, $\upsilon_{m-2}[\mathbf{k}]$, ⋯, $\upsilon_1[\mathbf{k}]$, $\upsilon_0[\mathbf{k}]$), where $\upsilon_i[\mathbf{k}]$ is the *i*-bit of the magnitude. As mentioned earlier, a sample bit being significant means that the magnitude bit of the sample location is '1' for the first time in the current bit-plane *p*, or that at least one of the magnitude bits at the sample position were '1' in the upper bit-planes. Based on this observation, we define two state variables as follows.

$$\sigma_{prev}[\mathbf{k}] = \upsilon_{p+1}[\mathbf{k}] \mid \upsilon_{p+2}[\mathbf{k}] \mid \cdots \mid \upsilon_{m-1}[\mathbf{k}] \quad (1)$$

$$\sigma_{curr}[\mathbf{k}] = \upsilon_{p}[\mathbf{k}] \mid \upsilon_{p+1}[\mathbf{k}] \mid \cdots \mid \upsilon_{m-1}[\mathbf{k}] \quad (2)$$

Eq. (1) designates whether the sample location has become significant in previous bit-planes, and Eq. (2) signifies whether the sample location has become significant in the current bit-plane. Using these two variables, we derive the following property.

Property I. If $\sigma_{prev}[k]$ is '1' for a sample bit, meaning that the sample location was already significant in the previous bit-plane, the sample bit is always encoded in P2.

Property I can be used to determine whether the sample bit should be encoded in P2. Yet, in order to derive its D/CX pair, we need to know whether the sample bit is the first bit being encoded in P2 for the sample location. [1] Conventionally, $\gamma[\mathbf{k}]$ is used to determine whether or not the sample location has been previously refined in P2. To eliminate the use of $\gamma[\mathbf{k}]$, we define an additional state variable.

$$\sigma_{pprev}[\mathbf{k}] = \upsilon_{p+2}[\mathbf{k}] \mid \upsilon_{p+3}[\mathbf{k}] \mid \cdots \mid \upsilon_{m-1}[\mathbf{k}] \quad (3)$$

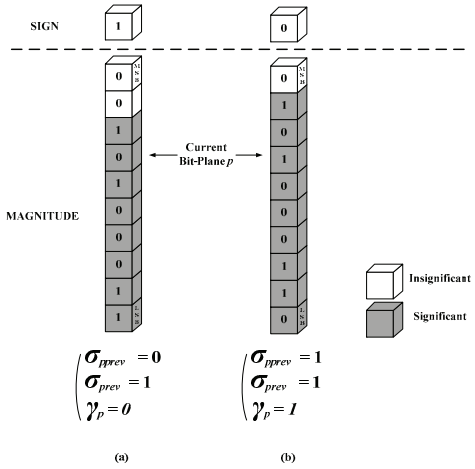

Fig. 2. Using Property II for the derivation of $\gamma_p[\mathbf{k}]$.

$\sigma_{pprev}[\mathbf{k}]$ shows whether the sample location has been significant in the second previous bit-plane. Using this, the second property is derived to eliminate the necessity of $\gamma[\mathbf{k}]$.

Property II. If $\sigma_{pprev}[k]$ is '0' and $\sigma_{prev}[k]$ is '1' for a sample bit, the bit is encoded by the magnitude refinement coding of P2 for the very first time. If $\sigma_{pprev}[k]$ is '1', the sample location was already encoded in P2 in previous bit-planes.

The sample bit shown in Fig. 2(a) satisfies Property II, meaning that '$\gamma_p[\mathbf{k}] = 0$' and that it is encoded in P2 for the first time. As $\sigma_{pprev}[\mathbf{k}]$ is '1' for the sample bit in Fig. 2(b), '$\gamma_p[\mathbf{k}] = 1$' designating that the sample location was already encoded in P2. Therefore, Property II can be utilized to eliminate the state memory related to $\gamma[\mathbf{k}]$.

2.2 Concurrent Stripe Coding Algorithm (CCCA)

Based on Eq. (1) to Eq. (3), we can derive $\sigma[\mathbf{k}]$ and $\gamma[\mathbf{k}]$ on the fly, which can eliminate the memory requirement for these two state variables. However, $\pi[\mathbf{k}]$ is still required to determine the pass membership of the sample locations. As mentioned in the previous sections, a sample location becomes significant only in P1 and P3. Because the sample bit can only be encoded in one of the three passes, clarifying whether the encoded bit location has become significant in P1 or in P3 is crucial in determining the pass membership of the bit and its neighbors. Accordingly, conventional architectures adopt a separate context window as shown in Fig. 1 in order to take into account the dependency issue in determining the pass membership of a sample bit. We propose the CCCA that could effectively eliminate the dependency issue and merge all three coding passes into a single context window. Based on this algorithm, all the relevant D/CX pairs can be generated within a single clock cycle.

To explain the proposed algorithm, we define an additional state variable, $\lambda[\mathbf{k}]$, to reflect the significance of eight neighboring bits. $\lambda[\mathbf{k}]$ is sub-categorized into the following two variables. First, $\lambda_{prev}[\mathbf{k}]$ designates whether there is a neighbor of which $\sigma_{prev}[\mathbf{k}']$ is 1, where $\mathbf{k}'$ represents the neighbor location. This variable is to show whether any one of the eight neighboring bits were already significant in the previous bit-plane. For example, if we were encoding the bit location of {1} in Fig. 3, $\lambda_{prev}[1]$ is derived as follows.

$$\lambda_{prev}[1] = \sigma_{prev}[D] \mid \sigma_{prev}[0] \mid \sigma_{prev}[E] \mid \sigma_{prev}[F] \mid \sigma_{prev}[G] \mid \sigma_{prev}[H] \mid \sigma_{prev}[2] \mid \sigma_{prev}[I] \quad (4)$$

Secondly, $\lambda_{curr}[\mathbf{k}]$ designates whether any one of the eight neighboring bits of the sample location becomes significant in P1 of the current bit-plane. It is used to take into account the significance changes of the sample bit's neighbors in P1 and to reflect the effect of significance propagation. For each sample location in the encoded column-stripe, $\lambda_{curr}[\mathbf{k}]$ is derived as follows. (Pass[**k**] refers to the pass membership of the location {**k**})

For the bit at {0}:

$$\begin{aligned}\lambda_{curr}[0] = {} & ((Pass[A]==P1)\&(\upsilon_p[A]==1)) \\ & \mid ((Pass[B]==P1)\&(\upsilon_p[B]==1)) \\ & \mid ((Pass[C]==P1)\&(\upsilon_p[C]==1)) \\ & \mid ((Pass[D]==P1)\&(\upsilon_p[D]==1)) \\ & \mid ((Pass[F]==P1)\&(\upsilon_p[F]==1))\end{aligned} \quad (5)$$

For the bit at {1}:

$$\begin{aligned}\lambda_{curr}[1] = {} & ((Pass[A]==P1)\&(\upsilon_p[A]==1)\&(\upsilon_p[0]==1)) \\ & \mid ((Pass[B]==P1)\&(\upsilon_p[B]==1)\&(\upsilon_p[0]==1)) \\ & \mid ((Pass[C]==P1)\&(\upsilon_p[C]==1)\&(\upsilon_p[0]==1)) \\ & \mid ((Pass[D]==P1)\&(\upsilon_p[D]==1)) \\ & \mid ((Pass[F]==P1)\&(\upsilon_p[F]==1)) \\ & \mid ((Pass[H]==P1)\&(\upsilon_p[H]==1))\end{aligned} \quad (6)$$

For the bit at {2}:

$$\begin{aligned}\lambda_{curr}[2] = {} & ((Pass[A]==P1)\&(\upsilon_p[A]==1)\&(\upsilon_p[0]==1)\&(\upsilon_p[1]==1)) \\ & \mid ((Pass[B]==P1)\&(\upsilon_p[B]==1)\&(\upsilon_p[0]==1)\&(\upsilon_p[1]==1)) \\ & \mid ((Pass[C]==P1)\&(\upsilon_p[C]==1)\&(\upsilon_p[0]==1)\&(\upsilon_p[1]==1)) \\ & \mid ((Pass[D]==P1)\&(\upsilon_p[D]==1)\&(\upsilon_p[1]==1)) \\ & \mid ((Pass[F]==P1)\&(\upsilon_p[F]==1)) \\ & \mid ((Pass[H]==P1)\&(\upsilon_p[H]==1)) \\ & \mid ((Pass[J]==P1)\&(\upsilon_p[J]==1))\end{aligned} \quad (7)$$

For the bit at {3}:

$$\begin{aligned}\lambda_{curr}[3] = {} & ((Pass[A]==P1)\&(\upsilon_p[A]==1)\&(\upsilon_p[0]==1)\&(\upsilon_p[1]==1)\&(\upsilon_p[2]==1)) \\ & \mid ((Pass[B]==P1)\&(\upsilon_p[B]==1)\&(\upsilon_p[0]==1)\&(\upsilon_p[1]==1)\&(\upsilon_p[2]==1)) \\ & \mid ((Pass[C]==P1)\&(\upsilon_p[C]==1)\&(\upsilon_p[0]==1)\&(\upsilon_p[1]==1)\&(\upsilon_p[2]==1)) \\ & \mid ((Pass[D]==P1)\&(\upsilon_p[D]==1)\&(\upsilon_p[1]==1)\&(\upsilon_p[2]==1)) \\ & \mid ((Pass[F]==P1)\&(\upsilon_p[F]==1)\&(\upsilon_p[2]==1)) \\ & \mid ((Pass[H]==P1)\&(\upsilon_p[H]==1)) \\ & \mid ((Pass[J]==P1)\&(\upsilon_p[J]==1))\end{aligned} \quad (8)$$

Eq. (5) to Eq. (8) only considers the locations that have already been coded within the context window and neglects the locations that are not visited yet. For example, Eq. (6) neglects the significance of {2,E,G,I} because these locations are not encoded yet in P1, while {D,F,H,0} locations are already visited when the context window is centered around {1}. In order to reflect the significance propagation effect of location {0}, locations {A,B,C} have been included in the derivation of $\lambda_{curr}[1]$. Therefore, based on Eq. (4) to Eq. (8) and the properties mentioned in the previous subsection, we attain the following property.

Property III. If $\sigma_{prev}[k]$ is '0' for a sample location, the bit is encoded in P1 or P3. If it is to be encoded in P1, either $\lambda_{prev}[k]$ or $\lambda_{curr}[k]$ must be '1', meaning that one of the neighboring bits was already significant or becomes significant in the current bit-plane. Otherwise, the sample bit is encoded in P3.

Property III designates the essence of the proposed CCCA which enables to determine the pass membership of all four bits within the column-stripe to be encoded and to derive the relevant D/CX pairs in a single clock cycle. Accordingly, the CCCA is defined as follows.

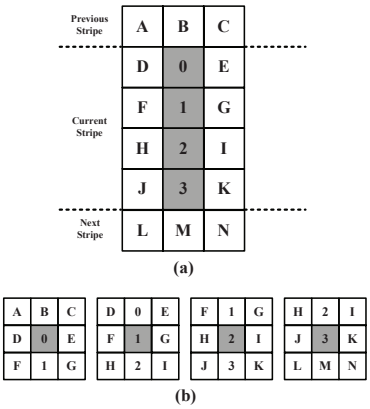

Fig. 3. (a) Context window for column-stripe scanning
(b) Individual context window for a bit location.

$$\begin{aligned}&\text{if } (\sigma_{prev}[\mathbf{k}]) \\ &\qquad Pass[\mathbf{k}] = P2 \\ &\text{else if}(\lambda_{prev}[\mathbf{k}] \mid \lambda_{curr}[\mathbf{k}]) \\ &\qquad Pass[\mathbf{k}] = P1 \\ &\text{else} \\ &\qquad Pass[\mathbf{k}] = P3\end{aligned} \quad (9)$$

3. PROPOSED ARCHITECTURE

The CCCA summarized in Eq. (9) is applicable to both the normal mode and the pass parallel mode of the bit-plane coding algorithm. It depends on how we compose $\lambda_{prev}[\mathbf{k}]$ and $\lambda_{curr}[\mathbf{k}]$ for bit locations {0} and {3}. If the proposed algorithm were to be used in the normal mode, λ[3] should take into account the significance of the bit locations {L},{M} and {N}, while the pass-parallel mode disregards these locations. Since these bits are located in the next stripe of the encoded stripe, only the $\lambda_{prev}[\mathbf{k}]$ values of these locations are needed for the normal mode. For the bit location of {0}, the significance of the bit locations {A}, {B} and {C} need to be considered. As these locations are located in the previous stripe, there is a possibility that these locations become significant in P1 of the current bit plane. In order to take into account this possibility, we utilize W registers of one bit to reflect the updated significances of the fourth row within the previous stripe. The 1-bit register is set to '1' when the sample bit of {3} was already significant or has become significant in P1 of the current bit-plane. As the context window scans the current stripe for W clock cycles, the W-bit register will be updated with the correct significance values of the fourth row within the current stripe. Accordingly, the W-bit register can be used for the encoding of the next stripe in order to correctly reflect the significance of the bit locations {A}, {B} and {C}.

Likewise, an additional 4-bit register is used to store the updated significance values of {0},{1},{2} and {3} within the currently encoded column-stripe. The 4-bit register is used for the encoding of the next column-stripe in order to adequately reflect the updated significance values of bit locations {D},{F},{H} and {J}.

TABLE II.
COMPARISON TO PREVIOUS ARCHITECTURES (BASED ON THE 64×64 CODE BLOCK SIZE)

Architecture	Proposed	[2]	[3]	[4]	[5]	[6]
Gate Count (NAND2)	5.6K	12K	N/A	N/A	8.6K	43K
Memory Requirements (bits)	0	12K	8K	8K	8K	6K
Processing Time for a W×W Code Block (cycles)	$0.25 \times W^2$	$1.3 \times W^2$	$1.3 \times W^2$	$0.25 \times W^2$	$1.0 \times W^2$	$0.6 \times W^2$
Minimum number of D/CX pairs generated per clock cycle	4	0	0	0	0	0

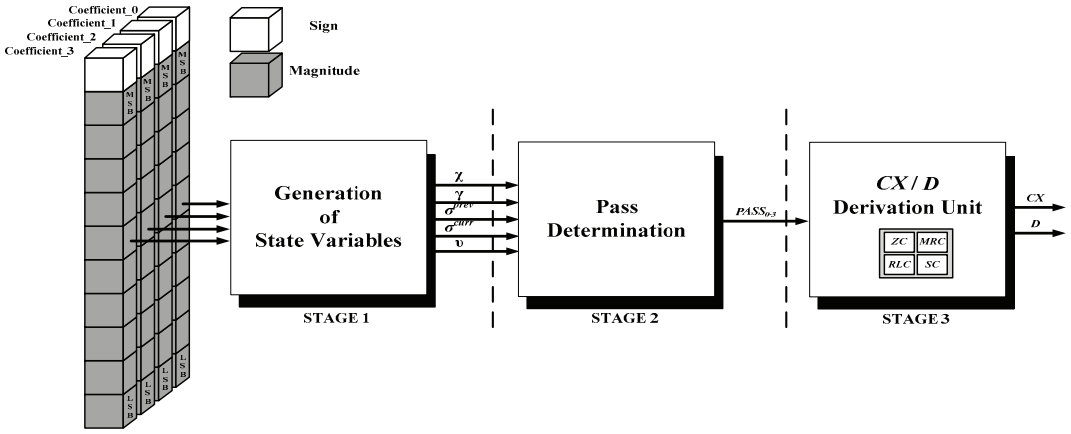

Fig. 4. Block diagram of the proposed BPC architecture.

Fig. 4 is a block diagram of the proposed BPC. It is based on the pass-parallel mode and the stripe-causal mode. [1] In order to increase overall throughput, the architecture is fully pipelined. Stage 1 uses the coefficient values forwarded from the DWT unit and generates the necessary state variables on the fly based on Property I and II for the pass determination. In stage 2, three shift-register banks are used to model the context window of Fig. 3, so the state variables can be utilized to identify the pass membership of bit locations {0} to {3}. Thanks to the CCCA of Property III, the dependence of the significance state variables is adequately resolved, so all three coding passes can be merged into a single context window as shown in Fig. 1. As a result, determining the pass membership of a column-stripe can be finalized in a single clock cycle effectively. The following stage 3 utilizes the pass membership data of the four bits and the relevant state variables in order to generate all D/CX pairs that are supposed to be generated within a column-stripe.

4. PERFORMANCE ANALYSIS

The proposed architecture was described in Verilog HDL and synthesized with a 0.18-μm cell library. The overall gate count is about 5.8K equivalent gates when the code block size is 64×64 and the number of nonzero bit-planes is 12. Because it derives all the state variables on the fly, no memory is required for its implementation. Also, compared to the conventional BPC architectures that adopt separate context windows, the proposed BPC uses a single context window and can generate all the D/CX pairs for a column-stripe in a single clock cycle. As shown in Table II, the implementation cost of the proposed BPC is substantially smaller than the traditional approaches. In addition, as the proposed BPC can completely encode a single column-stripe in a single clock cycle, the average number of D/CX pairs generated per clock cycle is the highest.

5. CONCLUSION

We have proposed a cost-efficient, memory-less BPC architecture that is optimized for ASIC implementation. We first eliminated the memory requirements of BPC by generating all state variables on the fly. Furthermore, the CCCA was presented to merge all three coding passes into a single context window, thereby generating all the relevant D/CX pairs in a single clock cycle.

6. REFERENCES

[1] ISO/IEC JTCI/SC29/WGI N1855, JPEG2000 Part1:Final Draft International Standard (ISO/IEC FDIS FDIS15444-1), Aug. 2000.

[2] C.J. Lian, K.F. Chen, H.H. Chen, and L.G. Chen, "Analysis and architecture design of block-coding engine for EBCOT in JPEG 2000," *IEEE Trans. Circuits Syst. Video Technol.,* vol.13, No.3, pp.219-230, Mar. 2003.

[3] Y.T. Hsiao, H.D. Lin, K.B. Lee and C.W. Jen, "High-speed memory-saving architecture for the embedded block coding in JPEG2000," *IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 5, pp.133-136, May. 2002.

[4] S. Rathi and Z. Wang, "Fast EBCOT Encoder for JPEG2000," *IEEE Workshop on Signal Processing Systems (SiPS)*, pp.595-599, Oct. 2007.

[5] J.S. Chiang, Y.S. Lin and C.Y. Hsieh, "Efficient Pass-Parallel architecture for EBCOT in JPEG2000," *IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 1, pp.773-776, May. 2002.

[6] Y. Li and M. Bayoumi, "Three-level Parallel High Speed Architecture for EBCOT in JPEG2000," *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 5, pp.5-8, March. 2005.