ABSTRACT
The key algorithm in JPEG2000 image compression system is embedded block coding with optimized truncation (EBCOT). The EBCOT scheme consists of a bit-plane coder coupled with a MQ arithmetic coder. Recently, the bit-plane coding can generate more than one symbol per clock cycle. Consequently, the coding speed is limited and bottlenecked at the interface between the output of the bit-plane coding and the input of the MQ arithmetic coder. Moreover, an efficient designed architecture for MQ arithmetic coder should compromise between processing speed and hardware cost. Therefore, a single symbol processor for arithmetic coder architecture implemented on FPGA is proposed in this paper, since it offers high throughput but requires low hardware cost. The proposed architecture is separated into 2 pipelined stages to break down the whole task into smaller sub-tasks, which leads to great reduction in the critical path. Consequently, it demonstrates no stall, high clock speed and high throughput in the encoding process. These benefits are achieved with the suitable hardware design, pipelining technique, pre-calculation, and prediction process. As a result, the coding speed can be at least 188.99 MHz with the throughput of 188.99 MCxD/S.

KEY WORDS
MQ-Coder, Arithmetic Encoder, JPEG2000, Hardware Architecture, FPGA Implementation

1. Introduction

JPEG2000 is a recent image compression standard [1]-[2]. It supports features like scalability, error resilience, high compression ratio, and others. The JPEG2000 adopts the discrete wavelet transform as its primary transform algorithm. The transformed coefficients are split into codeblocks and are sequentially processed by an entropy coding algorithm known as embedded block coding with optimized truncation (EBCOT).

The EBCOT algorithm consists of two main parts, which are Tier-1 and Tier-2. Tier-1 is a complete context-based entropy coder consisting of two processing stages: the bit-plane coding and the arithmetic coding (MQ-Coder). The bit-plane coding module generates the context of symbols while the arithmetic coder performs entropy coding. Tier-2 is a post-compression rate allocation processor. The major timing limitation is caused by the MQ-Coder, because it inherently depends on control statements and arithmetic operations. As a result, the MQ-Coder becomes bottleneck of the entire JPEG2000 encoding system.

In this paper, we examine and implement techniques that can be used to improve throughput thus occupy less hardware costs.

The rest of the paper is organized as follows. Section 2 overviews the algorithm of MQ-Coder while the proposed MQ-Coder architecture is described in section 3. Experimental results are given in section 4. Finally, concluding remarks are made in section 5.

2. Overview for MQ-Coder

“In JPEG2000, the bit-plane coding and MQ-Coder are the key modules for the EBCOT. Bit-plane coding is a context adaptive algorithm which produces a sequence of symbols, pairs of context (CX) and decision (D), for each coding pass. MQ-Coder then processes the context (CX) together with the decision value (D) to obtain entropy encoded symbols which will be output to Tier-2 to form separated bit streams for each code block. MQ-Coder is Binary Arithmetic Coder (BAC) based on the QM-Coder used by Joint Bi-level Image Experts Group (JBIG), but uses the byte emission technique of the Q-Coder. MQ-Coder uses context based probability estimation of 19 contexts. Each context has an associated probability state that identifies value of the most probable symbol (MPS) and index (I) pointing to a probability estimation table (PET). After a symbol has been coded, the corresponding state is updated according to the probability mapping rules. PET determines the probability estimation (Qe) for the least probable symbol (LPS), the next index value (NLPS, NMPS) and the exchange indicator (SWITCH) of the MPS. MQ-Coder module contains a set of registers: A, B, C, CT and L. MQ-Coder is implemented using fixed precision integer arithmetic and using an integer representation of fractional values in which 0x8000 is equivalent to decimal 0.75. The interval A is kept in the range 0.75 ≤ A < 1.5. Whenever, the value of register A falls below 0.75, its value will be doubled together with the value of the register C. Then, the down counter, CT, is decremented. If the CT value reaches 0, the previous value of the register B is released to the output.
The value of the byte counter (L) is incremented. Finally, the high order bits of the register C are moved out to temporal buffer B." [5-6]

Usually, the research of MQ-Coder architecture can be divided into 2 groups. One mainly processes a single symbol per clock ([8-10]) while the other processes multiple symbols per clock ([5-7]).

In the group of single symbol processors, the proposed MQ-Coder architecture in [8] is simple and direct which offers high clock rate and requires low hardware cost. Furthermore, the critical path in interval process is reduced by the prediction of the next index and the optimization of renormalization in [9]. Besides, in [10], the register C is split into 2 sub-registers (16-bit \( C_{\text{Low}} \) and 12-bit \( C_{\text{High}} \)) to reduce the critical path stemmed from the 28-bit adder. Each fragment of the register C is updated in consecutive pipeline stages. This helps reduce critical path in byte out process. Even through all the architectures in [8-10] provide fast implementation, it is still a challenging task in designing the architecture to accelerate coding speed faster and faster. The proposed systems in [5-6] demonstrate no stall, high clock speed and high throughput in the encoding process. These benefits are achieved with the suitable hardware design, pipelining technique, pre-calculation and prediction process.

On the other hand, in the group of multiple symbol processors, the architectures based on ‘hypothesis testing’ and ‘brute force method’ are proposed in [7]. In the hypothesis testing technique, MQ coder logic has been partially unrolled. This architecture contains stalls when renormalization occurs. In brute force method, the logic associated with two successive symbols encoding steps has been fully unrolled. Both methods can process two or more symbols in every clock cycle. In addition, the reduction of the critical path in the interval process of brute force method has been proposed in [5] using the prediction of the next index and next interval (A). Moreover, the partition of the register C into \( C_{\text{Low}} \) and \( C_{\text{High}} \) from [10] has been adopted in [6]. Although all the architectures in [5-7] offers high coding performance, the architectures are complex thus require long critical path and large memory consumption.

The simplicity, low hardware cost, high clock rate and high throughput are the benefits of the single symbol architecture. On the other hand, complexity, higher hardware space, low clock speed and high throughput are the characteristics of multiple symbol architecture. Accordingly, the proposed architecture chooses single-symbol processing for the proposed MQ-coder. The proposed architecture adopts several techniques in order to reduce critical path, thus accelerate the coding speed.

## 3. The Proposed MQ-Coder Architecture

At this stage, the key for good development is to compromise between processing speed and hardware cost. Therefore, the single-symbol architecture is chosen for the proposed MQ Arithmetic coder since it offers high throughput but requires low hardware cost. The proposed architecture, the “Fast MQ”, is illustrated in Figure 4. The Fast MQ is organized into 2 pipeline states. The first state is the RAM function. Next is the main operation, composing of index and MPS prediction, ROM, interval update, and RAM update, respectively. Using pipeline state separation technique helps to detach the independent tasks and break down whole task into smaller sub-tasks, which leads to great reduction in the critical path.

In the first state, RAM (Table I, Table MPS) is read and the values of index and MPS are then sent to the next state. Next, the index and MPS prediction will be activated. The prediction process is introduced in order to alleviate the problem occurring when the system needs to read the index and MPS values during the RAM update process. This is due to the delay time in updating RAM, which requires at least 1 clock cycle. The problem primarily takes place if renormalization occurs and the earlier and current context pairs are the same. Figure 2 illustrates a pseudo code for index updating process. The value of the index will be changed in the case of renormalization. Otherwise, its value will remain the same. Similarly, Figure 3 shows a pseudo code for MPS updating process, which its value will be changed when CODE LPS and SWITCH bit set. Moreover, the repeated input context pairs must be examined since it may introduce the previous mentioned problem and cause long critical path.

Next step, ROM (Table NLPS, Table NMPS, Table Qe, Table SWITCH, Table RENORM Qe, Table Qe+0x8000) is lookup from predicted index obtained from previous step.

Interval update process is the major step in the proposed Fast MQ encoder. It is a challenging task in designing the optimum hardware architecture to minimize the critical paths. In the proposed Fast MQ, we have revised conditions and operations in order to be suitable for hardware architecture. This leads to a great reduction in the critical path as illustrated in Figure 1.

![Figure 1 The renormalize of register A.](image)
(A\textless Qe+0x8000) is chosen instead of the old condition (A-Qe\textless 0x8000) [5]. The old condition requires a 16-bit subtractor while the new one requires no subtraction operation. The addition term is pre-calculated and kept in ROM (Table Qe + 0x8000). This helps to reduce the computational path. Moreover, A\textless=2\times Qe condition [5], which requires 16 bit left shifter, can be removed. The multiplication can be implemented using concatenation technique, where the result from table Qe (15 bit) will be concatenated with ‘0’ (1 bit) at the LSB. This can further decrease the critical path. Additionally, the possible results before renormalization can be Qe or A-Qe. We can calculate the renormalized results of these conditions in advance. For the term of Qe, the \text{RENORM}(Qe) will previously be calculated and stored in ROM. Meanwhile, for A-Qe condition, we can implement using 16 bit subtractor, leading zero detector circuit (2 bit) and barrel shifter (16 bit), respectively. The number of leading zero bits is suggested in [5] according to the condition of minimum difference between A and Qe. It has been stated that MIN(A) = 0x8000 and MAX(Qe) = 0x5601. This results in the minimum value of A-Qe (0x29FF) thus produces two leading zero bits at maximum. The less value of the A-Qe condition, the greater number of leading zero. The number of leading zero bits effect the number of renormalization. In this case, the maximum number of renormalization of A-Qe condition is 2.

Table 1 presents the select enable signals used to select the correct output for updating A. The select enable signals depend on the value of \text{CODE LPS} and \text{A\geq2\times Qe} condition. If \text{CODE LPS} and \text{A\geq2\times Qe} are both true or false, the result will be in term of Qe. Otherwise, the result will be in term of A-Qe. This relationship can be considered as XOR gate or XNOR gate characteristic. We choose XOR gate module to generate the select enable signals for Fast MQ. As previous mention, the proposed revised conditions and operations helps to accelerate encoding process while reduce the critical path of the system.

At the end of the second state, new index and new \text{MPS} will be determined in RAM update process. In fact, index may be changed or unchanged. The index will be changed according to either Code LPS (NLPS) condition or Code MPS (NMPS) condition as shown in Figure 2. Similarly, MPS may be changed or unchanged. If the value of MPS changes, the result is the complement of its current value as illustrated in Figure 3.

Fast MQ comprises of synchronous circuits. Its whole task can be separated into smaller sub-tasks, called the pipeline. Moreover, Fast MQ exploits buffers call pipeline registers for synchronization between two stages. When the pipeline registers are clocked, the data from previous stage will be transmitted to the current stage. With the assistance of the pipeline registers, it offers excellent synchronization between stages thus leads to great reduction in the critical path

As previously described, Fast MQ has been designed to simplify the hardware architecture as much as possible by revising conditions and operations. Its simplicity and uncomplicated architecture leads to a significant reduction in the critical path and consequently is suitable for implementation.

<table>
<thead>
<tr>
<th>CODE LPS</th>
<th>A \geq 2\times Qe</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Qe</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>A-Qe</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>A-Qe</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Qe</td>
</tr>
</tbody>
</table>

IF \text{D}_1 \neq \text{MPS(CX}_1) \text{ OR } A_1 < Qe_1 + 0x8000 \text{ THEN} \text{ index change}
ELSE
\text{ index not change}
END IF

IF \text{D}_1 \neq \text{MPS(CX}_1) \text{ AND SWITCH(CX}_1) = 1 \text{ THEN} \text{ mps change}
ELSE
\text{ mps not change}
END IF
4. Experimental Results

To evaluate the performance of proposed system, the test vectors of input symbols obtained from three ISO images (Lena, Baboon, Peppers) are used. All images are decomposed up to five levels using lossless wavelet transform. Using JAVA simulation [3], all codeblocks in each image are encoded and context pairs are generated for testbench. Fast MQ is implemented on Virtex-4 FPGA device using the Project Navigator 9.2.04 tool. The cost of designed hardware for the proposed system is illustrated in Table 2. The maximum frequency is 182.82 MHz while the equivalent gate count is only 8,252.

Table 3 shows a comparison of clock rate and throughput between Fast MQ and the systems proposed by [8] and [10]. The MQ-Coder proposed in [8] uses single-symbol processing and its architecture is partitioned into 3 stages: Interval Detector (ID), Renormalizer (REN) and Byte Out (BO). The ID stage checks either all contexts in the CxD input buffers are processed or not. If any context is left unprocessed, it is read and processed. The REN stage is responsible for adjusting the interval range. The BO stage captures upper 9 bits of C register (i.e. C[27:19]) which forms actual compressed data to be emitted in the next iteration. Besides, it is implemented using Altera’s Stratix FPGA.

The throughput of the system proposed by [8] (153 MHz.) is greater than clock speed (137.7 MHz.). This demonstrates that the encoding process contains stalls and this problem occurs in the byte out process. Since the proposed Fast MQ exhibits no stall, its clock rate and throughput are about 23.52% faster and 37.24% higher than those of [8].

In [10], the proposed MQ-Coder uses single-symbol processing and its architecture is also partitioned into 3 stages: Probability Estimation, Update Interval and Byte Out. Moreover, this architecture is supervised by a scheduler that has 3 functionalities namely interfacing, stall detection and stage controlling. By exploiting these functionalities, scheduler is able to handle different scenarios. Besides, the systems proposed by [10] implemented using Xilinx’s Virtex-5 FPGA.

The clock rate of the proposed Fast MQ (188.99 MHz.) outperforms about 43.17% to those of [10] (132 MHz.). Since the both systems use single-symbol processing and no stall in the encoding process thus the throughput outclasses likewise about 43.17%.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Device</th>
<th>Clock Rate (MHz)</th>
<th>Throughput (MCxD/S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[8]</td>
<td>Stratix</td>
<td>153</td>
<td>137.7</td>
</tr>
<tr>
<td>[10]</td>
<td>Virtex5</td>
<td>132</td>
<td>132</td>
</tr>
<tr>
<td>Proposed Fast MQ</td>
<td>Virtex5</td>
<td>188.99</td>
<td>188.99</td>
</tr>
</tbody>
</table>

5. Conclusion

This paper presents fast implementation of the interval process of JPEG2000 arithmetic encoder on FPGA. To solve long delay problem of a regular MQ-Coder, the proposed Fast MQ architecture is operated in 2 pipelined states to break down the whole task into smaller sub-tasks, which leads to great reduction in the critical path. Its architecture has been designed to simplify and to be suitable for the hardware architecture as much as possible by revising conditions and operations. Its simplicity and uncomplicated architecture help to accelerate encoding process and similarly is fast encoding for implement pre-calculation.

The proposed system demonstrates no stall, high clock speed and high throughput in the encoding process. These benefits are achieved with the suitable hardware design, pipelining technique, pre-calculation and prediction process. As a result, the coding speed can be at least 188.99 MHz with the throughput of 188.99 MCxD/S.

References


