Journal of Real-Time Image Processing (2024) 21:34
https://doi.org/10.1007/s11554-023-01410-8

RESEARCH

Implementation of JPEG XS entropy encoding and decoding on FPGA

Shuang Tian[1] · Qinghua Song[1] · Jialin He[1] · Yihan Wang[1] · Kai Nie[1] · Gang Du[1] · Ling Bu[1]

Received: 1 September 2023 / Accepted: 28 December 2023 / Published online: 19 February 2024

Abstract
JPEG XS is the latest international standard for shallow compression fields launched by the International Organization for Standardization (ISO). The coding standard was officially released in 2019. The JPEG XS standard can be encoded and decoded on different devices, but there is no research on the implementation of JPEG XS entropy codec on FPGAs. This paper briefly introduces JPEG XS encoding, proposes a modular design scheme of encoder and decoder on FPGA for the entropy encoding and decoding part, and parallelizes the algorithm in JPEG XS coding standard according to the characteristics of FPGA parallelization processing, mainly including low-latency optimization design, storage space optimization design. The optimized scheme in this paper scheme enables encoding speeds of up to 4 coefficients/clock and decoding speeds of up to 2 coefficients/clock, with a 75% reduction in encoding and decoding time. The maximum clock frequency of the entropy encoder is about 222.6 MHz, and the maximum clock frequency of the entropy decoder is about 127 MHz. The design and implementation of the FPGA-based JPEG XS entropy encoding and decoding algorithm is of great significance and provides ideas for the subsequent implementation and optimization of the entire JPEG XS standard on FPGAs. This work is the first in the world to propose the design and implementation of an algorithm that can implement the JPEG XS entropy encoding and decoding process on FPGA. It creates the possibility for the effective application of JPEG XS standard in more media.

Keywords DWT · Entropy codec · FPGA · JPEG XS · Shallow compression

1 Introduction

In 2016, the Joint Photographic Experts Group (JPEG), jointly established by the International Organization for Standardization and the International Telegraph and Telephone Advisory Committee (CCITT) under the International Telecommunication Union (ITU), launched a project for low-complexity, low-latency image compression codecs [1], the JPEG XS international standard, to support fields requiring low latency and high quality, such as autonomous driving, virtual reality (VR), and broadcast television. JPEG XS is the latest international standard in the shallow compression domain, called ISO/IEC 21122: JPEG XS low-latency lightweight image coding system [2, 3], the coding standard was officially released in 2019.

In 1991, the Joint Photographic Experts Group released the still image compression standard JPEG, which has been around for a long time. But is still widely used which is one of the most widely used digital image compression standards in the world. Moreover, the Joint Photographic Experts Group has been promoting the development of image compression standards for nearly 30 years. The group has continuously introduced JPEG-LS, JPEG 2000 [4], JPEG XR [5], and other better codec standards, but JPEG still dominates the market, which shows that the performance of JPEG can meet the demand compared to other more complex codec standards. Low complexity and easy implementation are

✉ Gang Du
dugang@cugb.edu.cn

✉ Ling Bu
lingbu@cugb.edu.cn

Shuang Tian
tshuang2002@126.com

Qinghua Song
song@cugb.edu.cn

Jialin He
745442763@qq.com

Yihan Wang
1286711270@qq.com

Kai Nie
954631557@qq.com

[1] School of Information Engineering, China University of Geosciences, Beijing 100083, China

more important [6, 7]. With the development of technology, in some scenarios, in addition to low complexity, strict bandwidth, and latency requirements need to be met, but low-complexity codecs such as JPEG [8, 9] and JPEG-LS [10, 11] cannot ensure compression to a given target bitrate while keeping latency below a single frame. As a result, new standards need to be developed to meet low-complexity, low-latency coding at high bit rates. JPEG XS is formulated to meet the above requirements, with visually lossless quality, multi-generation robustness, multi-platform operability (CPU, GPU, ASIC, FPGA), low complexity, and low latency [12]. The JPEG XS standard was developed with the possibility of running on a variety of platforms, including FPGA platforms. There have been studies of decoding on GPU [13]. There has also been a lot of research on implementing JPEG2000 on FPGA [14, 15]. A. Legrand et al. also discussed the deployment of JPEG XS in lossless packet networks and proposed an unequal error protection scheme with optimal rate distortion [16].

However, there is currently no research into FPGA implementations of JPEG XS entropy codecs. FPGA is becoming particularly popular as hardware accelerators and is known for their programmability, reconfigurability, and massive parallelism through large numbers of configurable logic blocks (CLBS) [17]. In this paper, the JPEG XS algorithm is implemented on the FPGA, and its encoding and decoding process is optimized in parallel. Four encoding types can be completed when the data is read once, and the encoding speed can reach 4 coefficients/clock, reducing the encoding time by 75%. In the entropy decoding process, parallel decoding optimization is carried out for unary decoding, and the decoding speed reaches 8 bits per clock. Then, the memory structure of the decoding is optimized. The decoding of the magnitude with uncertain delay is fixed to 1 clock cycle and the 4 clock cycles of symbol decoding are reduced to 1 clock cycle, reducing the decoding time by 75%. Finally, by adjusting the decoding timing, the memory space multiplexing optimization is completed, which can be synchronously decoded, so that a large amount of intermediate data does not need to be stored, and part of the storage space can be multiplexed, and finally, the decoding speed reaches 2 coefficients/clock.

2 JPEG XS standard

The encoding and decoding process of JPEG XS is shown in Fig. 1, The main steps of the encoder are image preprocessing, 5/3 DWT(Discrete Wavelet Transform) [18], quantization, entropy coding, and output code stream. The main steps of the decoder are code stream decomposition, entropy decoding, inverse quantization, discrete inverse wavelet transformation, and output image.

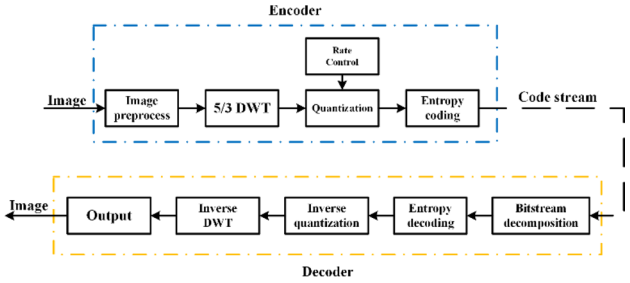

Fig. 1 The figure shows the JPEG XS encoding and decoding step. The image is converted into a code stream after processing and restored to an image after decoding

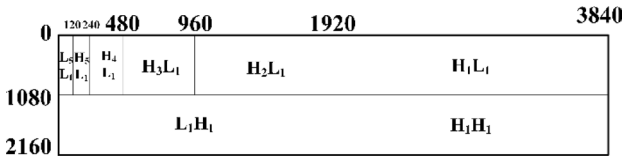

Fig. 2 The image structure after DWT, 3840×2160 image was used in the experiment, 1 vertical and 5 horizontal decomposition level DWT)

2.1 Image preprocess

To obtain more accurate intermediate results in subsequent transformations, after receiving the input image, the bit precision is first extended and the bit depth of the image data is expanded to 20 bits. The subsequent operation is an inter-integer calculation to avoid the loss of information during the calculation process.

Then perform a DC level shift, changing the sample value $I(x, y)$ from $0 \leq I(x, y) \leq 2B$ to a symmetric distribution of zero within $(-2B-1, 2B-1)$.

Since the input image is generally in RGB format, the human eye is more sensitive to the luminance information of the color than the chromaticity information. It is necessary to convert the RGB format into the *YCbCr* format so that the brightness information which the human eye is more sensitive to is concentrated on the Y component, and the insensitive chromaticity information is placed on the *CbCr* component. Then some of the *CbCr* components are removed by downsampling to improve the coding efficiency of JPEG XS.

2.2 Asymmetrical 5/3 DWT

The picture is divided into multiple bands by two-dimensional DWT, each band corresponds to different filter types,

and the calculation method and results are described in the ISO-IEC 21122–1-2019 standard. The divided bands are shown in Fig. 2, the general vertical decomposition level is 1, 2, 3, horizontal decomposition level is 5.

2.3 Quantization

The quantization process is to approximate the input value to the boundary of the quantization interval, mapping an input interval to an integer, converting some signals with similar amplitude into the same value, and reducing the amount of data by losing part of the accuracy to achieve compression. The quantization process that loses information is irreversible, and the information loss in the JPEG XS encoding process mainly comes from this. The quantification methods mentioned in the standard are deadzone quantization [19] and uniform quantization [20].

Deadzone quantization: enter the wavelet coefficients, the amplitude of the wavelet coefficients is shifted right by T bits (the parameter T is the truncation position, which needs to meet the specified constant bit rate, calculated according to the limit of the code rate, using a deadzone quantizer) The wavelet coefficient will be truncated from the lowest bit of the precision of T bits, and the wavelet coefficient in the zero value interval will be quantized to zero for subsequent coding.

2.4 Bitrate control DWT

Bitrate Control [21] is to control the size of the codestream after image compression and allocate bits to each coding unit. The JPEG XS standard allocates the appropriate number of bits to each coding unit according to the set bitrate limit. Then according to all possible quantization parameters, temporary coding, counting the budget bits of each coding unit, and obtaining a comprehensive budget table. In this process, only the budget value size of each coding unit needs to be calculated, without the actual coding process. Finally, the appropriate truncation position is selected through the lookup table so that the encoded bits are closest to, but not larger than the allocated bits.

To ensure the PSRN performance of the encoder and improve the visual quality of the image. The JPEG XS standard adopts different quantization parameters for each DWT subband, which are quantization Q and Precinct refinement R defined in the precinct_header for each precinct. The parameters Gain defined for each band in the two weights_ table G and Priority P decide. The truncation position T calculation formula is shown in (1):

$$T_{p,b} = Q_p - G_b + r \tag{1}$$

When $P_b < R_p$, r is 1, otherwise r is 0. G_b is band gains, and R_p is band priorities, these two parameters are fixed values only related to each sub-band after DWT, which is used to select different quantization parameters for different sub-bands obtained by discrete wavelet transformation.

2.5 Image structure

The image structure is shown in Fig. 3a. After image preprocessing, the image is divided into three components, storing Y, Cb, and C21r information. After 5/3 DWT, each component is divided into eight bands, a total of 24 bands. The ith line of each band is taken out and combined in order as precinct because the image is 3840×2160. So the image is divided into 1080 precincts each precinct has 2 lines, each line, every 4-pixel value is organized into 1 code group. Every 8 code groups are organized into 1 significance group, for the significance encoding in entropy coding. Output significance subpackets to the code stream. In-stream transfer, every eight precincts are grouped into one slice, as shown in Fig. 3b. and each precinct is divided into four packets. Packet 0 contains bands (0, 1, 2), (3, 4, 5), and (6, 7, 8). Packet 1 contains bands (9, 10, 11). Packet 2 contains bands (18, 19, 20). Packet 3 contains bands (21, 22, 23). A packet is the basic unit of transport in a code stream.

2.6 Entropy coding

In JPEG standard, there are two kinds of entropy coding Huffman coding [22] and arithmetic coding [23, 24]. The basic JPEG system provides for Huffman coding.

The following depicts the main steps of entropy coding in JPEG XS, including significance coding, bitplane count coding, magnitude coding, and sign coding.

2.6.1 Significance coding

Generate significance subpackets for significance groups encoding. The significance coding in the JPEG XS standard is similar to the Run Length Code and it is suitable for use in places with a large number of repeated data. For the wavelet transformed data, the coefficient in the high-frequency region will appear with a large number of zero values is very suitable for using Run Length Code, which can be encoded by recording data and the number of data repeats. However, since the number of repeats of the data is uncertain, the encoding length of the run code cannot be determined, resulting in increased coding complexity and uncertain delay.

The significance coding of JPEG XS is improved for these shortcomings. The number of repetitions is fixed, and the input coding group is divided into a significance coding

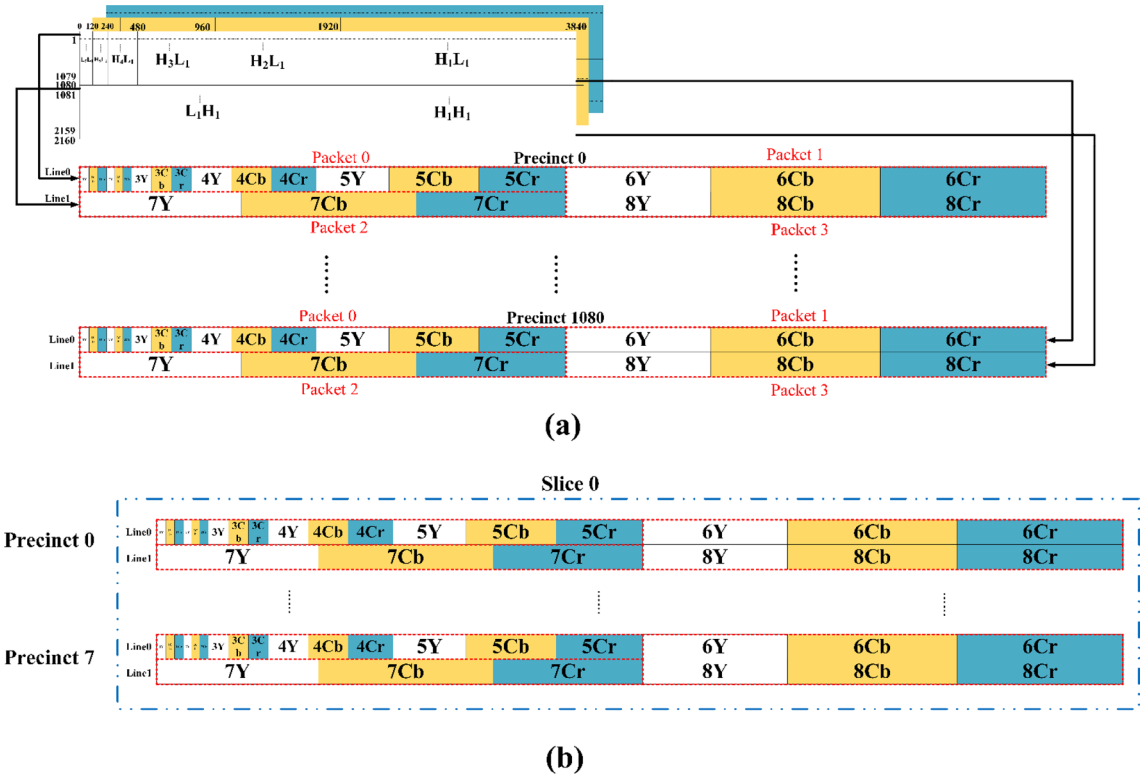

Fig. 3 The image structure of 3840×2160 is used as an example to describe the image structure. **a** The image was divided into 1080 precincts, each precinct containing 24 bands, and each precinct was divided into four packets. **b** The structure of the slice, each precinct is grouped into a slice. The experimental image of 3840×2160 contains a total of 135 slice

group every 8. Meanwhile, the significance coding group is marked with a 1-bit significance sign to determine whether the predicted residuals of the number of bit planes are all zero. If all is zero, the significance code is 1, the significance code is written to the significance subpacket, and the entire significance coding group is skipped in the subsequent bit plane number coding. If it is not all zeros, the significance code is 0, the significance code is written to the significance subpacket, and subsequent encoding is performed.

Significance encoding is an optional encoding method that is used only when the use of distinctive encoding can make the encoding shorter in length. Whether the significance encoding is used is recorded in the header information. If no significance encoding is used, there is no significance subpacket in the bitstream.

2.6.2 Bitplane count encoding

The bit-plane count is encoded to generate a bit-plane count subpackage. There are three ways to encode bitplanes: Raw coding mode, No prediction coding mode, and Vertical prediction coding mode.

Raw coding mode: relatively simple, does not do too much processing. The number of four-bit binary numbers is the plane number of each code group, directly transmitting the raw data of the number of planes. The original mode can ensure that the encoding length has an upper limit when the other two encoding modes get the encoding length exceeds the encoding length of the original mode. The raw mode must be used to encode, facilitate bitrate control, and determine the size of the decoding buffer. If the original mode encoding is used, you need to flag position 1 in the *precinct_header*, otherwise set 0.

No prediction coding mode: the bitplane count M and the truncation position T of the code group are encoded which is calculated in the bitrate control, and the coefficients of the code group will be quantified with T as the quantization parameter before entropy coding. The quantization result is that the T bit accuracy of the code group from the lowest will be truncated. When encoding with No Prediction Coding Mode, only the residual δ of bitplane count M and the truncation position T need to be encoded. Predictive residuals δ calculated as follows

$$\delta = M - T \tag{2}$$

When $T > M$, it means that the bitplane of the current code group is all truncated and quantized to zero. Residual δ at this time is zero. No prediction mode can be regarded

Table 1 No prediction mode residual coding code table

δ	Code
15	1111 1111 1111 1110
……	……
5	1111 10
4	1111 0
3	1110
2	110
1	10
0	0

Table 2 No prediction mode residual coding code table

δ	Code
15	1111 1111 1111 1110
``	```
5	1111 10
4	1111 0
3	1110
2	110
1	10
0	0

as a predictive code with truncated position T as the prediction base. Equation (2) is reversible and can be restored losslessly in decoding.

After calculating the residuals, the unary code [25] is performed. The residuals obtained with no prediction mode are all positive, and the residual values δ can be directly unvaried and encoded. The code table is shown in Table 1.

Vertical prediction coding mode: predicts the current code group using a code group vertically at the same location as the previous precinct, and then encodes the prediction residuals. The calculation of the predicted residual δ is shown in Eq. (3):

$$\delta = \max\left(M_g, T_{p,b}\right) - \max\left(\max\left(M_{g\prime}, T_{p\prime,b}\right), T_{p,b}\right) \tag{3}$$

where M_g is the number of bit planes of the current coding group, $T_{p,b}$ is its corresponding truncation positions, $M_{g'}$ is the bitplane count, $T_{p'}$ and b of the code group at the same position as the previous precinct are the corresponding truncation positions. This formula is also reversible, when $M_g > T_{p,b}$, M_g is mapped to δ according to the formula. When $M_g < T_{p,b}$, the bitplanes of the code group are all truncated and quantized to zero. The vertical differential prediction mode is a prediction code with $max(M'_g, T'_p, b), T_p, b)$ as the prediction base.

The code table is shown in Table 2.

2.6.3 Magnitude encoding

The quantized wavelet coefficient is generated as a data subpacket, and the magnitude plane of the quantized coefficient value is encoded in the magnitude encoding. Magnitude encoding is shown in Fig. 4 after the wavelet coefficient of the coding group is quantized by the truncation position T, T bit accuracy starting from the least significant bit will be truncated, so that the T bitplane of the code group is all zero. Therefore, it is only necessary to encode the bit plane between the bitplane count M of the code group and the truncation position T, starting from the highest non-zero plane with the sign bit removed. The bitplane is written to the code stream in order from high to low. If the coefficients in the code group are all quantized to zero, this code group is not encoded.

2.6.4 Sign encoding

The code stream structure is shown in Fig. 5, first *SOC_marker*, *capabilities_marker*, *picture_header*, *component_table*, *weights_table*, *extension_marker* (*optional*) mainly contains encoding and decoding configuration options such as *Ng*: Number of the coefficients per code group, *Ss*: Number of the code groups per significance group, etc. This is followed by the *packet_header* and *packet_body* of all packets in each precinct, which contains four subpackets encoded by entropy in *packet_body*.

3 FPGA optimization design for JPEG XS entropy encoding

3.1 JPEG XS entropy encoding architecture

The overall architecture of entropy coding is shown in Fig. 6. The top module of the entropy encoder is pack_precinct, which entropically encodes the wavelet coefficient after quantizing each region in the image and outputs the compressed code stream. The entropy coding top module consists of six sub-modules, which are the prec_header module (The zone header writing module, which writes the area header information to the code stream), the sigf_encoding module (The significance coding module, which encodes the prediction residuals of the bitplane count), the raw_mode module (The raw encoding module, which encodes the biplane count using the raw mode), the prediction_encoding module (The predictive Coding Module that encodes the bitplane count using vertical prediction coding mode or no prediction mode), data_encoding module (data coding

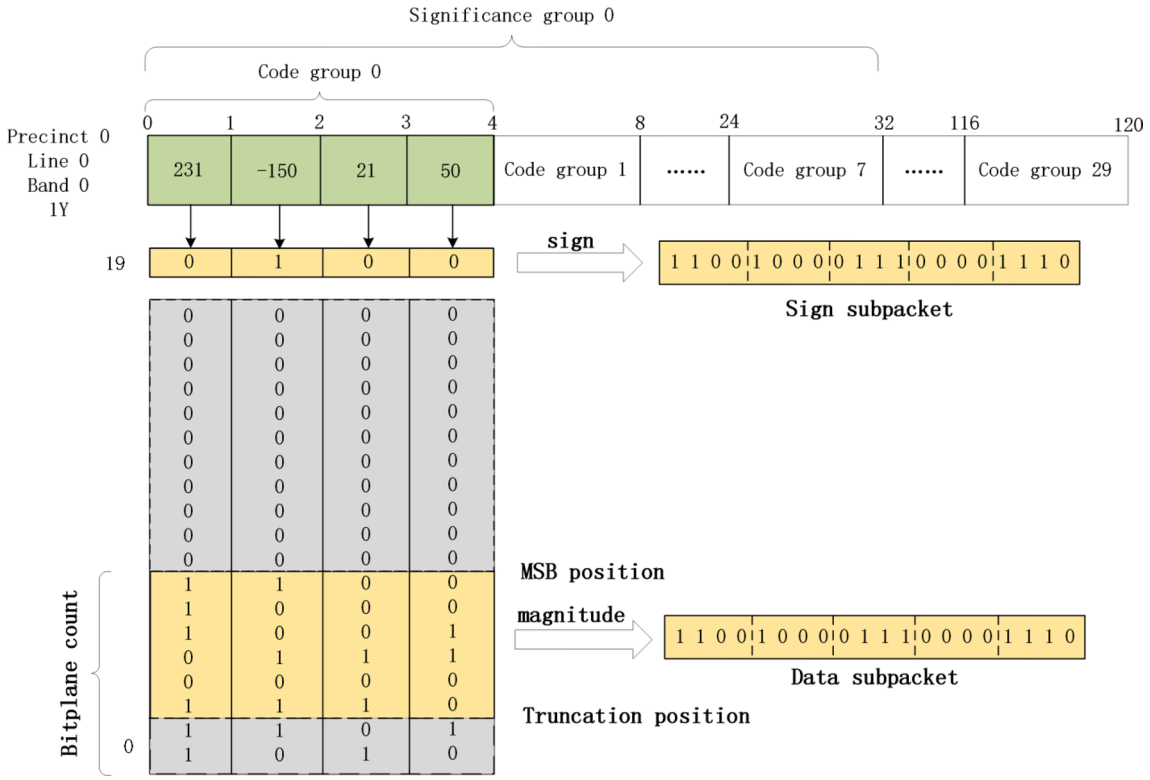

Fig. 4 The structure of code group and encoding methods. The four columns of each band's line form a code group. Each line of four binary numbers is a bitplane and encodes the bitplane information

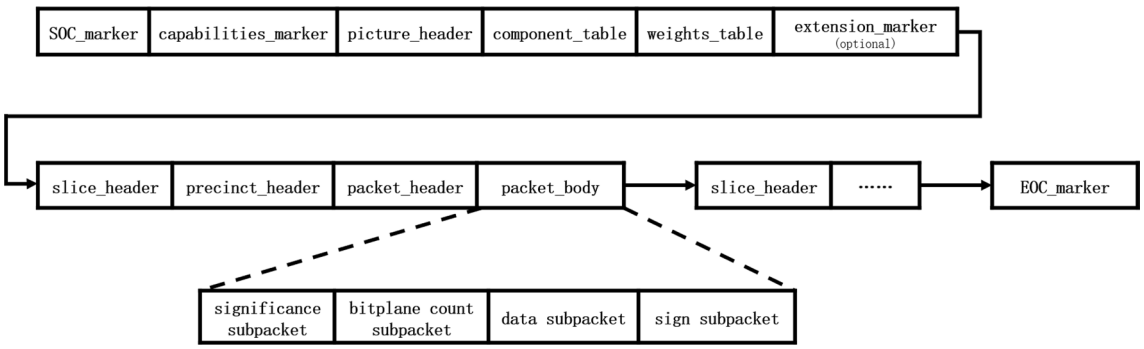

Fig. 5 The structure of codestream, sequence output header Settings file, and image information encoding

module, magnitude encoding and sign encoding of wavelet coefficients after quantization), packet_codestream module (code stream splicing module, code stream splicing of the four received data subpackets, and finally output code stream).

For the image data of a precinct, the entropy coding top module first receives the header information of a precinct. When encoding each packet in the precinct, it needs to receive the header information and encode each sub-band in the order of the sub-bands in the package. Entropy encoding is a parallel encoding for four encoding types. Finally, the output results of each encoding type are spliced with code streams, and 8-bit compressed code streams are output in the order of codestream.

3.2 JPEG XS entropy decoding architecture

The overall architecture of entropy decoding is shown in Fig. 7. The entropy decoding top module decomposes the received code stream. Firstly the module decomposes the header information according to the fixed number of bytes in the precinct header and the packet header, and

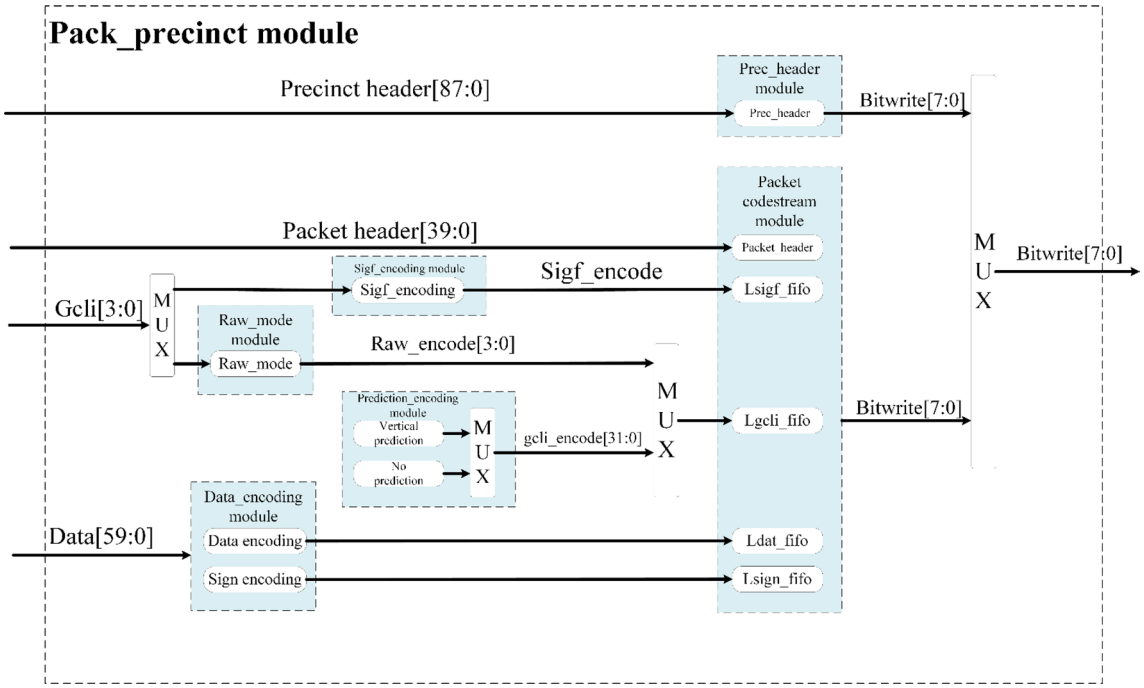

Fig. 6 The architecture of the entropy encoding module, which mainly includes precinct header module, raw coding module, significance coding module, sign coding module, data coding module, and packet codestream module

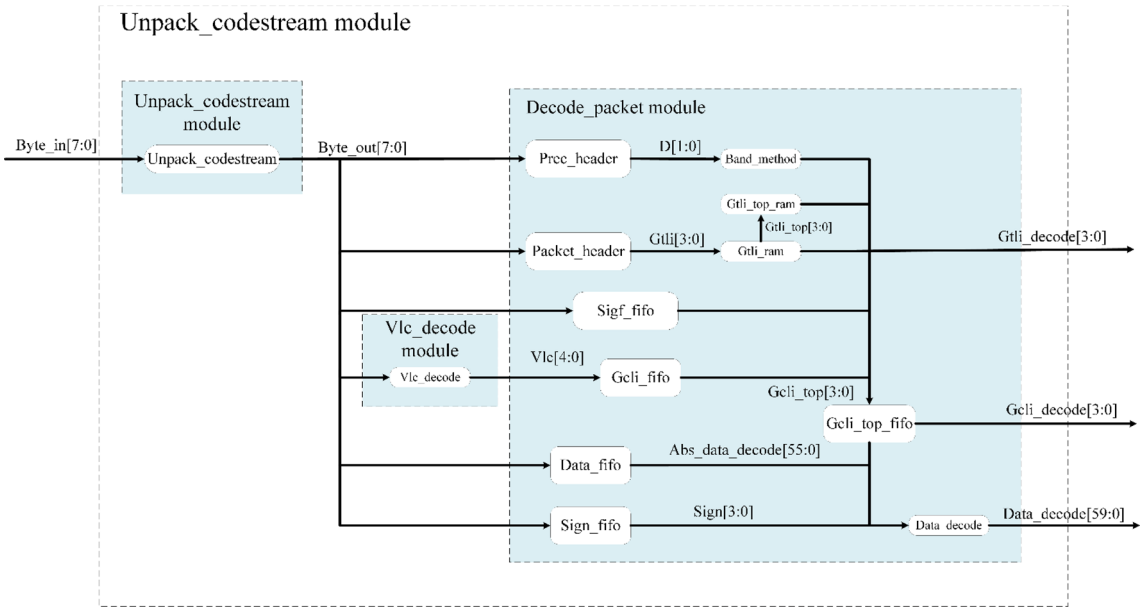

Fig. 7 The architecture of the entropy decoding module, which mainly includes the unpack codestream module, vlc decode module and decode packet module. Sequence output gtli_decode data, gcli_decode data, and Data_decode data

then divides the decoding stream according to the sub-packet information stored in the packet header to obtain significance subpackets, bitplane count subpackets, magnitude subpackets, and sign subpackets, and stores them in FIFO (First In First Out) respectively. Then, in the decode_packet module, the encoding mode and truncation position of each sub-band are restored according to the information stored in the precinct header. Then decoded in the order of the number of bit planes, amplitude, and symbols. Finally, obtain the quantized wavelet coefficient.

3.3 Low latency optimized design

3.3.1 Low latency optimized design

Pipeline operations can be used in significance coding and bitplane count coding if significance subpackets need to be added. The bitplane count of code groups first performs significance coding, receives the bitplane count, outputs the significance flag and the filtered prediction residuals, and there will be no situation where 8 consecutive prediction residuals are zero in the filtered prediction residuals. If there is no significance coding, the prediction residuals are output directly. The module responsible for bitplane count encoding receives the transmitted prediction residuals and encodes the bitplane count. The pipeline structure is shown in Fig. 8a.

Magnitude coding and sign coding are optimized in parallel. Magnitude coding encodes a code group at a time, and sign coding is a coefficient encoding in a code group, so the symbol encoding can judge the magnitude of each coefficient separately after receiving the coefficient of a code group, and then splice the valid sign bits together and output them together into the code stream. Magnitude encoding and sign coding parallelization are shown in Fig. 8b.

After parallelization of the four encoding types in entropy coding, you only need to enter the code group in the package once to get the significance subpacket, bitplane count subpacket, magnitude subpacket, and sign subpacket. But the codestream has a fixed structure, for the encoded subpackets need to wait for the output of the previous subpacket before output, when encoding, only the encoding length of each output of the significance encoding is fixed to 1 bit, and the length of the remaining coding types are not fixed

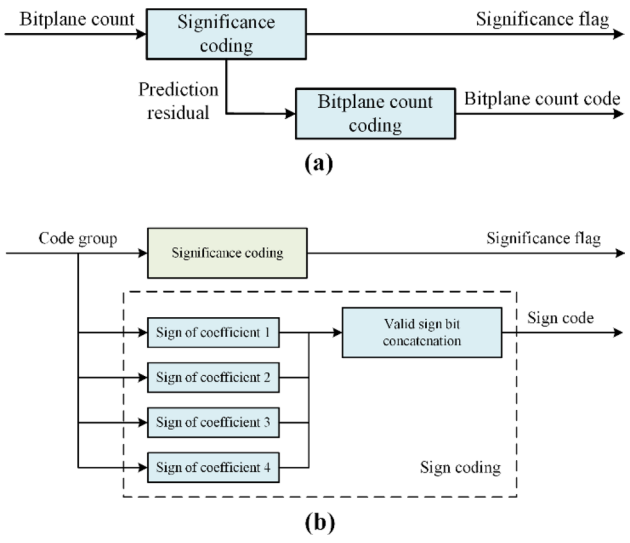

Fig. 8 The structure of the module with low latency optimized designed. **a** The Entropy decoding module. **b** The magnitude and sign parallel coding module

to ensure the validity of the codestream structure. It is also necessary to splice the output results of each encoding type to the codestream, which can ensure byte alignment after splicing, then store the storage space, and finally output each subpacket.

3.3.2 Univariate decoding parallel optimization

If there is a vertical prediction coding mode or no prediction coding mode in the encoding method, then the bitplane count subpacket of the codestream stores the univariate code of the prediction residuals. So the unary decoding needs to be performed first to obtain the prediction residuals. When unary encoding, for the encoded value n, 1 of n bits needs to be output, and then end with 0 of one bit as an interval, when decoding, it is necessary to calculate the number of 1 bits in the received stream, and output the decoded value when identifying 0 bits, and then clear the zero counter by counting the stream, the unary decoding process is shown in Fig. 9a.

JPEG XS compressed code stream has the characteristics of byte alignment, so when reading the codestream, you can read a byte (8 bits) of the codestream each time. If only one bit at a time is received for judgment when decoding, the limitation of decoding speed greatly increases latency and wastes storage space to store univariate encoding. So it is necessary to optimize the decoding method of unary decoding, each time the code stream inputs one byte to decode.

First of all, it is judged that several values can be decoded in the input byte, and it can be judged according to the number of 0 bits in the byte, and a maximum of 8 numbers can be decoded for one byte, that is, 8 zero values. The minimum number of zeros is solved, that is, 8 bits are 1, so a register that can store 8 values is used for buffering, and every 8 values decoded are stored in the storage space, otherwise, wait for the next byte to be decoded until all bitplane count subpackets are entered. The remaining values in the register are stored in the storage space. In the input bytes, the judgment order of unary decoding starts from the highest bit in the byte, each clock cycle recognizes one bit, each time 1 bit is recognized, the counter is added by one, and when 0 bits are recognized, the value of the counter is stored in the register, and the counter is cleared to zero. The univariate decoding parallelization process is shown in Fig. 9b, and the counter cache structure is shown in Fig. 10.

3.3.3 Storage fabric optimization

The four encoding types in JPEG XS entropy encoding are significance encoding, bitplane count encoding, magnitude encoding, and sign encoding, the encoding length of each encoding type output is different, and the corresponding encoding length that needs to be read when decoding each

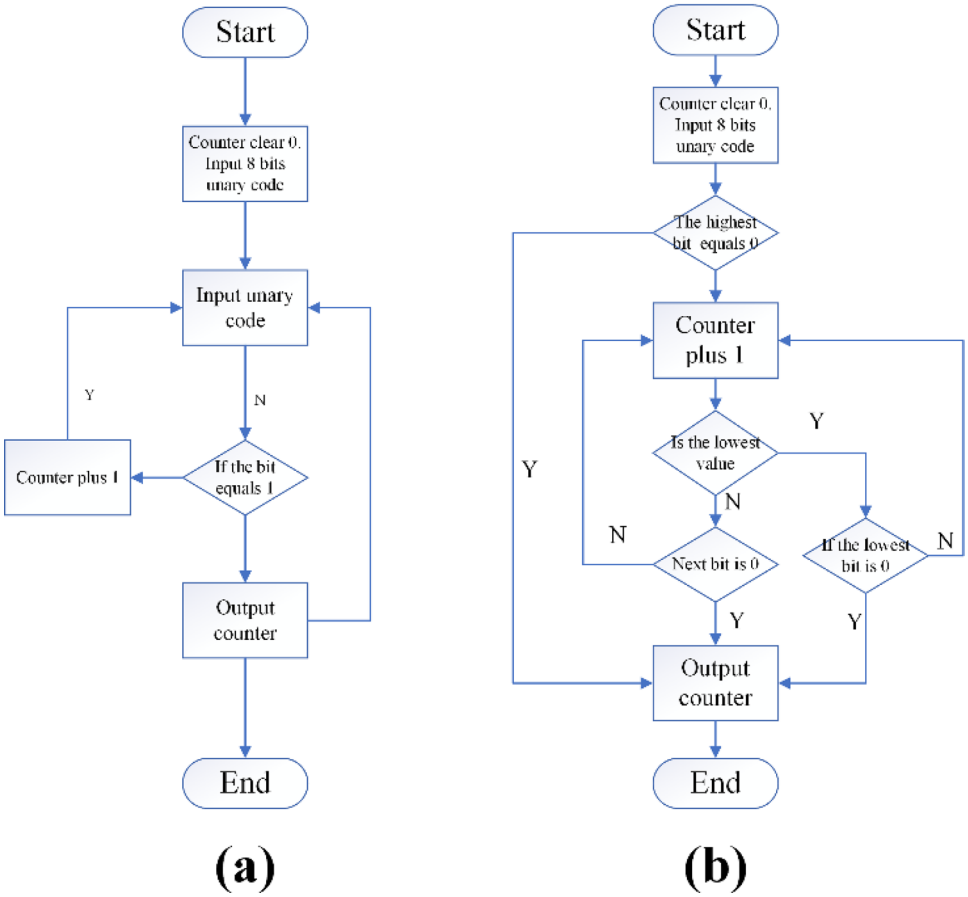

Fig. 9 The univariate decoding flowchart. **a** Original decoding step in JPEG XS, the encoding method is mainly performed sequentially. **b** The decoding module is optimized in parallel

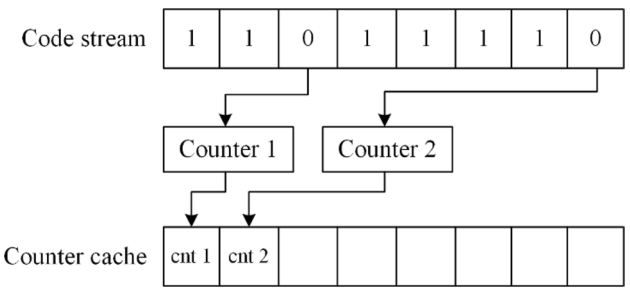

Fig. 10 The cache structure diagram in univariate decoding flowchart parallel optimization algorithm

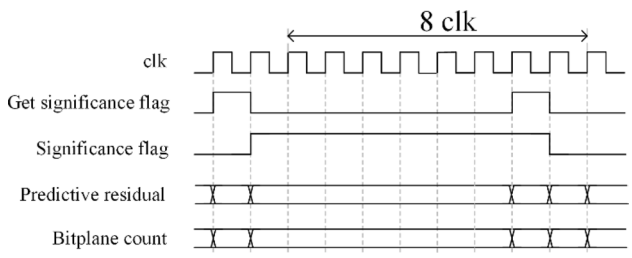

Fig. 11 Significance decoding timing diagrams in the standard. It needs 8 clock cycles to decode a significance code group

subpackage is also different. So the clock cycle required for each part of decoding will be different. Different clock cycles will cause the timing of each part of the decoding to be not aligned, and a part of the delay will be generated in the process of waiting for decoding.

The code stream will be read into the storage space of the corresponding subpacket in the module by bytes, and then taken out of the storage space according to the decoding requirements. When restoring the bitplane count, if significance coding is used, it is necessary to restore together according to the significance flag and the bitplane count, and the significance flag is 1, indicating that there are 8 consecutive prediction residuals of zero. So it takes 8 clock cycles to decode a significance code group when decoding and then predict and decode to obtain bitplane count after obtaining the prediction residuals. The significance of decoding timing is shown in Fig. 11.

When restoring the magnitude of the coding group, it is necessary to take out the data in the magnitude subpacket and arrange them in the order of the bitplane. The data taken out first is the highly significant bit arranged in the code group, until all the data between the most significant bit plane M (bitplane count) and the lowest significant bit plane

T (truncation position) are taken out, a total of 4 (M-T) bits, and then complete the T full zero planes to complete the magnitude restoration. After obtaining the magnitude of the code group, decide whether to take out the sign bits from the sign subpacket according to whether the magnitude of each coefficient is zero, and up to 4 bits need to be taken out of each code group.

When reading data, if the data is obtained from the magnitude subpacket in bitplane units, 4 bits will be read each time. It takes M-T clock cycles to complete the magnitude restoration of the code group, and then the sign bits are restored one by one according to the magnitude of each coefficient. The magnitude of a coding group takes 4 clock cycles to restore all the coefficients. The coefficient reduction timing diagram is shown in Fig. 13a, and the schematic diagram of reading data from the subpacket is shown in Fig. 12a.

However, reading data from the magnitude subpacket by bitplane for restoration results in an uncertain delay. Since the effective plane (M-T) of each code group may be different, the clock cycles used to restore the magnitude are different. At the same time, the result of each restoration of the amplitude is based on the code group, if the coefficients in the code group are filled one by one when decoding the sign subpacket, it is necessary to wait for another 4 cycles for a coded group to be fully output the decoding efficiency is low at this time.

Therefore, it is necessary to optimize the storage structure of magnitude subpackets and sign subpackets, so that the delay in decoding the data is reduced each time the data is read from the subpacket. Since the bit depth of each coding group is the same, there is a maximum value of the encoded encoding length. When storing the subpacket data, the storage space can be set according to the maximum length of the encoding as the bit width, so that the maximum encoding

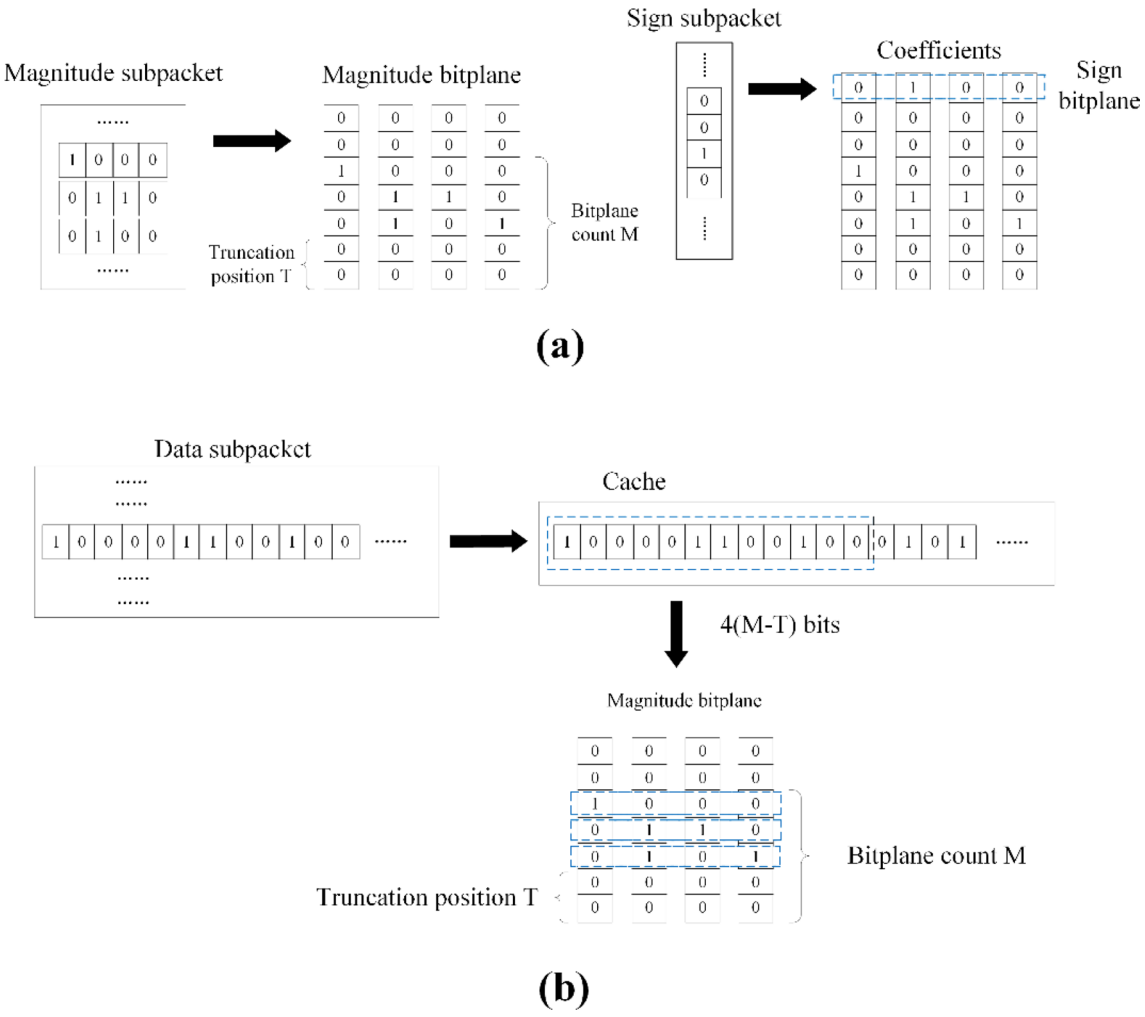

Fig. 12 The schematic diagram of decoding. **a** Reading magnitude and sign subpackets in JPEG XS, data is mainly read using a sequential structure. **b** Storage structure optimization for parallel processing

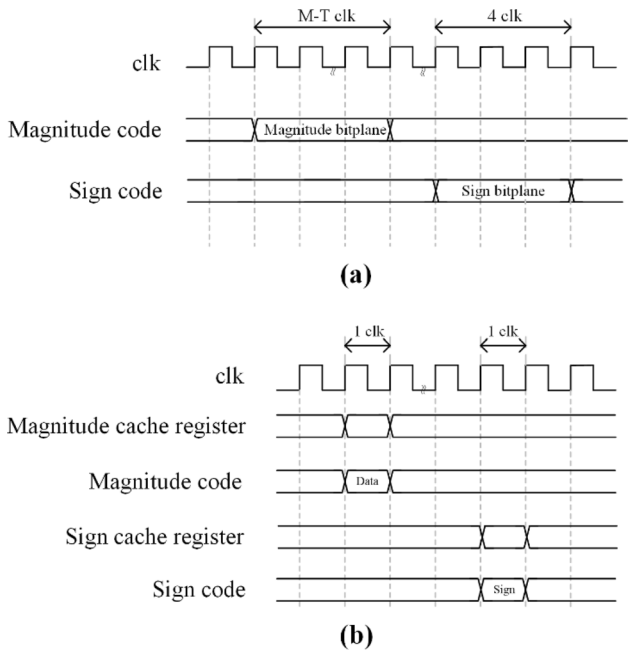

Fig. 13 The coefficient reduction timing diagram. **a** In JPEG XS, it needs M-T clock cycles to restore the magnitude of the code group, 4 clk to restore the sign bitplane, and some delay between the two parts of the date. **b** The optimized method only needs 1 clock cycle to restore the data of the code group, 1 clk to restore the sign, and some delay between the two parts of data

length bit width data is read from the storage space every time. The data read from the storage space is put into the cache register when decoding, and then the corresponding bit width data is obtained from the register according to the required number of bits. Taking magnitude decoding as an example, the required encoding length is calculated according to the bitplane count M and the truncation position T, that is, 4 (M-T) bits in the cache are obtained for magnitude decoding, and the storage structure optimization is shown in Fig. 12b.

The magnitude decoding process reads data from the magnitude subpacket with the maximum encoding length and stores it in the cache register first. The bit depth of the encoding group in this article is set to 15 bits, which is not less than the maximum encoding length so that the output bit width of the memory space is 64, which reduces the time delay caused by a single bitplane reduction eliminating the uncertainty of the magnitude decoding delay. Reorder the required encoding into the magnitude bitplane in one clock cycle to complete the magnitude decoding of the code group. Similarly, for 4-bit symbols in the code group, set the output bit width of the storage space to 4, and the sign decoding takes the sign from the corresponding cache register as a plane according to the number of non-zero values in the magnitude, which also only takes one cycle to complete the decoding, reducing the decoding time by 75% compared to the decoding of a single sign bit. The optimized coefficient reduction timing is shown in Fig. 13b.

3.4 Optimized design for storage space reuse

According to the JPEG XS standard, a large number of intermediate variables will be generated in the decoding process. First of all, four subpackets need to be stored after the codestream decomposition, significance subpackets, bitplane number subpackets, magnitude subpackets, and sign subpackets. The storage space size of each subpackage has been determined in the standard, where the bitplane count subpackets can be decoded in real-time after using parallelized unary decoding and directly decoded into prediction residuals without storage. So there is no need to store bitplane subpackets, but storage space is required to store the predicted residuals of a packet. When decoding the predicted

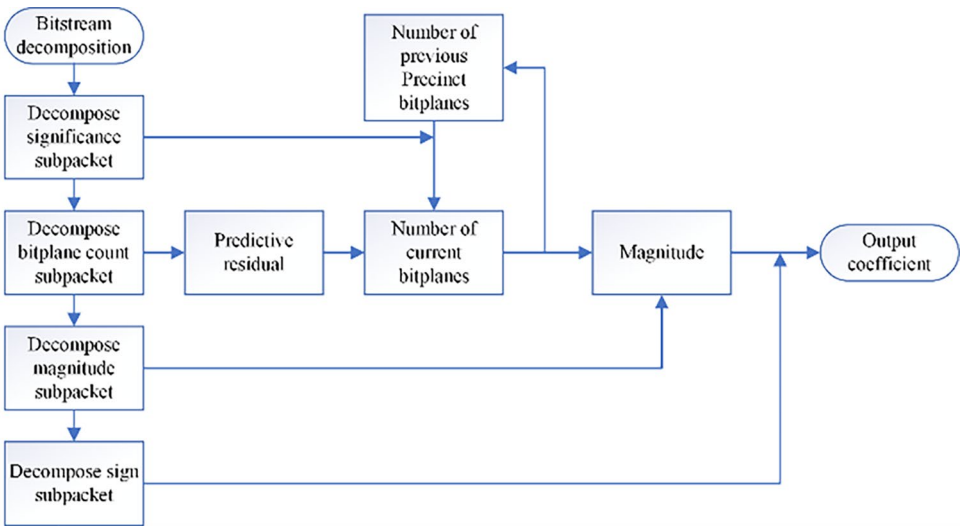

Fig. 14 Schematic diagram of entropy decoding storage space in the JPEG XS standard

residuals, the bitplane count of each code group in the previous region is required, and the bitplane count of the current region needs to be stored after decoding. Then the magnitude data of a packet will be generated when the magnitude is decoded. Finally, the coefficient is sent out by sign decoding. The entropy decoding storage space is shown in Fig. 14.

If the entropy decoding step in the JPEG XS standard is carried out step by step, multiple storage spaces will be required to store intermediate variables. Wait for the code stream of a package to be decomposed before synchronous decoding, and read data from the storage space according to the decoding needs. Read data from the significance subpacket FIFO, the predicted residuals FIFO, and the previous bitplane count FIFO, use the logic circuit to obtain the bitplane count of the current region, read the data from the magnitude subpacket FIFO according to the bitplane count in the current area, obtain the magnitude of the code group in one clock cycle. Finally, read the data from the sign subpacket FIFO according to whether the magnitude is zero and send the data after reducing the complete coding group coefficient in the same clock cycle, thereby reducing the magnitude storage space. At the same time, because of synchronous decoding, the storage of the current bitplane count can be reduced. After completing the magnitude decoding of a code group, the current bitplane count can be directly written to the previous area bitplane count FIFO, which is convenient for decoding the bitplane count of the next area, to realize the reuse of storage space. Figure 15a shows the optimized storage space multiplexing, and the decoding sequence of storage space multiplexing is shown in Fig. 15b.

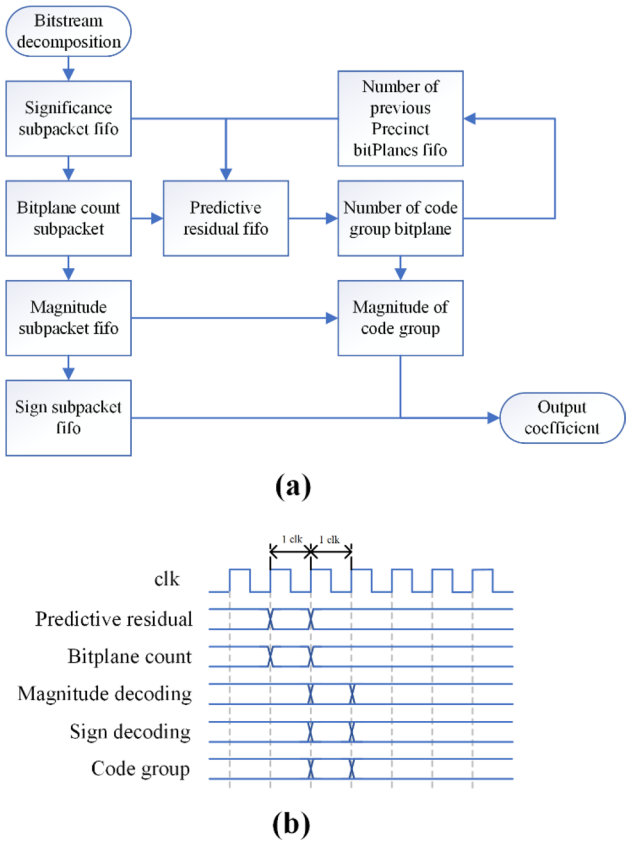

Fig. 15 The schematic diagram of entropy decoding after storage space reused design. **a** The optimized storage space after entropy decoding storage space multiplexing. **b** Optimized storage space multiplexing decoding timing diagram, The decoding of predictive residuals, bitplane count, magnitude, and sign only requires two clock cycles

4 Simulation verification

4.1 Entropy encoding

This section mainly simulates the significance coding module, prediction coding module, data coding module, and code stream splicing module.

4.1.1 Significance coding module

Part of the data simulation diagram of the significance coding module is shown in Fig. 16, when there is a significance code, that is, the *sigf_flag* is 1, the module stores the received predicted residual GCLI in the *gcli_fliter_buff* register, and counts at the same time as received. When the counter *sigf_group_cnt* is 8, the coding *sigf_encode* is output in the next cycle. If the registers are all zero, *sigf_encode* output 1 as shown in Fig. 16 circle A does not output the value in the register. When the data in Fig. 16 circle B is not completely zero, *sigf_encode* output 0 and output the data in the register one by one. As shown in Fig. 16 circle C, if there is no significance coding, the values in the registers are output one by one. After verification, the results of the module are correct, indicating that the module is functional.

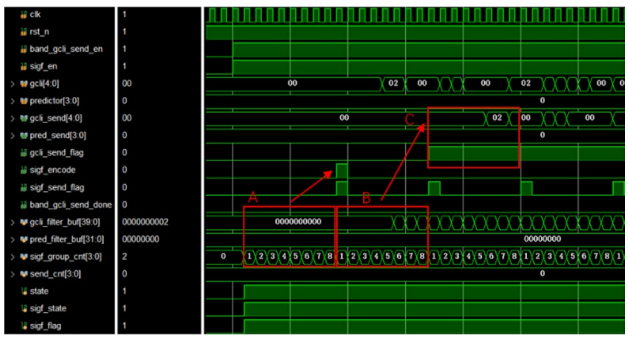

Fig. 16 Significance coding module simulation diagram. In circle A, gcli_fliter_buff are all zero, sigf_encode output 1. In circle B, the data is not completely zero, sigf_encode output 0 and output the data in the registers one by one. In circle C, there is no significance coding, the values in the registers are output one by one

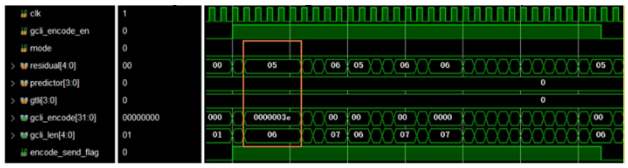

Fig. 17 Prediction code module data simulation diagram, in which the code mode = 1'h0, prediction residual = 5'h05, predictor = 4'h0, gcli_encode = 32'h0000003e, gcli_len = 5'h06

4.1.2 Predictive coding module

Part of the data simulation diagram of the predictive coding module is shown in Fig. 17, the *code mode = 1'h0* in the circle in the figure, that is, there is no prediction coding mode. The *prediction residual = 5'h05*, the prediction base value is *predictor = 4'h0*, and the *gcli_encode = 32'h0000003e* after code table encoding is obtained. The effective coding length gcli_len = 5'h06. After verification, the results of the module are correct, indicating that the module is functional and the module encoding can be completed in one clock cycle.

4.1.3 Data coding module

Part of the data simulation diagram of the data coding module is shown in Fig. 18, taking the value shown in the figure as an example. The input coding group coefficient value is data = 60'h800e003e004400d, where every 15 bits represents a coefficient value. Firstly, separate the magnitude bitplane of the code group, obtain 14 magnitude bitplanes, re-splice in the order of bitplanes from high to low, according to *gtli = 4'h0, gcli = 4'h4*, intercept the lower four-bit plane and store it in the *data_encode* register to complete the magnitude encoding as shown in circle A in the figure, and the effective encoding length is obtained *data_len = 6'h10*. The sign encoding is judged bit by bit in the sign_code according to the magnitude, as shown in circle B in the figure. The final sign code is *sign_encode = 4'hb*. After verification, the results of the module are correct, indicating that the module is functional and the module encoding can be completed in one clock cycle.

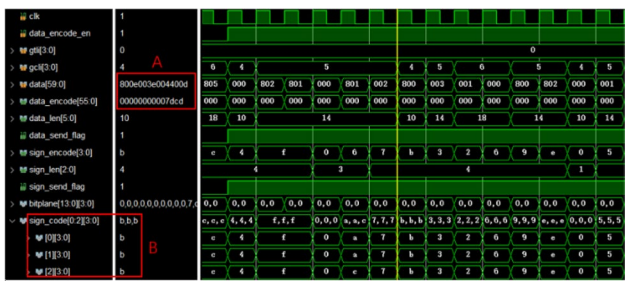

Fig. 18 Data encoding module simulation diagram. In circle A, because of the gtli = 4'h0, gcli = 4'h4, intercept the lower four-bit plane and store it in the data_encode. In circle B, the final sign code is sign_encode = 4'hb

4.1.4 Codestream splicing module

Part of the data simulation diagram of the code stream splicing module is shown in Fig. 19, taking the magnitude coding as an example to carry out the code stream splicing. For the input magnitude coding *data_encode* in the figure and its effective coding length *data_len*, respectively stored in the register *data_temp_w* and *data_temp_w_len*. When the next magnitude code arrives, determine whether the *data_len + data_temp_w_len* is longer than 64 bits, if it is longer than it, use the high significance bit of *data_encode* to complete the register *data_temp_w* and write it to the *data_temp_w*, that is, the FIFO, and the remaining code is written to the *data_temp_w*. The remaining effective code length is recorded; If it is not greater, only the encoding is written to the *data_temp_w*. The effective encoding length is updated, and the remaining subpackets are spliced in the same way. After verification, the results of the module are correct, indicating that the module is functional.

4.1.5 JPEG XS entropy encoder

To determine the accuracy of the JPEG XS entropy encoder results, the JPEG XS reference software provided in Part 5 of the JPEG XS standard is used to write the data required for entropy encoding and the entropy encoding compression results to the txt file. The text file of the input data and compression results pass the vivado's system function *$readmemh* reads into vivado. The required data is input into the JPEG XS entropy coding top module of the test platform, and the code stream output of the JPEG XS entropy coding top module is compared with the compression result of the reference software. If the output code stream is different from the result, the simulation stops, otherwise the simulation continues, to ensure that the function of the JPEG XS entropy encoder designed in this paper is correct. The comparison chart of entropy encoding results is shown in Fig. 20,

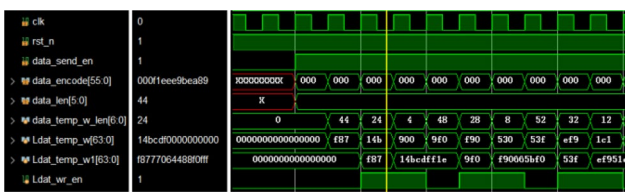

Fig. 19 Code flow splicing module simulation diagram, taking the magnitude coding as an example to carry out the code stream splicing

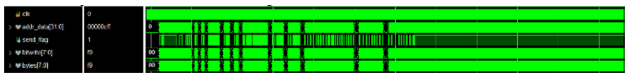

Fig. 20 Entropy encoding results comparison diagram, Can see that the value of the codestream output by the JPEG XS entropy encoding top module is the same as the value of the codestream output by the reference software

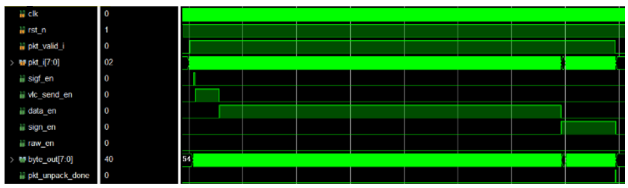

Fig. 21 The code stream decomposition module data simulation diagram, decomposes the codestream continuously input from the port pki_i and controls the enabled signal to write the codestream into the FIFO

where *bit write* is the codestream output by the JPEG XS entropy encoding top module, and bytes is the codestream output by the reference software. The comparison results are correct, so the JPEG XS entropy encoder designed in this paper functions normally and meets the requirements.

4.2 Entropy decoding

In this section, the code stream decomposition module, univariate decoding module, and packet decoding module are mainly simulated and tested. Finally, the quantized wavelet coefficients of the final output are guaranteed to be correct.

4.2.1 Codestream decomposition module

Part of the data simulation diagram of the code stream decomposition module is shown in Fig. 21, which decomposes the codestream continuously input from the port *pki_i* and controls the enable signal to write the codestream into the FIFO in the sub-module according to the number of subpacket bytes in the precinct header information. Wherein *sigf_en* is the code stream for writing important subpackets, *vlc_send_en* is the code stream for writing bit-plane count subpackets, *data_en* is the code stream for writing amplitude subpackets, and *sign_en* is the code stream for writing symbol subpackets.

4.2.2 Univariate decoding module

Part of the data simulation diagram of the unary decoding module is shown in Fig. 22. The input data where the cursor is located is unary decoded, the input byte is *vlc_byte = 8'b01111110*. The number of data that can be solved

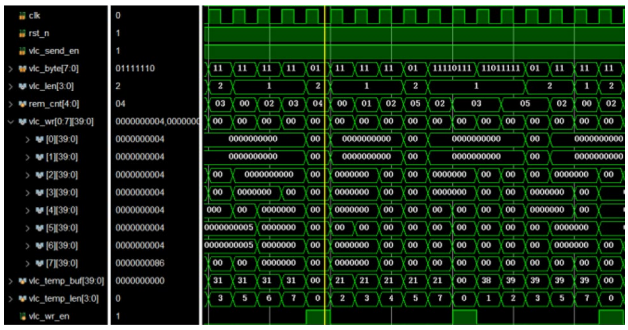

Fig. 22 Univariate decoding module data simulation diagram, in which the vlc_byte = 8'b01111110, the vlc_temp_len in the register is written to the FIFO when vlc_temp_len = 8

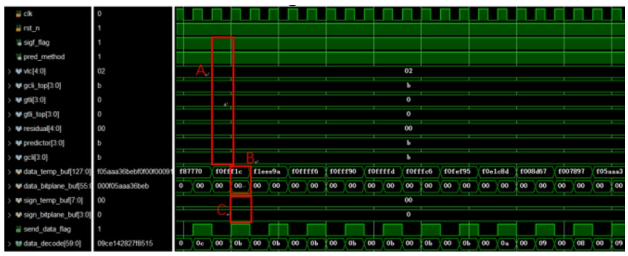

Fig. 23 Packet decoding module data simulation diagram. In Circle A, decode the bitplane count residual of the bitplane count to obtain the bitplane gcli of the code group. In circle B the bitplane count gcli is 11, and the truncated position gtli is 0

is calculated *vlc_len = 2*, and then judging bit by bit from the highest bit, whenever there is 0, the accumulated value will be written to the *vlc_wr*, the highest bit is 0 then output the value in the counter *rem_cnt*, and the lowest bit in the *vlc_wr* will be written to the register *vlc_temp_buf* according to the number of *vlc_len*. The registers are recorded in the *vlc_temp_len* to store the number of data, and the value in the register is written to the FIFO when *vlc_temp_len = 8*. A correct decoding result indicates that the module is functioning properly.

4.2.3 Packet decoding module

The data simulation diagram of the packet decoding module is shown in Fig. 23. Circle A in the figure is decoded to the bitplane count residual of the bitplane count to obtain the bitplane *gcli* of the code group. The result is obtained in one cycle through the combination logic circuit. When the magnitude is decoded, it will be taken out of the RAM in units of 64 bits and stored in *data_temp_buf* registers. The required number of bits will be obtained from the register and restored to the magnitude bitplane in the *data_bitplane_buf*, which can avoid the time delay caused by the single bit plane to restore the magnitude bitplane. As shown in

Fig. 23 circle B, the bitplane count *gcli* is 11, and the truncated position *gtli* is 0 if a single bit plane is restored one by one it takes 11 clock cycles. By increasing the register cache, the required 44 bits are taken from the register and restored to the magnitude bitplane in one clock cycle. Circle C in Fig. 23 is the symbol decoding, the same data firstly stored in the register *sign_temp_buf*. According to whether the magnitude of the coefficient is zero, restore the symbol bitplane in the *sign_bitplane_buf*, avoiding the delay caused by the decoding of a single sign bit. The result is obtained in one clock cycle, and finally, the symbol bit plane and the amplitude bit plane are concatenated in data_decode to output the coefficient values of the code set. After verification, the results of the module are correct, indicating that the module is functional.

4.2.4 JPEG XS entropy decoding module

To determine the accuracy of the JPEG XS entropy encoder results, the same JPEG-SX reference software is used to get the codestream and entropy decoding to decoding result into the txt file, read the txt file into vivado and input the codestream into the JPEG XS entropy decoding top module in the test platform. The wavelet coefficient output quantized by the JPEG XS entropy decoding top-level module is compared with the entropy decoding results of the reference software to ensure the functionality of the JPEG XS entropy decoder designed in this paper is correct. The comparison chart of entropy encoding results is shown in Fig. 24, where data is the code stream output by the JPEG XS entropy decoding top module, and the *data_comp* is the entropy decoding result output by the reference software. The comparison results are correct, so the JPEG XS entropy decoder designed in this paper functions normally and meets the requirements.

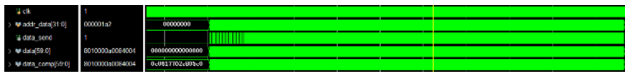

Fig. 24 The entropy decoding results comparison diagram. Can see that the value of the data_comp is the same as the value of the data

Table 3 JPEG XS entropy decoder timing report

Setup	Time
Worst negative slack (WNS)	5.508 ns
Total negative slack (TNS)	0.000 ns
Number of failing endpoints	0
Total number of endpoints	2918
Hold	**Time**
Worst negative slack (WNS)	0.022 ns
Total negative slack (TNS)	0.000 ns
Number of failing endpoints	0
Total number of endpoints	2918
Pulse width	**Time**
Worst negative slack (WNS)	4.458 ns
Total negative slack (TNS)	0.000 ns
Number of failing endpoints	0
Total number of endpoints	1081

4.3 FPGA timing verification

Timing simulation is conducted based on the module function to fulfill the requirements of layout and routing. This simulation encompasses crucial information such as device delay and line delay, mimicking the operational process of a real device and reflecting its actual working state. To ensure that the JPEG XS entropy encoder and entropy decoder designed in this paper meet the timing requirements, static timing analysis was performed on the xczu7egg-ffvc1156-2-i device using a clock frequency of 100 MHz.

4.3.1 Entropy encoding

Table 3 shows the timing report of the JPEG XS entropy encoder, where the worst negative timing margin (WNS) in the setup time (Setup) and the worst hold time margin (WHS) in the hold time (Hold) are both positive, indicating that the entropy encoder meets the timing requirements at 100 MHz. The maximum operating frequency of the clock can be calculated according to the formula Fmax = 1/(T-WNS), which is about 222.6 MHz.

Table 4 shows the resource occupation of JPEG XS entropy encoders, and it can be seen that the occupancy rate of logical resources is low, with LUT, FF, and BRAM accounting for only 0.84%, 0.23%, and 3.53%, respectively.

4.3.2 Entropy decoding

Table 5 shows the timing report of the JPEG XS entropy decoder, where WNS in set-up time and WNS in hold time are both positive, indicating that the entropy decoder meets the timing requirements at 100 MHz. The maximum operating frequency of the clock can be calculated according to the formula to be about 127 MHz.

Table 6 shows the resource usage of JPEG XS entropy decoders, in which the utilization rate of logical resources is low, with LUT, FF, and BRAM accounting for only 1.36%, 0.25%, and 3.53%, respectively.

Table 4 JPEG XS entropy encoder resource usage

Resource	Utilization	Available	Utilization %
LUT	1943	230,400	0.84
LUTRAM	4	101,760	0.00
FF	1052	460,800	0.23
BRAM	11	312	3.53
IO	275	360	76.39
BUFG	1	544	0.18

Table 6 The JPEG XS entropy decoder resource consumption

Resource	Utilization	Available	Utilization %
LUT	3144	230,400	1.36
LUTRAM	5	101,760	0.00
FF	1141	460,800	0.25
BRAM	11	312	3.53
IO	109	360	30.28
BUFG	1	544	0.18

5 Conclusion

Entropy encoding and decoding is an important step in JPEG XS encoding and decoding, this paper deeply studies JPEG XS coding standards and the entropy encoding process of each part of the modular design. According to the FPGA parallelization processing characteristics of the codec parallel optimization, storage structure optimization, and storage space multiplexing optimization. The parallel optimization of the four encoding types is completed for the entropy coding process so that the coding speed can reach 4 coefficients/clock, which reduces the encoding time by 75% compared to the serial 1 coefficients/clock. In the entropy decoding process, parallel decoding optimization is carried out for unary decoding, and the decoding speed reaches 8 bits/clock, which exceeds the 1 bit/clock of serial decoding. The memory structure of decoding is optimized, and the decoding of the amplitude value with uncertain delay is fixed to 1 clock cycle, and the 4 clock cycles of symbol decoding are reduced to 1 clock cycle, reducing the decoding time by 75%. By adjusting the decoding timing to complete the optimization of storage space multiplexing, synchronous decoding can be carried out, so that a large amount of intermediate data does not need to be stored and part of the storage space can be multiplexed. To reduce latency and resource occupation. Finally, the results show that the decoding speed reaches 2 coefficients/clock, and the optimization reduces the delay and resource consumption, making it more suitable for running on FPGA.

Table 5 JPEG XS entropy decoder timing report

Setup	Time
Worst negative slack (WNS)	2.130 ns
Total negative slack (TNS)	0.000 ns
Number of failing endpoints	0
Total number of endpoints	2748
Hold	**Time**
Worst negative slack (WNS)	0.026 ns
Total negative slack (TNS)	0.000 ns
Number of failing endpoints	0
Total number of endpoints	2748
Pulse width	**Time**
Worst negative slack (WNS)	4.458 ns
Total negative slack (TNS)	0.000 ns
Number of failing endpoints	0
Total number of endpoints	1173

Acknowledgements This work is supported by the Beijing College Students Innovation and Entrepreneurship Training Program (S202211415069), the National Innovation and Entrepreneurship Training Program for College Students (202211415086, 202211415031).

Author contributions Shuang Tian, Jialin He, Kai Nie and Yihan Wang wrote the main manuscript text. Shuang Tian prepared figures 1-7. Yihan Wang prepared figures 8-15. Jianlin He prepared figures 16-24. All authors reviewed the manuscript.

Data availability Due to the data confidentiality agreement of the laboratory and institution where the author of this paper works, the data in this paper cannot be disclosed for the time being.

Declarations

Conflict of interest The authors declare no competing interests.

References

1. Richter, T., Keinert, J., Foessel, S., Descampe, A., Rouvroy, G., Lorent, J.B.: JPEG XS—A high-quality mezzanine image codec for video over IP. SMPTE Motion Imag. J. **127**(9), 39–49 (2018). https://doi.org/10.5594/JMI.2018.2862098
2. Peng, W.H., Walls, F.G., Cohen, R.A., Xu, J., Ostermann, J., MacInnis, A., Lin, T.: Overview of screen content video coding:

Technologies, standards, and beyond. IEEE J. Emerg. Sel. Top. Circuits Syst. **6**(4), 393–408 (2016). https://doi.org/10.1109/JETCAS.2016.2608971
3. Walls, F.G., MacInnis, A.S.: VESA display stream compression for television and cinema applications. IEEE J. Emerg. Sel. Top. Circuits Syst. **6**(4), 460–470 (2016). https://doi.org/10.1109/JETCAS.2016.2602009
4. ISO, I. S., & JTC.: I. Information technology-JPEG 2000 image coding system-Part 1: Core coding system. ISO/IEC IS 15444–1. (2000)
5. Jadhav, S.S., Jadhav, S.K.: JPEG XR an image coding standard. Int. J. Comput. Electr. Eng. **4**(2), 137 (2012)
6. Ahmad, J., Raza, K., Ebrahim, M., & Talha, U.: FPGA based implementation of baseline JPEG decoder. In Proceedings of the 7th International Conference on Frontiers of Information Technology (pp. 1–6). (2009)
7. Shan, Y., Chen, X., Qiu, C., & Zhang, Y.: Implementation of Fast Huffman Encoding Based on FPGA. In Journal of Physics: Conference Series (Vol. 2189, No. 1, p. 012021). IOP Publishing. (2022)
8. ISO, I. S., & JTC.: I. Information technology-digital compression and coding of continuous-note still images: requirements and guidelines. ISO/IEC IS-10918–1. (1994)
9. Wallace, G.K.: The JPEG still picture compression standard. Commun. ACM **34**(4), 30–44 (1991)
10. Weinberger, M.J., Seroussi, G., Sapiro, G.: The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS. IEEE Trans. Image Process. **9**(8), 1309–1324 (2000)
11. ISO, I. S., & JTC.: I. Lossless and near-lossless coding of continuous tone still images (JPEG-LS). FCD 14495. (1997)
12. Descampe, A., Richter, T., Ebrahimi, T., Foessel, S., Keinert, J., Bruylants, T.: … & Rouvroy, G: JPEG XS—A new standard for visually lossless low-latency lightweight image coding. Proc. IEEE **109**(9), 1559–1577 (2021). https://doi.org/10.1109/JPROC.2021.3080916
13. Bruns, V., Richter, T., Ahmed, B., Keinert, J., & Fößel, S.: Decoding jpeg xs on a gpu. In 2018 Picture Coding Symposium (PCS) (pp. 111–115). IEEE. (2018). https://doi.org/10.1109/PCS.2018.8456310
14. Kumar, N. R., Xiang, W., & Wang, Y.: An FPGA-based fast two-symbol processing architecture for JPEG 2000 arithmetic coding. In 2010 IEEE International Conference on Acoustics, Speech and Signal Processing (pp. 1282–1285). IEEE. (2010). https://doi.org/10.1109/ICASSP.2010.5495418
15. Gangadhar, M., Bhatia, D.: FPGA based EBCOT architecture for JPEG 2000. Microprocess. Microsyst. **29**(8–9), 363–373 (2005). https://doi.org/10.1016/j.micpro.2004.10.006
16. Legrand, A., Macq, B., & De Vleeschouwer, C.: Forward error correction applied to JPEG XS codestreams. In 2022 IEEE International Conference on Image Processing (ICIP) (pp. 3723–3727). IEEE. (2022). https://doi.org/10.1109/ICIP46576.2022.9897287
17. Ravi, M., Sewa, A., Shashidhar, T.G., Sanagapati, S.S.S.: FPGA as a hardware accelerator for computation intensive maximum likelihood expectation maximization medical image reconstruction algorithm. IEEE Access **7**, 111727–111735 (2019). https://doi.org/10.1109/ACCESS.2019.2932647
18. Acharya, T., & Tsai, P. S.: JPEG2000 standard for image compression: concepts, algorithms and VLSI architectures. (2004)
19. Gish, H., Pierce, J.: Asymptotically efficient quantizing. IEEE Trans. Inf. Theory **14**(5), 676–683 (1968). https://doi.org/10.1109/TIT.1968.1054193
20. Richter, T.: Spatial constant quantization in JPEG XR is nearly optimal. In 2010 Data Compression Conference (pp. 79–88). IEEE. (2010). https://doi.org/10.1109/DCC.2010.14
21. ISO, I. S., & JTC.: I. Information technology-JPEG 2000 image coding system-Part 1: Core coding system. ISO/IEC 15444–1. (2001)
22. Bailey, D., Cressa, M., Fandrianto, J., Neubauer, D., Rainnie, H.K., Wang, C.S.: Programmable vision processor/controller for flexible implementation of current and future image compression standards. IEEE Micro **12**(5), 33–39 (1992). https://doi.org/10.1109/40.166711
23. Bilgin, A., & Marcellin, M. W.: JPEG2000 for digital cinema. In 2006 IEEE International Symposium on Circuits and Systems (pp. 4-pp). IEEE. (2006). https://doi.org/10.1109/ISCAS.2006.1693475
24. Maharshi, A., Tong, L., Swami, A.: Cross-layer designs of multichannel reservation MAC under Rayleigh fading. IEEE Trans. Signal Process. **51**(8), 2054–2067 (2003). https://doi.org/10.1109/TSP.2003.814465
25. Gonzalez-Perez, C., & Martín-Rodilla, P.: A metamodel and code generation approach for symmetric unary associations. In 2017 11th International Conference on Research Challenges in Information Science (RCIS) (pp. 84–94). IEEE. (2017). https://doi.org/10.1109/RCIS.2017.7956522