EBCOT coprocessing architecture for JPEG2000

Huakai Zhang, Jason Fritts,
Dept. of Computer Science and Engineering, Washington University,
One Brookings Drive, St. Louis, MO, USA 63130
Email: {hzhang, jefritts}@cse.wustl.edu

ABSTRACT

JPEG2000 is the latest still-image coding standard. It was designed to overcome the limitations of the original JPEG standard and provide high quality images at low bit rates. The JPEG2000 algorithm is fundamentally based on the Discrete Wavelet Transform (DWT) and Embedded Block Coding with Optimal Truncation (EBCOT). Both of the algorithms are computationally intensive and require significant memory bandwidth. In this paper, we propose a JPEG2000 hardware/software co-processing architecture that complements existing JPEG2000 software packages with efficient dedicated hardware for EBCOT tier-1 coding, enabling significant speedup.

Keywords: JPEG2000, EBCOT, Coprocessing

1. INTRODUCTION

Approved as an international standard (ISO/IEC 15444) [1] in January 2001, JPEG2000 is an emerging image coding standard for the next generation of digital imaging. It offers a host of features beyond the capabilities of conventional JPEG [2]. Superior low bit-rate performance, progressive transmission by pixel accuracy and resolution, region of interest (ROI) coding, random codestream access and processing, error resilience, and both lossy and loseless compression are among the important features of the JPEG2000 standard. JPEG2000 is targeted for rapidly growing diverse imaging applications, e.g. next-generation digital cameras, scanners, remote sensing, web browsing, medical imagery, digital photography, and wireless imaging devices.

All these features are possible because of the 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, necessitating dedicated hardware to perform the computationally complex portion of the applications in many embedded systems. Therefore, a thorough investigation of the algorithms is needed to find an efficient hardware implementation of JPEG2000 coding standard. Through runtime profiling and analysis, we find that the DWT and EBCOT functions are the most critical blocks in terms of computation. While a number of studies have already targeted the DWT for hardware implementations [11,12], the EBCOT algorithm is relatively new, with minimal research on hardware implementation. In our coprocessing architecture, it was decided that the tier-1 of EBCOT was best suited for this implementation because of its high computation complexity and relatively simple modularity. Along with some existing software on the host processor, it provides a complete JPEG2000 compression solution.

In this paper, we propose a co-processing architecture for JPEG2000. This involves replacing the software implementation of the EBCOT tier-1 coder [3] with dedicated hardware. The hardware implementation primarily consists of two parts: coefficient bit modeling and arithmetic entropy coding. The two modules interface with each other via a FIFO (First In, First Out) buffer. The hardware architecture has been implemented in VHDL and its performance has been evaluated.

The remainder of the paper is organized as follows. In section 2 we present a brief introduction of JPEG2000 Part I coder. The related work is discussed in section 3. In section 4 we propose the EBCOT coprocessing architecture and discuss some implementation method. The experimental results are illustrated in section 5. Finally, the paper is concluded in section 6.

2. JPEG2000 OVERVIEW

As with most of image compression algorithms, JPEG2000 is composed of a transform stage followed by a quantization stage and an entropy coding stage. The block diagram of JPEG2000 baseline is shown in Figure 1. The reader is referred to [1] for more details. We should note that JPEG2000 operates on independent, rectangular, non-overlapping tiles. The tile sizes are arbitrary, up to and including the entire image.

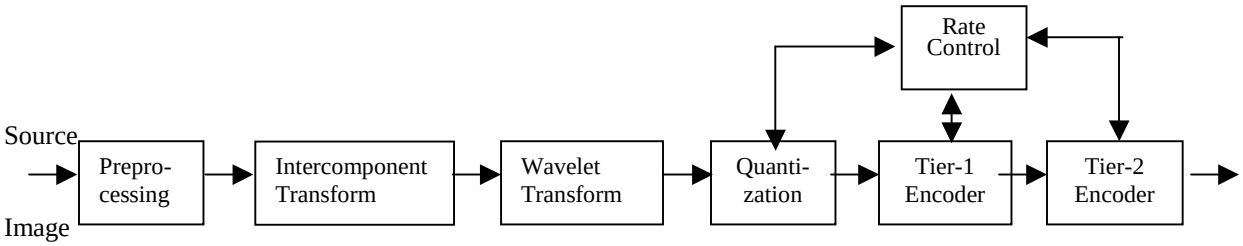

Fig. 1. JPEG2000 block diagram

Pre-processing: In the first step, the encoder transforms the input sample data to a nominal dynamic range so that it is approximately centered about zero. If the input sample data is signed, then assume they are centered at zero. Otherwise, unsigned sample values in each component are level shifted (DC offset) by subtracting a fixed value from each sample to make its value symmetric around zero.

Inter-component Transform: JPEG2000 optionally allows for an inter-component to be applied to the image after it has been level-shifted. This transform helps decorrelate the separate components for multi-component images. The common example of this transform is converting RGB data to YCbCr data.

Wavelet Transform: In this step, the Discrete Wavelet Transform (DWT) is applied on each tile to decompose it into a number of wavelet subbands at different levels and resolutions. In JPEG2000, the lifting-based DWT implementation is applied. There are two kinds of wavelet transform: reversible 5/3 integer wavelet transform and irreversible 9/7 floating-point wavelet transform. DWT is an important process to generate resolution-salable bit-streams, and decompose the spatial correlation. The output of this stage is a set of transformed coefficients.

Quantization: Quantization is used only in lossy compression. The wavelet coefficients in each subband are scalar quantized. The quantization step size is calculated based on the dynamic range of the subband values and constants defined in the standard. A purpose for quantization is to reduce in precision of subband coefficients so that fewer bits will be needed to encode the transformed coefficients.

Tier-1 Coder: Each subband is divided into code blocks which are entropy coded independently. The EBCOT entropy coding in JPEG2000 consists of two tiers. The tier-1 coder includes coefficient bit context modeling and arithmetic coding of the bit-plane data on block samples, and generates embedded block bit-streams.

Tier-2 Coder: Tier-2 coder operates on the compressed information of every block to arrange their contributions in each quality layer, in a process referred to as packetization. This data-ordering stage plays a vital role in creating resolution and SNR (signal to noise ratio) scalable compressed bit-stream.

Rate Control: To meet the target bit-rate constraints under lossy compression, the rate control module adjusts the quantizer step sizes and/or discards some of the coding pass information. In other words, the bit-stream of each block can be truncated to a variety of discrete lengths to render the possible best rate-distortion image with associated specific compression ratio.

3. RELATED WORK

JPEG2000 provides high compression with image quality superior to all other existing coding standards. However, it requires remarkable amounts of computation resources, which has led to growing interest about efficient and optimized implementations of the JPEG2000 standard. A number of JPEG2000 software implementations are currently available. Jasper[13], JJ2000[14] and Kakadu[15] are the three popular software implementations of Part 1 of the standard. Jasper

and JJ2000 are available free of charge and downloadable from the web, while Kakadu requires a license fee and offers better performance. There are also some commercial hardware implementations of the computationally intensive part of the standard. Analog Devices Inc. [16] launched the first chip of JPEG2000 accelerator - ADV-JP2000 in 2001. DSPworx [17] have a similar product of DWS2000S. Synopsys/InSilicon [18] have developed a JPEG2000 encoder Intellectual Property (IP) cores. Alma Technologies [19] and Amphion Semiconductor Ltd. [20] are also known to be developing their encoder cores. As the standard is applied in more and more areas, this number will increase without doubt.

In academic community, several papers [4, 5, 6, 7] have been published describing efficient hardware implementation of JPEG2000 coding standard. [4] and [5] presented a system level architecture for JPEG2000 with a strong focus on the bit plane coder, which described an efficient implementation of the coder by reducing memory interaction. [6] and [7] presented a high speed memory efficient EBCOT architecture for JPEG2000, which described some speedup methods, such as skipping no-operation samples and good memory arrangement. Some ideas of the work in this paper are referenced from these papers. The differences between this paper and them are several. Firstly, this paper has a more specific goal – to perform hardware coprocessing to accelerate one of the existing JPEG2000 software implementations. Secondly, this paper is a smaller implementation. It implements only a subset of the JPEG2000 standard identified by detailed profiling of execution time.

4. SYSTEM ARCHITECTURE

Before we get to the implementation details, we want to discuss the motivation of EBCOT coprocessing with dedicated hardware.

First, dedicated hardware is needed for the most computationally demanding stages to achieve high performance. A significant advantage of the hardware implementation over the software implementation is the ability of hardware to perform parallel processing. JPEG2000 operates on independent, non-overlapping code blocks, which makes the entropy coding part of EBCOT very suitable for parallel hardware implementation, since it is possible to encode several code blocks concurrently.

Second, from the documentation of JPEG2000 standard, it is foreseeable that the wavelet transform and EBCOT would be the most critical blocks in terms of required computational resources. Table 1 shows the runtime profiling results of JPEG2000 coding algorithms using Jasper [13] to encode the image monarch.ppm with the compression rate 0.1. The image size is 768*512 with 3 components (RGB), and the bit depth per component is 8. It is observed that EBCOT and the wavelet transform consume more than 80% of total execution time in software implementation. Therefore, it would be relatively straightforward to speed up the execution of JPEG2000 by exploiting some techniques in hardware implementation, e.g. pipeline, and parallel processing, without incurring too many overheads.

Table 1: Profiling results

	(5,3) filter	(9,7) filter
Preprocessing	2.77%	2.84%
Intercomponent transform	2.01%	3.81%
Wavelet transform	28.55%	62.01%
Quantization	0%	1.38%
EBCOT(tier 1 and 2)	66.67%	29.96%

Multimedia applications are generally understood to have certain characteristics unique from typical general-purpose applications. Such characteristics include intense computational loads, large amounts of streaming data, significant processing regularity that affords extensive parallelism, real-time constraints, and a tendency towards small integer data types [8]. As a typical multimedia application, JPEG2000 is believed to own most of the characteristic, which suggests us to take advantage of these characteristics to reduce the execution time of JPEG2000 by careful design. For example,

some existing implementations of JPEG2000 are not optimized for cache locality. We expect some degree of speedup with the design modification to improve cache locality. Efficient memory management and other strategies also help reduce the JPEG2000 processing time.

Most of the time spent in the wavelet transform is caused by cache misses due to discrepancies between memory access patterns of vertical and horizontal filtering in 2-D DWT (in fact, DWT has a relatively uncomplicated symmetrical nature). Better memory access pattern can be obtained by interchanging the loops and other means, and significant research already exists for wavelet hardware. Therefore, we are focusing primarily on EBCOT entropy coder. EBCOT is a two-tiered context-based adaptive arithmetic coder. EBCOT tier-1 has the highest computation complexity due to bit-wise processing and non-regular scan order, which make it well suited for hardware processing. EBCOT tier-2 contains ordering and decision, which is suitable for software implementation. So we focus only on the acceleration of EBCOT tier-1.

The proposed coprocessing architecture is shown in Figure 2. The architecture primarily consists of code block status memory, context bit modeling, context FIFO, arithmetic coder and bit-stream memory in a sequential fashion. The input to the coprocessor is a code block status buffer and the outputs are compressed codestreams. Existing software implementation is modified to allow it to produce a data file containing code block status information that EBCOT tier-1 consumes, and to receive the compressed bit-stream EBCOT tier-1 sends out. A VHDL test-bench was written to obtain data to apply the inputs, and to yield the compressed bit-stream at its output. Though the hardware implementation only covers a subset of JPEG2000 standard, it provides the preliminary software acceleration method for future improvement.

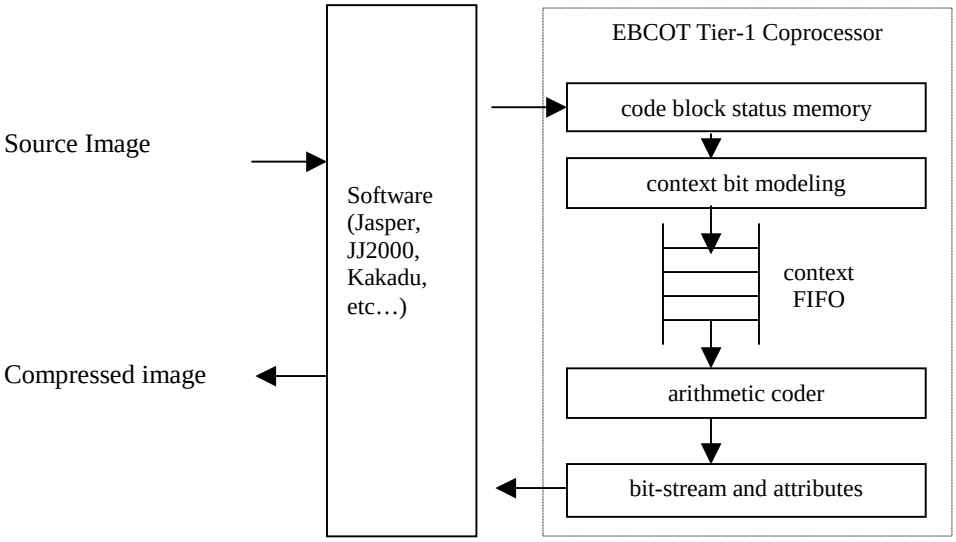

Fig. 2. JPEG2000 co-processing architecture

4.1. Code block status memory

Code block data produced by the software implementation of the JPEG2000 codec is stored in the *code block status memory*. The context bit model reads the block status data, including sign and magnitude bits, from the memory block stripe by stripe (a stripe is 4 consecutive rows of pixel bits in a code block bit-plane). Within a stripe, samples are scanned column by column as shown in Figure 3. A good memory management system proposed in [6] has been carefully designed to suit the needs of the non-regular scan order of the context bit modeling process. The carefully designed data arrangement in code block status memory allows using shift register glide over the memory with the stripe structure required for bit-plane coder. The obtained structure enables the possibility to achieve high throughput due to reduction of memory bandwidth requirements.

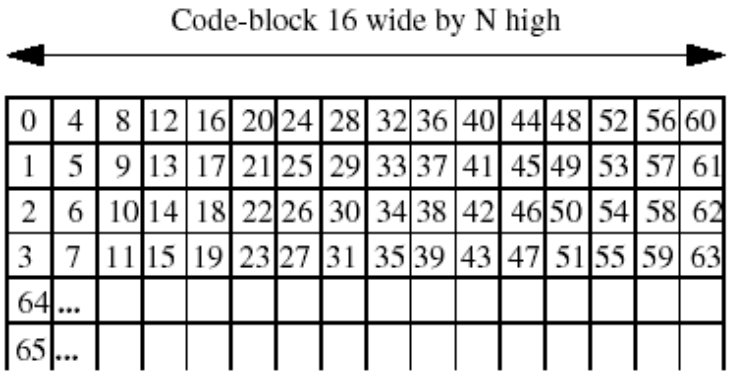

Fig. 3. Code block scanning pattern

4.2. Context bit modeling

Context bit modeling uses bit-wise processing to scan over the code block, and generates contexts according to the wavelet coefficients. It is also known as a bit-plane coder. Every bit-plane is coded using three passes: (1) significance propagation, (2) magnitude refinement, and (3) clean-up pass. Each coefficient bit is coded in exactly one of the three coding passes. Which pass a coefficient bit is coded in depends on the conditions for that pass. Each of three passes outputs a series of binary symbols, and these symbols are entropy coded using arithmetic coding.

The block diagram of this block is shown in Figure 4. Each context generation for each bit needs to reference its 8 neighboring bits in the bit-plane. Thus, significant memory and storage bandwidth is required in the bit-plane coder. Three states for each coefficient are maintained for three-pass context bit model. σ denotes whether the coefficient is significant, η denotes whether the coefficient is processed in the current bit-plane, and χ denotes the sign of the coefficient. The context symbols are generated according to the three state variables and the significance information from 8 neighboring samples for the coefficient bit in current bit-plane.

As mentioned before, an efficient memory arrangement is very important design to get speedup. The parallelism can be achieved by checking all 4 samples of a column concurrently to reduce the average number of memory access within a coding pass. Because independent relationship exists between the three coding passes, it also makes parallel processing

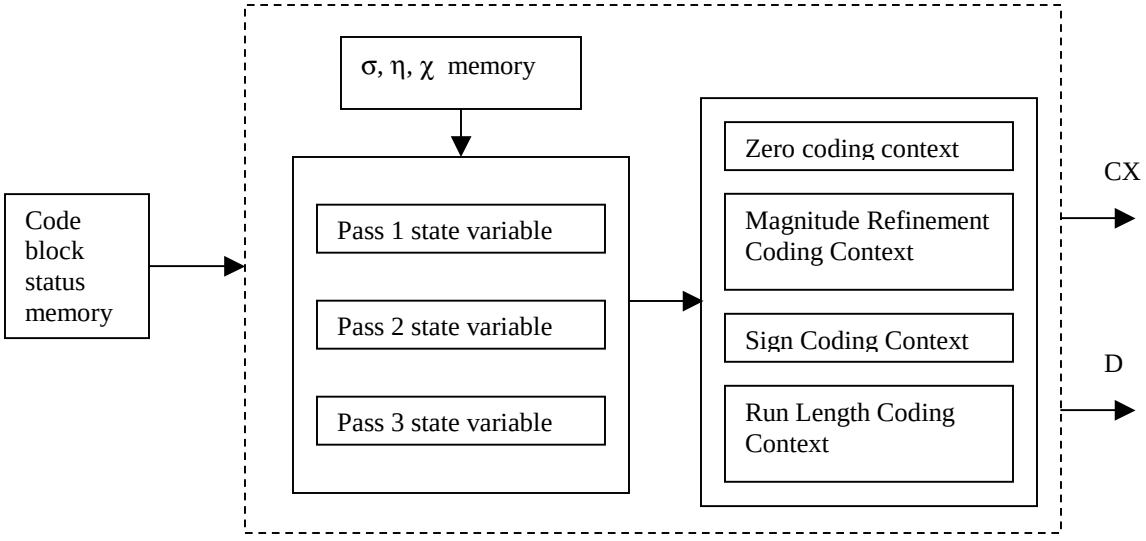

Fig.4. Context bit modeling

of different coding passes possible. We must note that magnitude refinement coding is only needed in pass 2, zero coding and run length coding is only needed when signs are encoded. Also there is no conflict between pass 2 and pass 3, or between pass 2 and pass1. Therefore, performance improvement can be obtained by low-level parallel processing of pass 2 and pass 1/pass 3 in different context windows.

4.3. Context FIFO

The *context FIFO* is simply a buffer where the temporary context information from context bit modeling is stored. The width of the FIFO is equal to the length of a context/data pair. The size of the FIFO can vary according to the processing throughput difference between the context bit modeling and arithmetic coder.

4.4. Arithmetic coder

The specific type of arithmetic coder in JPEG2000 is known as an "MQ coder"[9], an efficient, statistical binary arithmetic coder based on the recursive probability interval subdivision of Elias coding. It compresses wavelet coefficients into a bit-scream using context/data pair from bit modeling. The primary advantage of the MQ coder is that the probabilities associated with LPS (Less Probable Symbol) and MPS (More Probable Symbol) can be adopted. For every context label, there is a corresponding state machine associated with it. The context from bit modeling is used to index into a look-up table of LPS probability value (Qe). This block is control-intensive because it involves a complex probability estimation process. The statistics are maintained using a Qe-value look-up table. When the length of the probability interval (A register) falls below a certain minimum size(0.75), the interval must be renormalized to increase it above the minimum. The probabilities associated with the MPS and LPS also need to be updated when renormalization has occurred. The reader is referred to Appendix C of JPEG2000 Part I International Standard (ISO/IEC 15444)[1] for more detailed information.

The block diagram of this block is shown in Figure 5. The input context/data pairs are read from context FIFO. The key data path registers were identified by analyzing the MQ coder's flowcharts in the standard. Then control signals were created to select the operation on the datapath registers. Again, we can take advantage of hardware implementation to perform parallel processing. For example, both A(interval) and C(code word) registers can be updated in parallel to reduce the number of cycles. Similar to JBIG pipelined implementation [10], this block can also be implemented in multi-stage pipeline fashion to obtain potential speedup and reduce the memory requirement for context label outputs. A simple pipelined architecture for arithmetic coder is also illustrated in Figure 5. The output of compressed bit-stream is written in a buffer for the future access from software.

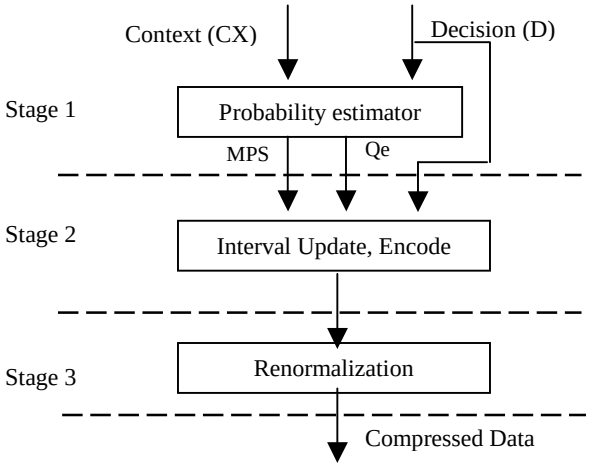

Fig. 5. Arithmetic encoder

4.5. Bit-stream memory

The compressed bit-stream obtained during arithmetic coding is provided to the bit-stream memory. It allows the software implementation to perform post-processing (EBCOT tier-2, etc) on the bit-stream until the whole compression process is finished.

5. EXPERIMENTAL RESULTS

The system level architecture has been developed and validated using VHDL language. The implementation of the EBCOT tier-1 encoder successfully passed all functional tests on real image data using the simulation tools--ModelSim. The output bytes that were generated by the VHDL encoder in simulation matched exactly the expected output stream of bytes obtained from Jasper [13] encoder.

Though the hardware implementation only covers a subset of JPEG2000 standard, it provides the preliminary software acceleration method for future improvement. It is expected from synthesizable structure view that the VHDL encoder could perform its computation in only a fraction of the processing time taken by the Jasper [13] arithmetic coder.

Mentor Graphic's Spectrum synthesis tools are used to synthesize VHDL codes into gate level codes. These gate level codes are then placed and routed to a 0.5 micron ASIC environment using Mentor Graphic's IC station. It is worth noting that synthesized maximum achievable clock frequency is of 45MHz for the current implementation technology and VHDL code. It takes around 0.82 million cycles in simulations to compress a 128*128 grayscale (8bit) image (lena.ppm), while it requires around 2.1 millions cycles to compress a 128*128 true color(24 bits) image (lena.ppm). We also tested 64*64 grayscale and true color image (baboon.ppm), which can be compressed in around 0.23 and 0.47 millions cycles, respectively. Generally, if the system works at 45MHz, it can process around 0.9 millions grayscale pixels or 0.4 millions true color pixels per second. In other words, it can process 150*200 grayscale video or 100*150 true color video at 30 frames per second. Expected on-chip power consumption of the simulated circuit is obtained from Mentor Graphic's Mach PA simulation tools. The power consumption is calculated by Vcc*average_current through the circuit, where the Vcc is the operating voltage of the simulated circuit. Preliminary measurements estimate power consumption for the EBCOT tier-1 coprocessor as 222.55mW (=40.51mA*5V).

6. CONCLUSIONS

In this paper, the software implementation of the JPEG2000 codec was profiled to find the critical stages in terms of computation. Analysis of the JPEG2000 encoder indicated that the EBCOT tier-1 encoder is best suited for hardware acceleration. Synthesis of the hardware EBCOT tier-1 codec in 0.5 micron technology resulted in a maximum clock frequency of 45MHz, processing rate of 0.4 millions true color pixels per second, and preliminary power consumption estimates of 223mW in a 5V operating range.

During the process of implementation, we found that the new emerging JPEG2000 compression standard is much more complex than today's JPEG standard. The new features provided by JPEG2000 are at the cost of more computational resources. To design a cost-effective and high-performance system is not an easy task. Power consumption, pipeline difficulty, chip size and memory usage are the main factors to impede the development of JPEG2000 chips. The transition to the new standard will be depended on the availability of JPEG2000 products and the size of the market.

The implementation of EBCOT tier-1 in hardware represents only a first step toward investigating the accelerating software implementation. As a future research we also plan to investigate the potential optimization of JPEG2000 in various areas, such as low-cost embedded system, reconfigurable hardware, DSP and Media Processor. Though currently the standard is not prevalent, it still is a promising standard for the future's demanding compression applications.

REFERENCES

1. JPEG2000 Image Coding System, JPEG2000 Part I, International Standard, (ISO/IEC 15444-1), Dec. 2000.
2. C. Christopoulos, A. Skodras, and T. Ebrahimi, "The JPEG2000 still image coding system: an overview", *IEEE Transactions on Consumer Electronics*, vol. 46, no. 4, pp. 1103-1127, 2000.

3. D. Taubman, “High performance scalable image compression with EBCOT”, *IEEE Transactions on Image Processing*, vol.9, July 2000.
4. K. Andra, C. Chakrabarti and T. Acharya, “A High performance JPEG2000 Architecture”, *Proc. International Symposium on Circuits and Systems (ISCAS’2002)*, Scottsdale, Arizona, U.S.A, May 2002.
5. K. Andra, T. Acharya and C. Chakrabarti, “Efficient VLSI implementation of bit plane coder of JPEG2000”, *Proc. of SPIE Applications of digital image processing XXIV*, vol. 4472, pp.246-257, San Diego, California, U.S.A, August 2001.
6. Kuan-Fu Chen, Chung-Jr Lian, Hong-Hui Chen, Liang-Gee Chen, “Analysis and Architecture Design of EBCOT for JPEG2000”, *Proc. International Symposium on Circuits and Systems (ISCAS'2001)*, Sydney, Australia, May 2001.
7. Chung-Jr Lian, Kuan Fu Chen, Hong-Hui Chen, Liang Gee Chen, “Analysis and Architecture Design of Lifting Based DWT and EBCOT for JPEG 2000”, *Proc. International Symposium on VLSI Technology, Systems and Applications (VLSI-TSA'2001)*, Hsin-Chu, Taiwan, April 2001.
8. Jason Fritts, Wayne Wolf, and Bede Liu, “Understanding multimedia application characteristics for designing programmable media processor”, *Proc. Of SPI Photonics West, Media Processor’99*, San Jose, California, U.S.A, January 1999.
9. J. L. Mitchell and W. B. Pennebaker, “Software implementations of the Q-coder”, *IBM J. of Res. Develop*, vol. 32, No. 6, pp. 753-774, Nov. 1988.
10. M. Oshita, M. Tarui, T. Onoye, I. Shirakawa: “Pipelined Implementation of JBIG Arithmetic Coder”, in *Proc. International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC '99)*, pp. 470-473, Sado, Japan, July 1999.
11. A. Grzeszczak, M. K. Mandal, S. Panchanathan: “VLSI Implementation of Discrete Wavelet Transform”, *IEEE Transactions on VLSI Systems*, Vol. 4, No. 4, pp. 421-433, Dec 1996.
12. G. Dimitroulakos , N. D. Zervas, N. Sklavos and C.E Goutis: “An Efficient VLSI Implementation for Forward and Inverse Wavelet Transform for JPEG2000”, in *Proc. Of 14th IEEE International Conference on Digital Signal Processing (DSP'02)*, Greece, July, 2002.
13. http://www.ece.uvic.ca/~mdadams/jasper.
14. http://jj2000.epfc.ch.
15. http://www.kakadusoftware.com/.
16. http://www.analog.com.
17. http://www.dspworx.com/.
18. http://www.insilion.com/.
19. http://www.alma-tech.com/.
20. http://www.amphion.com/.