

Received 3 November 2023, accepted 27 December 2023, date of publication 5 January 2024,
date of current version 12 January 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3350344

RESEARCH ARTICLE

Adaptive Pipeline Hardware Architecture Design and Implementation for Image Lossless Compression/Decompression Based on JPEG-LS

FANGJIA LIU^{1,2}, XIYAO CHEN^{1,3}, ZIJUN LIAO^{1,4}, AND CHONG YANG^{1,5}

¹School of Mechanical Engineering, University of Science and Technology Beijing, Beijing 100083, China

²Shunde Innovation School, University of Science and Technology Beijing, Foshan, Guangdong 528300, China

³Department of Infectious Diseases, Third Affiliated Hospital, Sun Yat-sen University, Guangzhou 510630, China

⁴School of Electronic Information and Communication, Huazhong University of Science and Technology, Wuhan 430074, China

⁵Beijing General Research Institute of Mining and Metallurgy, Beijing 100160, China

Corresponding authors: Xiyao Chen (chenxiyao2020@163.com) and Zijun Liao (zijun@hust.edu.cn)

This work was supported in part by the Characteristic Innovation Project of Colleges and Universities in Guangdong Province under Grant 2022WTSCX315.

ABSTRACT Lossless compression of medical images can reduce data size, save storage and transmission costs, and cope with the challenge of increasing resolution demand for medical images. This paper proposes an adaptive pipeline hardware architecture for lossless compression/decompression of medical images based on joint photographic experts group-lossless (JPEG-LS), which can achieve efficient image processing on field programmable gate array (FPGA). In addition, by adding a pipeline pause mechanism, the problem of coding errors caused by parameter updates is solved. On the one hand, the binary search method is used to optimize the calculation process of the Golomb coding parameter k in the pipeline, which reduces the parameter update delay. On the other hand, the construction process of the context causal template is optimized, which effectively improves the throughput. Based on the parallel block compression architecture, a hardware module for alpha red green blue (ARGB) image block compression/decompression with customizable block size is implemented, which improves the image compression/decompression throughput and reduces the compression and decompression delay. This paper implements the hardware architecture on Zynq7000 FPGA and evaluates its performance. Compared with existing schemes, the proposed architecture uses less resources and achieves higher compression ratio and clock frequency.

INDEX TERMS Lossless image compression/decompression, JPEG-LS, hardware design, FPGA.

I. INTRODUCTION

Medical centers generate massive amounts of digital medical images daily, including computed tomography, magnetic resonance imaging (MRI), ultrasound and capsule endoscopy (CE) images [1]. With the continuous improvement of image sensor resolution, the amount of image data that needs to be processed in real time is also increasing, posing great challenges to the storage capacity and transmission bandwidth of medical devices [2].

It is not cost-effective to solve the above problems by simply increasing the storage capacity and transmission

The associate editor coordinating the review of this manuscript and approving it for publication was Jun Wang¹.

technology, and the data transmission between different storage systems will also consume a lot of time [3]. However, the large amount of redundant information contained in digital images provides theoretical possibilities for image compression. In comparison, image compression is more economical for processing medical images.

Compared with that, image compression is a more cost-effective option for processing medical images. Image compression methods can be generally divided into two types according to whether the compressed image can be restored to the original image: one is based on lossy compression technology; the other is based on lossless compression technology. Medical images need to be effectively reconstructed

without any information loss [4], so they generally need to be losslessly compressed.

JPEG-LS algorithm is based on the low complexity lossless compression for images (LOCO-I) algorithm developed by Hewlett-Packard Laboratories [5], which provides lossless compression and near-lossless compression functions. The algorithm only needs to implement data subtraction, shift and other similar, simple processing processes [6]. Compared with joint photographic experts group (JPEG), JPEG2000 and other algorithms, JPEG-LS algorithm has the characteristics of low complexity, easy hardware implementation, etc. [7], [8]. In addition, these standards are considered in digital imaging and communication in medicine (DICOM), which is an industrial international standard for medical image representation and coding [9].

At present, JPEG-LS lossless compression is widely used in medical imaging, biometrics, satellite remote sensing and professional photography fields. The research on the algorithm mainly focuses on hardware implementation, and also makes many improvements on the algorithm itself for specific application scenarios. In 2001, Klimesh et al. [10] modified the run coding mode of LOCO-I algorithm, which was a significant advancement in hardware for space-efficient image lossless compression. In 2009, Merlino et al. [11] used pipeline structure to fully exploit the sequentiality of the algorithm. However, the image processing speed of the whole compression system was not fast. The maximum clock frequency of the encoder circuit was about 21MHz, and that of the decoder circuit was only 16MHz. In 2013, Mert [12] proposed an efficient full pipeline scheme to design JPEG-LS encoder. The processing speed of the encoder reached 120Mpixels/second. In 2016, Chen et al. [13] proposed a novel near-lossless color filter array image compression algorithm based on JPEG-LS using very large scale integration (VLSI), which had a gate count of only 10.9 K and an area of 30625 μm^2 using 90 nm complementary metal oxide semiconductor (CMOS) process synthesis, suitable for wireless video capsule endoscopy system. In 2018, Haddad et al. [14] proposed the first joint watermarking encryption JPEG-LS compression scheme for protecting medical images. In 2021, Wang et al. [15] proposed a hardware/software co-design method for pixel-level parallelized streaming image compressor based on JPEG-LS, which proposed pixel grouping scheduling scheme and pseudo-lossless (Pseudo-LS) method to exploit parallelism, and gave a design space exploration method to limit resource usage brought by parallelization. Cao et al. [16] proposed a convolutional neural network model with wavelet-based residual learning mechanism for JPEG-LS near-lossless compressed remote sensing image decompression recovery. In 2022, Liu et al. [17] used context-aware residual learning mechanism to realize JPEG-LS compressed remote sensing image recovery. In 2023, Sun et al. [18] proposed a lossless image compression and encryption algorithm combining JPEG-LS, neural network and hyperchaotic mapping to protect the privacy of digital images and reduce data storage space.

Although the software and hardware research on JPEG-LS algorithm has improved its compression performance to varying degrees, the popularity of high-speed interface devices and the increasing demand for higher compression performance of JPEG-LS algorithm in medical practice make improving the compression speed of JPEG-LS still one of the focuses in research. Although there are many very mature solutions for software-based compression at present, they generally have problems such as occupying too much central processor resources, high power consumption and processing speed that can no longer match the interface speed. In an environment where memory access speed and network bandwidth are gradually increasing, software-based image lossless compression gradually shows a performance bottleneck state [19]. Therefore, it is meaningful to study JPEG-LS algorithm with higher compression performance and its hardware implementation. At the same time, because FPGA has the advantages of powerful logic design function, short development cycle, low investment cost and fast processing speed, FPGA-based JPEG-LS encoder has become the focus of research.

This paper proposes an adaptive pipeline hardware architecture for medical image lossless compression based on JPEG-LS, which avoids pipeline idle waiting caused by data dependency in most cases. In order to improve the clock frequency and throughput of the designed JPEG-LS encoder, various methods are used to optimize it in the hardware design and implementation process, such as optimizing the construction process of context causal template and using binary search method to optimize the calculation process of Golomb coding parameter k . From the perspective of data compression, the bandwidth and access efficiency of memory are improved by reducing the amount of data accessing the color buffer; on this basis, by calculating the offset value of each two-dimensional block after compression, random access of ARGB image two-dimensional blocks can be realized, which can adapt to the needs of complex application environment.

The rest of the paper is organized as follows: Section II presents the analysis and design of the JPEG-LS compression/decompression algorithm; Section III describes the FPGA implementation of the JPEG-LS algorithm; Section IV conducts the FPGA board-level verification and analysis of the proposed encoder; Section V concludes and discusses future work.

II. JPEG-LS ALGORITHM

A. JPEG-LS COMPRESSION ALGORITHM

The core of JPEG-LS algorithm is LOCO-I algorithm, which includes two modes: normal coding mode and run coding mode. It mainly consists of context modeling, normal coding, run coding, interrupt coding and other parts, providing lossless compression and near-lossless compression functions. When the threshold of lossless compression and near-lossless compression is equal to 0, JPEG-LS performs lossless compression, otherwise it performs near-lossless compression. This paper only studies lossless compression, so the value is

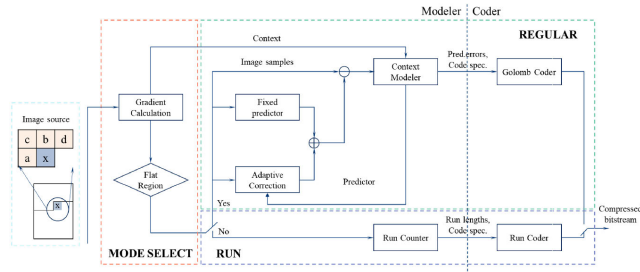


FIGURE 1. The operation structure of JPEG-LS algorithm.

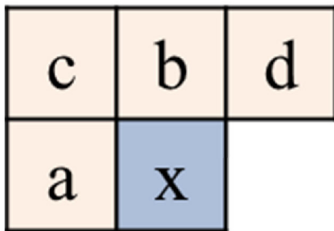


FIGURE 2. Causal template used for context modelling and prediction.

0. As shown in Figure 1, it is the overall framework of JPEG-LS Algorithm.

1) CONTEXT MODELING

Context modeling is to use the correlation between the current pixel to be encoded and its adjacent pixels to model it. There is correlation between image pixels, and the causal template of the current pixel to be encoded can be completed by using the four adjacent pixels of the current pixel to be encoded. As shown in Figure 2, *x* is the position of the current pixel to be encoded, corresponding to the pixel value *I_x*; *a*, *b*, *c*, *d* are the four adjacent positions of the current pixel to be encoded, corresponding to four reconstructed values *R_a*, *R_b*, *R_c*, *R_c*.

Context modeling first estimates the local gradients based on the values of adjacent pixels, as shown in equation (1).

$$\begin{aligned}
 D1 &= R_d - R_b \\
 D2 &= R_b - R_c \\
 D3 &= R_c - R_a
 \end{aligned}
 \tag{1}$$

After the local gradient values *D1*, *D2*, *D3* are calculated, the coding mode is selected. If the absolute values of the local gradient values are all less than or equal to *N*, then the current pixel to be encoded enters the run coding mode, otherwise the current pixel to be encoded enters the normal coding mode.

2) NORMAL CODING MODE

The normal mode consists of three parts: prediction of the reconstructed value *P_x*, calculation of the prediction error *Errval*, and Golomb coding of the prediction error *Errval*. As shown in equation (2), the predicted value *P_x* of the current pixel value *x* is formed by integrating the reconstructed values

R_a, *R_b*, and *R_c* of the three adjacent pixels *a*, *b*, and *c*.

$$P_x = \begin{cases} \min(R_a, R_b), & R_c \geq \max(R_a, R_b) \\ \max(R_a, R_b), & R_c \leq \min(R_a, R_b) \\ R_a + R_b - R_c, & \text{otherwise} \end{cases}
 \tag{2}$$

After edge detection is completed, the predicted values are corrected according to code segment 1. The predicted values are corrected by *C[Q]* and *SIGN*, which are the correction value of prediction error for context parameter *Q* and the sign of context, respectively. Where *C[Q]* is the correction value of the prediction error corresponding to the context parameter *Q*, *MAXVAL* is the maximum pixel value of the scanned source image, which is calculated by equation (3), and *p* is the bit depth of the source image.

$$MAXVAL = 2^p - 1
 \tag{3}$$

Code segment 1	Prediction correction from the bias
Input: <i>P_x</i> , <i>C[Q]</i> , <i>MAXVAL</i>	
Output: <i>P_x</i>	
1	if (<i>SIGN</i> == +1)
2	<i>P_x</i> = <i>P_x</i> + <i>C[Q]</i> ;
3	else
4	<i>P_x</i> = <i>P_x</i> - <i>C[Q]</i> ;
5	if (<i>P_x</i> > <i>MAXVAL</i>)
6	<i>P_x</i> = <i>MAXVAL</i> ;
7	else if (<i>P_x</i> < 1)
8	<i>P_x</i> = 0;

Using the corrected *P_x* value from the above procedure, the prediction error *Errval* is calculated according to code segment 2.

Code segment 2	Computation of prediction error
Input: <i>I_x</i> , <i>P_x</i> , <i>SIGN</i>	
Output: <i>Errval</i>	
1	<i>Errval</i> = <i>I_x</i> - <i>P_x</i> ;
2	if (<i>SIGN</i> == -1)
3	<i>Errval</i> = - <i>Errval</i> ;

The prediction error coding module mainly performs the calculation of the Golomb coding parameter *k*, the mapping of the prediction error *Errval*, and the Golomb coding of the mapped prediction error *MErrval*.

The function of the Golomb coding parameter *k* calculation module is to calculate the coding parameter *k* through the variables *A[Q]* and *N[Q]*, so that the Golomb coding can use the shortest codeword to represent the mapped prediction error. The function of the prediction error mapping module is to map the prediction error *Errval* that satisfies the bilateral geometric distribution to the prediction error *Errval* that satisfies the unilateral geometric distribution, and further improve the efficiency of Golomb coding. The function of the Golomb coding module is to encode the mapped prediction error *MErrval*.

The commonly used method to calculate the value of Golomb coding parameter *k* is to use sequential comparison, which requires at most 7 comparisons. Therefore, when implementing in hardware, there are at most 7 levels of comparator delay and 7 levels of shift register delay. This

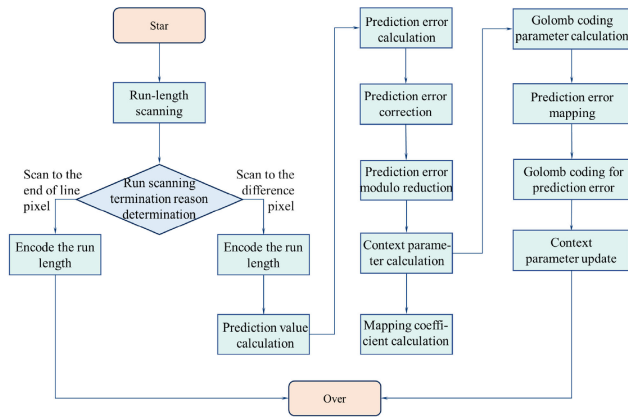


FIGURE 3. Flowchart of run and interruption coding.

paper proposes to use binary search to calculate the value of k , which requires at most 3 comparisons. Therefore, when implementing in hardware, there are at most 3 levels of comparator delay and 3 levels of shift register delay, which can effectively improve the clock frequency of the designed encoder. The specific implementation of using binary search to complete the calculation of Golomb coding parameter will be given in Chapter 3.

3) RUN AND INTERRUPT CODING MODE

Figure 3 shows the flowchart of the run-length coding mode in the JPEG-LS algorithm. After the current pixel to be coded enters the run-length coding mode, the pixel is first scanned for the run-length. The termination of the run-length scan is determined based on the reason for scanning. If the scan reaches the end-of-line pixel, the run-length is coded and the run-length coding mode is exited. However, if the scan is terminated due to a run interruption, the run-length is coded and the difference pixel is also coded. Subsequently, the predicted value of the difference pixel is calculated, followed by the calculation of the prediction error. This error is then corrected and reduced modulo. Additionally, the context parameters, mapping coefficients, and Golomb coding parameters are calculated. The prediction error is then mapped and subsequently Golomb coded. Finally, the context parameters are updated. Once all the variables are updated, the run-length coding mode is exited.

4) BITSTREAM SPLICING

The function of the bitstream splicing module is to splice the remaining bitstream from the previous splicing and the encoded bitstream of the current frame, and to output the spliced bitstream in units of 32 bits. Figure 4 shows the block diagram of the bitstream splicing module.

B. JPEG-LS DECOMPRESSION ALGORITHM

The decoding process is the inverse of the encoding process, that is, to obtain the value of x given the values of the context R_a , R_b , R_c , and R_d . Before decoding, the compressed image data must be preprocessed, that is, to perform corresponding

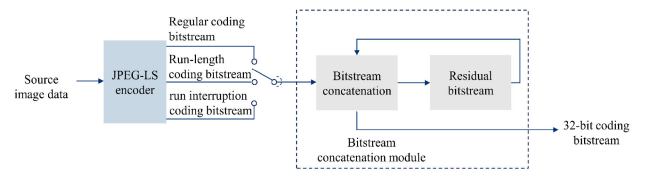


FIGURE 4. Bitstream splicing block diagram.

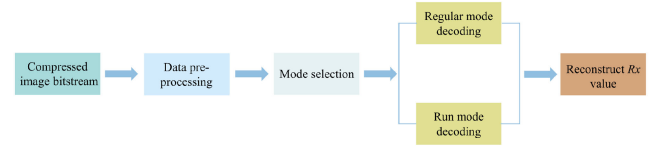


FIGURE 5. Image decompression flowchart.

operations on the header file of the image, remove some accessory information of the image, such as the height, width, and precision of the image, etc., and only then can the actual image bitstream participate in decoding. The decoding also starts with mode selection, which is the same as encoding mode selection and will not be repeated here. Figure 5 shows the structure diagram of the decoding module.

1) NORMAL DECODING MODE

The normal decoding mode obtains the values of Q_i , SIGN, P_x , and k by following the steps of gradient quantization, gradient merging, prediction, and prediction error coding in the encoding process. Then, the value of $MERRval$ is decoded from the value of k and the read-in bitstream.

2) RUN AND INTERRUPT DECODING MODE

First, read 1 bit R from the bitstream. If $R = 1$, fill the image with $2^{J[RUNindex]}$ samples of R_a (J is the index value of the $RUNindex$ entry vector table) until the end of the line. If $RUNindex < 31$, then decrement $RUNindex$ by 1 and continue filling. If the line is not finished, then continue reading the bitstream until the end of the line.

If $R = 0$, then read J bits from the bitstream and form a number x from these binary bits. Fill the image with x samples of R_a . If $RUNindex < 31$, then decrement $RUNindex$ by 1 and continue filling. When the run is interrupted, perform run interruption decoding, which is the opposite of run interruption encoding.

III. JPEG-LS HARDWARE DESIGN

This paper designs an ARGB image block compression/decompression hardware module, which uses pipeline technology, based on parallel block architecture, realizes random access of image two-dimensional blocks, and can also realize custom block size.

A. JPEG-LS COMPRESSION HARDWARE MODULE

The JPEG-LS compression module designed in this paper adopts pipeline architecture, and the hardware architecture is shown in Figure 6. The whole JPEG-LS encoder mainly includes context modeling module, normal coding module,

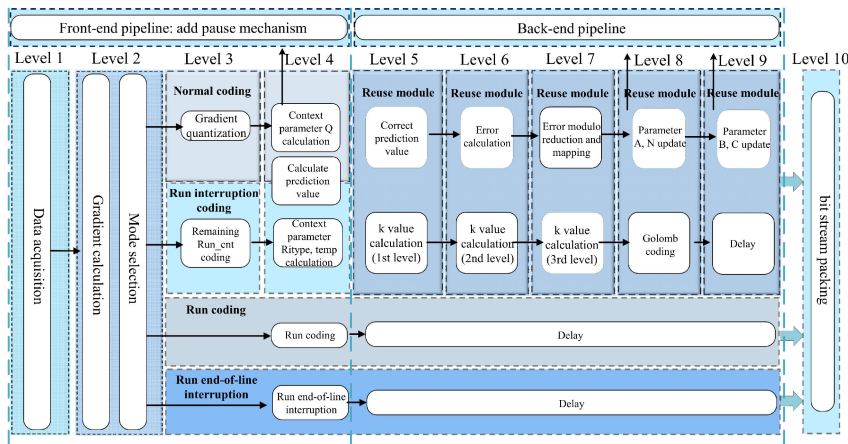


FIGURE 6. Compression hardware module.

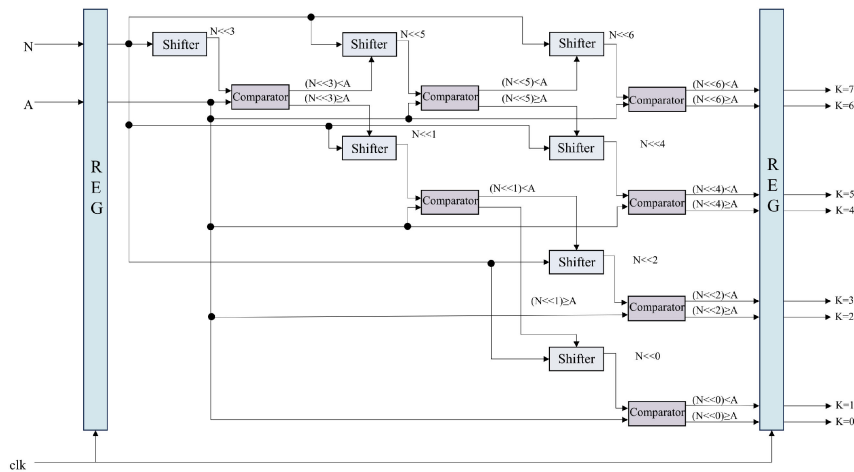


FIGURE 7. Structural block diagram of Golomb coding parameter k value calculation.

run-length coding module and output module. The whole encoder is divided into ten stages of pipeline, each column represents one stage of pipeline, and the parallel blocks in the pipeline represent the parallel execution part. After the second stage of pipeline, one of the four modes will be entered (each row represents one mode). The first to fourth stages of pipeline are added with pause mechanism. Whenever the Q value calculated by the fourth stage is equal to any Q value in the fifth to ninth stages, the first to fourth stages of pipeline need to be paused, which solves the delay problem caused by parameter update. The calculation of Golomb coding parameter k requires eight times of loop search. Here, using binary method, it can be realized by placing it in three stages of pipeline, which greatly reduces the delay. The main functions of each stage of pipeline are introduced below.

Stage 1: Data acquisition. According to the input pixel value x and index, output the pixel value x and the values of the adjacent pixels a, b, c, d , as well as the end-of-line and end-of-block flag signals. This is achieved by caching all the pixels of the previous line and the previous pixel.

Stage 2: Gradient calculation and mode selection. Calculate the current gradient value based on the adjacent pixels, and combine with the end-of-line flag signal to select the mode, which can be normal coding, run interruption coding, run-length coding or run end-of-line interruption mode.

Stage 3: According to the mode selection, perform gradient quantization, remaining Run_Cnt coding and run-length coding respectively.

Stage 4: If normal or run interruption coding mode is entered, calculate the context parameter Q, predicted value and Ritype, SIGN and other parameters simultaneously. If run-length coding or run end-of-line interruption mode is entered, only calculate the run-length coding. The subsequent run-length coding or run end-of-line interruption mode only need delay without any operation. The subsequent normal or run interruption coding mode reuse the same module.

Stage 5: Predicted value correction. Binary search for Golomb parameter k at the first level.

Stage 6: Error value calculation. Binary search for Golomb parameter k at the second level.

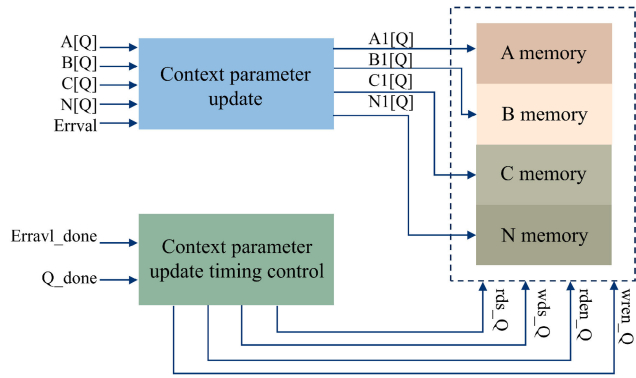


FIGURE 8. Structural block diagram of context parameter update.

Stage 7: Error modulo reduction and mapping. Binary search for Golomb parameter k at the third level.

Stage 8: Parameter update A , N and half of B update process. Golomb coding.

Stage 9: Parameter update for the remaining B and C .

Stage 10: Output according to one of the four modes selected by the mode selection.

1) BINARY SEARCH

In stages 5 and 6, this paper optimizes the calculation path of Golomb coding parameter k , and uses binary search to compare the sizes of $N[Q]$ and $A[Q]$ to determine the value of k . Figure 7 shows the block diagram of the optimized Golomb coding parameter k calculation, where A and N are $A[Q]$ and $N[Q]$ respectively.

2) CAUSAL TEMPLATE OPTIMIZATION

In this paper, the context update module completes the read, update and write of variables $A[Q]$, $B[Q]$, $C[Q]$, $N[Q]$. In this paper, dual-port RAM (Random Access Memory) IP core is used to store $A[Q]$, $B[Q]$, $C[Q]$, $N[Q]$. When the context update module detects that Q_done is high, it starts to read variables $A[Q]$, $B[Q]$, $C[Q]$, $N[Q]$. When the context update module detects that the completion flag $Errval_done$ of the prediction error calculation module is high, it updates variables $A[Q]$, $B[Q]$, $C[Q]$, $N[Q]$ and writes variables $A[Q]$, $B[Q]$, $C[Q]$, $N[Q]$ into corresponding dual-port RAM. Figure 8 shows the block diagram of context parameter update, where $A[Q]$, $B[Q]$, $C[Q]$, $N[Q]$ are variables before update, and $A1[Q]$, $B1[Q]$, $C1[Q]$, $N1[Q]$ are variables after update.

In JPEG-LS software compression process, variables $A[Q]$, $B[Q]$, $C[Q]$, $N[Q]$ are updated after each encoding is completed; in common hardware design process, the construction of causal template of next pixel to be coded is also performed after current pixel to be coded is encoded. However, due to FPGA hardware design parallelism, this paper optimizes context parameter update module. When context parameter update module detects that completion flag $Errval_done$ of prediction error calculation module is high, it starts to update variables $A[Q]$, $B[Q]$, $C[Q]$, $N[Q]$ without

TABLE 1. Basic signal list of normal coding module.

Signal name	Bit width	Interface status	Comment
SIGN	1	Input	correction symbol
Px	8	intermediate variable	predicted value
Errval	9	intermediate variable	prediction error
Errval_done	1	intermediate variable	modulo reduction done flag
rds_Q	9	intermediate variable	A, B, C, N memory read address signal
wds_Q	9	intermediate variable	A, B, C, N memory write address signal
rden_Q	1	intermediate variable	A, B, C, N memory read enable signal
wren_Q	1	intermediate variable	A, B, C, N memory write enable signal
A[Q]	14	intermediate variable	corresponding to the sum of absolute values of prediction errors
B[Q]	9	intermediate variable	Q corresponding to the sum of prediction errors
C[Q]	9	intermediate variable	corresponding to the correction value of prediction error
N[Q]	9	intermediate variable	Q occurrence count

waiting for encoding to be completed before updating $A[Q]$, $B[Q]$, $C[Q]$, $N[Q]$. This saves 5 to 8 clk clock cycles required by context parameter update module; meanwhile, since clock cycles required by context parameter update module are enough to perform two read operations on source image storage device, therefore when context parameter update module detects that completion flag Q_done of context modeling is high, it starts to read source image storage device and completes reading of next pixel to be coded I_x and pixel R_d at position d in causal template without waiting for current pixel to be coded encoding to be completed before constructing causal template of next pixel to be coded. This saves time of reading source image storage device twice. By optimizing encoder structure, including optimization of start signal of context parameter update module and optimization of causal template construction process of next pixel to be coded, 11 to 14 clk clock cycles are saved, which effectively improves encoder throughput.

B. JPEG-LS DECOMPRESSION HARDWARE MODULE

The top level of the decompression module is the run-length decoding finite-state machine (FSM), which is the upper part of the Figure 9, which continuously reads in the bit-stream and determines the next state according to the read bit. If normal decoding or run interruption decoding mode is entered, it enters a part similar to the encoding mode, which is the lower part of the Figure 9. This part is very similar to the encoding mode process, only a few parts need to be reversed and Golomb coding needs to be replaced with decoding mode. This module is also implemented by FSM. Note that, since the value of the current pixel is unknown

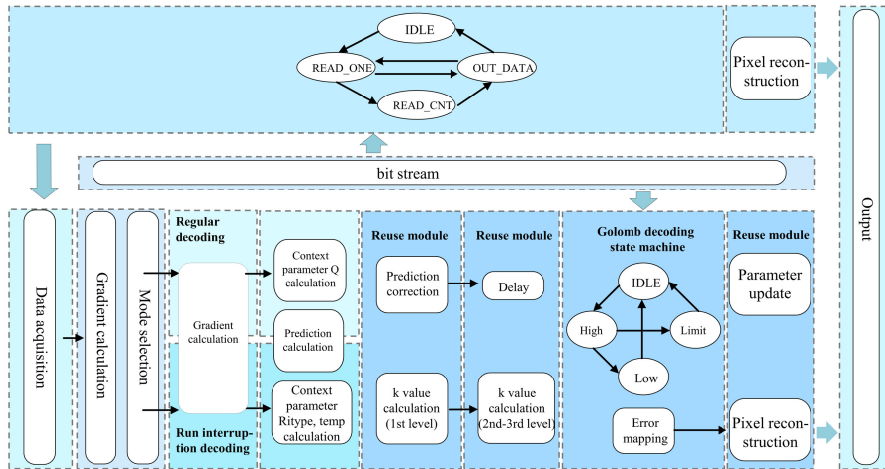


FIGURE 9. Decompression hardware module.

during decompression, pipeline architecture cannot be used, and pixels need to be decompressed one by one. The following introduces two FSM, and the rest of the parts are similar to encoding and will not be repeated.

- Run-length decoding FSM, This FSM has four states:

IDLE: Idle state. After reading the start signal, it enters the READ_ONE state.

READ_ONE: Read 1 state. Continuously read the bitstream. Whenever a 1 is read, it enters the OUT_DATA state and outputs the corresponding number of pixels. When a 0 is read, it enters the READ_CNT state and reads the remaining part of the run.

READ_CNT: Read the remaining length of the run. This value is the part that does not meet the cursor length when run-length coding. Continuously read the bitstream. After reading, it enters the OUT_DATA state and outputs the corresponding number of pixels.

OUT_DATA: Output data state. Output the pixel values according to the input value, and update J, RUN_index and other parameters. If the run is interrupted, it enters the IDLE mode and gives a signal to start the interruption decoding. If not interrupted, it continues to enter the READ_ONE state.

- Golomb decoding FSM, This FSM has four states:

IDLE: Idle state. After reading the start signal, it enters the High state.

High: High bit decoding state. Continuously read the bitstream. Whenever a 0 is read, the counter value increases by 1. When a 1 is read, stop counting. If the counter value is greater than Limit, then enter the Limit mode. Otherwise, assign the high bit decoding value to the counter value and enter the Low state.

Low: Low bit decoding state. Read k bits of bitstream and assign the low bit decoding value to these k bits and enter the IDLE state.

Limit: Limit mode, where the decoding value is specially processed according to the algorithm. See algorithm introduction for details.

C. COMPRESSION/DECOMPRESSION SYSTEM DESIGN

The compression/decompression system is designed on the zynq7020 platform. The compression/decompression modules are separately packaged as custom AXI IP cores. The IP core top level instantiates the compression/decompression module and two BRAMs to store the original/compressed and compressed/decompressed data. If the block size after compression is larger than the original block size, the original block is directly used as the compressed data, and the block size is written into the block header information. When decompressing a block, if the block size is larger than the original block size, it is not decompressed. The IP core also needs to complete the logic of AXI reading and writing DDR, with burst_length set to 16 and size set to 4, and adjust the number of burst transfers according to the block size. The block size can support 4, 8, 16, which can be adjusted in the compression/decompression module header file. The CPU is used to read the original image data from the SD card and complete the block function, and write it into DDR. The CPU also needs to read and configure registers to control the compression/decompression module. Figure 10 shows the system block diagram.

IV. EXPERIMENT PLATFORM AND RESULT

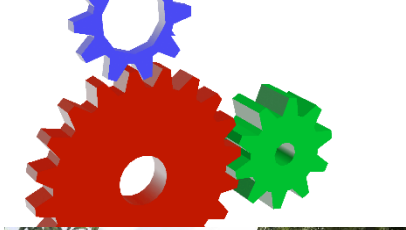
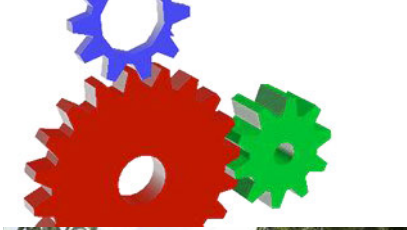


The following criteria are commonly used to evaluate the hardware compression and decompression performance:

Compression ratio: The compression ratio is the ratio of the information amount of the source image to that of the compressed image. A larger compression ratio indicates a better compression effect without reducing the image information entropy. The compression ratio of the image is shown in equation (4).

$$c = \frac{n_1}{n_2} \tag{4}$$

Here, c is the compression ratio, n_1 is the information amount of the source image, and n_2 is the information amount of the compressed image.

TABLE 2. Examples of the custom test image set.

Test image collection	original image	Decompressed image	size	No.
custom			256*256	1
custom			256*256	33

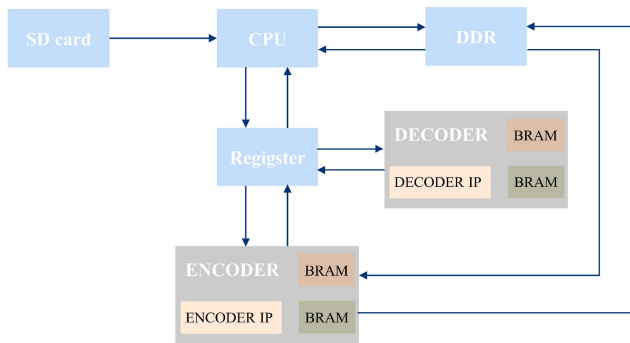


FIGURE 10. Compression/decompression system architecture diagram.

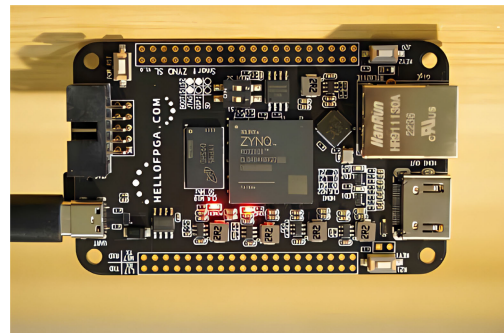


FIGURE 11. Hardware physical diagram of the experiment based on XC7Z020 chip.

Clock frequency: The clock frequency is the frequency of the digital clock signal inside the FPGA, which has a certain relationship with the FPGA’s computing capability. A higher clock frequency means that the CPU can execute more instructions per second. A higher clock frequency also means that the designed hardware architecture works faster.

Throughput: The throughput in compression is the amount of data compressed or decompressed per unit time, usually in bytes/second or bits/second. The throughput can be used to measure the performance and efficiency of compression software or hardware. Generally speaking, a higher throughput means a faster compression or decompression speed, but it may also mean a lower compression ratio.

Resource utilization: look up table (LUT) resources and slice resources are two types of logic resources in FPGA, which are used to implement different functions. LUT can store truth tables, implement combinational logic or distributed memory. Slice is a logic slice composed of multiple LUTs and flip-flops (FF), which can implement sequential logic, arithmetic operations, data selectors, etc. Generally speaking, a lower resource utilization means that the system

has more idle resources, which can handle more requests or loads.

The development board used in this paper is the Zynq7000 series chip from Xilinx, with the model number XC7Z020-CLG484-1, 53200 LUTs, 106400 FFs, 4.9Mb BRAM, 512MB DDR on board, and SD card slot on board, as shown in Figure 11. This scheme supports different compression block sizes.

To verify the robustness of the scheme for practical applications, and to facilitate comparison with other schemes, this paper selected a custom test image set that conforms to the JPEG-LS standard specification for testing.

A custom test image set consisting of 33 images of different types with 8-bit depth and 256*256 size was used for compression and decompression. The block size of the test image set was 8*8, and the clock frequency was 100MHz. In the compression test, LUT and FF occupied 1316 and 1110 respectively. The timing report generated under the 50MHz clock constraint showed that the maximum frequency could reach 108.6MHz after removing the 10.793ns slack. In the decompression test, LUT and FF occupied 789 and 803 respectively. The timing report generated under the

TABLE 3. Compression and decompression test results of 33 images.

sample	Total number of cycles for compression	Total number of cycles for decompression	Compression ratio	Width	Height	Total time for compression/ μ s	Total time for decompression/ μ s	Average compression delay per block/ μ s	Average decompression delay per block/ μ s
1	66541478	39142486	17.95	1920	1056	0.6650	0.3910	20.9912	12.3422
2	35779745	39345835	7.07	1240	960	0.3580	0.3930	19.2473	21.1290
3	41951603	135407463	3.20	1920	1080	0.4200	1.3540	12.9630	41.7901
4	68004241	306823787	2.53	2496	1600	0.6800	3.0680	10.8974	49.1667
5	153887419	590418154	2.88	3840	2160	1.5380	5.9050	11.8673	45.5633
6	113832341	707384025	2.12	3840	2120	1.1380	7.0740	8.9465	55.6132
7	17768520	94435881	2.26	1416	800	0.1780	0.9440	10.0565	53.3333
8	144989635	656355109	2.03	3840	2160	1.4500	6.5640	11.1883	50.6481
9	124948952	710330118	2.07	3840	2160	1.2490	7.1030	9.6373	54.8071
10	163918118	492954237	3.72	4000	2000	1.6390	4.9300	13.1120	39.4400
11	199306642	437322926	4.29	3840	2160	1.9930	4.3730	15.3781	33.7423
12	137776449	643752915	2.74	3840	2160	1.3780	6.4380	10.6327	49.6759
13	129104453	696193156	2.10	3840	2160	1.2910	6.9620	9.9614	53.7191
14	177096905	511681197	3.79	3840	2160	1.7700	5.1170	13.6574	39.4830
15	131443579	644803870	2.24	3840	2096	1.3140	6.4480	10.4485	51.2723
16	186695223	502349785	3.71	3840	2160	1.8670	5.0230	14.4059	38.7577
17	98325151	290033770	2.99	3168	1440	0.9830	2.9000	13.7907	40.6846
18	180573914	401948967	4.72	3456	2160	1.8060	4.0190	15.4835	34.4564
19	69280167	262314764	2.58	2560	1440	0.6930	2.6230	12.0313	45.5382
20	60671715	243102892	2.96	2304	1440	0.6070	2.4310	11.7091	46.8943
21	101646805	147590673	5.38	2552	1432	1.0160	1.4760	17.7930	25.8489
22	230276562	241287584	7.93	2880	2632	2.3030	2.4130	19.4444	20.3732
23	165033927	246591373	5.42	3000	2000	1.6500	2.4660	17.6000	26.3040
24	128907754	158508166	6.22	2744	1616	1.2890	1.5850	18.6040	22.8762
25	57011683	299238979	2.45	2560	1440	0.5700	2.9920	9.8958	51.9444
26	55289688	342917048	1.77	2520	1568	0.5530	3.4290	8.9569	55.5394
27	20125669	86774028	2.29	1416	800	0.2010	0.8680	11.3559	49.0395
28	38058058	148812584	2.58	1920	1080	0.3810	1.4880	11.7593	45.9259
29	31836750	174706030	1.98	1920	1080	0.3180	1.7470	9.8148	53.9198
30	29606519	180992551	2.01	1920	1080	0.2960	1.8100	9.1358	55.8642
31	31376562	175959318	1.89	1920	1080	0.3140	1.7600	9.6914	54.3210
32	53501360	93653993	4.23	1920	1080	0.5350	0.9370	16.5123	28.9198
33	30925248	177591819	1.63	1920	1080	0.3090	1.7760	9.5370	54.8148
Mean value	99257359	329718954	2.85	2790	1619	0.9925	3.2972	12.9244	42.5378

50MHz clock constraint showed that the maximum frequency could reach 128.5MHz after removing the 12.220ns slack.

The smoothness of the image greatly affects the compression performance of the JPEG-LS algorithm. Images with high smoothness have small variations in the gray values of adjacent pixels, and most of the prediction errors are small or zero, which can be represented by fewer bits, thus enhancing the compression ratio and speed. Images with low smoothness have large variations in the gray values of adjacent pixels, and the prediction errors are also large, seldom zero or near zero. Such images require more bits to represent the prediction errors, thus lowering the compression ratio and speed.

Due to the high smoothness of image 1, more pixels fall into the flat region, so the run-length coding mode can be

applied to most of the pixels, thus achieving efficient compression; in contrast, image 33 has low smoothness, fewer pixels fall into the flat region, that is, fewer pixels enter the run-length coding mode, so the compression ratio of the image is low. In fact, for medical images such as Computed Tomography (CT), MRI, etc., they are usually smooth, and it can be expected that this scheme can achieve good compression results.

Since there are few identical chip configurations in previous researches, this paper tries to compare the encoder designed in this paper with the original one under the same chip model. In order to compare with previous research results, the JPEG-LS encoder designed in this paper is based on the Zynq7000 series XC7Z020 chip and Virtex series XCV300 chip for logic synthesis and layout routing. Table 2

TABLE 4. Comparison to existing lossless compression works.

Work	Technology	Resource	CR	Clock Freq. (MHz)	Throughput (Mpixel/s)
Ours	Zynq-7XC7Z020	1316LUT	2.85	108.60	43.03
	VirtexXCV300	3161slice	2.85	63.60	28.7
Ref.[10]	VirtexXCV300	--	--	12	21.28
Ref.[20]	VirtexXCV300	4938slice	--	61.1	--
Ref.[21]	Virtex-6 XC6VCX75	8.3k LUT	2.14	--	52
Ref.[15]	Zynq-7XC7Z020	19.5k LUT	2.14	185	826

shows the performance comparison table of the encoder designed in this paper and previous research results.

The work in [14] combines the lossless compression standard JPEG-LS with bit substitution watermark modulation, which is simple but has low performance. The work in [10] modifies the run coding mode of the LOCO-I algorithm, which greatly reduces the complexity of the algorithm, but the compression performance is not high enough. The work in [15] proposes a hardware/software co-design method for a pixel-level parallelized streaming image compressor based on JPEG-LS, and proposes a pixel grouping scheduling scheme and a Pseudo-LS method to exploit parallelism, achieving high performance. This method is not only overly complex, but also the results in the paper show that the performance improvement is at the expense of high cache space and sacrificing some compression ratio, especially the Pseudo-LS method does not have significant improvement after being compatible with JPEG-LS. The work in [20] proposes a high-performance implementation, which solves the data dependency problem by pre-predicting the variable $C[Q]$, avoiding pipeline waiting. This implementation ensures that the worst-case performance of the algorithm is suitable for domains with high real-time requirements. The limitation of this method is that the logic of pre-prediction and comparison selection limits the number of pipeline stages, and it is difficult to further increase the clock frequency on this structure, and the performance of the decompression process is not considered. The hardware architecture designed in this paper achieves a good balance between performance and complexity, and realizes performance improvement of both compression and decompression processes, and has some advantages in compression ratio, which is very suitable for practical application in medical image lossless compression/decompression.

V. CONCLUSION

This paper proposes an adaptive pipeline hardware architecture design and FPGA implementation for medical image lossless compression/decompression based on JPEG-LS. The main contributions are as follows:

- We study and optimize the JPEG-LS lossless compression/decompression algorithm, and use Verilog hardware description language (HDL) to design the register transfer level (RTL) of JPEG-LS context modeling module, normal coding/decoding module, run

coding/ decoding module, and output module. We also implement a hardware module for block compression/decompression of ARGB images with customizable block size;

- In the ARGB image block compression/decompression module implemented in (1), we adopt an adaptive pipeline design, which can improve the throughput and clock frequency of the compression module, and solve the delay problem caused by parameter updates by adding a pause mechanism in the pipeline;
- Based on the ARGB image block compression/decompression module implemented in (1), we optimize the construction process of the context causal template. On the one hand, we use the causal template of the previous encoded pixel to optimize the construction process of the causal template of the current pixel to be encoded. On the other hand, we complete the construction of the causal template of the next pixel to be encoded in the context parameter update process. We also use a binary search method to optimize the calculation process of the Golomb coding parameter k ;
- Based on the ARGB image block compression/decompression module implemented in (1), we realize parallel compression/decompression of ARGB images by instantiating multiple ARGB image block compression/decompression modules in parallel, which improves the bandwidth and access efficiency of memory. Based on parallel block compression architecture, we calculate the offset value of each two-dimensional block after compression, and write the offset value to the header file, which realizes random access of ARGB image two-dimensional blocks;
- We test our design on FPGA development board, and compare it with previous research results. Our design achieves a good balance between performance and complexity, and realizes performance improvement of both compression and decompression processes. The test results show that our design can achieve lossless compression and decompression of images, which has some significance for the research of lossless image processing in medical practice.

The hardware design of the JPEG-LS lossless compression algorithm based on FPGA, which is completed in this paper, has certain application value, but there are still many

aspects that can be improved. At present, our work is only implemented on FPGA, and the frequency can only reach the level of 100 MHz. For the self-built image test set of 33 images with 8*8 blocks, the average compression time is 12.9 μ s. In the future, we can extend the compression and decompression module to the chip-level platform, which can reach the GHz level, and the compression time can be further reduced to milliseconds or less, achieving higher real-time performance of medical image compression.

REFERENCES

- [1] S. M. Anwar, M. Majid, A. Qayyum, M. Awais, M. Alnowami, and M. K. Khan, "Medical image analysis using convolutional neural networks: A review," *J. Med. Syst.*, vol. 42, no. 11, pp. 1–13, Oct. 2018.
- [2] L. Deng and Z. Huang, "The FPGA design of JPEG-LS image lossless decompression IP core," in *Proc. Chin. Autom. Congr. (CAC)*, Nov. 2015, pp. 2199–2203.
- [3] Md. A. Kabir and M. R. H. Mondal, "Edge-based transformation and entropy coding for lossless image compression," in *Proc. Int. Conf. Electr., Comput. Commun. Eng. (ECCE)*, Feb. 2017, pp. 717–722.
- [4] G. Xin and P. Fan, "A lossless compression method for multi-component medical images based on big data mining," *Sci. Rep.*, vol. 11, no. 1, p. 12372, Jun. 2021.
- [5] P. Ryan and J. Connell, "Real-time lossless image compression in a hardware environment," in *Proc. IEE Irish Signals Syst. Conf.*, Sep. 2005, pp. 68–73.
- [6] H. Niu, Y. Shang, X. Yang, D. Xu, B. Han, and C. Chen, "Design and research on the JPEG-LS image compression algorithm," in *Proc. 2nd Int. Conf. Commun. Syst., Netw. Appl.*, vol. 1, Jun. 2010, pp. 232–234.
- [7] G. Pavlidis, A. Tsompanopoulos, N. Papamarkos, and C. Chamzas, "A multi-segment image coding and transmission scheme," *Signal Process.*, vol. 85, no. 9, pp. 1827–1844, Sep. 2005.
- [8] S. D. Rane and G. Sapiro, "Evaluation of JPEG-LS, the new lossless and controlled-lossy still image compression standard, for compression of high-resolution elevation data," *IEEE Trans. Geosci. Remote Sens.*, vol. 39, no. 10, pp. 2298–2306, Oct. 2001.
- [9] S.-G. Miaou, F.-S. Ke, and S.-C. Chen, "A lossless compression method for medical image sequences using JPEG-LS and interframe coding," *IEEE Trans. Inf. Technol. Biomed.*, vol. 13, no. 5, pp. 818–821, Sep. 2009.
- [10] M. Klimesh, V. Stanton, and D. Watola, "Hardware implementation of a lossless image compression algorithm using a field programmable gate array," *Mars*, vol. 4, no. 4, pp. 5–72, 2001.
- [11] P. Merlino and A. Abramo, "A fully pipelined architecture for the LOCO-I compression algorithm," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 7, pp. 967–971, Jul. 2009.
- [12] Y. M. Mert, "FPGA-based JPEG-LS encoder for onboard real-time lossless image compression," *Proc. SPIE*, vol. 9501, pp. 45–52, May 2015.
- [13] S.-L. Chen, T.-Y. Liu, C.-W. Shen, and M.-C. Tuan, "VLSI implementation of a cost-efficient near-lossless CFA image compressor for wireless capsule endoscopy," *IEEE Access*, vol. 4, pp. 10235–10245, 2016.
- [14] S. Haddad, G. Coatrieux, M. Cozic, and D. Bouslimi, "Joint watermarking and lossless JPEG-LS compression for medical image security," in *Proc. Int. Conf. Watermarking Image Process.*, Sep. 2017, pp. 16–21.
- [15] X. Wang, L. Gong, C. Wang, X. Li, and X. Zhou, "UH-JLS: A parallel ultra-high throughput JPEG-LS encoding architecture for lossless image compression," in *Proc. IEEE 39th Int. Conf. Comput. Design (ICCD)*, Oct. 2021, pp. 335–343.
- [16] Z. Cao, T. Zhang, M. Liu, and H. Luo, "Wavelet-supervision convolutional neural network for restoration of JPEG-LS near lossless compression image," in *Proc. IEEE Asia Conf. Inf. Eng. (ACIE)*, Jan. 2021, pp. 32–36.
- [17] M. Liu, L. Tang, L. Fan, S. Zhong, H. Luo, and J. Peng, "CARNet: Context-aware residual learning for JPEG-LS compressed remote sensing image restoration," *Remote Sens.*, vol. 14, no. 24, p. 6318, Dec. 2022.
- [18] X. Sun, Z. Chen, L. Wang, and C. He, "A lossless image compression and encryption algorithm combining JPEG-LS, neural network and hyperchaotic system," *Nonlinear Dyn.*, vol. 111, no. 16, pp. 15445–15475, Aug. 2023.

- [19] K. Kanellopoulos, N. Vijaykumar, C. Giannoula, R. Azizi, S. Koppala, N. M. Ghiassi, T. Shahroodi, J. G. Luna, and O. Mutlu, "SMASH: Co-designing software compression and hardware-accelerated indexing for efficient sparse matrix operations," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2019, pp. 600–614.
- [20] M. E. Papadonikolakis, A. P. Kakarountas, and C. E. Goutis, "Efficient high-performance implementation of JPEG-LS encoder," *J. Real-Time Image Process.*, vol. 3, no. 4, pp. 303–310, Dec. 2008.
- [21] L. Chen, L. Yan, H. Sang, and T. Zhang, "High-throughput architecture for both lossless and near-lossless compression modes of LOCO-I algorithm," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 29, no. 12, pp. 3754–3764, Dec. 2019.



FANGJIA LIU received the B.S. degree in engineering mechanics from Northeastern University, Shenyang, China, in 2021. He is currently pursuing the M.S. degree in mechanical engineering with the University of Science and Technology Beijing, Beijing, China. His research interests include equipment diagnosis and intelligent manufacturing, digital image processing technology, and high-temperature material performance testing and evaluation.



XIYAO CHEN received the M.S. degree in internal medicine from the Third Affiliated Hospital, Sun Yat-sen University, Guangzhou, China, where she is currently pursuing the M.D. degree with the Infection Division.

She has published articles in *International Journal of Cancer*, *Canadian Journal of Gastroenterology and Hepatology*, and *European Journal of Gastroenterology and Hepatology*. Her main research interests include hepatitis B, liver cirrhosis, hepatic failure, and medical image processing.



ZIJUN LIAO received the B.S. degree in electromagnetic fields and wireless technology from the Huazhong University of Science and Technology, Wuhan, China, in 2021, where he is currently pursuing the M.S. degree in information and communication engineering.

His research interests include teleoperations, haptic communication, and FPGA.



CHONG YANG received the B.S. degree in mechatronics engineering from the Harbin Institute of Technology, Harbin, China, in 2021. His research interest includes medical robotics.