Analysis and Architecture Design of JPEG2000

Liang-Gee Chen, Chung-Jr Lian, Kuan-Fu Chen, and Hong-Hui Chen

DSP/IC Design Lab., Department of Electrical Engineering,
National Taiwan University, Taipei 106, Taiwan, R.O.C.
E-mail: lgchen@video.ee.ntu.edu.tw

ABSTRACT

Analysis and architecture design of the key modules in JPEG2000 are presented in this paper. For Discrete Wavelet Transform (DWT), a lifting based DWT core for the default 5-3 and 9-7 filters in part I of JPEG2000 is proposed. Folded architecture is adopted in DWT to reduce the hardware cost and to achieve the higher hardware utilization. For Embedded Block Coding with Optimized Truncation (EBCOT), column-based coding architecture of Tier-1 block coding engine is proposed. The context formation efficiency is increased by adopting two speedup methods. The computation cycle of the block coding engine is reduced to about 40% of previous work.

1. INTRODUCTION

JPEG2000 [1] is an emerging next-generation still image compression standard. It is designed to provide not only better compression performance than the existing JPEG standard [2], but also new features and functionalities unavailable in JPEG. The applications of JPEG2000 includes the digital still camera, internet and wireless transmission. Different from JPEG, JPEG2000 uses DWT as the transform coder, and EBCOT [3] as the entropy coder. Fig. 1 compares the functional block diagram of JPEG and JPEG2000. Wavelet transform is a subband transform. It transfers the whole image, instead of 8x8 blocks in JPEG, from spatial domain to frequency domain. To achieve efficient lossy and lossless compression within a single coding architecture, two wavelet transform kernels are supported in part one of JPEG2000. The 5-3 reversible and 9-7 irreversible filters are chosen for lossless and lossy compression, respectively. After wavelet transform, the coefficients are scalar quantized if lossy compression is chosen. Then, coefficients are entropy coded by EBCOT. EBCOT is a two-Tier coder. Tier-1 is a context based arithmetic encoder, and Tier-2 forms the bitstream according to the rate-distortion information. Different from Huffman coding, which can be implemented by table look-up, the implementation complexity of the adaptive arithmetic coder is much higher. In single pass JPEG coding, the compression ratio can not be accurately controlled. In JPEG2000, EBCOT Tier-2 can reorder the sub-bitstream segments contributed from each code-block according to the rate-distortion optimization criteria, and generate an JPEG2000 bit-stream with embedded characteristics. Therefore, the compression ratio can be accurately controlled by users, and the bitstream can be progressively decoded.

Since JPEG2000 is a brand-new standard, researchers are still working hard to find the optimized design. Software companies have provided their JPEG2000 codec, and kept promoting the adoption of this new standard. As for the hardware design, ADV-JP2000 [4] is claimed to be the first commercial chip to support the JPEG2000 compression. It is designed to be a JPEG2000 co-processor. DWT and EBCOT Tier-1 are implemented in this chip, and EBCOT Tier-2 is executed on the host PC or extra processor. Other researchers are trying another solutions such as DSP, FPGA and ASIC implementations. As we can see in the run time profile of JPEG 2000 in Table 1, EBCOT Tier-1 (context-based arithmetic encoding) is the most time consuming module, and DWT is the second one. In this paper, we mainly focus on the analysis of DWT and EBCOT Tier-1, and then propose architectures for these two parts.

The rest of this paper is organized as follows. In Section 2, the algorithm of lifting based DWT and its architecture design are discussed. In Section 3, the algorithm of EBCOT Tier-1 is detailed analyzed, and speedup schemes for the context formation are proposed. The experimental results and chip implementation of EBCOT Tier-1 are depicted in Section 4. Finally, a conclusion is given in Section 5.

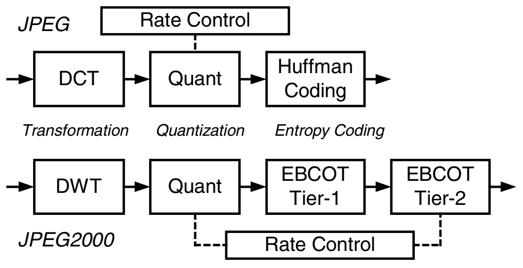

Fig. 1 Functional block diagrams: comparisons of JPEG and JPEG2000 encoder.

Table 1 Run time profile of JPEG2000 encoder (1,792x1,200 images, 5-level DWT, 9-7 filter, single layer, profiling platform: PentiumIII-733, 128MB RAM, Visual C++ and Windows ME)

Operation	Single Component		3 Components(RGB)	
	Lossless	Lossy	Lossless	Lossy
Color Transform			**0.91**	**14.12**
Wavelet Transform	**10.81**	**26.38**	**11.9**	**23.97**
Quantization		**6.42**		**5.04**
EBCOT Block Coding	**71.63**	**52.26**	**69.29**	**43.85**
context formation	*51.88*	*37.91*	*49.96*	*31.79*
arithmetic encoder	*19.75*	*14.35*	*19.33*	*12.06*
EBCOT R-D Optimization	**17.56**	**14.95**	**17.9**	**13.01**
layer formation	*10*	*9.52*	*9.94*	*7.95*
marker insertion	*7.56*	*5.43*	*7.96*	*5.06*

2. DISCRETE WAVELET TRANSFORM

2.1 Lifting-based Discrete Wavelet Transform

JPEG2000 is designed to support lossy and lossless compression within the same coding framework. Therefore, a compact architecture for both 5-3 and 9-7 filters is necessary. Different from convolution based implementations, the newly proposed lifting-scheme [5][6] for the computation of 5-3 and 9-7 filters has lower computational complexity than the classical one. There are some other significant features of lifting scheme. First, lifting scheme allows in-place computation of the wavelet

2001 IEEE International Conference on Multimedia and Expo
ISBN 0-7695-1198-8/01 $17.00 © 2001 IEEE

transform. The original signal can be replaced with the calculated wavelet transform coefficient. Second, no explicit boundary extension is needed. The symmetry mirroring is achieved by adjusting some coefficients at proper boundary positions.

2.2 Architecture Design of DWT

More and more lifting-based DWT architectures are proposed [7]-[9]. The basic processing unit for lifting scheme is shown in Fig. 2. Using one basic processing unit for prediction and another one for update forms one stage prediction and update functions. For 5-3 filter, there is only one stage of prediction and update. However, in 9-7 filter, there are two stages. In [9], for 5-3 filter, the extra layer is bypassed by setting the redundant lifting coefficients of extra layers to zero.

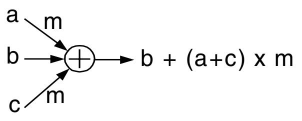

Fig. 2 Basic processing unit for lifting scheme

Another considerations are the input data rate. For low cost considerations, the folded architecture, as shown in Fig. 3, is proposed [10] under the assumption that only single read port and write port memory is available, and only single-phase clock signal is used for the system. Input data are read from memory one by one. After serial to parallel conversion, data are processed in pairs. The second stage of 9-7 filter operation is folded onto the same hardware. Stage 1 and stage 2 operations are interleaving. An extreme alternative is that the number of inputs are equal to the width or length of the images. This idea can be found in a RAM based implementation. There is always a trade-off between area and speed. The architecture design of DWT highly depends on the design of EBCOT since it is the bottleneck of the JPEG2000 system. A word in DWT is just one datum, but a N-bit datum becomes N data to be processed in EBCOT.

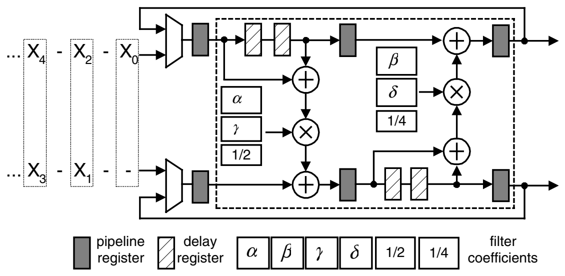

Fig. 3 Proposed folded architecture for both 5-3 and 9-7 filters

As for the extension of 1-D DWT core to 2-D DWT, the proposed 1-D DWT core can be used in the separable 2-D DWT implementation, and the line based [11] data flow is a feasible solution. The row and column filtering operations are interleaving, and the column filtering operations are executed as soon as possible. However, since EBCOT is not a line based entropy coder, a buffer is always necessary for the line scan to code-block scan conversion between DWT and EBCOT. The buffer size will depend on the code-block size, more specifically, the code-block height, and the maximum image width.

3. EBCOT TIER-1: BLOCK CODING

3.1 EBCOT Tier-1 Algorithm and Analysis

EBCOT is a two-tier coder. Tier-1 is an adaptive context-based arithmetic encoder, and Tier-2 is for bitstream formation according to the rate-distortion information from Tier-1. The input to EBCOT Tier-1 is code-block by code-block. The data in each code-block are arithmetic encoded to be a sub-bitstream. This is the reason EBCOT Tier-1 is called a block coding engine. The block coding engine can be viewed as two parts, Context Formation (CF) and Arithmetic Encoder (AE). CF scans all bits in code block in a specific order, and generates contexts for each bit. AE encodes each bit according to contexts generated by CF. The block coder is found to be the bottleneck of JPEG2000. The reason comes from the fact that EBCOT process the data bit-by-bit. The bit-level processing characteristics make it very inefficient to be implemented on a general processor.

EBCOT encodes the quantized wavelet coefficients bitplane by bitplane from MSB to LSB. The wavelet coefficients are converted to sign-magnitude format after quantization. Due to the consideration of compression performance, every bit-plane is further decomposed into three passes. Each bit in a bit-plane is encoded in one of the 3 passes, and the criterion is the expected importance of this bit. Fig. 4 shows the decision flow. Pass 1 is Significant Propagation Pass. Those bits having at least one significant neighbor are coded in this pass. They are viewed as more important bits, and are sent first. Pass 2 is Magnitude Refinement Pass. All significant samples are coded in this pass. The last pass is Clean Up Pass. Samples not coded in first two passes are encoded in this one. At the beginning, the first bits of all samples are always coded in pass 3. Once a sample becomes significant, all the following bits of this sample will be coded in pass 2. Another situation is that the sample itself has not been significant yet, but at least one neighbor has been significant. Then, this bit is to be coded in pass 1. After the sample itself becomes significant, it is then coded in pass 2. Fig. 5 illustrated the revolution map.

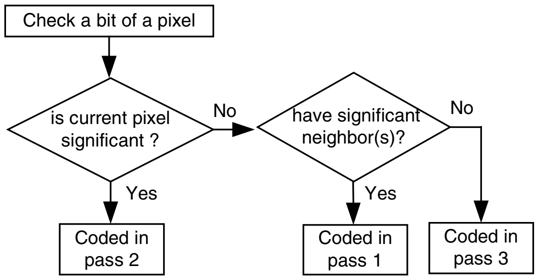

Fig. 4 Flow chart of the checking to determine in which pass this bit should be coded

According to this algorithm and the statistical data shown in Fig. 6, it can be found that the distribution is not uniform. At the beginning, most bits in a bit-plane belong to pass 3. After coding several bit-planes, more and more bits are coded in pass 2 since samples become significant at later bit-planes as long as the magnitude of this sample is not zero. The number of bits in pass 1 increases temporarily and then decrease. It is expected that the bits to be coded in pass 2 in the earlier bit-planes and those in pass 3 in later bit-planes are sparse. Since the arithmetic encoder in the block coding engine is the bottleneck, the CF should continuously feed data to AE in order to increase the throughput. A sequential checking of every bit in a bit-plane in every pass

2001 IEEE International Conference on Multimedia and Expo
ISBN 0-7695-1198-8/01 $17.00 © 2001 IEEE

will waste much time and have lots of "bubbles." The worst case is that every bit has to be checked for three times. Speedup schemes are therefore necessary for the context formation.

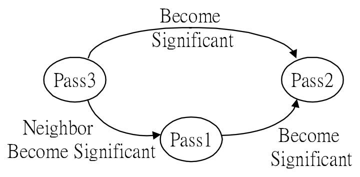

Fig. 5 Revolution map of the bits in a sample

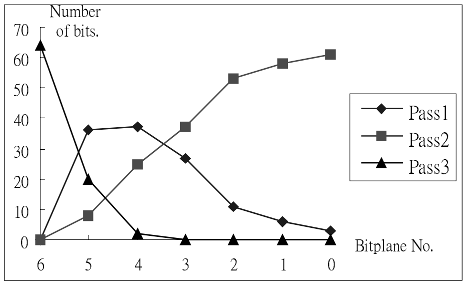

Fig. 6 The distribution of number of bits coded in the 3 passes (A 8x8 LL subband from 256x256 Lena image)

3.2 Design of the Block Coding Engine

Column-based architecture for context formation is proposed to check four bits in a column simultaneously. As for the arithmetic encoder, pipeline and look-ahead scheme are adopted to enhance the performance. Our strategy to speedup the block coder is to reduce the time required to check the samples in context formation. To achieve this, we have to keep processing NBC (Need-to-Be-Coded) samples, and skip no-operation samples.

The design of context formation is to support input data to the arithmetic encoder as continuously as possible. Data are supplied to the context formation processing elements (PEs) one column (four bits) at a time. There are two advantages of column-based operation. First, samples in a column can be checked simultaneously, so speedup methods proposed can be applied. Second, higher data reuse in significant and sign variables is achieved. Memory access frequency of these variables can therefore be reduced. Based on column-based operation, the two speedup methods are described below.

1) **Sample Skipping (SS)**: By column-based coding, samples in a column can be parallel checked to see if they are Need-to-Be-Coded (NBC) samples. NBC samples are processed directly and sequentially, while no-operation bits are skipped.
2) **Group-Of-Column Skipping (GOCS)**: In some bit-planes, successive columns without any NBC bits are possible. To further improve processing speed, these no-operation columns should be skipped. Due to the significant propagation in pass 1, we cannot decide whether one column is a no-operation column before previous neighboring column is coded. Therefore, columns in pass 1 have to check one by one. After coding pass 1, all NBC columns of pass 2 and pass 3 are decided. By grouping some columns together, this group of columns can be skipped if there is not any NBC bits in it. According to the simulation, eight columns grouped together has the better performance. A small overhead is the memory needed for the storage of the GOC-skip-or-not information for pass 2 and pass 3. For a 64 by 64 code-block, the memory size is 256 bits for the two passes.

One extreme case is happened when there is not any NBC bits in the whole pass. Once all the samples are significant, all the following bits are to be coded in pass 2. Pass 1 an Pass 3 , therefore, contain zero bits. Another situation is that once there is not any NBC bits in pass 3, the following bit-planes of pass 3 will no longer contain any bits. The idea can be implemented using two flags to record the two kinds of status.

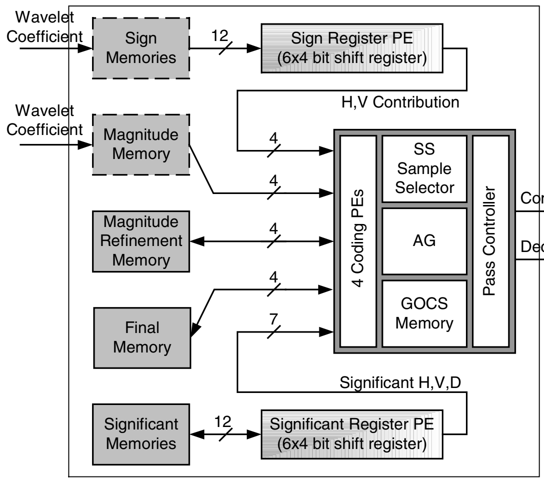

Fig. 7 Context formation block diagram

The block diagram of Tier-1 context formation is shown in Fig. 7. Quantized wavelet coefficients are loaded once a code-block from *Frame Memory* into *Code Block Memory*. The sign bit-plane is separated from the magnitude bit-planes. This code-block memory is necessary since every samples have to be accessed for many times. Contexts corresponding to every single bit are generated, which are the information needed for the adaptive arithmetic encoder. *State Variable Memories* are used for storing three different kinds of state variables (for magnitude refinement, final, and significant status) necessary for context generation. State variables needed are then loaded into *State Variable Register*, and are written back to *State Variable Memories* whenever updated. Not only state variables corresponding to the current coding sample, but also significant situations (H, V, D) of neighboring columns sent to PEs. *Sign Magnitude Register* works similar to *State Variable Register* expect no update operation is needed. The four PEs corresponding to the four coding primitives, Zero Coding, Run-Length Coding, Sign Coding, and Magnitude Refinement Coding, can generate contexts according to the state variables, sign and magnitude. Three Pass Controllers are used to control the four coding PEs.

4. EXPERIMENTAL RESULTS

Experiments of the block coding engine are made by encoding various images on our proposed architecture and Taubman's [12] for comparison. The processing time of EBCOT block coder is shown in Fig. 8. It is clear that the run time performance improved dramatically. Under the circumstances that only GOCS is applied, the processing time is reduced to 77% compared with Taubman's architecture. If only SS is applied, process time is reduced to 45%. By using SS and GOCS together, processing time can be further reduced to 40%. Through the simulation on various images, about 60% of improvement is achieved in all cases when SS and GOCS are applied together.

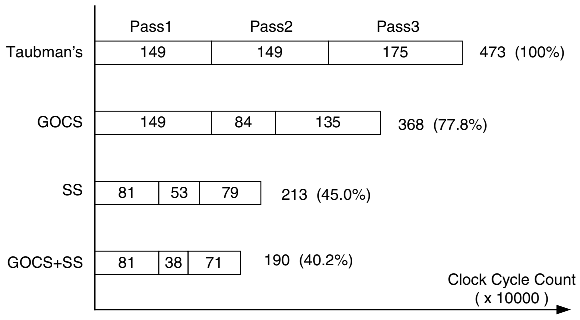

Fig. 8 Performance comparisons (clock cycles required)

A prototype chip of the block coding engine is implemented to verify the proposed speedup schemes. The specification of this chip is shown in Table 3. Simulation results show that this block coding engine can process 4.6 million pixels image within 1 second, corresponding to 2,400x1,800 image size, or 452x340 video sequence with 30 frames per second. It is believed that the performance of future version can be increased through more careful optimization of the critical path. As for the highest performance requirement, multiple block coding engines can be used in parallel since every code-block can be coded independently.

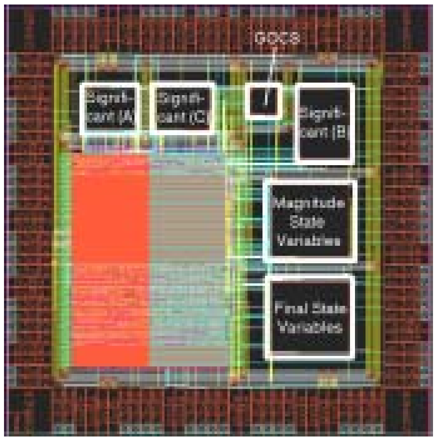

Fig. 9 Layout of the EBCOT block coding engine

Table 2 List of memories used in the prototype chip

Memory Size	function
1024x4	Magnitude Refinement State Variable
1024x4	Final State Variable
264x4	Significant State Variable (A)
528x4	Significant State Variable (B)
298x4	Significant State Variable (C)
64x4	Group-Of-Column Skipping Method

Table 3 Chip Specification

Process Technology	0.35μm CMOS 1P4M
Chip Size	3.67x3.67 mm^2
Gate Count	19,000 gates + 13 kbit memory
Clock Frequency	50 MHz (post-layout simulation)
Supply Voltage	3.3V
Power Consumption	115.49 mW

5. Conclusions

The architecture design of DWT and EBCOT Tier-1 are discussed in this paper. EBCOT block coding engine, as the bottleneck of the JPEG2000 encoding, is analyzed in detail and highly optimized. The computation cycles required is reduced to 40% of a straightforward implementation. It is a good accelerator for JPEG2000, and it is feasible to use multiple block coding engines to achieve the highest performance. As for DWT, a folded lifting based 1-D DWT core is proposed. The main considerations are to support both 5-3 and 9-7 filters.

References

[1] JPEG 2000 Part I Final Committee Draft Version 1.0, ISO/IEC JTC1/SC29/WG1 N1646R.

[2] ISO/IEC, International Standard DIS 10918, Digital Compression and Coding of Continuous-Tone Still Images.

[3] D. Taubman, “High Performance Scalable Image Compression With EBCOT,” *Proc. of IEEE International Conference on Image Processing*, Kobe, Japan, 1999, vol. 3, pp. 344–348.

[4] Analog Devices, Inc., “ADV-JP2000, JPEG2000 Co-processor,” Preliminary Technical Data, 16 May, 2001

[5] W. Sweldens, “The Lifting Scheme: A Construction of Second Generation Wavelets,” Tech. Rep. 1995:6, Industrial Mathematics Initiative, Department of Mathematics, University of South Carolina, 1995, (ftp://ftp.math.sc.edu/pub/imi_95/imi95_6.ps).

[6] W. Sweldens, “The Lifting Scheme: A Custom-Design Construction of Biorthogonal Wavelets,” *Appl. Comput. Harmon. Anal.*, vol. 3(2), pp. 186-200, 1996.

[7] J. M. Jou, Y. H. Shiau, and C. C. Liu, “Efficient VLSI Architectures for the Biorthogonal Wavelet Transform by Filter Bank and Lifting Scheme,” *IEEE International Symposium on Circuits and Systems (ISCAS'2001)*, pp. II-529 - II-532, May 2001.

[8] K. Andra, C. Chakrabarti and T. Acharya, “A VLSI Architecture for Lifting-Based Wavelet Transform,” *The 2000 IEEE Workshop on Signal Processing Systems (SiPS) Design and Implementation*, pp.70-79, October 2000.

[9] W. H. Chang, Y. S. Lee, W. S. Peng, and C. Y. Lee, “A Line-Based, Memory Efficient and Programmable Architecture for 2D DWT using Lifting Scheme,” *IEEE 2001 International Symposium on Circuits and Systems (ISCAS'2001),* pp. IV-330 - IV-333, May 2001.

[10] C. J. Lian, K. F. Chen, H. H. Chen, and L. G. Chen, “Analysis and Architecture Design of Lifting Based DWT and EBCOT for JPEG 2000,” *International Symposium on VLSI Technology, Systems, and Applications*, Hsinchu Taiwan, pp. 180-183, April 2001.

[11] C. Chrysafis and A. Orteg*a,* “Line-Based, Reduced Memory, Wavelet Image Compression,” *IEEE Trans. on Image Processing*, Vol. 9, No. 3, pp.378-389, March 2000.

[12] D. Taubman, “EBCOT: Embedded Block Coding with Optimized Truncation,” ISO/IEC JTC1/SC29/WG1 N1020R.