NOVEL HIGH-THROUGHPUT EBCOT ARCHITECTURE FOR JPEG2000

Ramy E. Aly, Beth Wilson and Magdy A. Bayoumi

University of Louisiana at Lafayette, LA 70504
Email: {ramy, beth, mab}@louisiana.edu

ABSTRACT

Embedded block coding with optimized truncation (EBCOT) consumes more than 50% of the processing time in JPEG200 encoding system. Hardware implementation with careful handling for the control nature of tier-1 is essential. Although, some architectures have been developed to speed up the coding operations, they still require a tedious checking mechanism to decide if each sample is eligible or not for coding. In this paper, we propose a novel checking scheme for the three coding passes for EBCOT that works in parallel with the encoding process to achieve the required high throughput. The simulation results show that the proposed architecture increases the throughput by 19% on average compared to other best known architectures.

1. INTRODUCTION

JPEG2000 becomes one of the most important image compression standards due to its features like low bit-rate performance, lossless and lossy compression, region of interest coding, continuous-tone and bi-level compression, and progressive transmission by pixel accuracy and resolution. JPEG2000 depends on three basic blocks, the discrete wavelet transform (DWT), tier-1 and tier-2. Starting with DWT and the scalar quantization, the transformed image is divided into tiles, named code blocks, with typical dimensions are 32x32 or 64x64 coefficients. Each code block is encoded individually in tier-1 and tier-2 coding blocks as shown in Fig. 1. The code block is divided into one sign bit-plane and several magnitude bit-planes. Tier-1 consists of two blocks, context formation block which generates the context formation (CX) and Decision (D) that are coded with the arithmetic encoder to form the compressed bitstream. In Tier-2, rate distortion optimization is used to truncate some of the compressed data to achieve the desired bit-rate. The context formation encodes the code block bit-plane by bit-plane, starting from the most significant bit-plane with at least a non-zero element to the least significant bit-plane. Each bit-plane is coded in three coding passes. Each bit in a bit-plane is coded once in one of the three coding passes. Every four rows in each bit-plane are called "stripe". In each pass, the bits are scanned stripe by stripe from top to bottom. Within a stripe, each 4-bits (samples) column is scanned column by column from left to right. Within a column, each bit location is scanned bit by bit from top to down as shown in Fig. 2. EBCOT has four primitives coding PEs: zero coding (ZC), sign coding (SC), magnitude refinement (MR), and run-length coding (RLC).

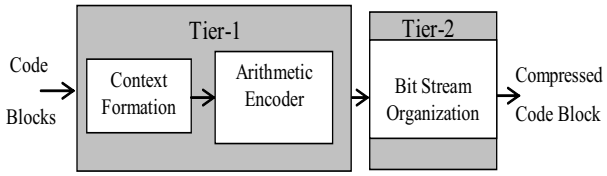

Fig. 1. Block Diagram of the EBCOT Encoders for JPEG2000

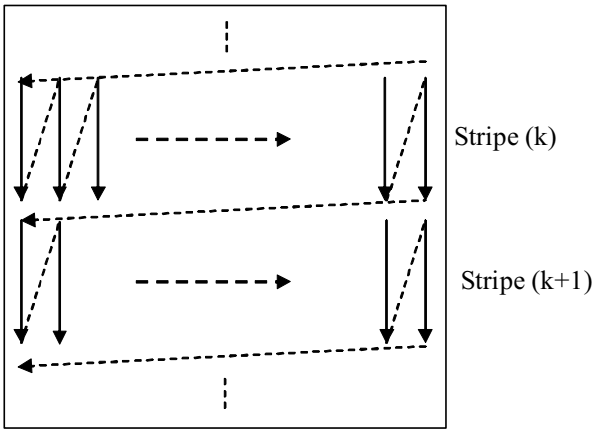

Fig. 2. Scanning order in every pass

If the bit is insignificant and any one of its neighbors is significant, it is coded in significance propagation pass (pass 1) that is based on ZC primitive. If the bit is significant, it is coded in magnitude refinement pass (pass 2) that is based on MR primitive. Other bits are coded in cleanup pass (pass 3) that is based on ZC and RLC primitive. The coding primitives generate context labels based on the sign and significance status of 8-connect neighbors of the current bit.

Studies show that embedded block coding with optimized truncation (EBCOT) consumes more than 50% of the processing time in JPEG200 encoding system [1]. Many EBCOT implementations have been proposed [2, 3, 4, 5, 6]. The context formation consists of two basic stages. The first stage is the checking mechanism to determine if the bit should be coded in this pass or not and the second stage is the coding using the suitable one or more of the four primitives encoders. Previous work only concentrated on the coding part trying to minimize the coding time to NxN clock cycles for an NxN code block. A check mechanism was proposed in [5] showing the importance of the speed up of the checking time and the power consumption for this stage. In this paper, we propose a novel checking mechanism that reduces the searching time for eligible samples, columns, and stripes aiming to increase the context formation throughput. The paper is organized as follows. In section II, we describe the context formation block. The proposed checking algorithm is discussed in section III. The performance of the proposed architecture is presented in Section IV, and section V concludes the paper.

2. CONTEXT FORMATION BLOCK

The context formation block obtains the sign and magnitudes bit planes for each code block as its input. Each sample in each magnitude bit plane should be encoded in one of the three passes. The context formation generates the context (CX) and decision (D) for each sample using the four coding primitives as its output. Direct implementation costs 4 clock cycles per column (1 cycle per bit) regardless if the sample is actually coded in this pass [7, 8]. After encoding, when non valid contexts are generated for samples that shouldn't be coded in this pass, these non valid contexts are ignored. The encoding process for one bit plane costs 3xNxN clock cycles. In [2], two speed up techniques are proposed that basically depend on checking the sample/column/group of columns first and according to that scanning, the control unit decides if there is at least one sample needs to be coded in the column. If not, the column is skipped. This implementation costs only 1xNxN cycles of coding time in addition to a checking time about 1 clock cycle per column. This technique costs at least 3xNxNx(1/4) clock cycles for checking process. In [5], an efficient architecture was proposed to perform the checking process faster and with low power consumption. The speed up is achieved by doing the check process early in the architecture by scanning the significance memory to determine if there is a sample that needs to be coded in the column. In the context formation block, the processing time equals the checking time and the coding time.

Fig. 3 shows the average number of samples coded in the three passes in each bit plane for many code blocks from 256x256 gray scale Lena image. It is clear that there are some bit planes with majority of the samples are coded in one pass, like pass3 for the higher bit planes and pass2 for the lower bit plane. The regular checking mechanisms in [2, 5] still treat all the passes equally. In reality the number of samples coded differs in each pass. The basic problem is finding eligible sample(s) quickly and to have the ability to skip the unneeded samples/columns/stripes/passes in short amount of time to increase the throughput of the system. The context formation block performs basically two operations, checking for eligible samples and encoding for these samples.

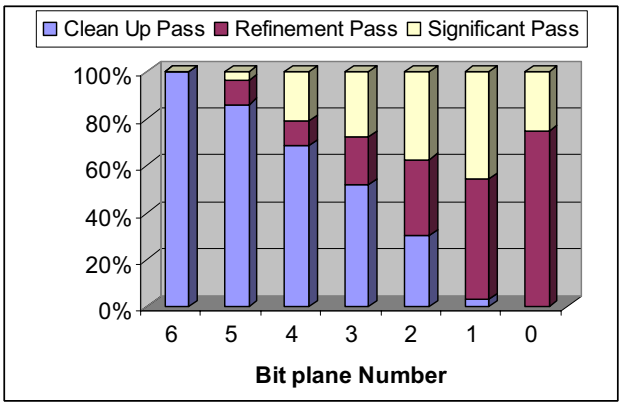

Fig. 3. The average percentage of number of samples coded in each bit plane

3. THE PROPOSED ARCHITECTURE

We propose a novel EBCOT architecture, shown in Fig. 4, which depends on a fast checking mechanism to decide the eligibility of each bit/column/stripe/pass to be coded in each of the three coding passes. The checking mechanism is implemented using two memories named, pass memory (PM) and pass status memory (PSM). The pass memory is of size NxNx2 and contains the sample's correct pass (i.e., 1, 2 or 3). The corresponding pass for each sample in the current processed bit plane is stored in the pass memory. For each coding pass, the control unit can easily check the pass numbers for the 4 samples in each column(s) in parallel to determine if there is at least one sample need to be coded in this column(s) by comparing the pass number in the pass memory with the current coding pass.

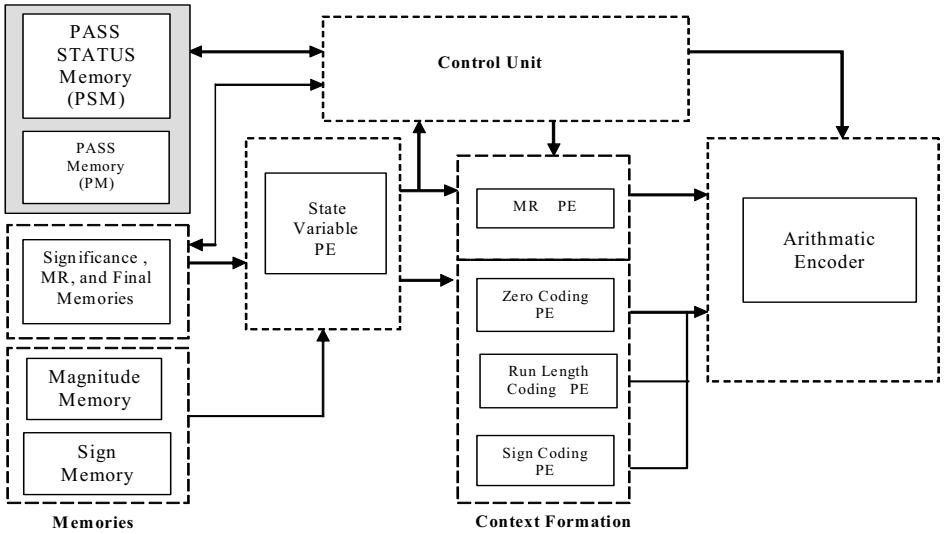

Fig.4. The proposed Architecture for EBCOT

If the pass number in the pass memory coincides with the current coding pass, the sample must be coded in this pass. If not, the control unit skips the sample. The proposed mechanism can be applied in serial and parallel coding architectures. In this paper, we concentrate on the serial coding architecture, meaning pass1 followed by pass2 and finally pass3. The pass memory doesn't change during the coding of pass1 and 2. After coding the first stripe in pass3, the control unit starts modifying the pass memory for the next bit plane in parallel to the coding process for pass3 as shown in Fig.5.

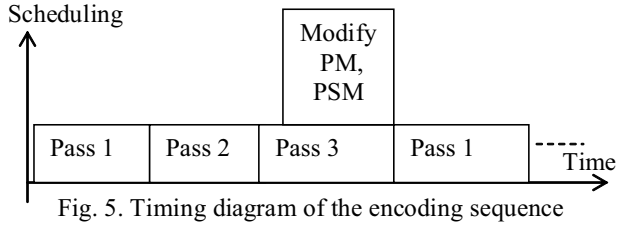

Fig. 5. Timing diagram of the encoding sequence

According to the statistical analysis shown in Fig. 3, we extend the proposed idea by adding the pass status memory (PSM) that summarizes the passes required for each group of 8 columns, each stripe and each pass as shown in Fig. 6. Assume we have M stripes (M= N/4) and each stripe has L*8 columns. We choose 8 columns as one group to achieve the best run-time performance [2]. Fig. 6 shows an example situation of the PSM where one bit is dedicated for each pass. The first column indicates the passes required to code of each stripe. The MSB is dedicated for pass1, the middle bit for pass2 and the LSB for pass3. In the example, the first stripe status is "111" that means there are samples need to be coded in each of the three passes that means the control unit can't skip the stripe while coding it for any of the three passes. For the M^{th} stripe, "010" means that the stripe need to be coded in pass 2 only and the control unit can skip checking/coding the whole stripe at pass1 and pass3. The last row, M+1, contains the passes required to code the entire bit plane. In the showing example, "111" means this the three passes are needed to code this bit plane. As shown in Fig. 3, there are some bit planes may not need to be coded in the three passes, others may only need 2 passes and the rest requires the three passes. The following columns in the PSM contain the status of each group of 8 columns in each stripe in the bit plane.

	Stripe	Group (1)	Group (2)	----------	Group (L)
1	111	001	111		010
2	111	111	111		001
M	010	010	010		010
	111				
	Passes				

Fig. 6. Status pass memory organization

At the beginning, PM contains "3" to indicate that all the samples in the MSB is coded in pass3. After coding of the first stripe, the control unit starts modifying PM according to the bit plane magnitude values as shown in Fig. 7. The PSM is also modified at least after the first stripe of the PM has been modified. Using gate logic and comparators, the PSM can easily modified for the next bit plane and all the stripes status bits can be determined. The proposed checking mechanism is summarized in Fig. 8.

The proposed architecture solves the checking problem in EBCOT design by doing all the decisions required for the three passes in one time in parallel to the encoding process to achieve high-throughput context formation stream.

```
Initial: All PM coefficients =3
   For all PM coefficients,
         If  bit plane k(n1, n2) =1, then PM(n1, n2)  = 2
   For all PM coefficients,
         If  PM(n1, n2) = 2,
                For the 8 neighbors for (n1, n2),
                     If  PM(neighbor) =3,
                          then PM (neighbor) = 1
```

Fig. 7. Pass Memory modification Algorithm

```
For each pass
    If  pass status =”0”, skip the pass.
    Else,
       For each stripe
          CPU checks the SPM (stripe status) to decide
          whether or not to encode the stripe.
           If “0”, skip the whole stripe.
           Else,
                 Check the SM (columns status),
                 Only encode the column with “1” status.
```

Fig. 8. The proposed checking mechanism

4. EXPERIMENTAL RESULTS

A Matlab program is designed to calculate the checking time for the proposed architecture and SS and GOCS techniques in [2] at serial coding. All of these architectures have the same encoding time but they have different checking time that we try to simulate. For each image, a DC shift (-128), wavelet transform for 3 levels using 5/3 filter and quantization for 256 gray levels are applied. The image is divided into 64x64 code blocks. The proposed architecture reduces the checking time by at least 19% than the other architectures, as shown in Fig. 9, due to its ability to skip passes/stripes/group of columns faster.

5. CONCLUSION

A novel checking mechanism is proposed to highly increase the throughput of the context formation block. Pass memory and status pass memory are added that contain the pass for each sample in the bit plane and a summary for the group/stripe/pass status to speed up the checking mechanism. The proposed mechanism can be used for serial and parallel EBCOT architecture. This paper describes the architecture for the serial coding passes. Compared to the other architectures, the proposed architecture shows on average 19% higher throughput than the other GOCS+SS.

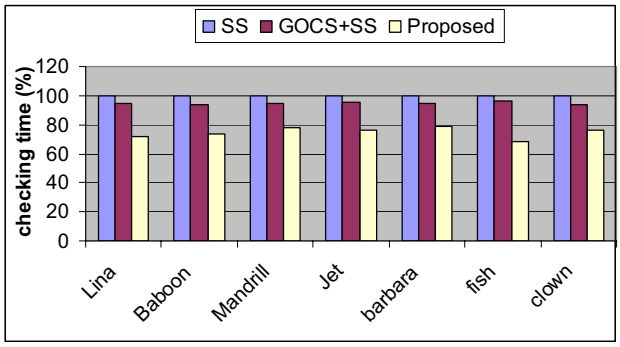

Fig. 9. Comparison of the required checking time

ACKNOLWLEDGEMENT

The authors acknowledge the support of U.S. Department of energy (DOE), EETAPP program DE97ER12220, The Governor’s Information Technology initiative, and the support of NSF, INF 6-001-006.

6. REFERENCES

[1] 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.

[2] Chung-Jr Lian, Kuan-Fu Chen, Hong-Hui Chen, and Liang-Gee Chen, “Analysis and Architecture Design of Block-Coding Engine in JPEG-2000,” IEEE Trans. on Circuits and Systems for Video Technology, vol. 13, no. 3, March 200.3

[3] Y. Li, R. E. Aly, B. Wilson, and M. A. Bayoumi, “Analysis and Enhancement for EBCOT in High speed JPEG2000 Architectures,” IEEE Midwest Symposium on Circuits and Systems, pp.207-210, July 2002.

[4] Y. Li, R. E. Aly, M. A. Bayoumi, and S. Mashali, “Parallel High-Speed Architecture for EBCOT in JPEG2000,” IEEE International Conference on Acoustics, Speech, and Signal Processing, vol.2, pp. 481-484, April 2003

[5] R. E. Aly, M. A. Bayoumi, “Low-Power and High-Speed Architecture for EBCOT Block in JPEG2000 System,” IEEE Midwest Symposium on Circuits and Systems, July 2004.

[6] K. Andra, C. Chankrabarti and T. Acharya, “A High-Performance JPEG2000 Architecture,” IEEE Trans. Circuit and Systems for Video Technology, vol. 13, no. 3, pp. 209-218, March 2003.

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

[8] D. Taubman and HP labs, “Report on core experiment CodeEff22, EBCOT: Embedded Block Coding with Optimized Truncation,” Tech. Rep. N1020R, ISO/IEC JTC1/SC29/WG1, Oct 1998.