A Hardware Accelerator IP for EBCOT Tier-1 Coding in JPEG2000 Standard

Tien-Wei Hsieh Youn-Long Lin

Department of Computer Science
National Tsing Hua University
Hsin-Chu, Taiwan 300
{ylin@cs.nthu.edu.tw}

ABSTRACT

We proposed a hardware accelerator IP for the Tier-1 portion of Embedded Block Coding with Optimal Truncation (EBCOT) used in the JPEG2000 next generation image compression standard. EBCOT Tier-1 accounts for more than 70% of encoding time due to extensive bit-level processing. Our architecture consists of a 16-bit parallel context formation module and a 3-stage pipelined arithmetic encoder. Compared with the known best design, we reduce 17% of the cycle count and have achieved within 5% of the theoretical lower bound. We have integrated our synthesizable RTL with an AMBA-AHB interface for SOC design. FPGA prototyping has been successfully demonstrated and substantial speedup achieved.

Keywords

JPEG2000, Embedded Block Coding with Optimal Truncation, Context Formation, Arithmetic Encoder.

1. INTRODUCTION

JPEG2000 is the next generation image compression/decompression standard. Like previous standard such as JPEG, it consists of three major phases: transformation, quantization and entropy coding. JPEG2000 achieves better quality and efficiency at the expense of more computation complexity. It employs discrete wavelet transform (DWT) in stead of the more traditional discrete cosine transform (DCT). For entropy coding, the embedded block coding with optimal truncation (EBCOT) is used. EBCOT consists of two tiers: Tier-1 codes each code-block into a sub-bit-stream and Tier-2 assembles sub-bit-streams into image stream.

In addition to computational intensive coding tasks, a JPEG2000 codec needs to handle other complicated control function such as rate distortion control. Therefore, it is advantageous to implement it with a hardware/software codesign approach. According to out profiling report, the Tier-1 portion of EBCOT accounts for more than 70% of the total encoding time due to its extensive bit-level processing. Hence, it is the most suitable candidate for hardwired implementation.

The EBCOT Tier-1 takes as its input a code-block ranging from 4X4 to 64X64 transformed and quantized pixels. Each pixel is a 9-bit signed magnitude number. The code-block is divided into bit-planes and coded one bit-plane at a time starting from the MSB bit-plane down towards the LSB bit-plane, Each bit-plane is further divided into horizontal 4-row stripes.

There are two phases in EBCOT Tier-1: Context Formation (CF) and Arithmetic Encoder (AE). For each bit, the CF generates a context/decision pair based on its surrounding information for the AE to perform adaptive coding. Each bit-plane is scanned by the CF in three passes. Each bit is coded in one of these passes. The scanning order is stripe by stripe, column by column, and bit by bit as depicted below Figure 1.

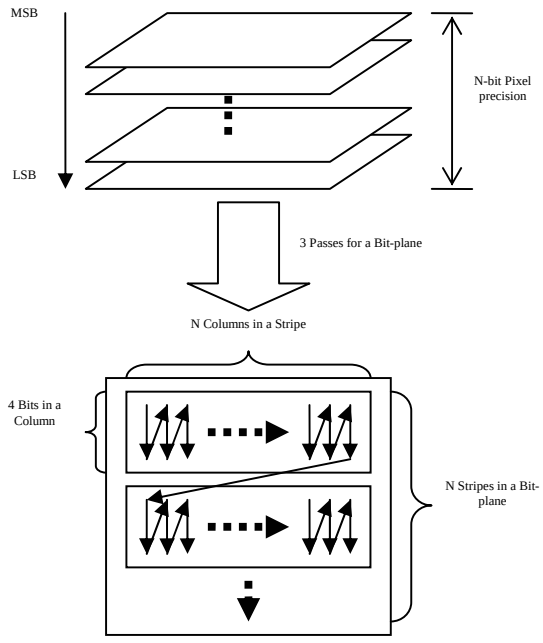

Figure 1. Scanning hierarchy

When a bit is scanned, whether and how it will be coded depends on the status of itself and its eight surrounding neighboring bits. After coding a bit and possibly generating its context/decision pair, we have to update its status information for successive coding of its yet to be coded neighbors.

The AE takes as its input the sequence of context/decision pairs generated by the CF and adaptively updates its probability tables for arithmetic coding. Coded information is output byte by byte.

2. PROPOSED ARCHITECTURE

Our proposed architecture consists of six parts as depicted in Figure 2. The code-memory supports input of code-block in a pixel by pixel fashion and output of 16 bits of a bit-plane. The state memory records the status of relevant bits. The address generator is responsible for supplying correct addresses to the above mentioned memory modules.

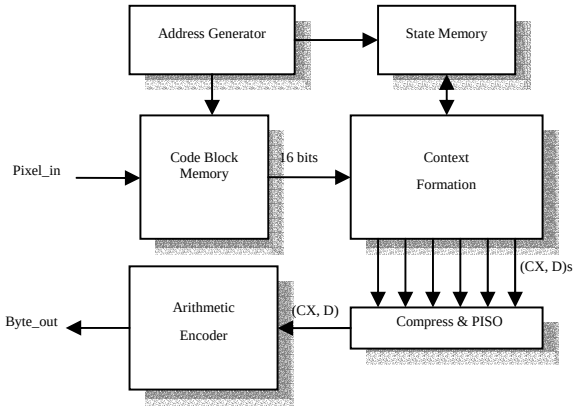

Figure 2. Proposed tier-1 block coder

CF is a parallel structure. It can process sixteen bits simultaneously. It generates up to 40 context/decision pairs for each set of 16 bits. The Compress & PISO (parallel-in-serial-out) module delivers all valid (CX, D) pairs to the AE one by one. AE is a 3-stage pipelined structure. It receives (CX, D) pairs from Compress & PISO and generates a series of bytes constituting sub-bitstream.

2.1 Context Formation

There are three factors affecting parallelism of CF: 1) scanning order, 2) checking neighbors, and 3) changing state. An example is illustrated in Figure 3. The number annotating a bit represents the scanning order of this bit. While the context of the sixth bit is generated, the states of the 8 bits in the context window should be checked. However, the first, second, third and fifth bit have already coded, and may have their states changed. It would affect the coding result of the sixth bit immediately.

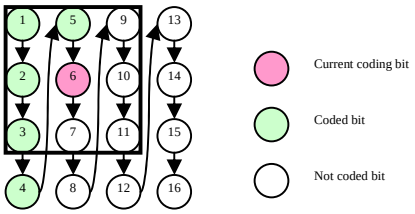

Figure 3. Scanning order

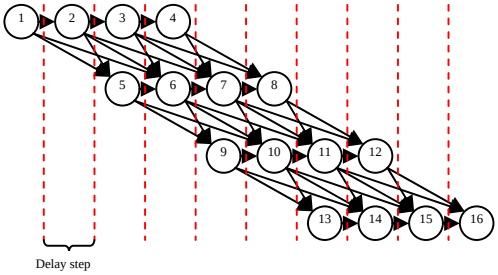

Figure 4. Data dependency during context

We depict the data dependency among these sixteen bits as a DFG shown in Figure 4. If we process the sixteen bits simultaneously, the clock period requires ten delay units. Since its delay is ten, not sixteen, times than sample-based architecture, we can use lower voltage to achieve the same level of throughput. Theoretically, we estimate that the 16-bit parallel architecture can save about 60% power consumption compared with a sample-based architecture.

If we alleviate the frequency of memory access and utilize efficient memory bandwidth, we can reduce the power consumption as well. The memory-saving algorithm and the data arrangement proposed by [4] are adopted. Furthermore, we adjust our memory arrangement to be suitable for 16-bit parallel processing. There are eight bits as a word, and words are placed in three memories with an interleaving format, as shown in Figure 5. During memory access, the order of memory data depends on the stripe. For example, the data order is (C, A, B) while we code Stripe n, and the data order is (B, C, A,) while we code Stripe n+1. Through this memory arrangement, we can utilize efficient memory bandwidth.

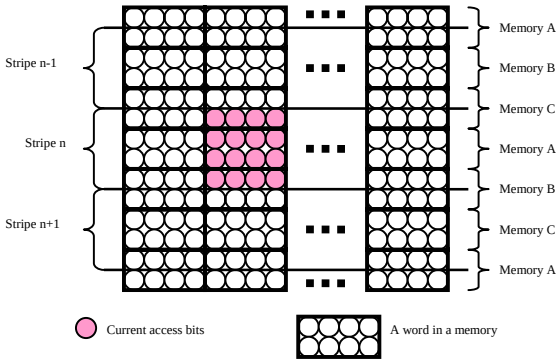

Figure 5. Data arrangement

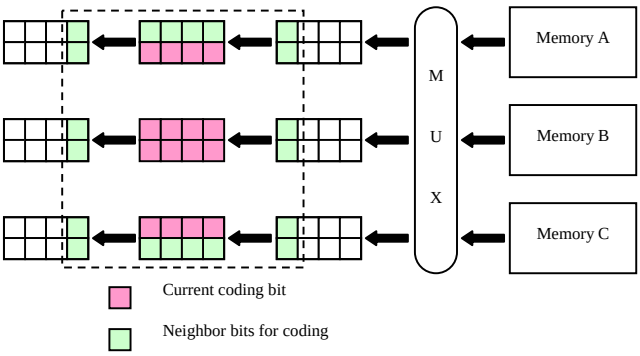

Figure 6. Shift registers

During the coding process, we use nine 8-bit shift registers for local data reuse as shown in Figure 6. By the multiplexer, the three memories can transfer data in the right order. The context window is the data requirement for the parallel processing. It includes sixteen current coding bits in the stripe and twenty neighbor bits for coding in the boundary. The context window would slide to the next four columns (sixteen bits) after each processing step.

The 16-bit parallel processing can reduce cycles consumed by context formatting. However, there are still wasted cycles because each bit is coded in only one of the three passes. We adopt the skipping scheme and propose a stripe-skipping method. It is easy to implement in our 16-bit parallel architecture. Moreover, the memory requirement of stripe-skipping is small. We just use three 16-bit registers to record the coding condition of all stripes in three passes and save about 2% of the cycles.

2.2 Arithmetic Encoder

Since the AE has feedback loops, we cannot employ a straightforward pipelined structure. We adopt a Modified Probability Estimation Table (MPET) [5] and the operand forwarding method commonly found in RISC design to overcome those problems.

Our 3-stage pipelined AE architecture is shown in Figure. 7. In the first stage, we read a context-decision (CX, D) pair from CF and use CX to look up the Context Table for the probability estimate. Since the probability estimate of CX can be updated by the feedback information from the second stage, we need to cope with two identical contexts come in continuously. We use the MPET that original PET data and two types of updating PET data are read simultaneously by one index. Moreover, the result of the second stage should forward to select correct PET data.

In the second stage, we calculate the updating values of the A register and the Context Table and dispatch the information of shift amount to the third stage. In the third stage, we either calculate the updating values of the C register and the counter CT or perform the renormalization procedure. After all bits of a code-block are coded, AE is terminated and flushed to get a complete sub-bitstream.

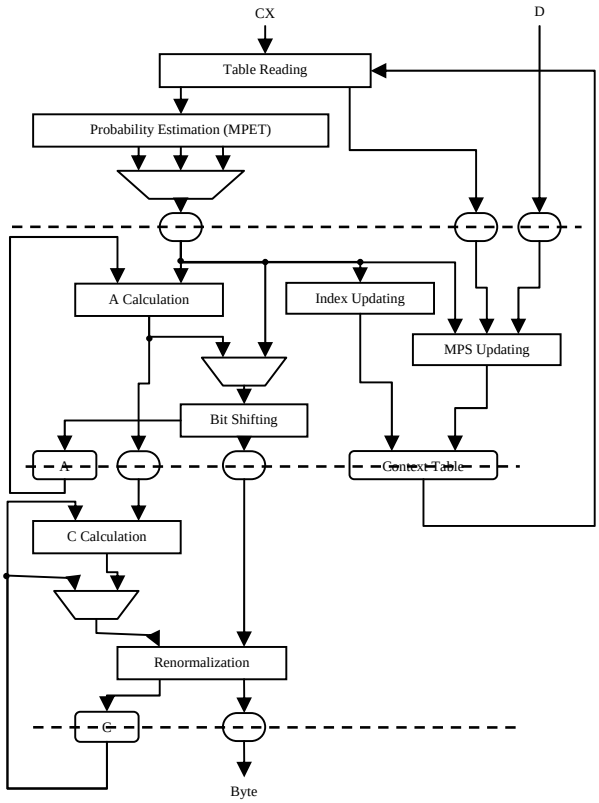

Figure 7. Proposed AE architecture

An important characteristic of the input symbols for AE are presented in [3]. Those symbols have a highly skewed distribution. In [4], they adopt a simple renormalization strategy, and their BYTEOUT sub-procedure would take more than one clock cycle to complete the C register operation. We use the operand forwarding method commonly found in RISC CPU design to overlap the actions of BYTEOUT and register-updating to achieve about 10% saving in cycle counts, as shown in Figure 8.

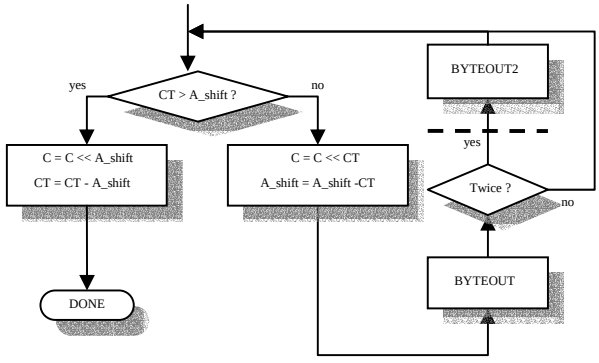

Figure 8. Renormalization using forwarding

3. EXPERIMENTAL RESULT

We have implemented the proposed architecture in Verilog RTL and synthesized it with Synopsys Design Compiler under the

worst case operating environment (WCCOM), and use PrimePower to analyze the power consumption, as shown in Table 1.

Table 1. Synthesis and power analysis

	Skipping methods[2]	Memory saving[4]	Our design
Tech. library	TSMC .35	TSMC .35	TSMC .35
Area (gate count)	19,000 + 13Kbit memory		25,706 + 45Kbit memory
Max. freq. (MHz)	50	142.8	43.38
Power (mW)	115	131.8 (under 100 MHz)	26.68

Table 2. Comparison by cycle count

Images	Lower bound	Column-based[2]	Our design
Airplane	1,315,525	1,755,450	1,405,830
Baboon	1,750,918	2,106,820	1,829,179
Lena	1,345,543	1,743,283	1,429,414
Peppers	1,455,349	1,880,388	1,540,763
Average	1,466,833.8	1,871,485.3	1,551,296.5

Our experiment uses four standard images Airplane, Babbon, Lena and Peppers. All of them are 512x512 gray image. Before EBCOT process, test images go through 5/3 DWT and three decomposition levels. Our process unit is a 64x64 code-block. We reduce the cycle count by 17% compared with column-based architecture. In addition, let the number of contexts be the lower bound on the cycle count, we have achieved 5% within the optimum, as shown in Table 2.

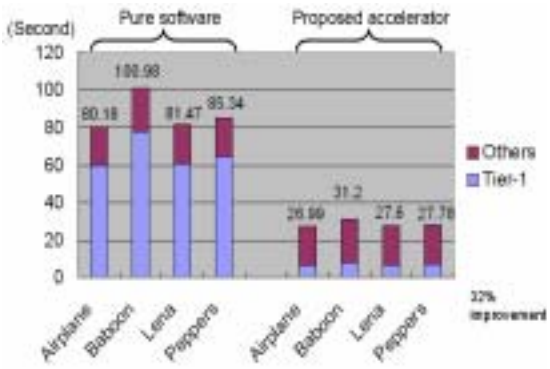

Figure 9. Experimental results on SoC platform

We integrate our design with JPEG2000 software on Altera Excalibur™ EPXA10DDR and compare the performance with pure JPEG2000 software, as shown in Figure 9.

4. CONCLUSION

We have proposed a hardware accelerator IP for EBCOT of JPEG2000. Our novel architecture leads to 17% reduction in cycle count compared with the state-of-the-art. We have achieved within 5% of the theoretical bound. We have also demonstrated the developed IP in an SOC platform using FPGA prototyping. The proposed IP can be used for any AMBA-based SOC design of IPEG2000 image coding.

5. REFERENCES

[1] *"JPEG 2000 Part I Final Committee Draft Version 1.0"*, ISO/IEC JTC 1/SC 29/WG 1 N1646R, March 2000. Available from http://www.jpeg.org.

[2] Chung-Jr Lian, Kuan-Fu Chen, Hong-Hui Chen and Liang-Gee Chen, *"Analysis and Architecture Design of Block-Coding Engine for EBCOT in JPEG 2000"*, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 13, No. 3, pp 219-230, March 2003.

[3] David Taubman, Erik Ordentlich, Marcelo Weinberger, Gadiel Seroussi, Ikuro Ueno and Fumitaka Ono, *"Embedded Block Coding in JPEG2000"*, Proceedings of the IEEE International Conference on Image Processing (ICIP), Vol. 2, pp 33-36, September 2000.

[4] 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.

[5] Masaya Tarui, Masaru Oshita, Takao Onoye and Isao Shirakawa, *"High-Speed Implementation of JBIG Arithmetic Coder"*, Proceedings of the IEEE Region 10 Conference, Vol. 2, pp 1291-1294, September 1999.