A high performance fully pipeline JPEG-LS encoder for lossless compression

Byung-Soo Kim¹a), Seungkyu Baek¹, Dong-Sun Kim², and Duck-Jin Chung¹

¹ School of Information and Communication Engineering, INHA University, 253 Yonghyun-Dong, Nam-Gu, Incheon 402–751, Korea
² Department of the Multimedia IP Research Center, Korea Electronics Technology Institute, 68 Yatap-dong, Bungand-Gu, Seongnam, Gyeonggi-do 463–816, Korea

a) soc_bskim@inhaian.net

Abstract: This paper presents the design of a JPEG-LS encoder for lossless image compression. The proposed encoder is mainly composed of a context determination block, a pixel prediction block, a prediction error encoding block, a context update block, a code appending block. All operations are fully implemented using efficient pipeline method to increase a JPEG-LS encoder’s throughput for real time applications and the original LOCO-I compression algorithm is not modified to compliant with the standard. The proposed encoder uses forwarding technique and pipeline method to avoid hazards, processing speed of the encoder has increased to 120 Mpixels per second. The proposed encoder has been verified by using FPGA implementation. Experimental results show that proposed JPEG-LS encoder is suitable for real time and mobile devices that high speed and low computational cost are a priority.

Keywords: lossless image compression, FPGA implementation, JPEG-LS, pipeline architecture

Classification: Integrated circuits

References

1 Introduction

Among the various lossless compression algorithms, the JPEG-LS compression algorithm is one of the established standards for lossless compression of continuous-tone, gray-scale, or color digital still images [1]. It is an attractive option for a variety of costumer electronic applications since it combines relatively high compression rates with high quality at low computational complexity. The JPEG-LS algorithm is comparable to other leading lossless compression algorithms in terms of compression capability [3]. But it is less complex than other leading algorithms since the JPEG-LS algorithm adapts LOCO-I (Low Complexity Lossless Compression for Images) which has been proposed by Weinbeberger [2]. The JPEG-LS algorithm is very suitable for hardware implementation due to its low storage requirement and low complexity. Also the JPEG-LS algorithm uses only simple operations, such as bit shift operation, addition, comparison, without the complex operations. Consequently, JPEG-LS would be a suitable candidate for FPGA implementations.

To increase the operating frequency of the JPEG-LS encoder using pipeline architecture is necessary. However, the LOCO-I algorithm has intrinsic problem to implement pipeline architecture, since the calculation of the prediction error and the update of the contexts parameter are data-dependant. If the contexts are the same, the second pixel will be elaborated based on an out-of-date values. We can solve this problem to stall the pipeline for several cycles, but stalling the pipeline can decrease the speed of the JPEG-LS algorithm. Hence, an appropriate method must be taken to implement pipeline architecture of the JPEG-LS. The solution in [4] is using two different context tables for achieving a speed up of almost 2. But the implementation requires a double size of memory, resulting into a design not fully compliant with the standard. In [5], multiple clock domains are proposed to implement fully pipeline architecture, but throughput is still the problem.

In this paper, in order to solve the problem and maintain the high processing speed and low power operating of the JPEG-LS encoder, forwarding technique and a fully pipeline architecture have been used. The rest of this paper is organized as follows. Overview of the JPEG-LS algorithm is presented in next section. Section 3 details the proposed hardware architecture and the implementation of the JPEG-LS encoder in depth. Section 4 shows the measurement results and the conclusions are presented in Section 5.

2 Overview of JPEG-LS algorithm

The JPEG-LS is an image compression standard designed to provide effective lossless and near lossless compression of gray scale and color still images. One of its main potential driving forces is the compression of high quality images without any quality loss. Besides lossless compression, JPEG-LS also provide a near lossless mode where the maximum error can be controlled by the encoder. Compression for JPEG-LS is generally much faster than JPEG2000 and much better than the original lossless JPEG standard. The JPEG-LS algorithm consists of four main stages, a context modeling, an error prediction, an error encoding and a context parameter.
updating. Run-length coding is used to encode flat region in the run mode of JPEG-LS and the Limited Length Golomb code encode a prediction error in the regular mode.

The block diagram of proposed architecture of JPEG-LS is illustrated in Fig. 1. The first step in encoding a JPEG-LS image is modeling the context. This is done using the values of the adjacent to the current pixels. The differences $D_1 = R_d - R_b$, $D_2 = R_b - R_c$, and $D_3 = R_c - R_a$ are each a difference of neighboring pixel values. These differences are each categorized into quantized gradients $Q_1$, $Q_2$, and $Q_3$ using thresholds. There are nine possible values that the quantized gradients may assume in the integer range $-4$ to $4$. With nine possible values for each quantized gradient, there are a total of 729 possible vectors. But the number of contexts may be halved because of symmetry. This gives 364 nonzero vectors and one zero vector, to arrive at the total of 365 possible vectors. And the last step in the context modeling is map the vector to an integer $Q$.

In the error prediction stages, a value $P_x$ is predicted based on the surrounding reconstructed pixels. This predicted value $P_x$ is the corrected with the prediction correction value $C[Q]$ and clamped in the appropriate range. Then, the prediction error, $Errval$, shall be computed by using the value of $P_x$ and reduced to the range relevant for coding. The next step is to encode the prediction error value. Before encoding of the prediction error, Golomb coding variable $k$ is computed and prediction error is mapped. The calculation of $k$ is performed, using the accumulated prediction error magnitude value $A[Q]$ and the number of occurrence of each context $N[Q]$. And the prediction error shall be mapped to a non-negative values, because the Golomb-Rice code was designed for coding non-negative values. Then, mapped error is encoded with the limited length Golomb codes. The last step of the JPEG-LS algorithm is to update the variables. The variables are updated according to the current prediction error. The magnitude of the quantized error and the signed, quantized error are added to $A[Q]$, $B[Q]$, respectively. And if $N[Q]$ has reached the $RESET$ value, $A[Q]$, $B[Q]$, and $N[Q]$ are halved to prevent the values from exceeding the limits of the memory. A detailed description of JPEG-LS algorithm is contained in [1].

![Block diagram of proposed JPEG-LS architecture](image)

**Fig. 1.** The proposed JPEG-LS block diagram

### 3 Proposed JPEG-LS implementation

The hardware architecture of the JPEG-LS encoder consists of a context determination block, a pixel prediction block, a prediction error encoding block, a context update block, a code appending block, as shown in Fig. 2.
A context determination block performs a local gradient computation, a mode selection, a local gradient quantization and a quantized gradient merging. First, the neighboring pixels stored in image line buffer are used to compose the predictive template. The next step in the context determination procedure shall be to compute the local gradient values by using neighborhood samples $Ra$, $Rb$, $Rc$ and $Rd$. Then encoding mode is decided regular mode or run mode by the local gradients. The context determination procedure shall continue by quantizing the local gradient values and generate 364 possible vectors and the variable $SIGN$.

In the regular mode, a pixel prediction block performs an edge detecting prediction, a prediction correction, and a prediction error computation. First, Edge-detecting predictor compute $P_x$ by using neighborhood samples. After $P_x$ is computed, prediction corrector corrects the prediction by using the $P_x$, a context table and a Q-table. Using the new value of $P_x$, corrected by the above procedure, the prediction error, $Errval$, shall be computed in a prediction error computation block. And prediction error shall be reduced to the range relevant for coding. In the regular mode, if consecutive pixels have the same $Q$, we wait for completion of the first on, for the updating of the context. It causes the reduction of throughput of the JPEG-LS encoder and it precludes the JPEG-LS encoder implementing fully pipelined. To solve this problem, we utilize pre-computation method in all update cases as shown in Fig. 3 (a). Since the next prediction correction value will either be the same, or incremented, or reduced by one in case of occurrence of same context, pre-computation method can be applied. In the next stage, we determine a prediction error $Errval$ of the three cases according to the parameter $SEL, C$ that indicates previous prediction correction value.

The next step of the regular mode is encoding the prediction error. A prediction error encoding block is divided into three parts, including golomb coding variable computation part, error mapping part and golomb coder. In the golomb coding variable computation part, a context table is used to compute the golomb coding variables $k$. Also, only simple comparator and shifter are used to implement for reducing the hardware complexity. And the prediction error, $Errval$ shall be mapped to a non-negative value, $MErrval$ in an error mapper. The prediction errors are

![Diagram](image)
encoded by golomb coder. Golomb codes are designed for one-sided distributions. Since the residual errors have a distribution that is usually approximately two-sided geometric centered at zero, the residuals need to be mapped into positive numbers using error mapper. And Coder has three inputs, MErrval is the data to be encoded, k is a golomb coding variables and limit is a parameter to restrict the maximum length of the output code, which is set to 32 on JPEG-LS encoder.

Context update block is the last step of the encoding of the sample X in the regular mode. After calculating the k and MErrval, context table is updated according to the current prediction error. The variables A and B accumulate prediction error magnitudes and values for context Q, respectively. The variable N accounts for the number of occurrences of the context Q since initialization. The bias variable B allows an update of the prediction correction value C by at most one unit every an iteration. The variables are clamped to limit their range of possible values. The prediction correction value C shall be computed according to the procedure, which also yields an update of B. In hardware implementation, context update logic, DPRAM (Dual-Port Random Access Memory) and multiplexer have used, as shown in Fig. 3 (b). To overcome the case that context is the same previous pixel and current pixel, we use Same Q flag that indicates appropriate update values. In case of same context, context
update logic utilizes the previous pixel’s updated parameter not current pixel’s out-of-date parameter according to the Same Q flag.

Code appending block converts variable length compressed data into 8-bit fixed length code stream, for the convenience of storage and compatibility of general JPEG-LS coding syntax. The code appending block fills up the end of the coded image data with 0-bits to complete final bytes of segment. Also, in case of X’FF’ byte generated by encoding process, 0-bit is stuffed in order to easy detection of marker segments.

4 Simulation and results

The proposed JPEG-LS encoder is designed by using verilog HDL and it is simulated and verified using the ModelSim Simulator. The design was synthesized for the Xilinx’s FPGA devices. Further evaluation of the designs was performed using the prototype board that consists of Xilinx Virtex4 device family. To evaluate the performance of a proposed JPEG-LS encoder, we measure the logic area, operating frequency, and throughput. The maximum clock frequency is up to 120 MHz that indicated in the synthesis report in Xilinx ISE design suite. In Table I, the characteristics of proposed implementation and other implementations are offered for FPGA technologies. Compared with previous design [6], the proposed JPEG-LS encoder reduced the area by 17% and speed up by 20%.

| Table I. Characteristics of proposed implementation and other implementations |
|----------------------------------|--------|-----------------|-----------------|
| Technology                      |       | [6]              | [7]              |
| Technology                      |       | Xilinx Virtex-4  | Xilinx Virtex-4  |
|                                 |       | XC4VLX15         | XC4VLX15         |
| Slices                          | 2,154 (35%) | 2,596 (42%)     | 10,752          |
| Operating Frequency             | 120MHz | 102.6MHz         | 75MHz           |
| Processing Speed (pixels/clock) | 1      | 1                | 1               |

5 Conclusions

In this paper, the hardware architecture of a JPEG-LS encoder for HD images has been presented. The proposed hardware architecture adapts a pipeline architecture to speed up the encoding path. The proposed JPEG-LS encoder is divided into four core parts, a context determination part, a pixel prediction part, a prediction error encoding part, a context update part, and a code appending part. The proposed implementation by using pre-computation and pipeline method performs the lossless image compression 1.2 ~ 1.6 times faster than previous JPEG-LS implementations. It is worth emphasizing again that this implementation performs compression of 1920 × 1080 pixels images with the speed of 24 frames per second. Later this JPEG-LS encoder architecture can be adapted to handle UHD images by using the multiple JPEG-LS encoder processing.

Acknowledgments

This work was supported by the IDEC’s CAD tool assistance.