IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 10 April 2024, accepted 21 April 2024, date of publication 26 April 2024, date of current version 13 May 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3393946

RESEARCH ARTICLE

A Real-Time Image Row-Compression Method for High-Definition USB Cameras Based on FPGA

LONGHUA XIE, ZHIYONG ZHANG, AND TINGYU ZHAO
School of Science, Zhejiang Sci-Tech University, Hangzhou, Zhejiang 310018, China
Corresponding author: Tingyu Zhao (zhaotingyu@zstu.edu.cn)

ABSTRACT A real-time image compression method based on field programmable gate array is proposed for the problem of high-frame-rate high-resolution camera image transmission under the limited bandwidth of universal serial bus (USB). This method quantizes image pixels on a per-row basis, taking advantage of the high correlation between adjacent pixels within a row, thus reducing the data volume of a single frame image transmission. The algorithm also aims to minimize the decoding complexity on the central processor of the receiving end. Corresponding hardware circuits and software programs are designed, and tests are conducted on an experimental platform. The experimental results show that this method effectively compresses image data losslessly on the board, improves the transmission frame rate. The maximum frame rate of 1280×1280 images tested in a USB 2.0 environment can reach 25.58 fps, an improvement of 11.67 fps compared to the original data transfer, with a compression rate of up to 55.8%. Furthermore, this method outperforms PNG decoding in terms of decoding speed, supports multi-core decoding, and achieves the highest decoding speed of 61fps when tested on the 1920×1080 image with 16 threads. This method provides a feasible transfer solution for real-time compressed transmission of high-speed and high-definition cameras in the industrial field.

INDEX TERMS Field programmable gate array, image compression, lossless compression, image relevance, industrial applications.

I. INTRODUCTION

With the advancement of image sensors and differential transmission technology, the resolution and frame rate of images continue to increase, posing greater challenges for data storage and processing during image acquisition. Due to the bandwidth limitations of traditional Universal Serial Bus (USB), a large amount of image data will be stored in the hardware system under high frame rate and high-resolution conditions. To avoid data loss, the output rate of the sensor must inevitably be reduced accordingly, which leads to poor real-time performance of the data received. Therefore, it is particularly important for the lower-level machine to implement a set of methods to compress the volume of image data, thereby reducing bandwidth occupancy.

Image compression technology can be divided into different categories based on various standards. According to the degree of image recovery, it can be divided into lossy compression methods (such as run-length encoding, fractal encoding, etc.) and lossless compression methods (such as Huffman coding, arithmetic coding, etc.) According to the image encoding and decoding methods [1], it can be divided into entropy coding, transform coding, predictive coding, etc. In recent years, some researchers have used FPGAs to implement different types of image or video encoding. To improve the time-consuming problem of Huffman coding during construction sorting, the parallel characteristics of FPGA (Field Programmable Gate Array) are utilized to improve the implementation of Huffman coding, and pipeline technology is used to increase the system operating frequency. This solution is a trade-off of space for time [2]. Since Huffman coding is entropy coding, when there is a lot of random data in the image, the amount of data after compression may significantly exceed the amount before compression, introducing instability. To solve this problem, the PNG (Portable Network Graphics) compression algorithm firstly compresses

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

the image using the LZ77 algorithm and then re-encodes it using the Huffman algorithm. To study the feasibility of implementing PNG compression in hardware, the best filter is selected by comparing different filters during the prediction phase and implemented in hardware. At the same time, the LZ77 compression algorithm and Huffman coding are implemented in hardware to compress the data, solving the problem that the amount of data after Huffman compression may severely exceed the amount before compression. However, due to the high data correlation of the LZ77 algorithm, it requires multiple cycles to complete in complex prediction block situations, making this solution more suitable for storing data with low real-time requirements [3]. the performance of PNG deployed in FPGA is very poor [4]. To solve the problem of spacecraft component crashes that may be caused by high-frequency direct transmission of image data on aerospace equipment, an adaptive predictive coding compression algorithm based on the least mean square error is used to compress camera images. This method encodes the difference between the predicted value and the actual value while updating the weight parameters. This solution relies heavily on context data and requires real-time updates of the predictive weights, making the algorithm relatively complex [5]. Many scholars have also tried inter-frame compression schemes on FPGAs, among which the more famous video encoding schemes are H.264 and H.265. The inter-frame transmission scheme requires the comparison of two or even more than two images, which is complex in algorithm logic and requires the storage of multiple images at the same time, making it unsuitable for use in camera systems [6], [7]. More scholars have turned their attention to JPEG-LS, a lossless or near-lossless image compression algorithm based on the LOCO-I algorithm [8]. Some scholars have evaluated the JPEG-LS algorithm and found it suitable for elevation data compression [9]. JPEG-LS is also applicable to real-time image compression, Due to the significant impact of LOCO-I's context update on the overall algorithm throughput, a high-performance global pipeline hardware structure based on fragment compression and featuring spatial parallelism is proposed [10]. In order to solve the problem of context writing and reading in JPEG-LS, a co-design method is proposed from both software and hardware aspects [11]. The hardware structure of the LOCO-ANS lossless and near-lossless image compressor based on the JPEG-LS standard is evaluated [12]. Some scholars have proposed a new compression method, which can predict based on linear and non-linear methods. The prediction error coding uses two-level compression, adopting adaptive Golomb coding and binary arithmetic coding [13]. Some scholars have simplified the hardware occupancy of JPEG-LS by specific optimization in the hardware implementation process [14]. Compressing raw images with JPEG-LS is also a good choice [15]. In order to improve the parallel capability of JPEG-LS, a new architecture has been designed which can process four pixels at one period time on FPGA [16]. However, the software decoding speed of JPEG-LS falls short of real-time standards. It can only achieve a speed of 156ms per frame when decoding an image of 1024*2048 resolution on software [17]. Based on the shortcomings of the above schemes, this paper proposes a row compression encoding algorithm suitable for FPGA implementation. By compressing the single-row pixels in the image, it saves data transmission bandwidth and avoids the instability introduced by the amount of data after compression far exceeding the amount before compression. Experimental results show that this algorithm overcomes instability, improves the efficiency of image compression encoding, and has the advantages of low resource occupancy and a short compression cycle.

The organization of this paper is as follows: Section II introduces the design principles of the compression method and the overall design of the FPGA system. Section III describes the implementation logic of the main modules of compression encoding. Section IV compares with the other algorithms and conducts on-board testing experiments. Section 5 is the summary. Compression algorithm principle and overall system design.

A. COMPRESSION ALGORITHM PRINCIPLE

To achieve image compression, this paper proposes a method based on intra-row data comparison. This method utilizes the characteristic that adjacent pixels in an image are correlated to compress and encode images captured by the camera: the image data is converted into an encoded sequence consisting of benchmark data, compression amount, and differences between pixel data and benchmark data. The intra-row compression operation is processed as shown in Algorithm 1. 1. Take the first pixel value of each row as the benchmark data and set a deviation range. 2. Sequentially compare the difference between the current pixel value and the benchmark data; if it is within the deviation range, use the difference as the encoding output and count the number of consecutive pixels within the deviation range in this compression union; if the deviation between the pixel and the benchmark data exceeds the range, place the counted pixel data at the head of the previous unit's encoding queue, take the current pixel value as the new benchmark data, and repeat the above steps. 3. Perform the above operations for each line, and when a row of data ends, determine whether the amount of data after compression exceeds the original pixel data amount; if so, use the original data. Repeat above operations until an entire frame of the image has been processed. This method ensures that the compressed data amount does not exceed the original pixel data amount, effectively improving the compression rate.

From the above pseudocode, it can be seen that the encoding system is divided into two situations. In the worst case, the time complexity is

$$T(m, n) = O(mn^2) \tag{1}$$

where m represents the number of rows in the image, and n represents the number of pixels per row. At this time, the pixel correlation within each row of the image is extremely poor.

Algorithm 1 Proposed Algorithm Working Flow

Input: pixel array: P^*, deviation range:D, line size:N
Output: codec data array: Q^*

```
b ← 0, i ← 0, k ← 0;
Q* ← empty array;
while i < N do
    if i = 0 then
        b ← P_i;
        Q_k ← P_i;
        k ← k + 1;
        Q_k ← 0;
        k ← k + 1;
    else
        if (i = N − 1) then
            if (k > N − 1) then
                Q* ← P*;
            end
        else
            if (|b − P_i| < D) then
                Q_k ← b − P_i;
                k ← k + 1;
            else
                b ← P_i;
                Q_k ← P_i;
                k ← k + 1;
                Q_k ← 0;
                k ← k + 1;
            end
        end
    end
    i ← i + 1
end
return Q*;
```

In the best case, the time complexity is

$$T(m, n) = O(mn) \tag{2}$$

where all pixels in each row can be compressed. However, since the encoding system is implemented on the FPGA, and the software does not participate in the encoding process, the FPGA can compensate for the high time complexity through pipelining. Interestingly, for the decoding system, since there will be a mark for whether each row of pixels in the image is compressed after the encoding process, the time complexity of the decoding system is

$$T(m, n) = O(mn) \tag{3}$$

which greatly alleviates the pressure of software decoding.

B. OVERALL SYSTEM DESIGN

This paper has constructed a hardware platform based on Efinix's T85 series FPGA chips, selected the GMAX0505 image sensor from GPixel Technology Co., Ltd., used DDR3 (double-data-rate 3) as the image cache storage device, implemented data transmission through the USB2.0 interface, and adopted the AMD Ryzen7 5800h processor as the host computer processor. The development was carried out using Efinity software, with Verilog HDL as the design language.

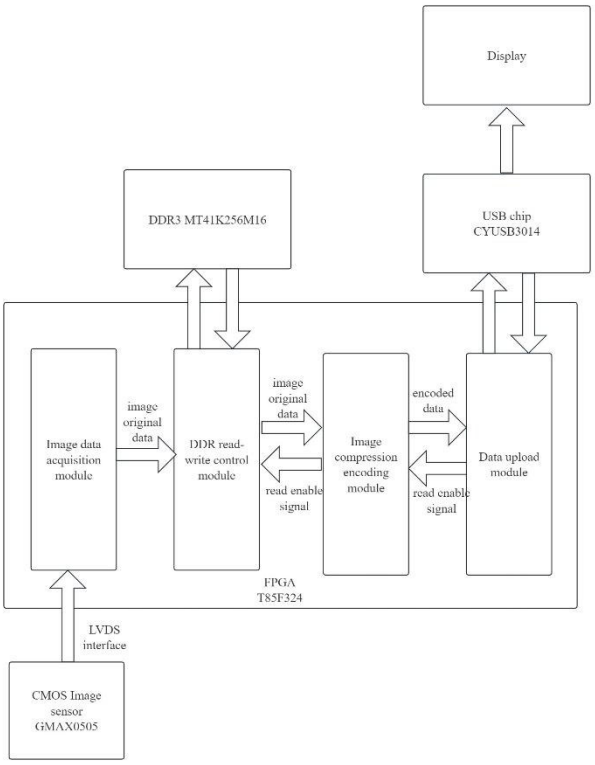

FIGURE 1. System architecture diagram.

FIGURE 1 shows the architecture of the system, which consists of four modules: the image data acquisition module, DDR read-write control module, image compression encoding module, and data upload module. The image data acquisition module receives the data output by the sensor through the LVDS (Low Voltage Differential Signaling) interface and sends it to the DDR3 for buffer storage after channel adjustment conversion through the DDR read-write control module. The image compression encoding module reads data from the DDR through the DDR read-write control module and encodes it. The encoded data is then passed to the CYUSB3014 chip by the data upload module and transmitted to the host computer to display by the CYUSB3014. When the CYUSB3014 detects that the internal data is full and has not been all received by the host computer, it will send a read pause signal to inform the data upload module to stop transmission.

II. IMAGE COMPRESSION AND ENCODING MODULE

The function of the image compression encoding module is to analyze and compress images. It consists of three sub-modules: the benchmark calibration sub-module, the compression amount statistics sub-module, and the encoding sub-module. The relationship between these three sub-modules is shown in FIGURE 2.

A. BENCHMARK CALIBRATION SUB-MODULE

The function of the benchmark calibration sub-module is to determine the position of the benchmark value for each

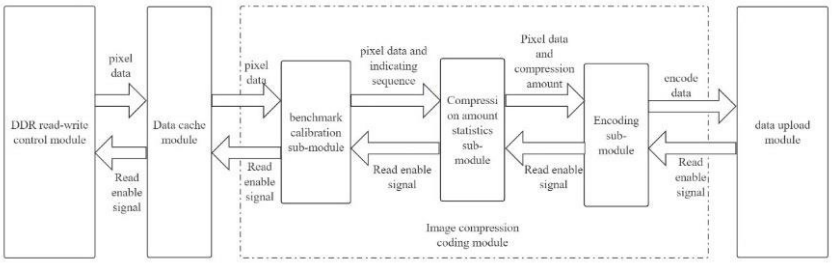

FIGURE 2. Internal structure diagram of image compression encoding module.

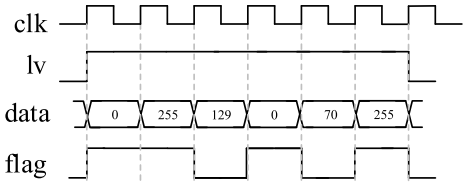

FIGURE 3. An example of output diagram of the benchmark calibration sub-module.

compression unit in a row. Upon receiving a row of data, it stores the original data in RAM (Random Access Memory) and generates a binary sequence to mark whether each position is a benchmark data, i.e., if the pixel value at that position is the benchmark value of the compression unit, then the flag is set to 1, otherwise, it is 0. This module also compares the difference between subsequent pixel values and the current benchmark value. The flag is set to 0 if the difference is within the deviation threshold, otherwise the flag is set to 1, and the current pixel value becomes the new benchmark data. The output of this module includes the original data and the calibration sequence.

Taking the deviation range [-127, +127] as an example, its illustration is shown in FIGURE 3. 0 is the first data in the row. Currently, the row valid signal lv is detected to rise, so 0 is used as the benchmark data. The flag is marked as 1. The difference between the second data and the first reference data exceeds the threshold of 127. Therefore, 255 is set as the new benchmark data, the flag is marked as 1, and the difference between 129 and 255 is less than 127, so it is judged that the value is not the benchmark data, and the corresponding flag is 0. By analogy, when it is detected that 255 is the last data in the row, the flag is also set to 1. To improve compression speed and reduce timing requirements, the benchmark calibration sub-module processes multiple pixels within one clock cycle. Let's take processing two data in one cycle as an example. Four signals are required for processing: frame valid signal (fv), line valid signal (lv), handshake signal (valid), and original data (pix_data, 32-bit data, where the high 16 bits are the first pixel data, and the low 16 bits are the second pixel data), and benchmark flag signal (flags, 2-bit data, where the high bit indicates whether the first pixel data is benchmark data, and the low bit indicates whether the second pixel data is benchmark data). At the start of the line valid signal, the module marks the first data as the benchmark data and calculates the difference between the two data currently input.

TABLE 1. Correspondence between benchmark data, input data, and benchmark flags.

Is the difference between the first data and the benchmark data within the deviation range?	Is the difference between the second data and the benchmark data within the deviation range?	Is the difference between the first data and the second data within the deviation range?	Benchmark Flags
Yes	Yes	Yes	00
Yes	Yes	No	00
Yes	No	Yes	01
Yes	No	No	01
No	Yes	Yes	10
No	Yes	No	11
No	No	Yes	10
No	No	No	11

If the gap between the second pixel data and the first pixel data is less than the given deviation, the benchmark data will not be updated, and the benchmark flags signal will set to 10; otherwise, the benchmark flag signal is set to 11, and the benchmark data will update to the second data. Subsequently, with the handshake signal high, the relationship between two input data and the benchmark data is determined, as shown in Table 1.

Repeat the operation until the module detects the falling edge of the line valid signal, then mark the second pixel as the benchmark position. This position corresponds to a sequence marked as 1. Wait for the start of the next row. The relevant timing is shown in FIGURE 4 (taking the deviation [-127, +127] as an example). When the sub-module detects the

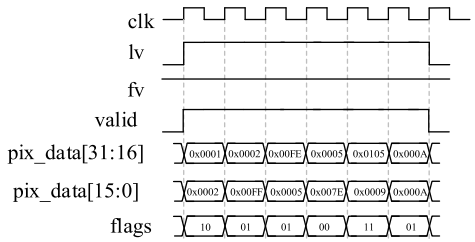

FIGURE 4. An example of output timing diagram of the benchmark calibration sub-module.

rising edge of lv, it takes 0×0001 as the benchmark data, and sets the high bit of the flags to 1. At this time, it detects that the difference between the data 0×0002 and 0×0001 is less than the threshold 127, so the low bit of the flags is set to 0. In the second clock cycle, the difference between the data signal 0×0002 and 0×0001 is less than 127, so the high bit of flags is set to 0, and the low bit $0 \times 00FF$ and 0×0001 have a difference of more than 127, so $0 \times 00FF$ is set as the benchmark data, and its flag low bit output is 1. And so on, to the last data of this line, at this time, it detects that the difference between the high bit of the data signal $0 \times 000A$ and the benchmark data 0×0009 is less than the threshold, so the high bit of the flag is set to 0, and the low bit of the data signal is the last data of this line, so the corresponding flag low bit is set to 1.

B. COMPRESSION AMOUNT STATISTICS SUB-MODULE

The compression amount statistics sub-module uses a ping-pong operation. Ping-pong operation is to use two FIFO (First Input First Output), one for writing and one for reading at the same time. In this sub-module, a line of original data is set to facilitate the encoding sub-module to encode. This sub-module is mainly responsible for counting the amount of data in each compression unit, that is, judging the number of pixels between the two flags with a value of 1 in the flags stream passed by the benchmark calibration sub-module. When a new row of data is detected, the counter is started. When the current sequence is detected as 1, the value of the counter is stored in RAM, and then the counter is reset; otherwise, the value of the counter continues to accumulate. At the same time, the sub-module also calculates the equivalent compression amount for that line. For different flags, the method of accumulating the equivalent compression amount is shown in Table 2. When the line needs to be output, the equivalent length will be compared with the valid data amount length of the input row. The sub-module outputs the original data stream, benchmark sequence, and the compression amount of the corresponding unit, as well as a flag signal indicating whether to compress that row.

The sub-module requires five signals: frame valid signal, line valid signal, handshake signal, original pixel data, and benchmark flags signal. Similarly, to ensure compression speed, this module will also process multiple data in one cycle. Taking the example of processing two data in one clock cycle, the sub-module internally uses 4 RAMs, with 2 RAMs as a group, adopting a method of writing one group while reading another. When the rising edge of the line valid signal is detected, the internal counter is reset, and in all cases where signals are valid, the counter operation performs different actions based on the incoming benchmark flags, as shown in Algorithm 2. If the incoming benchmark flags are 00, then the two counters each add 2; if the incoming benchmark flags are 01 or 10, different values are added to the previously selected RAM, and the RAM for this time is set; if the input benchmark flags are 11, then the current counter is saved to the last selected RAM and the counter is reset, at the same time, the RAM for this time is set as the second RAM; when the falling edge of the line valid signal is detected, then switch to another group of RAM for writing, and read from that group of RAM.

TABLE 2. The cumulative relationship between benchmark flag signals and equivalent compression amount.

Benchmark Flags	Equivalent Compression Amount
00	16
01	40
10	40
11	64

C. ENCODING SUB-MODULE

The compression encoding sub-module encodes image data using a method of benchmark data encoding. This module requires seven signals: frame valid signal, line valid signal, handshake signal, original pixel data, benchmark flags signal, compression identification, and unit compression amount. Internally, the module recalculates the deviation value and generates encodings of different lengths based on the benchmark flag signal, with the relationship between encoding length and benchmark flag signal as shown in Table 2.

The sub-module internally has a register with a width of 256. The sub-module will sequentially write the generated encoding into the high bits of register. When it is detected that the count exceeds 128, the module indicates that the data output is valid and outputs the low 128 bits of data from register to the data upload module. When the module detects the falling edge of the frame valid signal, it will output the remaining signals and after a delay of one cycle, output an end code to facilitate the host computer's recognition and termination of data reception. The related process is shown in Algorithm 3.

III. EXPERIMENTS

A. COMPARATIVE EXPERIMENTS

This paper compares the image performance parameters of different algorithms through simulation. The computer used for the experiment is configured with an AMD Ryzen R7

Algorithm 2 Last Select Ram and Benchmark Flags Corresponding Operator Ralationship

Input: frame valid signal:fv, line valid signal:lv,benchmark flag:$bf[1:0]$, reset signal:rst
Output: Register array:R_1, R_2

```
if (rst = 1) then
    LastSelectRam ← 0, count1 ← 0, count2 ← 0;
    R1 ← [0, 0...0];
    R2 ← [0, 0...0];
end
if (fv ∧ lv) then
    if (LastSelectRam = 0) then
        if (bf = 2'b00) then
            count2 ← count2 + 2;
            count1 ← count1 + 2;
        end
        if (bf = 2'b01 ∨ bf = 2'b11) then
            count1 ← count1 + 2;
            R1 ← [R1[0], R1[1]...count1];
            LastSelectRam ← 1;
            count1 ← 0, count2 ← 0;
        end
        if (bf = 2'b10) then
            count1 ← count1 + 1;
            R1 ← [R1[0], R1[1]...count1];
            LastSelectRam ← 0;
            count1 ← 0, count2 ← 0;
        end
    else
        if (bf = 2'b00) then
            count2 ← count2 + 2;
            count1 ← count1 + 2;
        end
        if (bf = 2'b01) then
            count2 ← count2 + 1;
            R2 ← [R2[0], R2[1]...count2];
            LastSelectRam ← 1;
            count1 ← 0, count2 ← 0;
        end
        if (bf = 2'b10) then
            R2 ← [R2[0], R2[1]...count2];
            LastSelectRam ← 0;
            count1 ← 0, count2 ← 0;
        end
        if (bf = 2'b11) then
            R2 ← [R2[0], R2[1]...count2];
            LastSelectRam ← 1;
            count1 ← 0, count2 ← 0;
        end
    end
end
```

Algorithm 3 Encoding Sub-Module

Input: pixel data: $P[31:0]$, frame valid signal: fv, data valid signal: dv, compress flag: cf, benchmark flag: $bf[1:0]$, union compress amount: $uc1[15:0]$, $uc2[15:0]$, Reset signal: rst
Output: codec data: $Q[127:0]$ output data valid: dvo

```
if (rst = 1) then
    i ← 0,arr[255:0] ← 0, count ← 0, b ← 0;
end
if (fv = 1) then
    if (cf = 0) then
        if (dv = 1) then
            arr[count + 31 : count] ← P[31:0];
            count ← count + 32;
        end
    else
        if (dv = 1) then
            if (bf = 2'b00) then
                arr[count + 15 : count] ← (b − P[31 : 16], b − P[15:0]);
                count ← count + 16;
            end
            if (bf = 2'b01) then
                arr[count + 39 : count] ← (b − P[31 : 16], P[15:0], uc2);
                count ← count + 40, b ← P[15:0];
            end
            if (bf = 2'b10) then
                arr[count + 39 : count] ← (P[31 : 16], uc1, b − P[15:0]);
                count ← count + 40, b ← P[31:16];
            end
            if bf = 2'b11 then
                arr[count + 63 : count] ← (P[31 : 16], uc1, P[15:0], uc2);
                count ← count + 64, b ← P[15:0];
            end
        end
    end
    if (count ≥ 128) then
        Q[127:0] ← arr[127:0];
        count ← count − 128;
        dvo ← 1;
    else
        Q[127:0] ← 0;
        dvo ← 0;
    end
else
    if (count ≠ 0) then
        Q[127:0] ← arr[count − 1, 0];
        dvo ← 1;
    else
        Q[127:0] ← 0;
        dvo ← 0;
    end
end
```

5800H CPU, which has 8 cores and 16 threads. The base clock speed is 3.2Ghz, and it can be overclocked up to 4.4Ghz. The L1 cache is 512KB, the L2 cache is 4MB, and the L3 cache is 16MB. The operating system is Windows 11 22H2. The computer has 40GB of RAM and a 1.5TB SSD. This paper randomly extracts a subset of data (1000 images) from the COCO dataset for compression encoding, in order to obtain more fair performance parameters. The images randomly extracted are normalized to a resolution of 1920 × 1080, pixel depth is 8bit per pixel. and a comprehensive evaluation is conducted on the compression ratio, decompression time, PSNR, and Throughput. The data results obtained from the experiment are shown in Table 3 (All algorithms have been set to single-thread decoding). From the table, it can be seen that the best compression effect is achieved by JPEG, with an average compression ratio of 0.206237. However, in terms of frame rate and image information loss, it is not as good as the algorithm proposed in this paper. JPEG-LS (with

TABLE 3. Comparison table of various parameters and indicators across different algorithms.

Algorithm	Compression ratio	Decompression frame rate	PSNR	Throughput
Proposed algorithm	0.739192	47.5667	Lossless	94.065MB/s
JPEG[18]	0.206237	42.3082	55.10712	83.666MB/s
PNG[19]	0.411754	29.6609	Lossless	58.656MB/s
JPEG-LS[17]	0.292913	26.2578	Lossless	51.926MB/s

TABLE 4. The relationship between the number of decoding threads and the decoding frame rate.

threads num	1	2	4	8	16
average frame rate(fps)	47.5667	52.7124	57.7731	60.9811	61.4435
CPU usage ratio(%)	4.538	4.810	5.584	8.016	20.581
Memory usage (MB)	777.1994	762.4731	763.0640	761.1025	762.2080

the Near parameter set to 0, at which point the decompression speed is the fastest) has a compression effect second only to JPEG, with an average compression ratio of 0.292913, but it lacks performance in software decoding, with an average frame rate of only 26.25778fps. PNG (with the compression parameter set to 1, at which point the decompression speed is the fastest) can achieve a compression ratio of 0.411754, but the average frame rate during decoding is only 29.6609fps.

At the same time, the algorithm proposed in this paper supports multi-threaded parallel decoding. The paper tested the average frame rate of decoding coco dataset(normalized to a resolution of 1920×1080, pixel depth is 8bit per pixel) under different threads by OpenMP, as shown in Table 4. We can see that as the number of threads increases, the decompression frame rate also increases accordingly. However, when the decompression thread exceeds 8 threads, the increase in decompression frame rate will decreases. This is because the program also generates some time overhead when creating threads, which leads to the reduction of frame rate. The best performance is achieved with 16-thread decoding, reaching a speed of 61.4435fps. This paper also tested the CPU usage and memory usage under different decoding threads. The CPU usage increases with the increase of the number of threads, reaching the highest at 16 threads, with an average usage rate of 20.581%. However, the increase in the number of threads does not have a significant impact on memory usage.

In order to simulate a real scenario, we added noise to FIGURE 5(normalized to a resolution of 1920×1080, pixel depth is 8bit per pixel) and then used different algorithms for compression and decompression to compare the relevant parameters. After adding salt-and-pepper noise and random noise to FIGURE 5, the relationship between various compression algorithms and compression rates is shown in FIGURE 6. The horizontal axis represents the proportion of noise added, and the vertical axis represents the compression rate. It can be seen from the figures that after the insertion of salt-and-pepper noise, as the proportion of noise increases, the compression rates of all compression algorithms are increasing. Among them, PNG compression performs better because it uses a mix of LZ77 and Huffman compression, so it can still control the compression rate around 0.7 even when the proportion of noise is large. Next is the algorithm of this paper and JPEG-LS, which can control the compression rate around 1.0 as the noise increases. Through random noise, it can be observed that the PNG algorithm still maintains a better compression rate. The algorithm of this paper is based on the principle of row compression, so when it is detected that the data size after row compression exceeds the original transmission data size, the original data will be automatically used for transmission, therefore, the compression rate can still be maintained around 1.0 under the worst conditions. It is worth mentioning that JPEG-LS does not provide a solution when the data size after compression is larger than before compression. In order to have a better comparison, when this situation occurs, this paper fills in with the last time the data size after compression is less than before compression.

FIGURE 5. Image to be processed.

The comparison of the decoding frame rate and noise ratio for each algorithm is shown in FIGURE 7. As the number of noise ratio increases, the decoding speed of the JPEG algorithm will rapidly decrease and eventually stabilize. All algorithms, except for the one proposed in this paper, other algorithm are at a lower decompression frame rate. In contrast, the row-based compression algorithm proposed in this paper will significantly increase the decoding speed as the number of noise points increases. When the amount of noise data reaches a point where the data size after compression is greater than before compression, the algorithm will automatically transmit the original pixel data. The host computer only needs to directly copy the data to the target location after parsing the uncompressed identifier. Therefore, in such cases, the algorithm of this paper can achieve a decoding speed

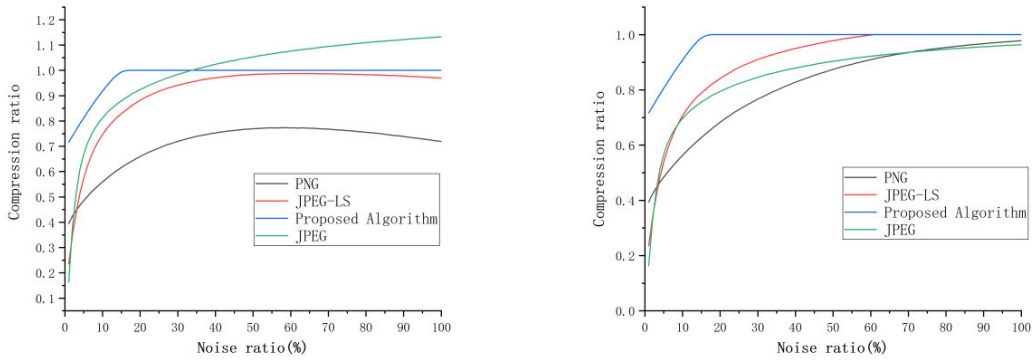

FIGURE 6. The relationship between compression ratio and noise ratio of various algorithms after inserting salt-and-pepper noise (left)and random noise (right).

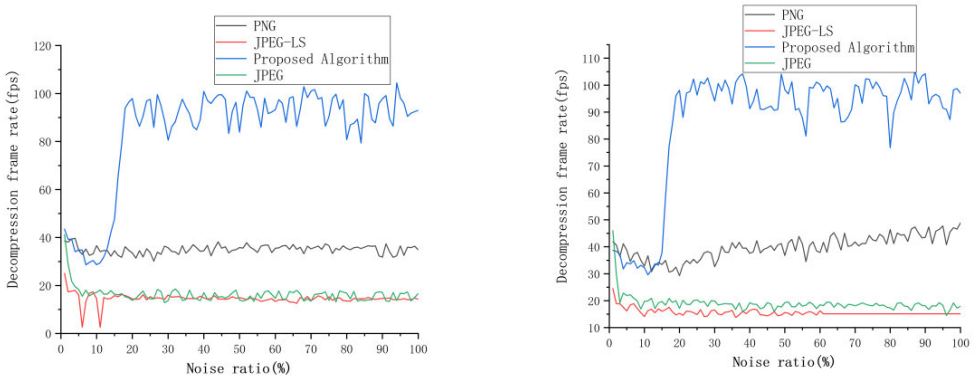

FIGURE 7. The decompression frame rate and noise ratio of various algorithms after inserting salt-and-pepper noise (left) and random noise (right).

TABLE 5. FPGA resource utilization.

Resource	Origin data output	Codec data output	Available
Logic Elements	13847	16865	84096
Memory Blocks	202	494	792

of over 80 frames, ensuring the stability of camera image transmission and minimizing the impact of the environment on the compression algorithm as much as possible.

B. ON-BOARD EXPERIMENTS

After the program compilation is completed, the resource utilization rate is as shown in Table 4. Compared to the original data output, the logic resources occupied by the output after adding compression encoding increased by 3%, and the storage resources increased by 36.8%.

FIGURE 8 shows the image captured by the sensor, which has been successfully decoded and displayed by the host computer. The image resolution is 1280 × 1280, with a bit depth of 12 bits (transmitted as 16 bits per pixel). Under USB 2.0 (The ideal bandwidth is 32 Mbyte/s, but the actual test can reach 42 Mbyte/s), the original transmission frame rate is 13.91 fps, and after compression, the transmission frame rate can reach 25.58 fps. the experimental results have proven that

FIGURE 8. Captured image.

this compression algorithm can reduce the data transmission size of a single frame image and increase the frame rate.

IV. CONCLUSION

In response to the phenomenon of low transmission frame rates and poor real-time performance of high-resolution cameras under the condition of limited bandwidth transmission media, this paper proposes a lossless compression

algorithm based on the principle that adjacent pixels in an image have high correlation, and implements this algorithm on an FPGA. At the same time, this paper compares the proposed algorithm with common compression encoding and decoding algorithms in terms of compression rate, decompression frame rate, and image performance. The experimental results show that the algorithm in this paper is superior to the PNG algorithm in decompression frame rate, while ensuring lossless compression in image performance. Considering the current development of processors towards multi-core and multi-threading, this paper tests the frame rate of the algorithm under multi-threaded decompression. The algorithm in this paper can achieve optimal processing of images with a resolution of 1920 × 1080 under 16-thread decompression, reaching up to 61fps. Considering the worst-case, this paper also adds noise to the images to simulate the worst case and compares different algorithms in terms of decoding frame rate. The experimental results show that the higher the random pixels in the image, the higher the frame rate that can be achieved by the algorithm in this paper. The experimental results demonstrate that under certain bandwidth conditions, the algorithm in this paper can reduce the data size of single-frame image transmission and improve the real-time transmission of the camera on limited bandwidth media. Moreover, the decoding algorithm requires low computational power on the receiving end and can support multi-threaded processing. For cameras with high resolution that only support limited bandwidth media transmission, this algorithm can save the transmission volume of single frames and improve the frame rate. With the continuous development of sensor technology and differential transmission technology, cameras will inevitably develop towards faster frame rates and higher resolutions, but this also poses challenges to the transmission media, especially traditional media such as USB, GIGE (Gigabit Ethernet), etc.

REFERENCES

[1] R. M. Thanki and A. Kothari, "Classification in data compression," in *Hybrid and Advanced Compression Techniques for Medical Images*. Cham, Switzerland: Springer, 2019, pp. 17–29.

[2] Y. Chen, G. C. Wan, L. Y. Tang, and M. S. Tong, "Huffman coding method based on parallel implementation of FPGA," in *Proc. Prog. Electromagn. Res. Symp.*, 2017, pp. 2207–2211.

[3] R. Kachouri and M. Akil, "Hardware design to accelerate PNG encoder for binary mask compression on FPGA," *Proc. SPIE*, vol. 9400, pp. 15–27, Feb. 2015.

[4] A. Filipe and R. Merendeira. (2024). *Verilog PNG Encoder*. Semanticscholar. Accessed: Apr. 8, 2024. [Online]. Available: https://www.semanticscholar.org/paper/Verilog-PNG-Encoder-Filipe-Merendeira/c317ab87bd7e5738bd9cd570108111c9454cfbb3#related-papers

[5] L. Chun-Xiang and G. Yong-Fe, "Real-time composing and compression of image within multi-channel spaceborne camera," *J. Jilin Univ.*, vol. 43, no. 6, pp. 1680–1684, Jan. 2013.

[6] W. Chen, Q. He, S. Li, B. Xiao, M. Chen, and Z. Chai, "Parallel implementation of H.265 intra-frame coding based on FPGA heterogeneous platform," in *Proc. IEEE 22nd Int. Conf. High Perform. Comput. Commun., IEEE 18th Int. Conf. Smart City, IEEE 6th Int. Conf. Data Sci. Syst. (HPCC/SmartCity/DSS)*, Dec. 2020, pp. 736–743.

[7] M. Balasubramani and S. P. Sunder, "A novel dual stage pipeline method of H.264 CAVLC video coding using FPGA," *Indo-Iranian J. Sci. Res.*, vol. 2, pp. 74–82, Feb. 2018.

[8] M. J. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I lossless image compression algorithm: Principles and standardization into JPEG-LS," *IEEE Trans. Image Process.*, vol. 9, no. 8, pp. 1309–1324, Aug. 2000.

[9] 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.

[10] 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.

[11] 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.

[12] T. Alonso, G. Sutter, and J. E. López de Vergara, "An FPGA-based LOCO-ANS implementation for lossless and near-lossless image compression using high-level synthesis," *Electronics*, vol. 10, no. 23, p. 2934, Nov. 2021.

[13] G. Ulacha and M. Lazoryszczak, "Lossless image coding using non-MMSE algorithms to calculate linear prediction coefficients," *Entropy*, vol. 25, no. 1, p. 156, Jan. 2023.

[14] Y. M. Mert, "FPGA-based JPEG-LS encoder for onboard real-time lossless image compression," *Proc. SPIE*, vol. 9501, pp. 45–52, May 2015.

[15] Y. Zheng, T. Li, W. Li, F. Lei, J. Liu, and Y. Fan, "A low-complexity algorithm for JPEG-LS-based RAW domain compression," in *Proc. IEEE 15th Int. Conf. ASIC (ASICON)*, Oct. 2023, pp. 1–4.

[16] X. Dong and P. Li, "Implementation of a real-time lossless JPEG-LS compression algorithm based on FPGA," in *Proc. 14th Int. Conf. Signal Process. Syst. (ICSPS)*, Nov. 2022, pp. 523–528.

[17] J. Li, Z. Wang, X. Pan, and X. Shi, "Design and FPGA implementation of JPEG-LS image decompression," in *Proc. 2nd Int. Conf. Optoelectronic Inf. Comput. Eng.*, vol. 12752, Nov. 2023, pp. 92–98.

[18] M. W. Marcellin, M. J. Gormish, A. Bilgin, and M. P. Boliek, "An overview of JPEG-2000," in *Proc. Data Compress. Conf.*, Nov. 2002, pp. 523–541.

[19] *PNG (Portable Network Graphics) Specification Version 1.0*, document RFC2083, 1997.

LONGHUA XIE received the B.Sc. degree from Fujian University of Technology, in 2021. He is currently pursuing the master's degree in optoelectronic information engineering with Zhejiang Sci-Tech University. His current research interest includes camera development and processing. His research interest includes logic and image processing.

ZHIYONG ZHANG is currently pursuing the master's degree in optoelectronic information engineering with Zhejiang Sci-Tech University. His research interests include logical and formal methods and digital circuits.

TINGYU ZHAO received the Ph.D. degree from the College of information Science and Electronic Engineering, Zhejiang University, in 2009. She is currently a Professor with the School of Science, Zhejiang Sci-Tech University. Her research interests include the integrated design of optomechanical computing, including CAD design of imaging optical systems, mechanical structure design of optical instruments, and digital image processing algorithms.