Analysis and Architecture Design of Lifting Based DWT and EBCOT for JPEG 2000

Chung-Jr Lian, Kuan-Fu Chen, Hong-Hui Chen, and Liang-Gee Chen
DSP/IC Design Lab., Department of Electrical Engineering,
National Taiwan University, Taipei, Taiwan, R.O.C.
E-mail: {cjlian, lgchen}@video.ee.ntu.edu.tw

ABSTRACT

Analysis and architecture design for two major parts of JPEG2000, Discrete Wavelet Transform (DWT) and Embedded Block Coding with Optimized Truncation (EBCOT), are presented in this paper. For DWT, a configurable lifting based 1-D DWT core for both 5-3 and 9-7 filters is proposed. Folded architecture is adopted in DWT to reduce the hardware cost and to achieve the higher hardware utilization. For EBCOT, column-based coding architecture of Tier-1 coding with three speedup methods is proposed. Computation time of context formation in EBCOT can be reduced up to 70%.

I. INTRODUCTION

JPEG2000 [1] is an emerging standard for still image compression. It not only has better compression performance than the existing JPEG [2] standard at low bit rate, but also provides new features not available in JPEG. JPEG 2000 is composed of two major parts: DWT and EBCOT [3]. Fig. 1 shows the functional block diagram of JPEG2000. Wavelet transform is a subband transform. It transfers images 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 coding algorithm proposed by David Taubman. Each wavelet subband is divided into code blocks, and Tier-1 coding engine encodes these code blocks into independent embedded bit-streams using context-based arithmetic coding. Then, Tier-2 reorders the code block bit-streams into the final JPEG 2000 bit-stream with rate-distortion optimized property and the features specified by user.

As we can see in the run time profile of JPEG 2000 in Table I, two major parts of computation load are discrete wavelet transform and Tier-1 coding of EBCOT. Therefore, in this paper, we mainly focus on discrete wavelet transform and Tier-1 coding of EBCOT, analyze, and propose architecture for these two parts.

The rest of this paper is organized as follows. In Section II, the algorithm of lifting based DWT and EBCOT are described and analyzed in detail. The proposed architectures of these two parts and the experimental results are depicted in Section III. Finally, a conclusion is given in Section IV.

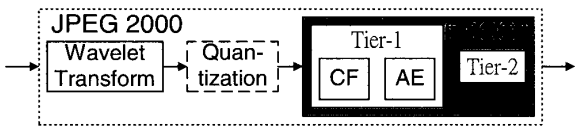

FIG. 1. JPEG 2000 BLOCK DIAGRAM

TABLE I. RUN TIME PROFILE OF JPEG 2000 (IMAGE 1792X1200, 5 LEVEL WAVELET DECOMPOSITION WITH 9-7 FILTER, 1 LAYER, PROFILE AT PENTIUMIII-733, 128MB RAM, VISUAL C++ 6.0 AND WINDOWS ME)

Operation	Single Component	Three Components	
	Lossy	RGB Only	RGB + Intracompone
intercomponent transform			**14.12**
Wavelet transform	**26.38**	**27.64**	**23.97**
quantization	**6.42**	**5.78**	**5.04**
EBCOT Tier 1	**52.26**	**51.89**	**43.85**
pass 1	*14.82*	*15.48*	*12.39*
pass 2	*7*	*6.75*	*5.63*
pass 3	*16.09*	*15.64*	*13.77*
arithmetic encoder	*14.35*	*14.02*	*12.06*
EBCOT Tier 2	**14.95**	**14.7**	**13.01**
layer formation	*9.52*	*8.99*	*7.95*
marker insertion	*5.43*	*5.71*	*5.06*

II. ALGORITHM AND ANALYSIS

Lifting Based DWT

In designing DWT for JPEG 2000, a compact architecture for both 5-3 and 9-7 filter operation is necessary. Fig. 2 shows the classical and lifting based implementations of wavelet transform. A number of architectures of DWT based on the classical implementation have been proposed in the literature [4][5]. The newly proposed lifting-scheme [6][7] for the computation of DWT often has lower computational complexity than the classical implementation. However, there are still fewer discussions about the hardware architecture based on lifting scheme in the literature [8][9].

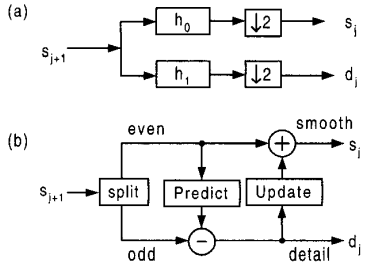

FIG. 2. WAVELET TRANSFORM: (A) CLASSICAL IMPLEMENTATION, (B) LIFTING-BASED IMPLEMENTATION

There are some significant features of lifting scheme. First, by using the similarities between the high and low pass filters, the computation complexity is lower than traditional two-band subband transform scheme. The number of multiplications and additions needed for two points 5-3 and 9-7 1-D DWT by convolution and lifting scheme respectively are listed in Table II for comparison. Second, lifting scheme allows in-place computation of the wavelet transform. The original signal can be replaced with the calculated

0-7803-6412-0/01/$10.00©2001 IEEE.

wavelet transform coefficient. Third, no explicit boundary extension is needed. The symmetry mirroring effect is achieved by a multiplied-by-two operation at proper boundary positions.

TABLE II. COMPLEXITY COMPARISON OF CONVOLUTION AND LIFTING-BASED IMPLEMENTATION

	Two pixels, 1-level, 1-D DWT			
	Convolution		Lifting Scheme	
Filter	Multiplications	Additions	Multiplications	Additions
5-3	4	6	2	4
9-7	9	14	6	8

Embedded Block Coding with Optimized Truncation (EBCOT) Tier-1

Tier-1 of EBCOT utilizes context-based arithmetic coding to encode each code block into independent embedded bit-stream. Tier-1 coding 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. EBCOT encodes the quantized wavelet coefficients bitplane by bitplane from MSB to LSB. Every 4 rows in a bitplane are called as a "stripe," and the scanning order of each pass in every bitplane is stripe by stripe. In every stripe, bits are scanned column by column. Every column is composed of 4 bits. The wavelet coefficients are converted to sign-magnitude format after quantization. A pixel is called "significant" after the first '1' bit is met while encoding magnitude from MSB to LSB bit-plane. Otherwise it is called "insignificant." Contexts for all bits are generated according to their neighbors using four coding methods: (1)Zero Coding (ZC): used to code whether insignificant pixels become significant in current coding bitplane. (2)Run-Length Coding (RLC): used to code 4 consecutive insignificant pixels in a column together if all their neighbors are insignificant. (3)Sign Coding (SC): used to code the sign of every pixel right after the pixel become significant. (4)Magnitude Refinement (MR): used to code significant pixels.

Every bit-plane is encoded using 3 passes. Each bit in a bit-plane is encoded in one of the 3 passes. Pass 1 is "Significant Propagation Pass." Pixels having at least one significant neighbor are coded in this pass, using ZC and SC. Pass 2 is "Magnitude Refinement Pass." All significant pixels are coded in this pass using MR. Pass 3 is "Clean Up Pass." Pixels not coded in first two passes are encoded in this pass. The coding methods used in pass 3 are ZC, RLC and SC. Every bit in a bit-plane is checked once in all 3 passes to determine if this pixel should be coded. In Taubman's architecture [10], a straightforward method is used. Every single bit is checked and (or) coded in all 3 passes, which cost total 3 clocks. In order to speedup context formation, more than one bit has to be processed every clock. In proposed architecture, column-based operation is adopted instead of pixel-based operation. 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: 1) pixels in a column can be checked and (or) encoded simultaneously, so speedup methods proposed can be applied. 2) Higher data reuse in significant and sign variables. Memory access frequency of these variables can be reduced.

Each pixel is coded in one of 3 passes, so one column in every pass may contain 0~4 Need-to-Be-Coded (NBC) pixels. Table III shows the analysis results of column-based operation. Columns are classified in every pass according to number of NBC pixels in them. For example, there are 181076 columns with zero NBC pixels in pass 1, 159223 columns in pass 2, and 258328 columns in pass 3. There are total 598627 (47%) columns contain 0 NBC pixels. The percentage of columns having four NBC pixels (means no processing time wasted in Taubman's architecture [10]) is only 22.06%. According to Taubman's architecture, coding a column costs 4 clocks for checking no matter how many NBC pixels in it. That is inefficient because most of process time is spent on no-operation pixels.

TABLE III. COLUMNS CLASSIFIED BY NUMBER OF NBC PIXELS (BABOON, 512X512, 5-LEVEL WAVELET, CODE BLOCK SIZE 64X64)

NBC pixel no	number of column in each pass				
	Pass1	Pass2	Pass3	Sum	
0	181076	159223	258328	598627	47.85%
1	72650	47663	14437	134750	10.77%
2	60921	49313	10532	120766	9.65%
3	51098	63132	6568	120798	9.66%
4	29391	75805	170807	276003	22.06%

III. PROPOSED ARCHITECTURE

A block diagram of the JPEG 2000 system is shown in Fig. 3. The 2-D DWT is realized by recursively applying horizontal and vertical 1-D DWT to the LL band. The quantized wavelet coefficients are then entropy encoded by EBCOT. The frame memory is used for the data storage before and during wavelet transform, and for the compressed bit-streams of each code block after EBCOT. Proposed architecture for 1-D DWT core and EBCOT Tier-1 are introduced in the following sub-sections.

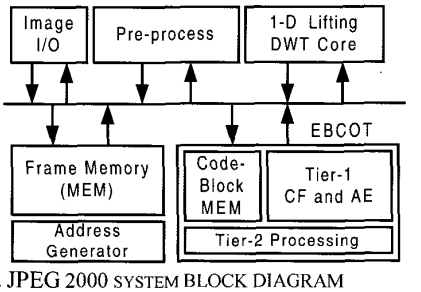

FIG. 3. JPEG 2000 SYSTEM BLOCK DIAGRAM

Lifting Based DWT

It is feasible to calculate both 5-3 and 9-7 filter using the architecture proposed in [8], which is redrawn in Fig. 4 for illustration. The computation of 5-3 filter can be done by alternating the coefficients needed for 5-3 filter, and by taking the output from the first stage. However, the hardware utilization is only 50% when calculating 5-3 filter using this architecture. Also, provided only single read port and single write port memory is available, samples come in serially one sample per cycle and buffered, and then enter the DWT core two samples every other cycle. The hardware utilization is 50% lower due to the sub-sampling effect. To solve the hardware area in-efficiency, a folded 1-D DWT core is proposed based on the lifting scheme. It is configurable for 5-3 and 9-7 filters. The detailed architecture is shown in Fig. 5.

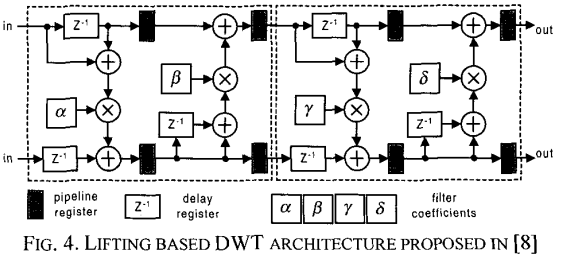

FIG. 4. LIFTING BASED DWT ARCHITECTURE PROPOSED IN [8]

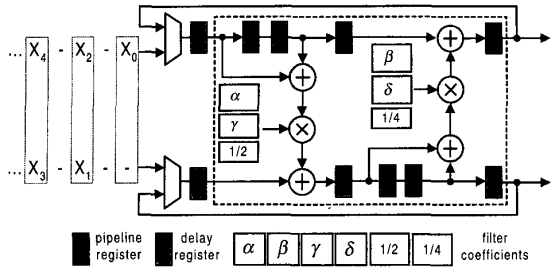

FIG. 5. PROPOSED FOLDED ARCHITECTURE FOR 5-3 AND 9-7 FILTER OF DWT

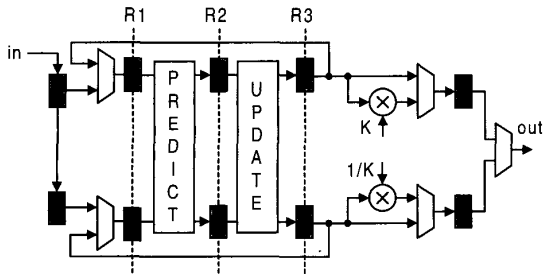

FIG. 6. BLOCK DIAGRAM WITH PRE- AND POST- DATA FORMATTER

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, data read from memory one per cycle, and write back one per cycle. In the split phase of lifting scheme, the data are inputted into two shift registers, and two samples are read into the predict stage every other cycle. At the output, two output data are available in every other cycle, and a parallel to serial circuit is also added for the constraint on single write port memory. That means the input and output data rate of the DWT core are both one sample per clock cycle.

In the 9-7 filter mode, there are two stages of predict and update operation. Data after the first stage computation are feedback (folded) to R1 in Fig. 6 for the second stage computation. The computation of the first stage and the second one are interleaved. The hardware utilization is 100%. While in the 5-3 filter mode, no folded computing is necessary since there is only one stage for lifting based operation for 5-3 filter. Another difference is that the multiplication in 5-3 filter is in fact only shift operation. More specifically, the number of bits to be shifted right is a constant, and only hardwired shifting with sign bit extension is necessary. The computation load in 5-3 is much lower than in 9-7. Also, since no interleaving computation of two stages exists in 5-3 mode, the computation time in predict and update phase can be equivalently two times of the clock period. Therefore, the pipeline registers of R2 and R3 in Fig.6 can be bypassed in 5-3 filter mode, with the effect that the latency is reduced without increasing the clock frequency. Being a dedicated DWT core for JPEG2000, the multiplications of 9-7 filter operation can also be further optimized. Hardwired multipliers are used instead of real multipliers, and the coefficients can be represented in their canonic signed-digit (CSD) format to achieve a more compact design.

Tier-1 of EBCOT

Based on column-based operation, the key ideas to improve the process speed of context formation are as follows: first, skip no-operation pixels and columns (columns with zero NBC pixel). Second, parallel process a column. These ideas can be realized in three methods described below:

1) **Pixel Skipping (PS)**: PS is designed for pass 1 and pass 3. By column-based coding, pixels in a column can be parallel checked to see if they are NBC pixels. Only NBC pixels are processed, other no-operation bits are skipped. If there are n NBC pixels in a column, only n clocks will be spent on coding these NBC pixels and save 4-n clocks.
2) **Magnitude Refinement Paralleling (MRP)**: MRP is used in pass 2. Since only pixels that have been significant in previous bit-planes will be coded in this pass, all pixels in a column can be parallel processed. Only one clock needed for processing a column no matter how many NBC pixels (0~4).
3) **Column Skipping (CS)** : by using first two speedup methods (PS and MRP), every no-operation column will cost only 1 clock for checking. To further improve process 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. So 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. Address of these NBC columns can be recorded. While encoding pass 2 and pass 3, only those NBC columns recorded are processed and all no-operation columns are skipped. As shown in Table III, about 70% of no-operation columns are skipped (Number of no-operation columns in pass 2 and pass 3 divided by all no-operation columns).

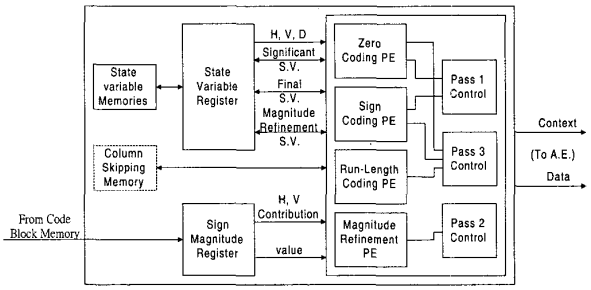

FIG. 7. TIER-1 CONTEXT FORMATION BLOCK DIAGRAM

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 coefficients of a code block are stored in the *Code Block Memory* in a specific bit-plane separated format for EBCOT coding. Contexts corresponding to every single bit are generated, which are the information needed for arithmetic encoder (A.E.). *State Variable Memories* are used for storing three different kinds of state variables (significant, magnitude refinement, final) necessary for context formation. 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 current coding pixels, 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 PEs corresponding to the four coding methods can generate contexts according to the state variables, sign and magnitude. Three Pass Controllers control four coding PEs in the way described in Section II.

Column-Based Operation: The key point to apply column-based operation is to check and (or) process 4 bits in a column simultaneously. 3 columns of significant state variables needed for encoding a column are registered in *State Variable Register*, as shown in Fig. 8. It can provide not only the significant state variables needed, but also the sum of significant neighbors of each pixel in current coding column simultaneously with only a little more hardware cost. In a continuous coding case, after finish coding one

column, we just shift the significant state variables in register array to left by one column, and load one new column into right side of this array.

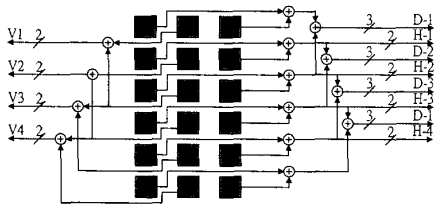

FIG. 8. SIGNIFICANT STATE VARIABLE REGISTERS

Three Speedup Methods: To implement pixel skipping (PS) method, pixels in current coding column are checked at the same time. Indexes for NBC pixels in this column are then generated one by one, skipping all no-operation pixels. The architecture for pixel skipping is shown in Fig. 9. A 4-bit bus indicates if pixels in current coding column are NBC pixels or not. The pixel counter counts the number of pixels (0~3) already coded in this column. The indexes of NBC pixels are generated. Pixel counter is compared with the number of NBC pixels. If all NBC pixels in this column are coded, a change column signal is generated. We can skip all no-operation pixels with only a little hardware cost. Magnitude Refinement Paralleling (MRP) can be implemented easily by parallel encode all pixels in a column using 4 magnitude refinement coding units. Since the contexts table of magnitude refinement coding is quite simple, extra hardware cost is small in MRP. System performance can be improved dramatically (from 158 to 39), as shown in Fig. 10.

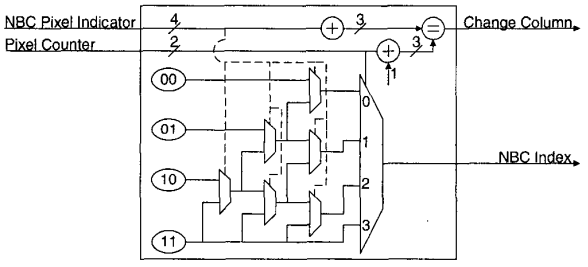

FIG. 9. PIXEL SKIPPING ARCHITECTURE

Column skipping (CS) is applied on pass 2 and pass 3. All no-operation columns in these two passes can be skipped. Since the addresses of all NBC columns in pass 2 and pass 3 must be recorded in pass 1 coding, a memory for storing these addresses is required. Dotted line rectangular (*Column Skipping Memory*) in Fig. 7 means this memory is optional, and added only when CS is used.

Experiments are made on encoding various images on our proposed architecture and Taubman's [10] for comparison. The processing time in context formation in Tier-1 coding of both architectures is shown in Fig. 10. It is clear that PS+MRP improve the performance dramatically. By using CS+PS+MRP, 73% of processing time is reduced. Since PS and MRP can be implemented with only a little more hardware cost, and the performance can be improved dramatically, PS+MRP is recommended for general applications. In this circumstance, memory requirement are similar to Taubman's design. In high-speed applications, CS can be used to further reduce processing time. Overhead of CS is that an additional memory is needed.

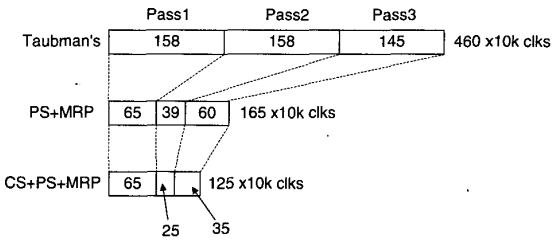

FIG. 10. EXPERIMENTAL RESULTS FOR PROCESSING TIME COMPARED WITH DAVID TAUBMAN'S ARCHITECTURE [10]

IV. CONCLUSIONS

As shown in profiling and analysis for JPEG 2000, most computation load is on DWT and Tier-1 coding of EBCOT. Architectures for these two major parts of JPEG 2000 are proposed in this paper. For DWT, a configurable lifting based 1-D DWT core is proposed. Folded architecture is adopted in DWT to reduce the hardware cost and to achieve the higher hardware utilization. It is a compact and efficient DWT core for the hardware implementation of JPEG2000 encoder. For EBCOT, column-based coding architecture of Tier-1 coding with three speedup methods is proposed. Computation time of context formation in EBCOT can be reduced up to 70%.

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] K. K. Parhi, T. Nishitani, "VLSI architectures for Discrete Wavelet Transforms," *IEEE Tran. on VLSI Systems*, Vol. 1, No. 2, pp.191-202, June 1993.

[5] M. Vishwanath, R. M. Owens, and M. J. Irwin, "VLSI Architectures for the Discrete Wavelet Transform," *IEEE Transactions on Circuits and Systems –II: Analog and Digital Signal Processing*, Vol. 42, No. 5, May 1995.

[6] 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).

[7] W. Sweldens, "The Lifting scheme: A custom-design construction of biorthogonal wavelets," *Appl. Comput. Harmon. Anal.*, vol. 3(2), pp. 186-200, 1996.

[8] Chin-Chi Liu, Yeu-Horng Shiau, and Jer-Min Jou, "Design and Implementation of a Progressive Image Coding Chip Based on the Lifted Wavelet Transform," *Proceeding of the 11th VLSI Design/CAD Symposium*, Taiwan, August 2000.

[9] 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.

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