A HIGH-PERFORMANCE MEMORY-EFFICIENT ARCHITECTURE OF THE BIT-PLANE CODER IN JPEG 2000

Grzegorz Pastuszak

Institute of Radioelectronics, Warsaw University of Technology, Poland
G.Pastuszak@ire.pw.edu.pl

ABSTRACT

The paper presents a high-performance architecture of the bit-plane coder for the embedded block coding algorithm in JPEG 2000. The architecture adopts a pipeline structure and is dedicated to generate two context-symbol pairs per clock cycle. A novel method called Dynamic Significance State Restoring (DSSR) allows reduction of on-chip memories. The overall design is described in VHDL and synthesized for FPGA and ASIC technologies. Simulation results show that for FPGA Stratix devices, the engine can process about 22 million samples at the frequency of 66 MHz.

1. INTRODUCTION

The JPEG 2000 image compression [1], [2] can be exploited in such applications as surveillance systems, digital cameras and digital cinema chain. The use of Motion JPEG 2000 instead of video compression standards (MPEG-1/2/4, H.26x) may be desired. In particular, JPEG 2000 enables the use of the same resources as for image capturing, ability to handle many image sources in any order, the ease of changing the frame-rate, and flexible access to an arbitrary frame.

Since JPEG 2000 involves a computational-intensive algorithm, hardware acceleration may be indispensable to achieve real-time performance. The bottleneck of the JPEG 2000 system arises from the block performing embedded block coding with optimised truncation (EBCOT). This limitation is caused mainly by time-dependent operations in the context adaptive binary arithmetic coder (CABAC) included in the EBCOT block. However, the bit-plane coder (BPC), which is another part of EBCOT module, may also decrease the throughput because of bit-level operations and intervals introduced by fractional bit-plane coding.

There are various architectures for EBCOT proposed in literature [3]-[7]. Most of them focus on optimisation methods for the BPC and assume that the CABAC can process at most one context-symbol pair per clock cycle. In [8], we presented the VLSI implementation of the CABAC able to code two context-symbol pairs per clock cycle. To benefit from the increased throughput of the CABAC, the BPC has to adjust its speed to balance the CABAC. This paper presents the architecture which meets this requirement. The core adopts pipeline arrangement optimised to achieve the clock rate as high as possible. Moreover, a novel method called Dynamic Significance State Restoring (DSSR) is proposed to reduce on-chip memories. The architecture of the BPC has been described in VHDL and synthesized for commercial FPGA and ASIC technologies. The estimated clock rate is 66 MHz for FPGA Stratix devices.

The rest of the paper is organized as follows. Section 2 reviews the algorithm of the bit-plane coder. The proposed DSSR method is described in Section 3. Architecture design is illustrated in Section 4. Implementation results are given in Section 5, and the Conclusion in Section 6.

2. BIT-PLANE CODING ALGORITHM

The bit-plane coder is the first stage of the EBCOT algorithm. The BPC generates context-symbol pairs on the basis of quantization indices grouped in code-blocks. Input data are read in the sign-magnitude format and analysed bit-plane wise starting from the most significant bit-plane (MSB) with a non-zero element to the least significant bit-plane (LSB). Each bit-plane is scanned in three coding passes called significance propagation, magnitude refinement, and cleanup. A bit plane is divided into horizontal stripes of four rows (see Figure 1). They are scanned from top to bottom. Each stripe is scanned in column fashion from left to right. Each column is scanned bit by bit from top to bottom. Each coefficient in a code-block has an associated variable which indicates whether or not the coefficient is significant. The state changes from insignificant to significant at a bit-plane where the most significant non-zero magnitude bit of the corresponding coefficient is found. Two additional variables are necessary to identify coding pass and bit-plane at which a coefficient changes its state.

At the beginning of each bit-plane processing (exclusive of the first one), the significance coding pass is

0-7803-9332-5/05/$20.00 ©2005 IEEE

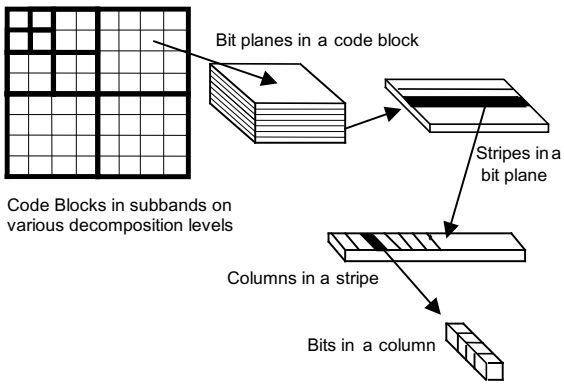

Figure 1. Hierarchy of a code-bock

performed. In this pass, all coefficients which are insignificant and have at least one of their immediate eight neighbours significant are coded. The neighbour coefficient states are mapped into one of nine contexts. The generated symbol equals the value of the scanned bit. If it is one then sign coding follows taking into account states and signs of four (horizontal and vertical) neighbour coefficients. They are mapped onto five contexts. In conjunction with the sign, they determine symbol. The magnitude refinement pass follows the significance propagation pass and includes coefficients which have become significant in previous bit-planes. There are three possible contexts. The cleanup pass processes the remaining coefficients which have not been coded in the first two passes. Here, contexts and symbols are produced as in the significance propagation pass. Additionally, the standard defines the run-length and UNIFORM contexts.

3. REDUCTION OF THE STATE MEMORY

The state information is used to determine the context and the pass in which a coefficient is coded. Each coefficient state consists of three binary variables: the significance state, the first refinement indicator, and the significance pass membership. Direct mapping onto hardware involves the use of a 12K bits memory so as to match the maximal possible size of the code block [2]. An approach demonstrated in [3] and [7] makes it possible to reduce the state variables to two bits per sample, decreasing the size of the state memory by 4K bits. The proposed dynamic significance state restoring (DSSR) technique allows the reduction of the size of the state memory to 1K bits. Instead of storing state variables in the memory, the technique reconstructs them by analysing several bit-planes in parallel. Most notably, prior to coding a bit-plane, the significance state of each coefficient can be determined on the basis of bit-planes located above the current one. For example, if a coefficient has any non-zero bit in these bit-planes, its state is significant. In the magnitude refinement pass, it is necessary to distinguish in

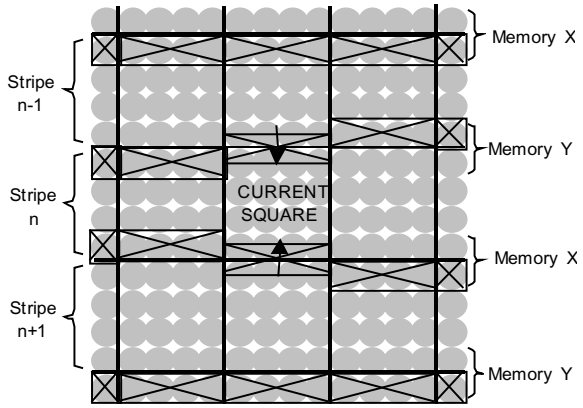

Figure 2. Proposed memory arrangement for significance pass membership variables

which bit-plane such a coefficient has become significant (the first magnitude indicator). This information is derived by checking the position of the most significant non-zero bit.

The significance pass membership variable indicates which coefficients have been checked at the significance propagation pass of the current bit-plane. Remaining coefficients which are insignificant have to be coded in the following cleanup pass. Additionally, the variable allows the context formatter to determine which coefficients have become significant at a given point of processing. In particular, coefficients having first non-zero bit in the current bit-plane become significant after checking at either the significance pass or the cleanup pass. The choice depends on the significance pass membership variable. The variable corresponding to a given coefficient is set active whenever at least one of immediately neighbouring coefficients has changed its significance state earlier than the current one, which has happened either during the processing of one of upper bit-planes or in the significance pass of the current bit-plane. The first case is derived from upper bit-planes, as described above. In the second case, the changes of the significance pass membership variables propagate through neighbouring coefficients which follow in the scan order and have their first non-zero bit in the current bit-plane. The propagation terminates on coefficients which remain insignificant, i.e. they have zero-valued bits in the current bit-plane. As a consequence of these propagation dependencies, the significance propagation has to be performed in each pass to restore the significance pass membership variables.

The context formation process for bits located on boundaries of the current stripe refers to states of coefficients located in neighbouring stripes. This means that states should be reconstructed for boundary coefficients located just above and below of the current stripe. Moreover, the context formation for sign coding refers to their signs. When all available coefficients belong only to the current stripe, the sign and the state

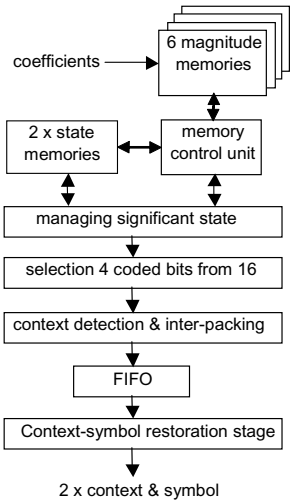

Figure 3. Diagram of the bit plane coder

information for coefficients located in neighbouring stripes has to be conveyed by means of a dedicated memory. In this case, each state of a boundary coefficient occupies three bits to store the sign, the significance state, and the significance pass membership. If boundary coefficients from neighbouring stripes are available, only the significance pass membership indicator, which occupies one bit per a boundary coefficient, has to be stored in the memory. In each pass, the states from neighbouring stripes are read only at the current stripe. Hence, rewriting them by states of coefficients located on boundaries of the current stripe saves the memory space. This arrangement does not disturb the coding process because two adjacent stripes share the same memory entries exchanging data in compliance with the scan order. The memory has to be divided into two parts to accesses two boundaries of the stripe in parallel, as shown in Figure 2.

4. ARCHITECTURE

The designed architecture for the BPC block is shown in Figure 3. The bit-plane coder adopts the pipeline arrangement to shorten critical paths and employs on-chip memories to provide quick data access. All memory modules use two separate ports to write and read. Input memories buffer coefficients and have the capacity matching maximal possible size of the code-block (64 x 64). Consecutive coefficients from a given row are grouped so as to read/write four of them at one clock cycle.

Square-based scanning requires access to 16 coefficients, each of which provides its sign and one bit from a selected bit-plane. However, using the DSSR technique involves reading additionally 20 coefficients surrounding the scanned square. To facilitate access to coefficients from six rows in parallel, the architecture incorporates six memory modules (see Figure 4). The left

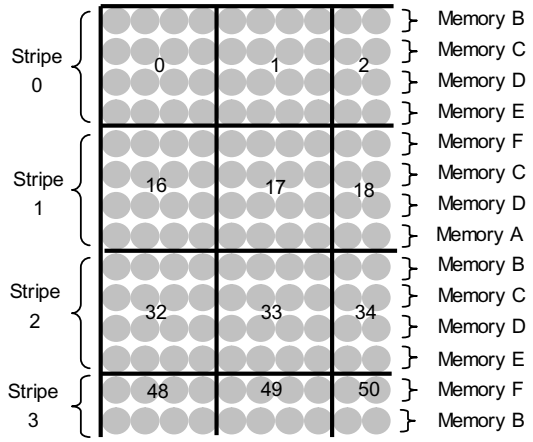

Figure 4. Memory arrangement for coefficients

border column of six states is obtained by registering one column from the previous square. The right border column is obtained by means of the look-ahead technique which analyses coefficients from the following square available at the preceding pipeline stage. Two memory modules are used to convey the significance pass membership variable between stripes, as discussed in Section 3. Sizes of memories incorporated into the design are summarized in Table 1.

At the beginning, input memories are initialised with quantized coefficients. Following the initialisation, 16 coefficients from a selected square along with eight ones located in neighbouring stripes are read in each clock cycle. Next, bits corresponding to some bit-planes are selected and loaded into registers. The selected bit-planes include the currently scanned one, three bit-planes located just below (computation of reductions in distortion), and the sign one. Additionally, state registers are set for two variables that are restored by checking upper bits of coefficients. These operations are accomplished at the stage 1 of the pipeline architecture. The stage 2 performs the significance propagation to restore states of coefficients corresponding to the analysed square of bits and immediate neighbourhood. The result for border coefficients of the current stripe is written back into memories. Four bit-planes of the processed square, updated significance state variables, signs, and pass membership indicators, which are necessary to form contexts and symbols, are provided to the stage 3. At the stage 3, four (or less) bits coded in the current pass with state variables of their neighbours are selected and forwarded to the next stage. If there are more than four coded bits, this operation requires additional clock cycles and halting previous stages. The stage 4 identifies context labels for selected bits on the basis of neighbour state variables and signs. Calculated contexts and symbols are written into the FIFO buffer. A single cell of the FIFO is comprised of 32 bits and is able to keep four or more context-symbol pairs (special packing method is applied).

Table 1. Summary of memories incorporated into the design

function	index	Number of entries	Data width [bits]
coefficients	A	128	40
coefficients	B	128	40
coefficients	C	256	40
coefficients	D	256	40
coefficients	E	128	40
coefficients	F	128	40
states	X	128	4
states	Y	128	4
contexts-symbols	FIFO	512	32

The last stage is not synchronized with others because it reads data from the FIFO and restores context labels and symbols to submit them to the arithmetic coder. The FIFO is accessed when necessary to adjust the rate of two context symbol pairs per clock cycle required by the CABAC block.

5. IMPLEMENTATION RESULTS

The proposed architecture is described in VHDL. Performance analysis was conducted with a set of images. The results are summarized in Table 2. Provided the BPC block generates data continuously, number of clock cycles utilized for encoding equals near the half of the total of symbols.

Digital synthesis has been performed targeting FPGA Stratix devices and AMS 0.35 m technology. The architecture attains the working frequency of 66 MHz for ALTERA Stratix devices. It enables the system to encode about 22 million pixels greyscale image within 1 s, which corresponds to the colour image size of 2400 x 3000. The designed module consumes about 4K Logic Elements for Stratix devices, which corresponds to 14K gates in AMS 0.35 m technology. The total size of on-chip memories is 57K bits. The amount of logic resources in the architecture not enhancement by DSSR is similar. So, the reduction of on-chip memories has been achieved without an additional cost.

6. CONCLUSIONS

This paper describes the architecture of bit-plane coder able to generate two symbols per one clock cycle. The design adopts the pipeline structure and benefits from the DSSR method to reduce on-chip memories. Digital synthesis has been performed for FPGA and ASIC technologies to verify timing performances. The estimated working frequency is 66 MHz targeting FPGA Stratix devices. It allows the system to encode about 22 million coefficients within 1 s, which corresponds to the colour image size of 2400 x 3000.

Table 2. Number of clock cycles needed to encode 512x512 images, lossless compression (5x3 filter core), two decomposition levels

Image	Clock cycles			
	Code Block 16 x 16		Code Block 64 x 64	
	grayscale	color	grayscale	color
Baboon	855237	2527810	907628	2664069
Lena	616321	2002392	691754	2182245
Jet	621466	1753902	699838	1990875
Peppers	699795	2196092	777866	2378313

7. ACKNOWLEDGEMENT

The work presented was developed within activities of VISNET, the European Network of Excellence, (http://www.visnet-noe.org), founded under the European Commission IST 6FP programme.

11. REFERENCES

[1] ISO/IEC 15444-1, Information technology - JPEG 2000 image coding system - Part I: Core Coding System, 2000.

[2] D. S. Taubman and M. W. Marcellin, JPEG2000: Image Compression Fundamentals, Standard and Practice. Norwell, MA: Kluwer, 2002.

[3] Yun-Tai Hsiao, Hung-Der Lin, Kun-Bin Lee and Chein-Wei Jen, "High Speed Memory Saving Architecture for the Embedded Block Coding in JPEG 2000," 2002.

[4] K. Andra, C. Chakrabarti, and T. Acharya, "A high performance JPEG2000 architecture," IEEE Trans. Circuits and Systems for Video Technology, vol. 13, no. 3, pp. 209–218, March 2003.

[5] C.-J. Lian, K.-F. Chen, H.-H. Chen, and L.-G. Chen, "Analysis and architecture design of block-coding engine for EBCOT in JPEG 2000," IEEE Trans. Circuits and Systems for Video Technology, vol. 13, no. 3, pp. 219–230, March 2003.

[6] Yijun Li; Aly R.E, Wilso B and Bayoumi M.A, Analysis and Enhancement for EBCOT in high speed JPEG 2000 Architectures," The 45th Midwest Symposium on Circuits and Systems, 2002. MWSCAS-2002., Volume: 2, 2002

[7] Hung-Chi Fang, Tu-Chih Wang, Chung-Jr Lian, Te-Hao Chang and Liang-Gee Chen, "High Speed Memory Efficient EBCOT Architecture for JPEG2000".

[8] Grzegorz Pastuszak, "A High-Performance architecture for arithmetic Coder in JPEG2000", ICME'04, Taipei, Taiwan, 2004.