Frontiers in Computing and Intelligent Systems
ISSN: 2832-6024 | Vol. 3, No. 2, 2023

Image lossless compression algorithm optimization and FPGA implementation

Fangbin Dang

School of Information Science and Technology, Southwest Jiaotong University, Chengdu 610031, China

Abstract: In this thesis, a minimum redundant prefix coding with higher compression ratio and lower time complexity is proposed for lossless compression of HD images. The compression algorithm is based on Canonical Huffman coding, preprocesses the data source to be compressed according to the image data features, and then compresses the data in batches using the locally uneven features in the data, which improves the compression ratio by 1.678 times compared with the traditional canonical Huffman coding. During the implementation of the algorithm, the counting sorting method with lower time complexity and the code-length table construction method without relying on binary trees are used to reduce the complexity of the algorithm and achieve the purpose of real-time data processing by the system. Finally, the proposed compression algorithm is deployed in FPGA to improve the encoding rate by parallel hardware circuit and pipeline design.

Keywords: Image lossless compression; Compression ratio; Time complexity; FPGA.

1. Introduction

Entropy coding is one of the most commonly used compression methods, and its common ways are Golomb coding[1,2], arithmetic coding[3,4], Asymmetric Numeral systems (ANS) coding[5,6], and Huffman coding[7,8]. Among these methods, Golomb coding is an encoding method that can only encode non-negative integers, which divides the integer to be encoded into two parts, quotient and remainder, according to the parameter N. When N is not a power of 2, the encoding complexity is high; arithmetic coding is a full-sequence coding, which encodes the data source as binary values of size between 0 and 1. ANS coding is a compression algorithm that balances coding speed and algorithm complexity, and has two implementations: rANS and tANS. rANS require arithmetic operations of high complexity, which is slow and not suitable for applications with real-time requirements. tANS need to store the whole coding table, which has a high memory requirement and is not suitable for low-memory scenarios such as mobile devices. Huffman coding is a statistically implemented optimal grouping code that makes the average code length of the encoded symbols close to the lower bound of the average code length of variable-length codes given in Shannon's theorem. Although it does not compress as well as arithmetic coding, the coding process is simpler than arithmetic coding and more suitable for implementation in hardware.

When Huffman coding is implemented on the software side and the amount of data to be compressed is relatively large, the consumption of CPU resources is large and the data processing speed is slow, which is not suitable for time-critical situations. When Huffman coding is implemented using FPGAs, the speed of decompression can be improved and the ability to process data in real time can be realized. Moreover, when the task is handled in cooperation with a computer, a large amount of storage resources, computational resources can be saved and the energy consumption of the whole system will be reduced with the improvement of computational efficiency.

2. Algorithm improvement based on Canonical Huffman coding

2.1. Pre-processing

In order to obtain a more ideal compression effect, the compression algorithm and the implementation need to be improved according to the different characteristics of different compression objects. The preprocessing module proposed in this thesis for HD images contains two parts: residual operation and mapping, and the structure is shown in Figure 1.

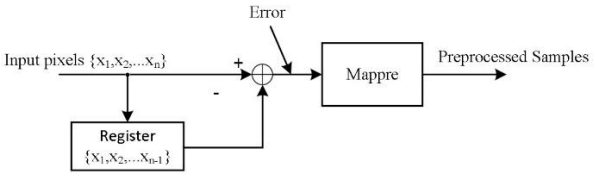


Fig. 1 Pre-processing structure

According to the characteristics of the image data, it is known that the rest of the image has a higher correlation between neighboring pixel points except for the adjacent physical demarcation region in the image. Therefore, when doing residual processing, the first pixel point of each channel is used as the reference value to restore the original pixel value. For the other pixel points use the method of subtracting adjacent pixels to construct the residual value, as shown in Figure 2, the three color regions in the figure are distributed to represent the R, G and B channels, and the residual value is subtracted in the direction indicated by the arrow, for the first pixel value of each row the first pixel of the previous row that is closer is selected as the subtracted number, and the rest of the part takes the left pixel point as the subtracted number.

The resulting residual value range is -255 to 255, if the value is directly encoded, the range of statistical values becomes larger, requiring more resources, so the residual value is converted to a non-negative integer by positive mapping, the mapped residual value range is 0 to 255, the mapping is calculated as follows the mapping is calculated as follows.

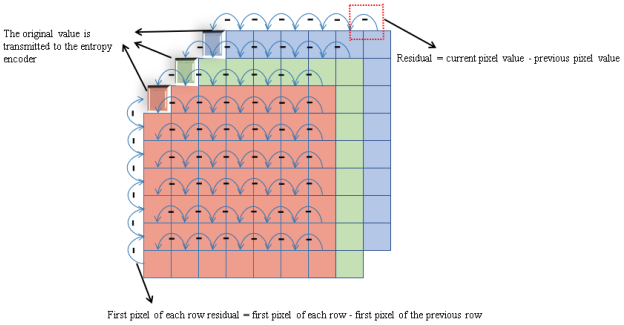


Fig. 2 Schematic diagram of residual calculation

$$map\_res_i = \begin{cases} theta_i + |\delta_i|, & |\delta_i| > theta_i \\ 2 \times |\delta_i| - 1, & \delta_i < 0 \\ 2 \times \delta_i, & otherwise \end{cases} \quad (1)$$

Where, δ_i is the residual value obtained by subtracting between pixel points. $theta_i$ is calculated as in (2):

$$theta_i = min\{x(i-1) - x\_min, x\_max - x(i-1)\} \quad (2)$$

Where x is the input pixel value, and x_min and x_max are the minimum and maximum values of the input pixel value, respectively. The " Compressed data-frequency" distributions are shown in Figures 3 to 5 for the original image data, residual data, and positive mapping data, respectively:

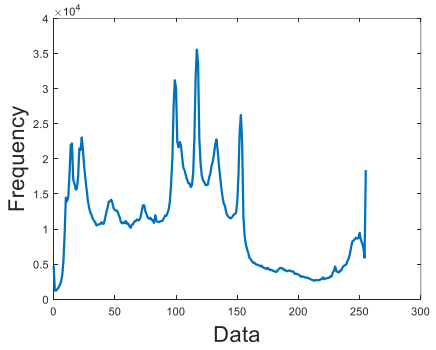


Fig.3 Frequency distribution of raw data

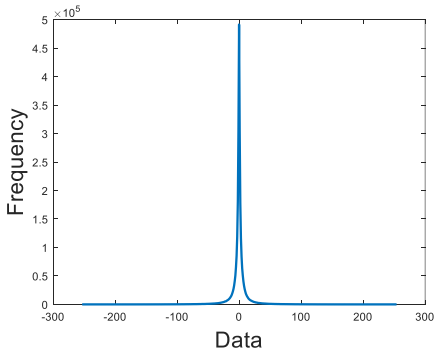


Fig.4 Frequency distribution of residual data

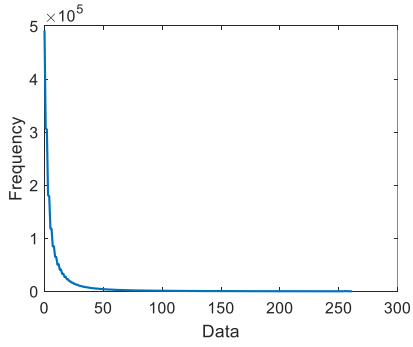


Fig. 5 Frequency distribution of positive mapping data

From the above figure, it can be seen that the frequency difference of the data after pre-processing increases. Shannon's entropy value and compression ratio were calculated with the original data and the preprocessed data, respectively, and the results are shown in Table 1:

Table 1. Comparison of compression effect between raw data and preprocessed data

Encoede Data Type	Shannon Entropy	Compression ratio
Raw image data	7.7183	1.0332
Pre-processed data	4.7812	1.6611

The result of Shannon's entropy represents the optimal average coding length that can be compressed to. From the above table, it can be seen that the optimal average coding length that can be theoretically achieved after pre-processing is 4.7812 bits, and the compression ratio is improved by 1.608 times.

2.2. Grouping of data to be compressed

Huffman coding compresses the data by counting the frequency of each pixel value, which generates a large delay when the amount of compressed data is large and inhibits the real-time transmission effect of Huffman coding. In order to improve the coding rate, the data is compressed in groups, and the coding between groups does not interfere with each other, which can effectively reduce the delay. In addition, this grouping process makes full use of the characteristics of the local character frequencies of the data to be compressed, which can achieve better data compression results.

Using HD image as the compression object, the results obtained after pre-processing are grouped and compressed with 1024, 2048, 4096 values as the amount of data compressed each time, and the results of Shannon entropy and compression ratio obtained are shown in Table 2:

Table 2. Compression effect under different group size

Compressed data volume	Number of groups	Average Shannon entropy	Compression ratio
512	5400	4.4775	1.7741
1024	2700	4.5842	1.7335
2048	1350	4.6358	1.7144
4096	675	4.6680	1.7025

Comparing Table 2 with Table 1, we can see that the data is better compressed after grouping. By considering the compression effect, the compression processing time, and the resource consumption during hardware implementation, this thesis chooses to compress the images after dividing the data into 2700 groups.

In addition to improving the compression ratio effect, grouping compression also facilitates the limitation of the encoding length. Assuming that the data to be compressed contains a total of N (N>1) different values, the longest possible encoding length when using Huffman coding compression is N-1. While using grouping for encoding, the depth of the Huffman tree is reduced due to the reduction of the maximum occurrence frequency, which can greatly reduce the number of encoded character bits.

2.3. Linear frequency table construction

In order to reduce the time complexity of the canonical Huffman coding algorithm, the comparison-based sorting algorithm is replaced by the counting sorting method in this thesis. The worst-case time complexity of any comparison-based sorting algorithm is O(nlogn). The time complexity of counting sort is O(n+k), where k represents the maximum value in the input data range, and the method is suitable for sorting among integer-type elements whose k values are not very large. When the object to be compressed is an RGB image and the amount of data to be compressed in each group is 1024, the time complexity of the counting sort O(1024+255) is significantly smaller than that of the comparative sorting algorithm O(1024×log1024).

The steps for constructing the linear frequency table proposed in this thesis are summarized as follows:

(1) The frequency of each symbol appearing in the data to be compressed is counted by the counting and sorting method, and the frequency table F is obtained.

Taking the array Data=[15, 2, 11, 11, 8, 2, 8, 8, 3, 15, 8, 10] as an example, a storage space with data arranged from smallest to largest is constructed based on the values in the array, as shown in Figure 6. Iterate through each data in the array Data, and use the result of subtracting it from the smallest value as the storage space index value, so that the corresponding F is added by 1.

S	2	3	4	5	6	7	8	9	10	11	12	13	14	15
F	2	1	0	0	0	0	4	0	1	2	0	0	0	2

Fig. 6 Constructing a counted sorted storage space

(2) In order to reduce the amount of data involved in subsequent statistics and sorting, nodes with frequency 0 in F are removed.

(3) After removing the zero-frequency nodes, F is used as input, and the number of values with the same frequency is counted again using counting order, and the results are shown in Figure 7.

S2	1	2	3	4
F2	2	3	0	1

Fig. 7 Frequency statistics results

(4) According to the same number of frequencies, the starting address of each frequency data in the frequency sorting memory . Take Figure 7 as an example, the node with the largest frequency is S2 (4), the starting address of this frequency node is set to 1, and the statistical result shows that there are only 1 node of this frequency, so the starting address of the node with the next largest frequency, S2 (3), is 2, but the number of nodes of this frequency is 0, so the second frequency needs to be changed to S2 (2), and there are 3 nodes of this frequency, so the starting address of the node with frequency 1 is The node with frequency 1 is delayed to the 5th position. The final result is addr=[5,2,2,1].

(5) Find the index Idx of the sorted position of each non-zero frequency data in F. Iterate through each non-zero value in F and use Eq. The result of k(i) is used as the index value:

$$k(i) = F(i) - (S\_min - 1) \quad (3)$$

where S_min is the smallest value in S. Based on this rule, the F non-zero frequency ordering position in Figure 6 is: Idx=[3,1,5,6,2,4].

(6) The data in F are rearranged according to Idx so that the frequencies are stored in the order from largest to smallest. As shown in Figure 8:

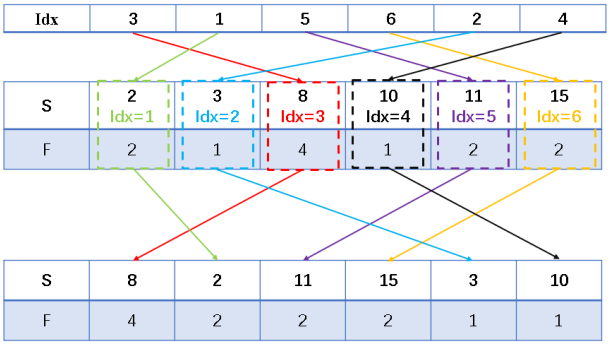


Fig. 8 Sorting by Idx

2.4. Linear frequency table construction

Canonical Huffman coding assigns a unique translatable code to each symbol, and the entire code table can be reconstructed by only the code length table, i.e., the storage of the entire Huffman coding table is converted to the storage of the code length of each symbol. Canonical Huffman coding, like Huffman coding, requires generating the code length table by constructing a Huffman tree, but the Huffman tree structure is more complicated and the efficiency of generating codes is lower. Therefore, this thesis proposes a method that does not require the construction of a Huffman tree. The code length can be obtained by the operation between each element of the array through the mapping relationship of the parent node pointing to the child nodes, and the specific process is summarized as follows:

(1) The frequency sorting memory F is scanned from left to right, and two values (i.e., the least frequent node in the current array) are taken from F at a time and added together to construct the root node. Since only the depth of each node needs to be known, the root node value does not need to be stored in the memory, but only the pointer to each root node can be recorded.

Suppose the initial value of frequency sorting memory F is: F=[2, 2, 4, 4, 5, 14, 15, 20],according to step (1) we can get the value of each root node pointer as [3, 4, 4, 6, 7, 7]. In order to understand more intuitively the calculation process among the elements of the above array, it is represented in the form of a binary tree as shown in Figure 9:

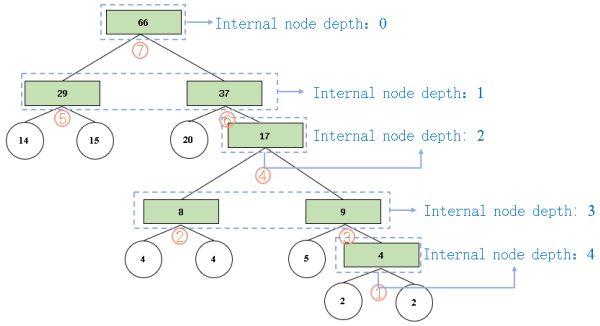


Fig. 9 Binary tree structure

The green boxes in the figure indicate internal nodes, the circles indicate external nodes, and the red circles are marked with the order of tree building.

(2) Set the maximum depth of root node F (n-1) as 0, and then put the result obtained from step (1) into equation (4) to

calculate the depth of each internal node.

$$F(i) = F(F(i)) + 1, i = n-2, n-3, \ldots, 1 \quad (4)$$

n in the above equation is the length of F. Substituting the example result in step (1) into in the equation, we can get the depth of each internal node as F = [4, 3, 3, 2, 1, 1, 0].

(3) According to the depth of internal nodes, the depth of each leaf node is obtained: firstly, the number of nodes contained in different node depths is counted in the result of step (2), and then the mapping relationship between parent and child nodes shows that when the total number of nodes in a certain depth is greater than the number of internal nodes, it means that the nodes in that depth contain external nodes in addition to internal nodes, and the depth of these external nodes can be determined. The depth of these external nodes can be determined.

According to the above rules, the example final result is F = [5, 5, 4, 4, 4, 4, 2, 2, 2]. This value is the code length of each node.

3. FPGA hardware architecture design

According to the optimized compression algorithm, a hardware circuit structure based on FPGA implementation is designed in this thesis. The structure mainly contains preprocessing module M0, frequency statistical sorting module M1, constructing code length module M2, code word module M3, and mapping module M4. The overall structure of its hardware implementation is shown in Figure 10:

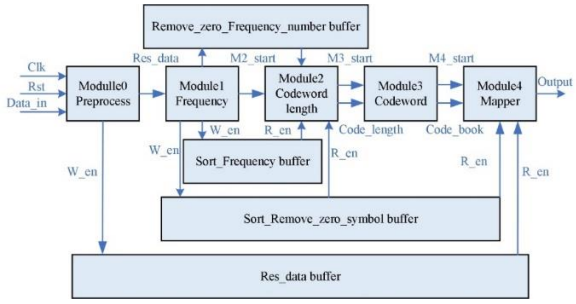


Fig. 10 Hardware implementation architecture

3.1. Frequency statistics module

In this thesis, the data to be compressed are divided into groups of 1024 samples each for group compression, and the statistical circuit needs to be cleared to zero after every 1024 data are counted, and this operation brings the problem that the statistics of one cycle of residual data will be missed when it is cleared to zero. In order to prevent incomplete data statistics, this thesis adopts the ping-pong structure, so that when the register is cleared after the previous batch of 1024 data is finished counting, the next batch of 1024 data can continue to be counted.

In order to enhance the hardware circuit speed, the module adopts parallel processing. Sixteen residual data are sent to the accumulation module every cycle, and 16 sets of accumulation results can be obtained after 64 cycles, as shown in Figure 11. Each blue box represents a different residual data, and the residual value is used as the index of the position of different blue boxes in the register, and the count value of the corresponding position is added one when each box is indexed once.

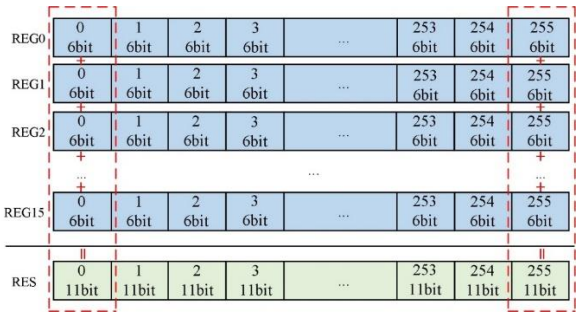


Fig. 11 Cumulative register structure

Each of the accumulation registers in Figure 11 contains the statistical intermediate results of 256 values, and the values at the same index position are summed to obtain the frequency of the residual value. When multiple data are added, direct summing can cause circuit delays and excessive resource utilization. To solve this problem, the method often used today is the Carry Save Adder (.). The constructed addition tree is shown in Figure 12 and contains both 3-2 compression and 4-2 compression structures.

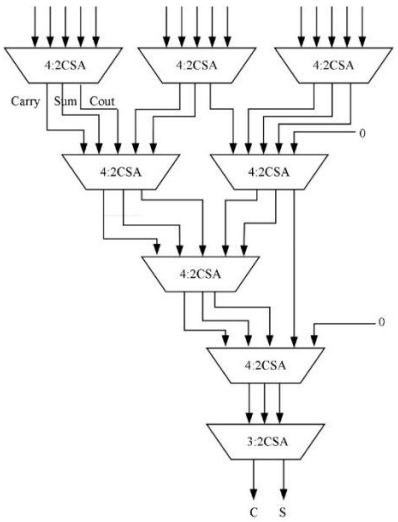


Fig. 12 CSA compression tree

In Fig. 12, the logical expression of the 3:2 compressor is shown in (5) and (6):

$$C = a_0 \times a_1 + a_2 \times (a_0 + a_1) \quad (5)$$

$$S = a_0 \oplus a_1 \oplus a_2 \quad (6)$$

The 4:2 compressor composed of two full adders in series needs to go through four heterodyning gates, and the circuit timing delay is large, which can be changed to the structure shown in Figure 13, which only needs to go through three heterodyning gates.

The logic expression corresponding to the circuit structure of Fig. 13 is

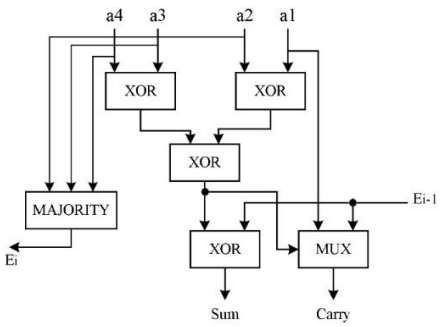


Fig. 13 4:2 compressor structure

$$E_i = (a_4 \& a_3) || (a_3 \& a_2) || (a_4 \& a_2) \quad (7)$$

$$Sum = a_4 \oplus a_3 \oplus a_2 \oplus a_1 \oplus E_{i-1} \quad (8)$$

$$Carry = (a_1 \& \overline{(a_4 \oplus a_3 \oplus a_2 \oplus a_1)}) || (E_{i-1} \& (a_4 \oplus a_3 \oplus a_2 \oplus a_1)) \quad (9)$$

$$a_1 + a_2 + a_3 + a_4 + E_{i-1} = 2 \times (E_i + Carry) + Sum \quad (10)$$

3.2. Removal of zero-frequency node module

The number of data involved in sorting can be reduced by eliminating zero frequency nodes, and the hardware implementation can achieve the purpose of circuit acceleration by using this method. The flag register is set according to the statistical frequency of the data to be compressed, as shown in Figure 14:

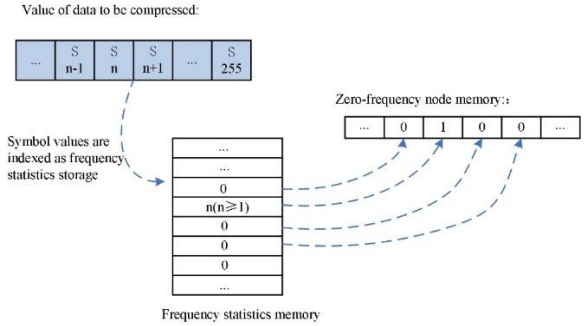


Fig. 14 Setting zero-frequency node memory based on statistical frequency

In this thesis, a set of zero-frequency node flag bit registers are designed to distinguish the nodes to be sorted and zero-frequency nodes. Each bit in this flag bit register corresponds to the statistics of a symbol, and if the statistical frequency of a symbol is not 0, the corresponding position in the flag bit register is set to 1'b1, otherwise it is set to 1'b0.

The process of setting the zero-node flag bit register is simple, but the operation of selecting the corresponding bit and assigning the value is prone to generate critical timing paths. In the process of removing the zero-frequency node, the corresponding bit in the register flag bit needs to be selected according to the input data. Taking the HD image as an example, the pre-processed data contains a total of 256 symbols and requires a logic circuit containing 256 conditional judgment branches, which is equivalent to an 8-256 decoder structure. In this thesis, the decoder is designed as a 4-16 decoder composed of 5 pieces of 2-4 decoders (one piece as a piece-selected decoder, 4 pieces cascaded), and then 17 pieces of 4-16 decoders to form an 8-256 decoder structure (one piece as a piece-selected decoder, 16 pieces cascaded), where 16-chip cascade), in which each 8-256 decoder output will pass through two levels of 4-16 decoder combination logic units, and each 4-16 decoder needs to pass through two levels of 2-4 decoder combination logic units, which will generate a large timing delay. By interrupting the timing path by inserting registers into the 4-16 decoder structure, the timing delay can be effectively reduced, and the 4-16 circuit structure is shown in Figure 15:

The output of each result of the decoder composed of Figure 15 requires a delay of 4 clock cycles, but the corresponding clock frequency of the circuit is boosted and its timing path is broken into 3 segments, each of which is used to pass through only one level of combinational logic circuitry.

Before frequency sorting, the flag bit registers are sequentially scanned and the non-zero frequency nodes are passed into the sorting module from the statistical memory corresponding to the flag bits. This bit-by-bit scan of the registers consumes a large number of clock cycles, and the data to be processed is batched in order to quickly remove valid nodes. The zero-frequency node sign bit register contains frequency statistics for a total of 256 symbols, and assuming that 4 bits of data are scanned in parallel per cycle, all the bits in the sign bit register can be scanned after 64 cycles.

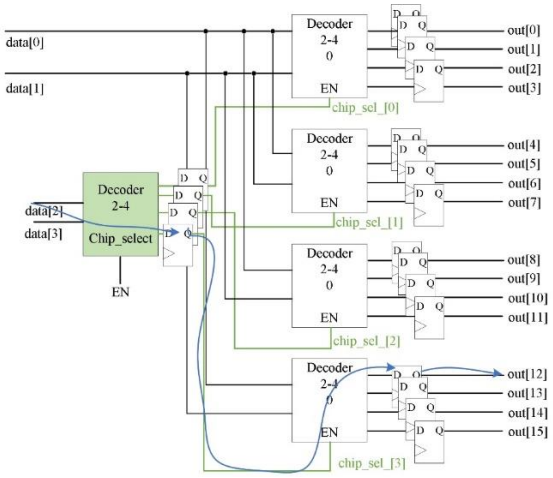


Fig. 15 4-16 decoder structure with inserted registers

3.3. Frequency sorting module

The frequencies are sorted according to the method in Section 1.3 of this thesis. The first step is to find the most value in the frequency by constructing a comparison tree, as shown in Figure 16.

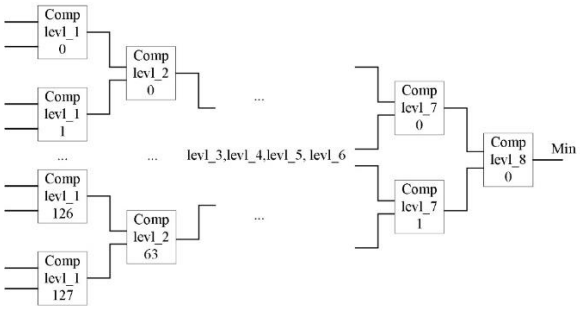


Fig.16 Comparison tree

The index of each frequency value in the statistical memory of the number of data of the same frequency can be obtained by subtracting the minimum value from each data in the statistical memory with zero frequency removed, and the count value of the memory cell corresponding to each index value is added by 1. In order to get the statistical results faster, this thesis constructs four memories for storing intermediate statistical results, and inputs one index value into each of the four memories every cycle to add 1 to the corresponding address space statistical value, and after traversing all the index values, the data in the same addresses of the four memories are added together to get the final statistical value.

3.4. Code Length Table Building Block

This module contains three parts, which are calculating the root node pointer, calculating the internal node depth, and calculating the leaf node depth. When calculating the internal node value, the data in the two smallest addresses are directly taken from the sorted frequency memory and added as the

first root node, after which the values in the frequency memory are traversed one by one and compared with the root node, and the value of the internal node is constructed by adding the two nodes with the smallest frequency values each time according to the comparison result, and the structure is shown in Figure 17:

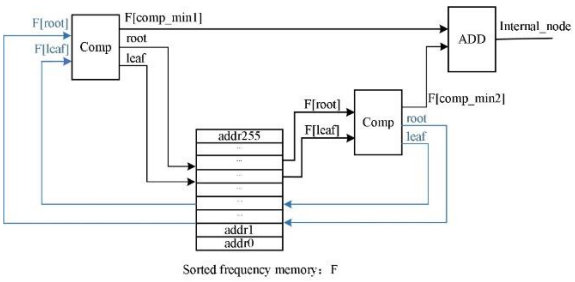


Fig. 17 Schematic diagram of internal node calculation

The computation of the three parts of the module is dependent, and the computation of the later step depends on the result of the previous step. In this thesis, the data flow in each step and the parallel loop computation are controlled by a state machine. The control state machine of the module is shown in Figure 18:

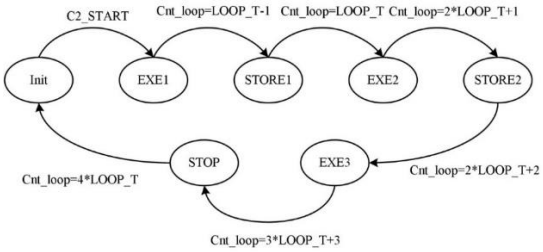


Fig.18 Code length table control state machine

Figure 18 state machine outputs three flag bit signals, start_loop1, start_loop2, start_loop3. Init is the initialization stage; EXE1 is the calculation of the root node pointer stage, flag bit starts_loop1 is 1, the rest of the flag bits are 0; when the state jumps to STORE1, the root node pointer calculation is complete, the root node pointer will be stored in the register. When the state jumps to EXE2, the internal node depth is calculated, the start_loop2 flag bit is pulled high, and the operand is read from the register assigned by STORE1; similarly, in STORE2, the internal node depth result is stored in the register, and the three flag bits are low; when the state is EXE3, start_loop3 goes high and starts to calculate the leaf node depth; STOP state, get the final code length result.

3.5. Pipeline design

From the hardware realization point of view, the packet compression method is beneficial to save storage resources and parallel computing. The improved Canonical Huffman coding process in this thesis mainly includes four processes: statistics, sorting, code length table construction and mapping output. Among them, statistics and mapping all need to read the pre-processed residual data. In the whole encoding process, the residual data to be compressed should always be stored in memory and cannot be updated until the encoding output is completed. If all residual data is read at once for compression, a memory that can accommodate all residual data sizes needs to be built, which will cause a great waste of resources. If the residuals are grouped and compressed group by group, only a set of data size memory holds residuals to support the statistical and coded output process.

Grouped compression is also more conducive to parallel computation by the hardware circuit. Figure 19 is a schematic diagram of the data flow of the hardware circuitry executing the entropy coding system in parallel.

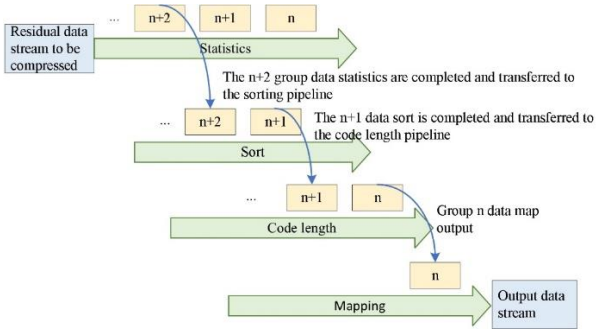
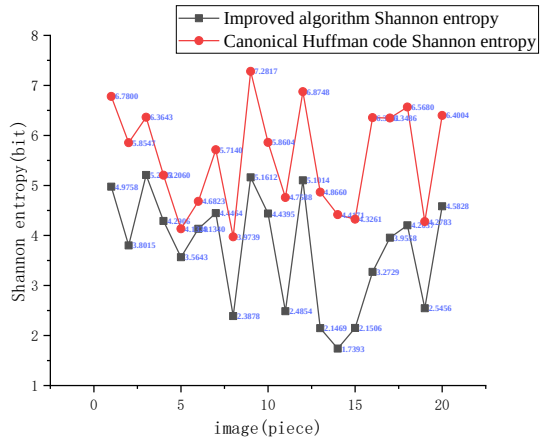


Fig.19 Pipeline

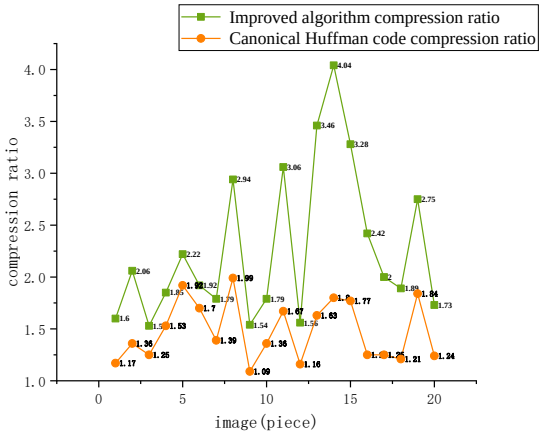
As shown in Figure 20, when the nth group of data is finished encoding for mapping, the nth+1st group of data is calculating the code length of each symbol, while the nth+2nd group of data is in the sorting stage.

4. Results

In order to test the effect of compression ratio improvement after pre-processing and grouping, this thesis tests the landscape and people pictures separately:

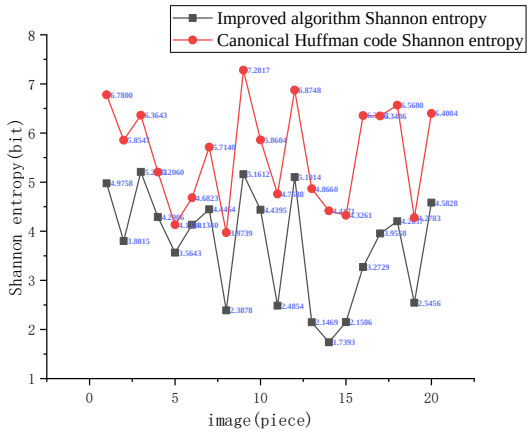


(a) Shannon entropy comparison

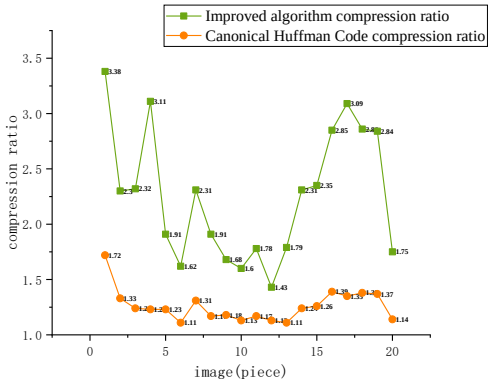


(b) Compression ratio comparison

Fig. 20 Comparison of the compression effect of the landscape map



(a) Shannon entropy comparison



(b) Compression ratio comparison

Fig. 21 Comparison of the compression effect of the figure

From the above figures, it can be seen that the optimized compression algorithm achieves a large improvement in both the theoretical value of compression and the actual compression ratio.

In the coding process, the statistics and sorting modules are the core modules in the whole circuit system, with the most resource utilization as well as time consumption. In order to verify the improvement effect of this system in coding speed, the running time of these two modules in FPGA and the software running time are compared. The hardware circuit designed in this thesis consumes 305 cycles for the first group of data when implementing statistics and sorting, and only 66 cycles for each of the remaining groups due to the pipelined design approach, as shown in Figure 22.

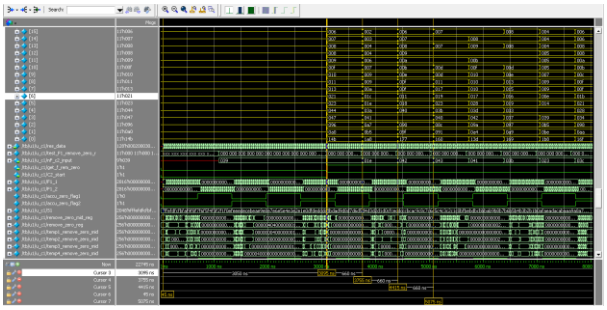


Fig. 22 Counting and sorting the number of cycles consumed by each group of data

The actual maximum frequency of this circuit system after testing is 125 MHZ, that is, the first group of data statistics, sorting a total consumption of 2440 ns, the rest of each group of data statistics, sorting is consumed 528 ns. And the PC side of the same picture of each group of data statistics, sorting the consumption of time range are more than 26000 ns, and even individual after the statistics of zero frequency data less, participate in sorting data more data groups, the consumption time is more than 0.01 s, that is, 10000000 ns. The consumption time is more than 0.01 s, i.e., more than 10000000 ns. Therefore, compared with the serial operation on the software side, the FPGA-based parallel circuit system can greatly improve the coding efficiency.

5. Summary

In this thesis, the HD image compression algorithm is improved by preprocessing to remove correlation, removing zero-frequency nodes, constructing linear frequency tables, and constructing code-length tables without relying on Huffman trees, resulting in a 1.67 times higher compression rate than traditional Huffman coding, and the time complexity of the compression algorithm is greatly reduced to achieve the purpose of real-time data processing. In addition, this thesis designs the corresponding hardware circuit structure to improve the coding rate by using the feature of parallel computing of hardware circuits.

References

[1] Bin W E N, Minghua H E, Minqi H. AVS Exponential-Golomb Code Algorithm[J]. Computer Engineering, 2011, 37(15): 218-220.

[2] Wernik C, Ulacha G, Ieee. Application of adaptive Golomb codes for lossless audio compression[C]. Signal Processing - Algorithms, Architectures, Arrangements, and Applications (SPA), 2018: 203-207.

[3] Hashempour H, Lombardi F. Application of arithmetic coding to compression of VLSI test data[J]. Ieee Transactions on Computers, 2005, 54(9): 1166-1177.

[4] Dyer M, Taubman D, Nooshabadi S, et al. Concurrency techniques for arithmetic coding in JPEG2000[J]. Ieee Transactions on Circuits and Systems I-Regular Papers, 2006, 53(6): 1203-1213.

[5] Dube D, Yokoo H, Ieee. Fast Construction of Almost Optimal Symbol Distributions for Asymmetric Numeral Systems[C]. IEEE International Symposium on Information Theory (ISIT), 2019: 1682-1686.

[6] Wang N, Wang C, Lin S-J. A simplified variant of tabled asymmetric numeral systems with a smaller look-up table[J]. Distributed and Parallel Databases, 2021, 39(3): 711-732.

[7] Patel R, Kumar V, Tyagi V, et al. A Fast and Improved Image Compression Technique Using Huffman Coding[C]. IEEE International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), 2016: 2283-2286.

[8] Kasban H, Hashima S. Adaptive Radiographic Image Compression Technique using Hierarchical Vector Quantization and Huffman Encoding[J]. Journal of Ambient Intelligence and Humanized Computing, 2019, 10(7): 2855-2867.