Fast FPGA Implementation of EBCOT block in JPEG2000 Standard

Anass Mansouri, Ali Ahaitouf, and Farid Abdi

UFR SSC, LSSC, Electrical Engineering Department Faculty of sciences & technology BP: 2202 FES MOROCCO
Phone: +212 35 61 13 26, Fax: +212 35 60 82 14, web site: http://www.fst-usmba.ac.ma

Abstract

Embedded block coding with optimized truncation (EBCOT) is an important feature of the latest digital still-image compression standard, JPEG2000; however, it consumes more than 50% of the computation time in the compression process. In this paper, we propose a new high speed VLSI implementation of the EBCOT algorithm. The main concept of the proposed architecture is based on parallel access to memories, and uses an efficient design of the context generator block. The proposed architecture is described in VHDL language, verified by simulation and successfully implemented in a Cyclone II and Stratix III FPGA. It provides a major reduction in memory access requirements, as well as a net increase of the processing speed as shown by the simulations.

Keywords: *JPEG20, EBCOT algorithm, VLSI architecture, FPGA implementation.*

1. Introduction

JPEG2000 is the latest still image compression standard, developed by ISO/IEC JTC1/SC29/WG1 (commonly referred to as the Joint Photographic Experts Group JPEG) [1]. JPEG 2000 is not only a competitive compression performance, but also provides many new features for different types of still images [2]. It offers quality scalability, resolution scalability, region of interest (ROI) coding, and supports both lossless and lossy coding in the same framework.

All these features are possible by adoption of the Discrete Wavelet Transform (DWT) and Embedded Block Coding with Optimal Truncation (EBCOT) originally proposed by Taubman [3]. Unfortunately, both algorithms are computation and memory intensive.

The EBCOT is one of the main resources intensive components of JPEG 2000, it accounts for nearly 50% of the total computation time of encoding process [4,6], and then it represents the most critical part in the design and implementation of the JPEG2000 standard. Besides the intensive computation, EBCOT needs massive memory locations. In conventional architectures, the block coder requires at least 20K-bit memory [10]. In order to decrease the EBCOT algorithm time execution two main speedup methods have been suggested. Sample skipping (SS) and Group Of Column Skipping (GOCS) [4]. In the first, one skips no operating samples and in the second all non operating columns are skipped, i.e. all the four bits of the column are skipped. This last technique allows to save a clock cycle naturally wasted in the SS method when a complete column is empty.

Many implementations of hardware architectures have been proposed and designed for EBCOT algorithm to improve the encoding speed, such as Andra's state-machine based bit plane encoder [9], which presents VLSI architecture for embedded bit-plane encoding in JPEG 2000 that reduces the number of memory accesses. This architecture has been implemented in VHDL and the estimated frequency of operation was 200 MHz. Chaing and al. [10] have proposed a Pass-Parallel Context Modeling (PPCM) to implement the EBCOT algorithm, this implementation can work at 180 MHz. They claimed that when compared with the previous context-modeling architectures, there solution improve the throughput rate up to 25%.

In this paper we present an efficient VLSI architecture for EBCOT. It's based on an optimized data organization and a new memory arrangement as well as a simple state machine and combinatorial logics of encoding part. Our proposed architecture makes the four bits to be processed and their neighbors available at one clock cycle and consequently a complete column is processed in only four clock cycles during each pass. This proposed architecture is implemented on FPGA without using any external memory.

The following part of the paper is organized as follows. Section II reviews the EBCOT algorithm; Section III describes the analysis of bit plane coding algorithm. The proposed EBCOT architecture is presented in section IV; the hardware implementation performances and results discussion are described in section V, and section VI concludes the work.

2. EBCOT Algorithm

As illustrated in Fig. 1, the encoding process in JPEG2000 standard follows the typical still image encoding operations. The DWT block transfers the image information from spatial domain to frequency domain and removes the spatial correlation. The redundant information can be rejected by quantization process which is the mainly lossy block in JPEG2000 standard. After the quantization step, many coefficients become zero then the entropy coder can encode the quantized coefficients more efficiently and generates the compressed bit stream. The entropy coding and generation of compressed bit stream in JPEG2000 are two tiers coders, tier-1 is a context based adaptive arithmetic coder and tier-2 is a bit stream layer formation.

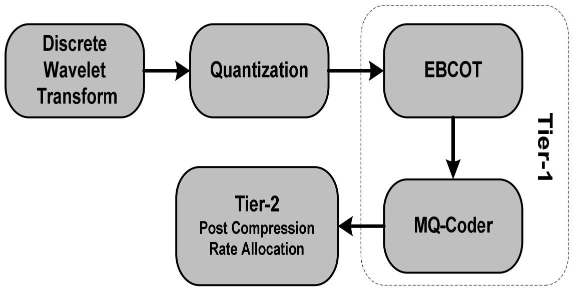

Fig. 1 Functional block diagram of JPEG2000 standard.

2.1Tier-1 coding: context based adaptative arithmetic coder

2.1.1 Bit-plane coding:

The key of JPEG2000 is the EBCOT algorithm. The DWT sub-bands are partitioned into relatively small blocks, called code-blocks (typically 64×64 or 32×32) [1, 2]. The code-blocks are encoded independently. Each code-block is decomposed into n bit-planes. Sequentially, they are encoded from the most significant bit-plane (MSB) to the least significant bit-plane (LSB). Each bit-plane is partitioned into a set of stripe, which spans the full width of the code block and consists of four rows. The stripes are scanned from the top to the bottom one by one. Within each stripe, the scan proceeds column by column, within a column, each sample location is scanned by a top-down manner, until all samples of the column have been visited as shown in Fig. 2.
The encoding process is done by fractional bit-plane coding (BPC) mechanism to create a context and a binary decision value for each bit position. JPEG2000 uses the EBCOT algorithm for the BPC.
This algorithm encodes each generated bit-plane in one of three coding passes [2, 3]: significant propagation pass (PASS-1), magnitude refinement pass (PASS-2), and cleanup pass (PASS-3).. For a bit-plane, these three passes are processed sequentially. The Pass-1 processes

coefficients which are insignificant [3] and have at least one significant neighbour among its 8 immediate neighbours, Pass-2 processes all significant coefficients except those becoming significant in Pass-1, finally Pass-3 processes all remaining coefficients not encoded in the Pass-1 and Pass-2. According to the information contained in each bit, four coding primitives are used to generate its context: zero coding (ZC), sign coding (SC), magnitude refinement (MR), and run-length coding (RLC) [3].

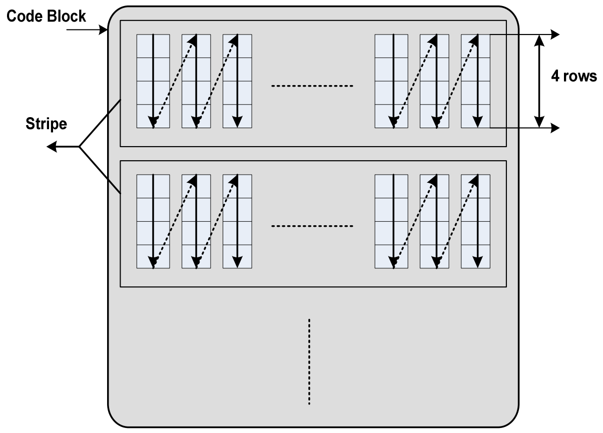

Fig. 2 The scanning order within a bit-plane.

The generated contexts are based on the contextual information (the sign and significance states of the 8-connect neighbors) of the sample scanned in current coding pass. A detailed description about these coding primitives can be found in references [3] and [11].

2.1.2 Arithmetic Coding:

The BPC outputs are entropy encoded using a MQ-coder which is a derivative of the Q-coder [12]. According to the provided context, the coder chooses a probability for the bit to encode, among predetermined probability values supplied by the JPEG 2000 standard and stored in a look-up table. From this probability the MQ-coder progressively generate the compressed code-words. These data are the output of the first tier, and send to Tier 2 for further selection to form the final JPEG2000 bitstream.
2.2 Tier-2 coding: bit stream layer formation
In this second step, each code-block is efficiently represented as a layer and block summary information [2]. A layer is a consecutive bit-plane coding passes from each code-block in a tile, including all sub-bands of all components in the tile. The block summary information consists of the length of compressed code words of the code-block and the truncation point between the bit-stream

layers. The compressed code-words generated in the Tier-1 coding step are encoded using a Tag Tree coding mechanism.

3. Analysis of the algorithm

Table 1 shows the complexity estimation for JPEG2000 coding obtained by using the modified software implementation [13]. The run-time table uses a P-IV1.2G CPU, 256M RAM, and VC++6.0 with WINXP system. The size of the test image Lena is 512 x 512 pixels, and we use a configuration with the flowing parameters (lossless and lossy filter, 4 levels wavelet decomposition and one layer with spatial scalability).
Our results (see table 1) as well as others works [5, 6] clearly show that the EBCOT algorithm takes the great part of the processing time compared to the others blocks, this is because EBCOT operations are bit-level processing as illustrated in Fig. 3, which requires important memory resources and several memory accesses.

Table 1: execution time for JPEG2000 encoder using the 512 x 512 Lena image

JPEG2000 blocks	Lossless compression	Lossy compression
DWT	11,4 %	21,2 %
EBCOT	67,8 %	61,3 %
MQ-Coder	18,2 %	14,1 %
Others	2,6 %	3,4 %

Therefore, the key to increase the processing speed of JPEG2000 consists of both new researches and new developments of more efficient VLSI system architecture of EBCOT algorithm.
In order to optimize the hardware design of EBCOT block, a detailed analysis of the algorithm is needed. Fig. 4 shows the analysis results obtained using a 512x512 Lena image which is decomposed by (5,3) filter with 4 levels decomposition for DWT block. Each bit in a bit-plane is encoded only in one of the three coding passes and skipped in the two others.
At first, all samples of the first bit-plane (MSB) are insignificants and encoded in PASS-3 of the coding process, for the lower bit-plane, some samples with neighbouring significant bits will be encoded in Pass-1 and bits which have been significant will be encoded in Pass-2. Therefore from Fig. 4 only a small number of coefficients are encoded by all three coding passes, so two speed-up methods have been proposed in order to accelerate the encoding process: sample skipping (SS) and group-of-column skipping (GOCS) [4, 7, 8]. The key idea of the SS method is to skip those no-operation samples in a single column. The SS is more efficient compared with the straightforward method, but a clock cycle is still wasted when a stripe column is "empty", that means none of the samples of the stripe column belongs to the current coding pass. Therefore, the second speed-up method, GOCS, is designed to further improve the processing speed. It skips

a group of "empty columns" simultaneously at the cost of an extra GOCS memory. Besides, the number of column in a group is a compromise between processing speed and area cost.

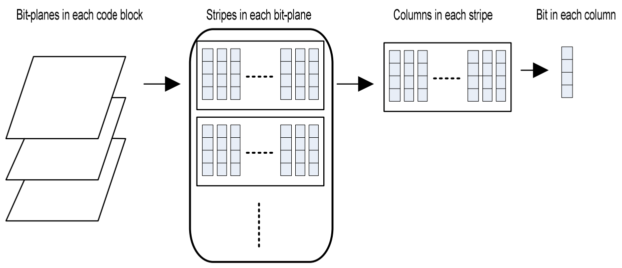

Fig. 3 The hierarchy of a code block.

In this work we adopt a hierarchical scanning for a given code block ordered from lower level (bit) to upper level (bit-plane) as illustrated in Fig. 3. If we can skip from upper level, several iterations will be saved.
The bit-plane coding performs context selection by examining state information for the surrounding neighbours of a sample. Three state variables are necessary for the context formation algorithm (significance state variable, magnitude refinement state variable and visited state variable), as defined in [3] and [11]. These variables are stored in three memories and two others memories are needed to store the sign bit-plane and magnitude bit-plane. Each memory is 4K bits to support the maximum block size. Hence, it is essential to reduce internal memory by using the optimized memory saving algorithm [7].

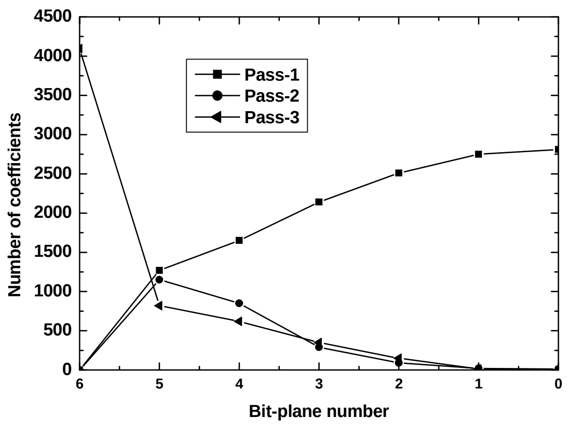

Fig. 4 Coefficients distribution in 3 passes with 64×64 sub-band from 512×512 Lena image.

4. Proposed architecture

In the proposed architecture, a complete column is processed in a single clock-cycle. The four bits to be encoded and their two neighbours are all available at the same time. Therefore to increase the speed of computation and reduce the memory requirement for EBCOT, we exploit the parallel access to memories and propose a new design of context generator for EBCOT tier-1 architecture. The key idea is to bypass the redundant coefficients-bits in each coding pass; it can be done by adopting a new method in the data organisation and memory arrangement. This method proves the efficiency of reducing both access number and memories bandwidth.
The proposed hardware architecture of EBCOT algorithm is presented in Fig. 5. The architecture reads the DWT coefficients data (LL, LH, HL, and HH sub-bands) from the code blocks memories, carry out the discrete wavelet transform, and output the context and data.
This architecture is based on an original register modules (data register module, and state variables register module) communicating through by using two controller blocks and working in parallel with a tight synchronization.
The block diagram of Fig. 5 consists mainly of: 1) Memory blocks 2) Switches and counters 3) Context generator and data selector (mux). All these blocks are controlled and synchronized by two state machines; one manages the pass of the algorithm and the second controls the columns processing in each pass, these blocks are described below.

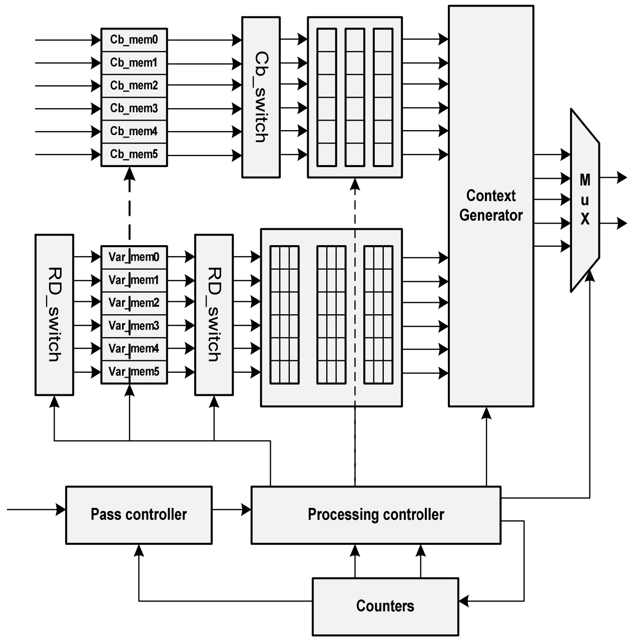

Fig. 5 Block diagram of the proposed EBCOT hardware implementation.

4.1 Data organization and memory arrangement

In order to achieve an efficient data and state variables memories access and to reduce the required memory access clock cycle, we propose a new data arrangement and memory organization to implement the Bit-plane coding. As shown in Fig. 6 memory blocks are organized in six partitions.

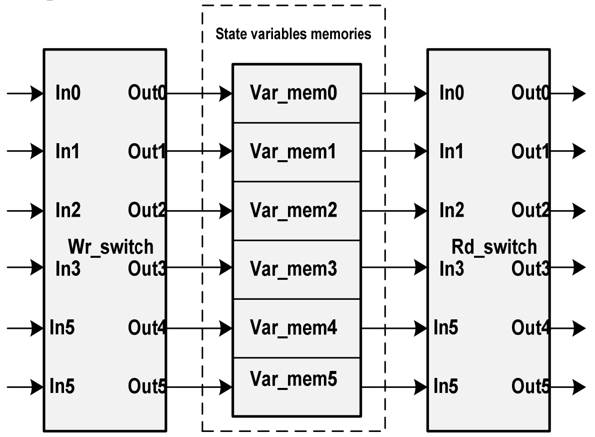

Fig. 6. State variables memories.

MEM0 to MEM5 contain the state variables data. The same organization is adopted for code block coefficients. In a single clock cycle they supply the four bits to be processed and their vertical neighbours at the same time (Fig. 7).
By using this memory arrangement, we can:

- Reduce the complexity of addressing.

- Perform read and write operations at the same clock cycle.
- Prevent the operational conflicts (simultaneous read or write).

Within this arrangement, samples of two nearby stripes, the previous and the next stripes, are loaded into the state variables register simultaneously in only one clock cycle. So, only a single clock cycle is spent by reading data from the six memories and shifting state variable register.

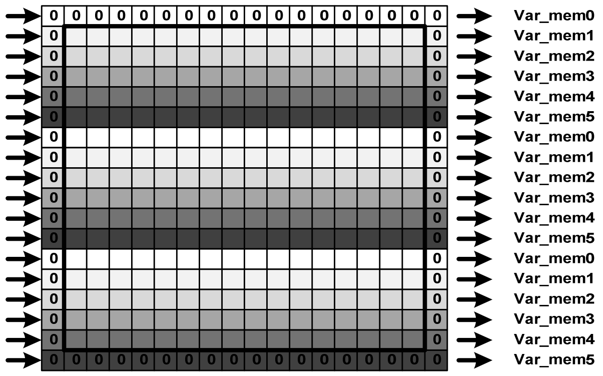

Fig. 7 Code block lines and code block memories association.

For the code block memories, we use a tile splitter block, which extracts the code blocks from each memory of wavelet coefficients sub-band. The extracted data are stored in six memories, in each one; a line of code block is stored as shown in Fig.8. The register blocks are implemented as two six parallel shift registers, one for code block coefficients and the second for state variables. The data shift from memories to registers column by column.

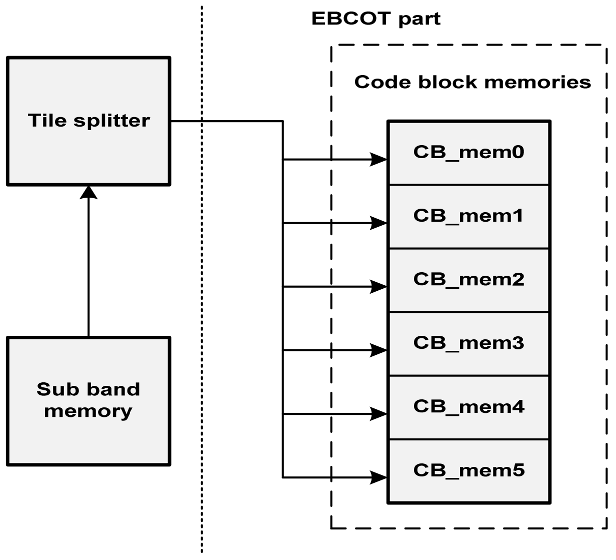

Fig. 8 Code block memories.

4.2 Switches and counter

To manage the code block memories and state variable memories outputs, three switches are used according to the stripe witch is going to be processed. Therefore, there are four switching modes for each switch block.
In this architecture we use one counter to count the columns and stripes of the code block to be processed, and generate the current pass bit plane to be encoded. This counter also allows detecting the position of the bit to be processed inside the code block.

4.3 Context generator and mux

The multiplexer chooses the context from the outputs of ZC(Zero Coding) context block, SC(Sign Coding) context block, MR(Magnitude Refinement) context block, or the hard encoded RLC(Run Length Coding) contexts (17 or 18 contexts).

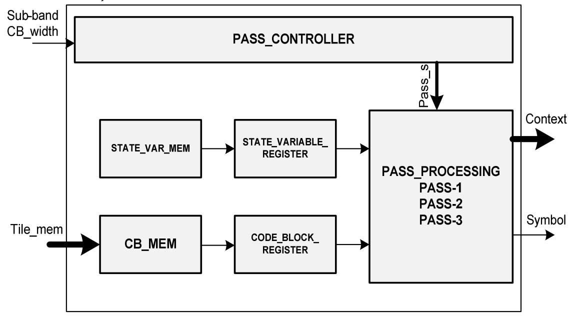

Fig. 9 Context information generator.

For the three passes of EBCOT algorithm, we propose a code block processing model shown in Fig. 9. In this architecture all passes are merged into one component. The pass to be processed is selected by the “pass” signal. This block is managed by the pass-controller block, which is based only on five states, and the symbol is encoded in 3 clock cycles. To better improve the speed of the proposed EBCOT architecture, we use the by pass mode presented in [1].

5. Implementation and experimental results

The proposed architecture was implemented in VHDL. The EBCOT algorithm was also developed using C language (ISO/IEC 15444-1 [1] compatibility) to validate the architecture by comparison with the hardware behavioral.
When synthesized and target to an FPGA ALTERA Cyclone II and Stratix III [14, 15], our design performs at 285 MHz.
Table 2 gives the complete device usage summary. We can see that our design spent 1.5K gates and 40K bits of internal memory.

5.1 Implementation Results

The proposed EBCOT architecture is tested by encoding three test images: Baboon, Jet and Lena with size of 512x512. The code block size is 64×64 or 32×32, and each coefficient can be represented by 12 bits of width.

Table 2: synthesis results of EBCOT block implemented in a altera Cyclone II and stratix III.

	Cyclone II	Stratix III
Total logic elements	1,501 (3%)	1,501 (1%)
Total registers	365	234
Total memory bits	40196 (11%)	40196 (5%)
CLK(EBCOT)	225 MHZ	285 MHZ
Operating voltage	1.8 V	1.8 V

In table 3 we show a comparison between our results and those of the most recent work [4, 16]. It is clearly shown that our proposed architecture reduces the processing time by about 45%. This decrease in the processing time are mainly explained by the saving both the wasted clock cycle in the SS method and the required additional clock cycles for memory access in the GOCS architecture.

Table 3: performance of proposed architecture compared with other techniques.

Architecture	Area (cells)	Cycles /code-block	CLK (MHz)
Single Sample[8]	631	156590	51.7
Sample-Skipping[14]	710	89170	38.6
This work	985	41250	285

A high processing frequency is achieved with a suitable number of the cycles/codes-block, thus the proposed architecture is faster than SS [4] and GOCS [16] methods, with low number of the required clock cycles, it reduces the processing time by about 45% as shown in Fig. 10. This increasing in the speed of the EBCOT algorithm is mainly due to the minimization of the number of memory access by adopting an efficient memory architecture to store the state variables and the code block data. The proposed architecture not only overcomes the complexity of state machine, but also has faster computation than PPCM.
However the proposed architecture needs some additional hardware resources, which is relatively low compared to the gain in the processing time, and represents a good compromise between speed and hardware resources for applications such as digital cinema.

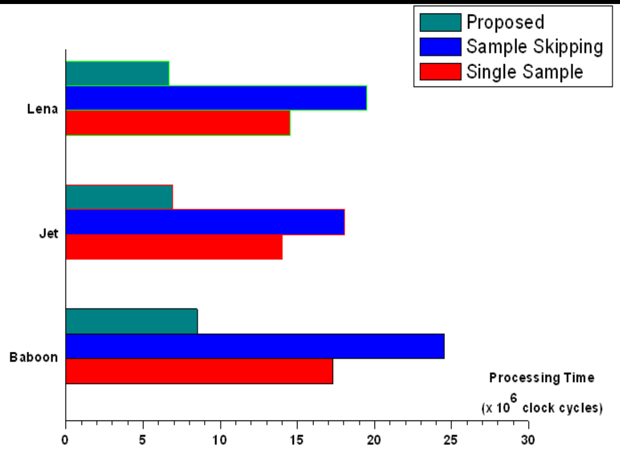

Fig. 10 Comparison of proposed architecture versus other techniques.

5.2 Integration

The proposed EBCOT architecture was first integrated in the JPEG 2000 encoder. It is designed for I-frames in standard definition television (SD, 720 x 480 30 fps) format at 54 MHz and supports high-definition television (HD720p, 1280 x 720 30 fps) format at 100 MHz. Our EBCOT architecture is capable of processing Motion JPEG2000 in real time with 12 bits Bitdepth, 4:4:4 video encoding and a compression ratio of 11.

6. Conclusion

We have proposed an efficient VLSI architecture to implement the Bit-plane coding. The design was implemented in VHDL, and synthesized and routed in an ALTERA Cyclone II and Stratix III FPGA.
This new architecture is based on a parallel access to memories, and uses a new design of context generator block. A working frequency of 285 MHz is achieved, and

the number of the required clock cycles is reduced, which increase the processing speed, by comparison with previous works.
The proposed EBCOT encoder is fully compatible with ISO/IEC 15444-1 [1], and can be adopted for supporting real-time applications rates.
The system is secure because no external memory is used and the data flow is protected during the whole encoding process. So, it can be widely used in the application of the futur-generation digital cinema.

References

[1] ISO/IEC 15444-1: Information Technology-JPEG 2000 image coding system-Part 1: Core coding system, 2000.
[2] D. S. Taubman and M. W. Marcellin, JPEG2000: Image Compression Fundamentals, Standards and Practice. Norwell, MA: Kluwer, 2002.
[3] D. Taubman, “High Performance Scalable Image Compression with EBCOT”, IEEE Transactions on Image Processing, Vol. 9, No. 7, July 2000, pp. 1158-1170.
[4] M. D. Adams and F. Kossentini, “Jasper: a software-based JPEG-2000 codec implementation,” Proc. IEEE Int. Conf. Image Processing, vol. 2, pp. 53-56, Sep. 2000.
[5] M. Rabbani, and R. Joshi, “An Overview of the JPEG2000 Still Image Compression Standard”, Signal Processing: Image Communication Journal, Vol. 17, No. 1, October 2001.
[6] M. Dyer, D. Taubman, and S. Nooshabadi, “improved throughput arithmetic coder for JPEG 2000,” accepted in IEEE Int, Conf. Image Processing, pp.2817-2820, 2004.
[7] K.-F. Chen, C.-J. Lian, H.-H. Chen, and L.-G. Chen, “Analysis and architecture design of EBCOT for JPEG2000,” Proc. IEEE Int. Symp. Circuits and Systems, vol. 2, pp. 765-768, May 2001.
[8] 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 and Systems for Video Technology, vol. 13, pp. 219-230, March 2003.
[9] Paul R. Schumacher, “An Efficient JPEG2000 Tier-1 Coder Hardware Implementation for Real-Time Video Processing”, IEEE Transactions on Consumer Electronics,Vol. 49, No. 4, November 2003.
[10] Kishore Andra, Chaitali Chakrabarti and Tinku Acharya, “A High Performance JPEG2000 Architecture”, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 13, No. 3, pp 209-218, March 2003.
[11] ISO/IEC JTC 1/SC 29/WG 1 WG1N1878, JPEG 2000 Verification Model 8.5 (Technical description), September 13, 2000.
[12] Cyclone-II platform FPGAs: Complete Data Sheet. ALTERA. [Online]. Available: http://www.altera.com.
[13] Stratix-III platform FPGAs: Complete Data Sheet. ALTERA. [Online]. Available: http://www.altera.com.
[14] Y. Li, R.E. Aly, B. Wilson and M.A. Bayoumi, “Analysis and enhancements for EBCOT in high speed JPEG2000 architectures,” Mid-west Symp. on Ckts. And Systems, vol.2, pp.207-210, Aug. 2002.
[15] Yun-Tai Hsiao, Hung-Der Lin, Kun-Bin Lee and Chein-Wei Jen, “High-Speed Memory-Saving Architecture for the Embedded Block Coding in JPEG2000”, IEEE International Symposium on Circuits and Systems, Vol. 5, pp 133-136, May 2002.

Anass MANSOURI received M.S. and Ph.D degrees in Microelectronics and Telecommunication from Faculty of sciences & technology, Fes, MOROCCO, in 2005 and 2009, respectively. He is a Assistant Professor in National School of Applied Sciences, Fes.
His major research interests include VLSI and embedded architectures design, video and image Processing.

Ali AHAITOUF received the Ph.D. degrees in electronics from the Metz University in France 1992. He is a Professor in electrical engineering department at Faculty of sciences & techniques, Fes, MOROCCO, when he obtained the Doctor Title in Physics at 1998.
His major research interests include Digital and Analog VLSI architecture, EMC Simulation and Physics of Semiconductor Components. He is managing the Microelectronics and Components research group.

Farid ABDI received the Ph.D. degrees in Physics from the Metz University in France 1992. He is a Professor in electrical engineering department at Faculty of sciences & techniques, Fes, MOROCCO.
His major research interests include Optical Components, Image, Audio and video processing. He is managing the optical research group.