LOW-POWER HIGH-THROUGHPUT MQ-CODER ARCHITECTURE WITH AN IMPROVED CODING ALGORITHM

Alireza Aminlou, Maryam Homayouni, M. R. Hashemi, Omid Fatemi

Multimedia Processing Laboratory, Faculty of Electrical and Computer Engineering,
University of Tehran, Tehran, Iran

ABSTRACT

In this paper, a high speed architecture with a well designed pipeline is presented for the arithmetic encoder in JPEG2000 algorithm. The coding algorithm has also been improved by changing the renormalization and byteout extraction steps. The proposed algorithm has been implemented with a four stage pipelined architecture in VHDL. The new design has reduced, and in some cases removed data dependencies. The proposed architecture has been synthesized on a Virtex2 FPGA and its power consumption, working frequency and encoding rate were analyzed. Simulation results indicate that the proposed design is able to encode a CIF video sequence at 47 frames/s.

1 INTRODUCTION

The recent growth of real-time multimedia applications elucidates the need for more efficient and faster encoders. For still image compression the new JPEG2000 standard has been approved by ISO/IEC JTC1/SC29/WG1 Joint Photographic Experts Group (JPEG). The applications of this standard are far-reaching, from consumer applications such as multimedia devices (e.g., digital cameras, personal digital assistants, etc.) and client/server communication (e.g., the internet), to other specific applications such as military/surveillance, medical, and storage of motion sequences, entertainment (e.g., digital cinema) [2]. The MQ-Coder, the context-based adaptive arithmetic coder, is the process with the highest computational cost within the complete JPEG2000 encoding algorithm. When aiming at real-time applications in mobile environments, it is extremely important to be able to reduce the coding time as well as the power consumption. In general the former is achieved through a reduction of the number of clock cycles required per unit of time, but achieving the latter requires more considerations.

Various architectures have been proposed to improve the throughput and performance of MQ-Coder [3]-[10]. In [8], long critical paths in updating CX table, renormalization and byte-out have been addressed. Also a short delay circuit has been proposed to detect the number of shifts. In the architecture proposed in [9], the serial shift operation in Renormalization has been modified to one judgment and one shift operation. Using this technique, only one clock cycle is required for renormalization. In [4-8] an arithmetic coder, capable of consuming more than one CX-D pair per clock cycle, is presented. In these papers various pipelined MQ coder architectures have been proposed that can process exactly two [3],[4],[7], two or three [6], and up to 10 context-data pairs[5] in one clock cycle.

In this paper a modified algorithm has been proposed that shortens the longest critical path caused by the updating of C register, in several steps. In addition and in order to realize the proposed algorithm a new 4-stage pipelined architecture with an improvement in renormalization and byteout generation has been implemented. The stages of this pipelined architecture are designed to shorten or remove the data dependencies that exist in traditional MQ-coder architectures.

In the rest of this paper, an overview of arithmetic coding is provided in section 2, our modified algorithm and the proposed architecture are presented in section 3, followed by simulation results and conclusions.

2 ARITHMETIC CODING OVERVIEW

The block coding engine of a JPEG2000 encoder consists of a bit-modeler and an MQ-Coder. The bit-modeler generates context-decision (CX-D) pairs which are then entropy coded by the MQ-coder. Each decision has an associated context, which determines its probability. The MQ-Coder, adaptive binary arithmetic coder, is based on recursive probability interval subdivision, which is initially set to [0, 1). According to the probability of the decision, the interval will be partitioned into two subintervals, the more probable symbol (MPS) and the less probable symbol (LPS). In Figure 1 below, the interval register A provides the length of the current interval, and the codeword register C the lower bound of the interval.

Depending on the decision, whether it is Less Probable Symbol (LPS) or Most Probable Symbol (MPS), these two registers are typically updated according to (1). Qe is the probability of the input symbol, determined by CX.

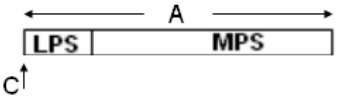

Figure 1: A represents the size of the interval and C points to its lower bound.

$$MPS : A = A - Qe \;, C = C + Qe$$
$$LPS : A = Qe \qquad , C = C \tag{1}$$

In the MQ-Coder algorithm, A and C registers are renormalized in order to keep A greater than 0x8000 (the required precision for A register). This is performed by doubling them both in a renormalization procedure whenever A falls below 0x8000. Periodically – to keep C from overflowing – a byte of data is removed from the high order bits of C and placed in an external compressed data string buffer during the BYTEOUT procedure. Then the extracted bits of C register are replaced with zeros in the Mask_C step. For more information on MQ-Coder please refer to [1].

3 THE PROPOSED ARCHITECTURE

The data flow of MQ-coder and its data dependencies are shown in Figure 2. In this paper we propose a modified algorithm to remove the data dependencies caused by updating C register ⑤, update CT ⑥ and byte out extraction ⑦. Also update/renormalization of A and C registers' path (3), (4) were shortened. The modified algorithm is then implemented with a proposed four-stage pipelined MQ-Coder architecture with balanced delay of each stage. For more details in Figure 2, please refer to [1].

3-1 Modified algorithm

Generally, data dependencies happen when we partition pipeline stages of an architecture. For instance when reading from and writing back to a register are performed in different stages. Data dependencies must be managed using data forwarding or stalling techniques. The former technique increases the critical path and the latter consumes extra clock cycles. As shown in Figure 2, the C register is modified in Update_C, Renormalization_C, and Mask_C which are not located normally in the same stage. To reduce or remove these dependencies (4 and 5), we proposed two modifications in the MQ-Coder algorithm. The first modification is to divide the C register into two 16-bit registers: C-High and C-Low. This modification is feasible because the most significant bits of C, C-High, do not take part in the Update_C step computations and are used only for extracting byteout. Instead, C-Low is only used in updating C. As a result, instead of a 32-bit adder, a 16-bit adder can be used in Update_C which reduces the critical path delay in dependency (4).

The second modification is to eliminate the data dependency (5) in Figure 2 by removing Mask_C step. As a result the value of C-High register remains unchanged after byteout extraction. According to the standard, when the byteout is extracted from the most significant bits of C, the extracted bits are replaced by zeros in Mask_C step.

C-High register and the location of extracted bytes are depicted in Figure 3. As a general case, byteout is extracted from bits 3 to 10 of C-High, the shaded bits in Figure 3-a. Since we eliminate Mask_C step, when the next byteout (i+1) is ready to be extracted as byteout, the previous byteout (i) has been shifted left and is placed in bits 15-11, shaded bits shown in Figure 3-b.

In a standard MQ-Coder, the eleventh bit of C-High (Figure 3-b) is the carry propagated from C-Low computations and shift operations. But due to the elimination of Mask_C, the previous byteout (i) remains in C-High. Thus, C-High(11) would be the exclusive OR of the first bit of byteout (i) and the carry propagated from C-Low. The value of this bit, C-High(11), is checked in a standard MQ-Coder algorithm to change the manner of extracting byteout as shown in Figure 4 (Standard). Therefore, in the proposed method, this condition is changed accordingly, as shown in Figure 4 (Modified).

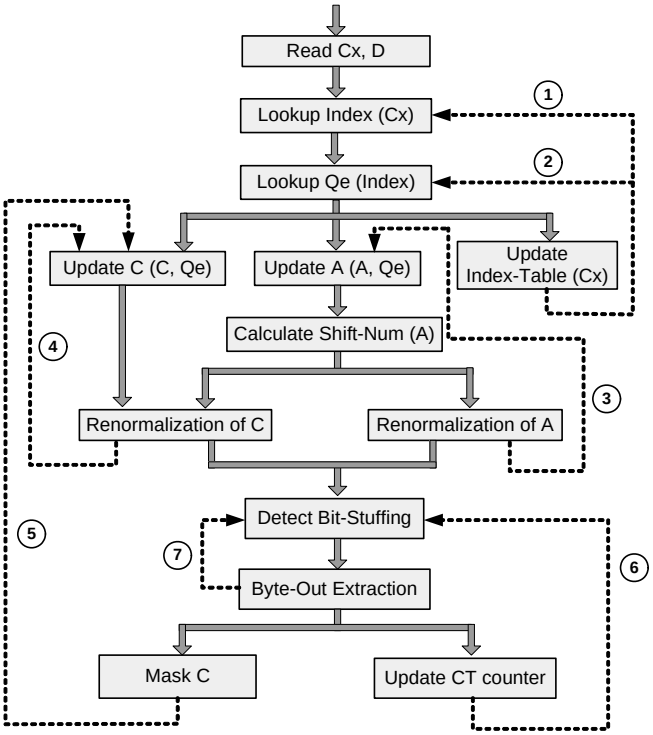

Figure 2. The data flow of Binary Arithmetic Coder

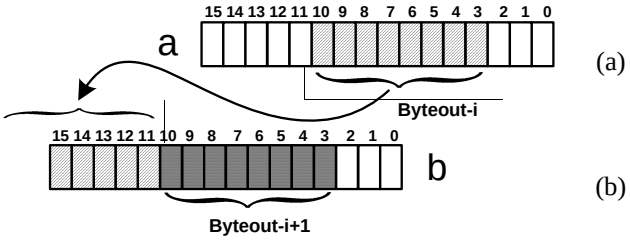

Figure 3. (a) C-High register in extracting Byteout-i (b) C-High register in Byteout-i+1 extraction

Standard	If (C-High (11)='0') then ...
Modified	If ((C-High(11) **XOR** byteout-i(0)) ='0') then ...

Figure 4: The effective condition in byteout extraction

3-2 Hardware Architecture

The modified algorithm has been realized with a four-stage pipelined architecture that balances the critical paths of different stages and shortens data dependencies, shown in Figure 2. The overall architecture of the pipelined arithmetic encoder is shown in Figure 5. The first stage includes probability estimation and MPS/LPS coding recognition. In the second stage update and renormalization of A and C-Low registers is performed. The third stage includes updating CT register and renormalization of C-High register and the fourth stage contains byteout extraction with bit-stuffing.

The inputs of the first stage are Context-Decision (Cx,D) pairs. In stage1, the Index of the current Cx is looked up from Cx table. Moreover the more probable symbol (MPS) of current Cx is read from Cx-table. Index of the context table is also updated from the feedback information from the second stage. To have the correct Index-MPS pair, a forwarding technique is used to deal with the case of identical successive contexts. In case of identical Cx, Index-MPS might be selected from the one, which is forwarded from second stage. Then using the correct index, corresponding Qe value is looked up. Also, according to the value of D and the correct MPS, MPS-coding or LPS-coding is identified.

The second stage includes update and renormalization of A and C-Low registers. The probable carry occurring during update and the number of shifts performed in renormalizing C-low are dispatched to the next stage. Dividing C register to C-Low and C-High and achieving update and renormalization of C-Low in one stage resulted in a shorted dependency path no. 4 in Figure 2. Dependency path no.3 is also shortened due to performing partial renormalization of A register in the same stage, which is discussed below.

The number of shifts is between 0 and 15 bits which requires either a large barrel or a small serial shifter. The barrel shifter leads to large area and long critical path. The later imposes several stalls in the pipelined architecture. An optimized hidden shifting method is also proposed and used in the second stage. Using hidden shifting technique not only shortens the critical path of shift operation, but also decreases the number of required stalls. Our proposed shifting block consists of two shifters: shifter_1 and shifter_2. Shifter_1 follows update_A_C and performs almost all the required shifts in one clock cycle. Shifter_2 is placed in parallel with update_A_C and assists performing shift operating in remaining cases. In the other words, the critical path of shifter_2 is hidden under the critical path of Update_A_C.

Critical path of design refers to Updating_A_C in stage2 followed by forwarding index and index table in the first stage. Shifter_1 must be designed such that the sum of Upadate_A_C and Shifter_1 delays do not affect the system's critical path. Therefore, we designed Shifter_1 with 3 bit shifter supporting 0 to 3 bit shifts. As a result, the delay of the data dependency (3) is reduced. We also designed Shifter_2 as a 0-4-8 shifter in parallel with Upadate_A_C to utilize its delay path.

Table 1 shows the average percentage of shift counts for encoding 11 different color images. The tested images were of size 800×600 with 8 bit precision, and a tile size of 256×256, a codeblock size of 64×64 and three resolution levels were used for encoding. In Table 1, in addition to the average, the results for the test sequences 2, 5, and 11 are presented, because their percentages for zero shift number are the minimum, the maximum and average value.

Experimental results indicate that in 99.81% of renormalization step, the number of required shifts is between 0 to 3 bits. Using shifting method explained above, the renormalization step lasts only one cycle in these cases. In the remaining cases (0.19%), when the number of required shifts is between 4 and 15, data goes through the modules "Update A C_low" and "Shifter_1" during the first clock cycle and through "Shifter_2" and "Shifter_1" in the second cycle.

The third stage consists of updating C-High and CT registers. C-High register is updated, using the dispatched carry, shift number and C-Low from previous stage. Besides CT register is decremented based on the number of shifts performed in the previous stage. The byteout extraction is then activated for the next stage, when CT reaches zero or as it rolls over.

Table 1: Percentage of shift counts in 11 images

Shifts	0	1	2	3	4-15
Average	39.68	48.09	11.56	0.48	0.19
Image #2	29.68	56.51	13.30	0.39	0.124
Image #5	65.78	26.87	6.49	0.59	0.276
Image #11	43.12	44.26	11.79	0.63	0.210

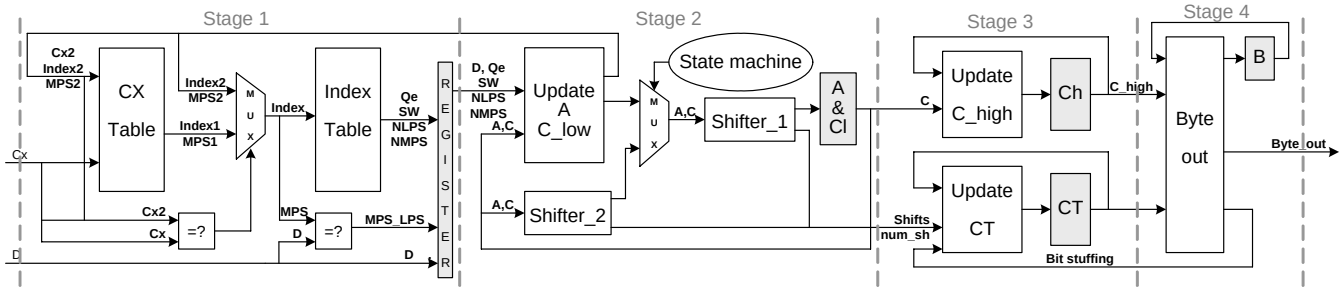

Figure 5. The proposed four stage pipelined MQ-Coder architecture

The fourth stage includes Detect_bit_stuffing and Byte_out_extraction steps. Placing these two steps in a same step removes data dependency path (7) in Figure 2. Also updating CT register before bit stuffing detection, in former stage, result in elimination of the data dependency path (6). According to the standard, Detect_bit_stuffing may cause an additional decrement in CT. Therefore, a bit_stuffing signal is dispatched to the third stage, notifying the bit stuffing detection. We also eliminate Mask_C step from this stage as mentioned before. As a result, the bits of the C register extracted as byteout are not filled with zeros and thus C-High remains unchanged. Therefore, to preserve the way the byteout operation is performed, which depends on the carry propagated to the 11th bit of C-High, a modified condition will be used instead of the standard one, as Figure 4.

4 EXPERIMENTAL RESULTS

The proposed design is described by VHDL, verified in ModelSim and synthesized using ISE and Quartus tools on various FPGAs with the lowest speed grade. The simulation parameters are the same as what presented for shift count, above Table 1. The synthesis results, presented in Table 2, provide working frequency of 86 MHz and power consumption of 100mW. Based on the number of clock cycles for different shift values and their percentages in Table 1, the proposed design processes 85.8 MegCx/sec at 86 MHz. Simulation results show that each pixel is equivalent to about 12 contexts for color images. Hence the proposed design is able to process 7.15 million pixels per second, corresponding to CIF (352×288) video sequence at 47 frames per second or a 2400×3200 color image at one frame per second. According to Table 3, where synthesis results for our design are presented inside () for different targets, the proposed architecture has a higher working frequency (and hence a higher throughput) and lower area compared to other existing architecture. This is while it has a reasonable power consumption (100mW), suitable for mobile applications. Note that since none of the existing designs have reported their power consumption we are not able to perform an exact fair comparison in this regards.

Table 2: Synthesis results of the proposed MQ-Coder

	Stage1	Stage2	Stage3	Stage4	Total
Gates	2879	2593	813	826	7100
Delay (ns)	9.09	10.22	6.79	8.1	11.6
Power (mW) (net+Logic)	51 38+13	56 29+27	9.5 5+4.5	5.5 3.5+2	100 58+42

Table 3: The proposed and existing designs

	Target	Freq. (**Ours**)	Area (**Ours**)
[3]	Apex	26 (**66**)	1811LC (**756 LC**)
[6]	Stratix	77 (**110**)	4K LE (**755 LE**)
[9]	0.18um	200 (--)	7 KGates (**7.1 KG**)
[8]	Stratix	107 (**110**)	1256 LE (**755LE**)
[9]	0.35um	50 (--)	11 KGates (**7.1 KG**)

5 CONCLUSION

In this paper, an improved arithmetic coding algorithm for the JPEG2000 standard has been proposed. The corresponding architecture for the optimized encoder has been implemented with a 4-stage pipeline. By removing the complexities in interval division and byteout extraction and using a hidden shifting method in renormalization, our proposed scheme is able to shorten the data dependency paths, resulting in a higher clock frequency and less stall cycles. Synthesize on a Xilinx Virtex2, xc2v6000 FPGA results in 7.1 KGates and 100 mW at 86 MHz. The encoding rate is 85.8 millions contexts per second, corresponding to CIF video sequence at 47 frames per second.

REFERENCES

[1] M. Boliek, C. Christopoulos, and E. Majani (Editors), “JPEG 2000 Part1 Final Publication Draft”, ISO/IEC JTC1/SC29/WG1 N2678, July 2002.

[2] P. R. Schumacher, “An Efficient JPEG2000 Tier-1 Coder Hardware Implementation for Real-Time Video Processing ”, IEEE Transactions on Consumer Electronics, vol. 49, no. 4, pp. 780–786, 2003

[3] M. Dyer, D. Taubman, and S. Nooshabadi, "Improved Throughput Arithmetic Coder for JPEG2000", International Conference on Image Processing, vol.4, pp. 2817-2820, Singapore, 2004.

[4] Y. Li, M. Elgamel, and M. Bayoumi, "A Partial Parallel Algorithm and Architecture for Arithmetic Encoder in JPEG2000", Vol. 5, pp. 5198-5201, ISCAS, Japan, 2005.

[5] A. K. Gupta, D. Taubman, and S. Nooshabadi, “High speed VLSI architecture for bit plane encoder of JPEG2000”, 47th Midwest Symposium on Circuits and Systems, vol. 2, pp. II-233–II236, Japan, 2004.

[6] G. Pastuszak, "A novel architecture of arithmetic coder in JPEG2000 based on parallel symbol encoding", Inter. Conf. on Parallel Computing in Electrical Engineering, pp. 303-308, Germany, 2004.

[7] Y. Z. Zhang; C. Xu; L. B. Chen, "A Dual-Symbol Coding Arithmetic Coder Architecture Design for High Speed EBCOT Coding Engine in JPEG2000", 6th ICASIC, vol.1, pp. 261-264, China, 2005.

[8] M.kuizhi, Z.Nanning, U.Ji, L.Yuehu, "Design of high speed Arithmetic Encoder", 7th International Conference on Solid-State and Integrated-Circuit Technology, vol. 3, pp. 1617-1620, China, 2004.

[9] K.Zhu, F.Wang,X.Zhou, and Q.Zhang, "An efficient accelerating architecture for tier-1 coding in JPEG2000", 7th International Conference on Solid-State and Integrated-Circuit Technology, vol. 3, pp. 1653-1656, China, 2004.

[10] C.-J. Lian, K.-F. Chen, H.-H. Chen, L.-G. Chen, "Analysis and Architecture Design of Block-Coding Engine for EBCOT in JPEG 2000", IEEE Tran. on Circuit and Systems For Video Technology, vol. 13, no. 3, pp. 219-230, 2003.