SMPTE ST 2073-1:2017
Revision of
SMPTE ST 2073-1:2014

SMPTE STANDARD

VC-5 Video Essence — Part 1: Elementary Bitstream

Table of Contents

Approved
April 5, 2017

Figures

Tables

Foreword

SMPTE (the Society of Motion Picture and Television Engineers) is an internationally-recognized standards developing organization. Headquartered and incorporated in the United States of America, SMPTE has members in over 80 countries on six continents. SMPTE's Engineering Documents, including Standards, Recommended Practices, and Engineering Guidelines, are prepared by SMPTE's Technology Committees. Participation in these Committees is open to all with a bona fide interest in their work. SMPTE cooperates closely with other standards-developing organizations, including ISO, IEC and ITU.

SMPTE Engineering Documents are drafted in accordance with the rules given in its Standards Operations Manual.

SMPTE ST 2073-1 was prepared by Technology Committee 10E.

Intellectual Property

At the time of publication no notice had been received by SMPTE claiming patent rights essential to the implementation of this Engineering Document. However, attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. SMPTE shall not be held responsible for identifying any or all such patent rights.

Introduction

This section is entirely informative and does not form an integral part of this Engineering Document.

The VC-5 codec is a variable-bit-rate codec intended for high-quality video acquisition and post-production, capable of encoding diverse image and video formats.

(1) The key design goals of the VC-5 codec are:

(2) Very high visual quality (visually lossless in most applications);

(3) Efficient implementation of both decoders and encoders;

(4) Support for any color space or color difference component sampling;

(5) Direct encoding of camera sensor output without conversion to a different format;

(6) Adaptability and ease of use in video editing and post-production.

The VC-5 codec uses reversible integer wavelet transforms applied to the entire image to prevent blocking artifacts. Since the transforms are reversible, the only losses are due to quantization. Lowpass wavelet coefficients are encoded without quantization to preserve visual quality.

The wavelet transforms can be implemented efficiently using only integer shift and add operations. Highpass wavelet coefficient values equal to zero are run length coded. Runs of zeros and non-zero coefficients are represented using variable-length codes that are efficient for both decoding and encoding.

The VC-5 bitstream can represent any image that can be unpacked into component arrays by an image unpacking process prior to encoding (Figure 1). A component array is like an image plane and typically would contain a single type of color or data, although component arrays containing multiple types of image data are not precluded by this standard. Each of the component arrays represented in the bitstream can have different dimensions.

The image unpacking process is not specified by this standard and can implement image pre-processing algorithms, which also are not defined in this standard. The encoding process takes an ordered set of component arrays and represents the component arrays in the bitstream such that a decoding process can reconstruct the set of component arrays that were input to the encoding process in the same order as input to the encoding process. The reconstructed set of component arrays duplicates the component arrays input to the encoder except for losses due to compression.

The decoding process can be followed by an image repacking process that converts the ordered set of component arrays into a packed representation of the image. The image repacking process is not specified by this standard. The image repacking process can implement image post-processing algorithms as required by the application. An image display process that transforms the packed image into a displayable picture can follow the image repacking process. For example, the packed image can include data values that must be mapped to colors to create a displayable image.

The VC-5 codec is intended to accept common image formats, including images with RGB and YCbCr color components with an optional alpha channel. Image formats can use color difference component sampling, for example YCbCr with 4:2:2 or 4:2:0 sampling.

The VC-5 codec is not restricted to a particular set of image dimensions, pixel formats, or color standards. Color Filter Array (CFA) images, including Bayer pattern images, can be unpacked into component arrays that can be represented in a VC-5 bitstream without conversion to a different image format. Component arrays containing non-color data, such as disparity values, can be represented in a VC-5 bitstream. Component arrays can use up to 24 bits of precision to represent each value in the component array.

The syntax of a VC-5 bitstream is based on tag-value pairs that allow for future extensions. Each tag is a 16-bit number and only a small number of tags have been assigned. The value is a 16-bit number, but a tag-value pair can be used to represent the type and size of a large block of data, providing support for values larger than 16 bits.

The number of component arrays and the width and height of each component array is limited only by the scheme for representing information as tag-value pairs.

A VC-5 bitstream can include all of the information required by a decoding process to reconstruct the component arrays represented in the bitstream, but the bitstream does not have to include information that is provided by an external source. For example, a bitstream that is embedded in a media container can omit information that is provided by the container.

The VC-5 codec can adapt to specific applications by representing an image using minimal syntax or including extensive information required to describe the output of sophisticated image sensors or complex workflows.

Wavelet transforms are applied recursively to each component array with the lowpass band from each wavelet transform input to the next transform. Fewer inverse transforms can be applied during decoding to produce a lower resolution proxy image, allowing a decoder to balance decoding speed and image quality as required by specific applications.

1 Scope

This standard specifies the compressed representation and decoding process for the core VC-5 bitstream.

2 Conformance Notation

Normative text is text that describes elements of the design that are indispensable or contains the conformance language keywords: "shall", "should", or "may". Informative text is text that is potentially helpful to the user, but not indispensable, and can be removed, changed, or added editorially without affecting interoperability. Informative text does not contain any conformance keywords.

All text in this document is, by default, normative, except: the Introduction, any section explicitly labeled as "Informative" or individual paragraphs that start with "Note:"

The keywords "shall" and "shall not" indicate requirements strictly to be followed in order to conform to the document and from which no deviation is permitted.

The keywords, "should" and "should not" indicate that, among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required; or that (in the negative form) a certain possibility or course of action is deprecated but not prohibited.

The keywords "may" and "need not" indicate courses of action permissible within the limits of the document.

The keyword “reserved” indicates a provision that is not defined at this time, shall not be used, and may be defined in the future. The keyword “forbidden” indicates “reserved” and in addition indicates that the provision will never be defined in the future.

A conformant implementation according to this document is one that includes all mandatory provisions ("shall") and, if implemented, all recommended provisions ("should") as described. A conformant implementation need not implement optional provisions ("may") and need not implement them as described.

Unless otherwise specified, the order of precedence of the types of normative information in this document shall be as follows: Normative prose shall be the authoritative definition; tables shall be next; followed by formal languages; then figures; and then any other language forms.

3 Normative References

The following standards contain provisions which, through reference in this text, constitute provisions of this engineering document. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this engineering document are encouraged to investigate the possibility of applying the most recent edition of the standards indicated below.

None.

4 Terms and Definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- IEC Electropedia: available at http://www.electropedia.org/
- ISO Online browsing platform: available at http://www.iso.org/obp

4.1 big-endian

Order of bytes that comprise a number with more significant bytes before less significant bytes.

4.2 bitstream

Ordered sequence of bits that forms the representation of an image.

4.3 bitstream position

Number of bits counted from the beginning of the bitstream.

4.4 byte-aligned

Bitstream position that is a multiple of 8 bits (one byte).

4.5 byte

Set of eight consecutive bits starting at a bitstream position that is a multiple of 8 bits.

4.6 camel caps

Refers to the practice of writing compound words using one or more internal upper case letters. Such words are usually formed from phrases where the individual words or elements are joined without spaces, and where each element begins with a capital letter within the compound word. For example CamelCaps

4.7 channel

Representation of a component array in the bitstream.

4.8 chunk

Sequence of bytes in the bitstream used to represent information that cannot be represented by the value in a tag-value pair.

4.9 codeblock

Representation of a two-dimensional array of coefficients corresponding to a wavelet band.

4.10 codec

Truncation of the terms “coder” or “encoder” and “decoder”.

4.11 codec state

Set of parameters in effect at a specific position in the bitstream that govern how the decoder processes codeblocks and reconstructs component arrays.

4.12 codeword

Contiguous sequence of bits that represents an entry in the codebook.

4.13 coefficient

Scalar frequency value that is the result of a wavelet transform.

4.14 companding

Transform using a monotonic non-decreasing function to reduce the range of quantized coefficients.

4.15 component array

Two-dimensional array of scalar values.

4.16 component value

Scalar value in a component array.

4.17 convolution filter

Filter bank.

4.18 decoder

Embodiment of a decoding process and optionally an image repacking process.

4.19 decoding process

Process that consumes a bitstream to produce an ordered set of component arrays.

4.20 display process

Post-processing of the decoded image into a picture for display or further processing (informative).

4.21 encoder

Embodiment of an encoding process and optionally an image unpacking process.

4.22 encoding process

Process that consumes an ordered set of component arrays to produce a bitstream.

4.23 entropy coding

Mathematical process intended to reduce the number of bits required for a lossless representation of data.

4.24 horizontal wavelet transform

Wavelet transform applied to each entire row in a two-dimensional array.

4.25 image

Ordered set of one or more component arrays.

4.26 image data

Component values that comprise an image.

4.27 image dimensions

Image width and height.

4.28 image height

Upper bound of the height of the component arrays represented in a bitstream (see Annex B.4).

4.29 image repacking process

Post-processing and repacking the decoded component arrays into a picture (informative).

4.30 image unpacking process

Unpacking and pre-processing a picture into an ordered set of component arrays for input to the encoding process (informative).

4.31 image width

Upper bound of the width of the component arrays represented in a bitstream (see Annex B.4).

4.32 initial codec state

Codec state at the beginning of the bitstream.

4.33 inverse companding

Monotonic transform that restores companded values to an approximation of the quantized coefficient value before companding.

4.34 inverse horizontal wavelet transform

Inverse wavelet transform applied to each entire row of coefficients in an array.

4.35 inverse quantization
dequantization

Mapping a quantized value to a representative value for the sub-range identified by the quantized value.

4.36 inverse spatial wavelet transform

Inverse vertical wavelet transform followed by application of the inverse horizontal wavelet transform to the result of the inverse vertical wavelet transform.

4.37 inverse vertical wavelet transform

Inverse wavelet transform applied to each entire column of coefficients in an array.

4.38 inverse wavelet transform
inverse discrete wavelet transform
IDWT

Inverse of the wavelet transform.

4.39 native order

Order of bytes on the encoding or decoding device.

4.40 network order

Byte order that is big-endian.

4.41 parameter

Variable in the codec state that takes one of a range of values.

4.42 picture

Displayable array of source, coded, or reconstructed image data with color values that are packed or planar depending on the image format.

4.43 quantization

Process that partitions a range of discrete values into sub-ranges that are not necessarily equal and represents each partition by a unique value.

4.44 raster-scan order

Processing a two-dimensional array from the top line (row) to the bottom line (row) and from the left column to the right column within each line (row).

4.45 reversible wavelet

Discrete wavelet transform (DWT) and inverse discrete wavelet transform (IDWT) pair using elementary integer arithmetic operations such that the composition of the DWT and IDWT is the identity function.

4.46 run

Signed value and the number of times that the value is repeated.

4.47 spatial wavelet transform

Two-dimensional wavelet transform that is the separable composition of the horizontal wavelet transform followed by the vertical wavelet transform.

4.48 spatial wavelet

Decomposition of an array into four wavelet bands by application of a spatial wavelet transform.

4.49 subband

Wavelet band that must be present in the bitstream.

4.50 subband coefficient

Value of a coefficient in a subband.

4.51 upper bound

Element in a range of values that is greater than or equal to the maximum element in a subset of the range.

4.52 variable length coding

Reversible procedure for entropy coding.

4.53 VLC

Variable length coding or the variable length code used for entropy coding.

4.54 vertical wavelet transform

Wavelet transform applied to each entire column in a two-dimensional array.

4.55 wavelet band

Array of coefficients resulting from the application of the same combination of lowpass and highpass wavelet filters by the spatial wavelet transform.

4.56 wavelet level

Number of spatial wavelet transforms applied recursively to a component array to compute a spatial wavelet.

4.57 wavelet transform
discrete wavelet transform
DWT

Transform that computes a set of lowpass coefficients and a set of highpass coefficients through the iterated application of a filter bank.

5 Notation

5.1 Codec State Parameters

In this standard, parameters in the codec state are represented in camel caps and boldface with the first letter capitalized as in **ImageWidth**.

5.2 Number Representations

Numbers are decimal integers except that hexadecimal numbers are prefixed with “0x”.

The operators and functions used in this standard operate on and return numerical values with unlimited precision and perform internal calculations with unlimited precision.

Negative integers shall be represented using two’s complement arithmetic.

Integer division shall round towards zero.

5.3 Arithmetic Right Shift

The function $\mathrm{ash}(x, b)$ shall be an arithmetic right shift equivalent to the expression:

$$\left\lfloor \frac{x}{2^b} \right\rfloor$$

where x is a signed integer and b is a non-negative integer. Arithmetic right shift truncates towards negative infinity.

6 Overview (Informative)

The encoding process can be preceded by an image unpacking process that unpacks the input image into component arrays that are encoded in the bitstream as separate channels. A reversible wavelet transform is applied recursively to each component array to compute a sequence of three wavelets. The lowpass band for the smallest wavelet in the sequence and all of the highpass bands are encoded into the bitstream.

A decoding process decodes each channel separately into a component array and can deliver the component arrays to an image repacking process that packs the component arrays into an output image (Figure 1).

The image unpacking and repacking processes are not part of this standard.

Decoding the bitstream corresponding to each component array consists of the following steps:

(1) The lowpass band in the bitstream is unpacked into the array for the lowpass band in the smallest wavelet in the sequence.

(2) Each entropy coded highpass subband in the bitstream is decoded into the corresponding array for the wavelet band in the sequence.

(3) The inverse companding curve is applied to the highpass band coefficients.

(4) The decompanded highpass band coefficients are dequantized.

(5) The inverse wavelet transform is applied recursively from the smallest to largest wavelet to reconstruct the lowpass bands that were not present in the bitstream.

(6) The inverse wavelet transform is applied to the largest wavelet in the sequence to reconstruct the component array.

The highpass bands are entropy coded as runs of zeros and non-zero coefficients using a Huffman codebook.

Parameters that control decoding, such as the dimensions of each component array, are encoded in the bitstream as tag-value pairs to provide extensibility. The codec state is the current value of all parameters in effect at any point in the bitstream.

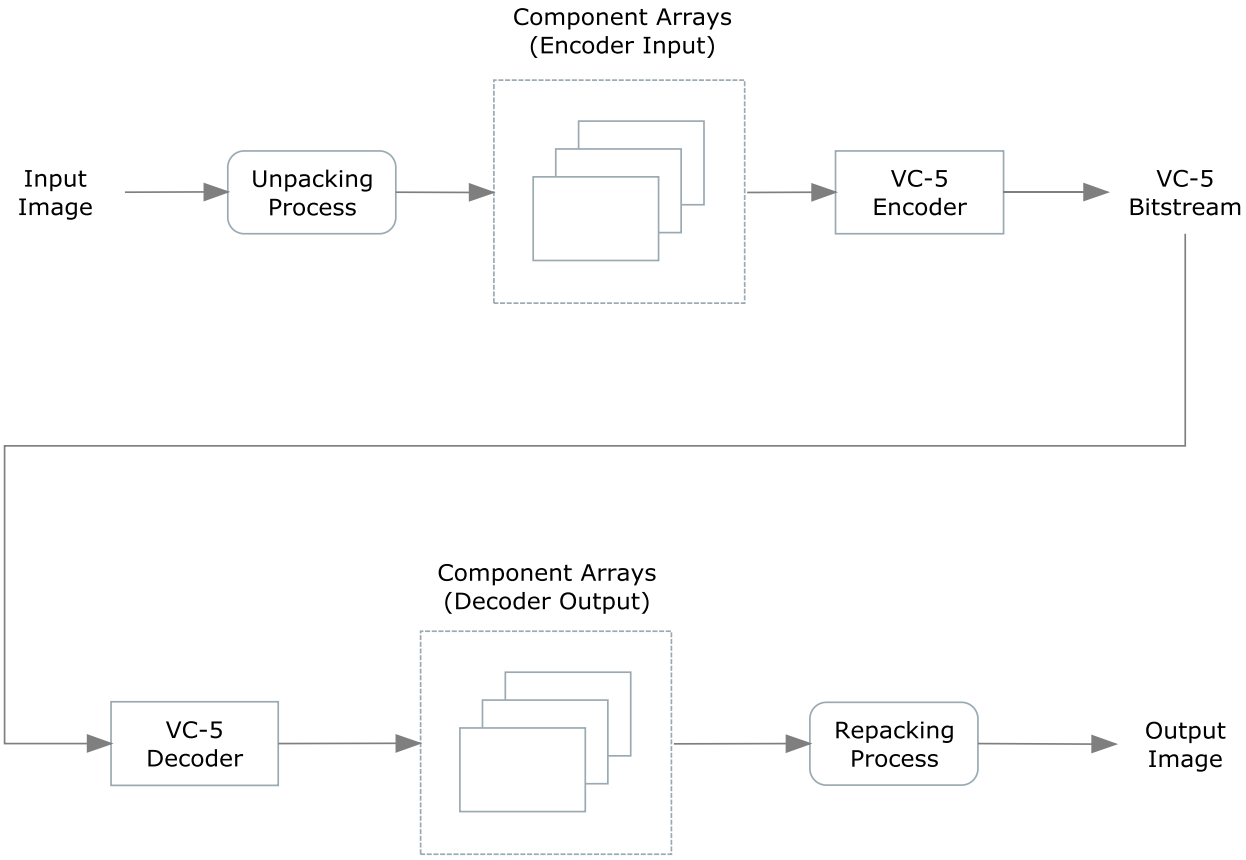

Figure 1 – Overview of the encoding and decoding processes (Informative)

7 Component Arrays

7.1 Component Array Requirements

The set of component arrays represented in the bitstream shall satisfy the following conditions:

(1) All component values shall be unsigned integers.

(2) The number of component arrays shall be the value of **ChannelCount**.

Note: In many applications, the component arrays are the output of an image unpacking process.

7.2 Component Array Representation

Each component array in the bitstream shall be represented as a separate channel in the bitstream (Figure 2). Channels shall be identified by channel numbers. Channels need not be present in the bitstream in numerical order.

7.3 Component Array Dimensions

ChannelWidth shall be the number of component values per row in a component array.

ChannelHeight shall be the number of rows in a component array.

7.4 Component Value Precision

BitsPerComponent shall be the number of significant bits per component value in a component array.

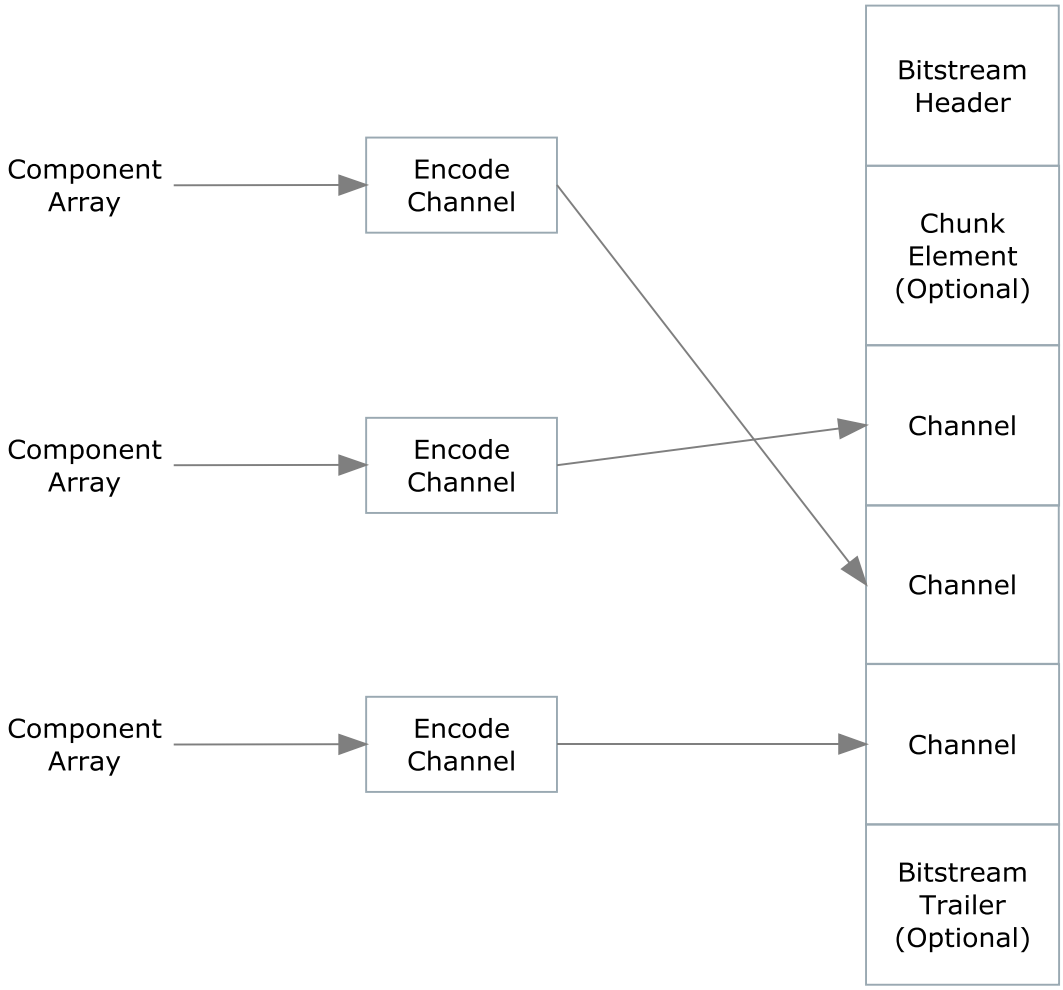

Figure 2 – Component arrays are represented as separate channels in the bitstream

8 Bitstream Syntax

8.1 Syntax Elements

A VC-5 bitstream shall consist of a contiguous sequence of syntax elements.

A syntax element shall be a start marker, tag-value pair, codeblock element, or chunk element.

8.2 Bitstream Segments

A segment shall be a contiguous sequence of 32 bits (four bytes) in the bitstream that begins on a segment boundary. A segment boundary shall be a bitstream position that is a multiple of 32 bits (four bytes). The bytes comprising a segment shall be in network order.

8.3 Tag-Value Pairs

8.3.1 Tag-Value Syntax

A tag value pair shall consist of one segment. The tag shall be the most significant 16-bit integer in the segment and the value shall be the least significant 16-bit integer in the segment.

There are two types of tag-value pairs:

(1) Required tag-value pairs;

(2) Optional tag-value pairs.

8.3.2 Required Tag-Value Pairs

A required tag-value pair may be present in the bitstream. The tag shall be a positive number corresponding to a codec state parameter (Annex B).

8.3.3 Optional Tag-Value Pairs

The tag in an optional tag-value pair shall be a negative number. An optional tag-value pair may be present in the bitstream. A decoder shall not be required to process the value in an optional tag-value pair. The two's complement negation of the tag shall be the tag number that identifies the corresponding codec state parameter. The tag number need not be listed in Table B.2.

Note: Optional tag-value pairs are out of scope.

8.4 Chunk Elements

8.4.1 Chunk Element Syntax

Chunk elements shall be one of two types: required or optional.

A chunk element shall comprise a chunk header immediately followed by the chunk payload. Chunk elements shall be required or optional as indicated by the sign of the tag in the chunk header (Sections 8.4.3 and 8.4.4).

If the chunk element is required:

(1) The decoder shall be required to process the chunk payload.

(2) The chunk payload shall be in network order.

If the chunk element is optional:

(1) The decoder shall not be required to process the chunk payload.

(2) The byte order of the chunk payload is out of scope.

(3) The decoding process for the chunk is out of scope.

8.4.2 Chunk Payload Size

The size of a chunk payload shall be in units of segments. The minimum size of a chunk payload shall be zero. Chunk payloads shall be small or large.

8.4.3 Small Chunk Payloads

The chunk header shall consist of one segment that comprises a 16-bit integer chunk tag and a 16-bit unsigned integer that is the size of the chunk payload.

If the chunk tag is positive, the chunk element is required, and the chunk tag that identifies the chunk shall be the 16-bit integer chunk tag.

If the chunk tag is negative, then the chunk element is optional and the chunk tag that identifies the chunk shall be the two's complement negation of the 16-bit integer chunk tag.

The payload of a small chunk element is limited to $2^{16} - 1$ segments = 262,140 bytes.

Note: The syntax of a small chunk header is equivalent to a tag-value pair.

8.4.4 Large Chunk Payloads

The chunk header shall consist of one segment that comprises an 8-bit integer chunk tag followed by three bytes that are used to compute the size of the chunk payload.

If the chunk tag is negative, then the first two bytes of the chunk header shall comprise a 16-bit two's complement negative integer. The two's complement negation of that 16-bit integer shall comprise:

A one byte tag that identifies the chunk.

The most significant byte in the three bytes (including the last two bytes in the chunk header) that comprise a 24-bit unsigned integer that is the chunk payload size.

If the chunk tag is positive, then the chunk tag shall be listed in Annex B.3 and the three bytes that follow the chunk tag in the chunk header shall comprise a 24-bit unsigned integer that is the chunk payload size.

If the chunk tag is negative, then the chunk element is optional; otherwise, the chunk element is required.

The payload of a large chunk element is limited to $2^{24} - 1$ segments = 67,108,860 bytes.

Note: The syntax of the chunk header for an optional large chunk element is equivalent to an optional tag-value pair with the least significant byte in the tag used for the payload size.

8.5 Bitstream Constraints

A valid bitstream conforms to the following restrictions:

(1) All syntax elements shall consist of an integer number of segments.

(2) The first segment in the bitstream shall be a start marker segment (Section 8.6).

(3) The bitstream header (Section 8.7) shall occur immediately after the start marker segment.

(4) Codeblocks from different channels shall not be interleaved in the bitstream.

(5) The bitstream shall terminate immediately after the last codeblock of the last channel.

Note: The restriction on not interleaving codeblocks allows the same set of wavelets and buffers to be used for decoding all channels.

8.6 Bitstream Start Marker

The **StartMarkerSegment** shall be the 4 byte sequence 0x56, 0x43, 0x2D, 0x35.

8.7 Bitstream Header

The bitstream header:

(1) Shall consist of zero or more tag-value pairs that correspond to header parameters listed in Table B.2.

(2) Shall not contain a tag-value pair for a header parameter that is not defined in this standard (Table B.2).

If a tag-value pair that corresponds to a header parameter (Table B.2) is present in the bitstream, the tag-value pair:

(1) Shall occur at most once in the bitstream.

(2) Shall be present in the bitstream header.

8.8 Codeblock Elements

A codeblock element shall be a large chunk element as follows:

(1) The large chunk element shall be a required chunk element.

(2) The chunk tag shall be **LargeCodeblockTag**.

(3) The payload of the large chunk element shall be the codeblock (Section 10.2).

(4) The payload shall comprise the minimum number of segments required to contain the codeblock.

Immediately before any codeblock element:

(1) The **ChannelNumber** in effect shall be the number of the channel for the codeblock.

(2) The **SubbandNumber** in effect shall be the subband number of the codeblock.

9 Codec State

9.1 Codec State Parameters

The codec state is the set of parameter values used by the decoding process to decode the codeblocks and reconstruct the component array for each channel.

The parameter values used to process a syntax element shall be the codec state in effect immediately before the syntax element.

Changes to the codec state that result from processing the syntax element shall take effect immediately after the syntax element.

The values of the parameters in the codec state shall be obtained from:

(1) Default codec state parameters set during initialization of the decoder (Section 9.2).

(2) External parameters provided to the decoder from the calling application or the media container (Section 9.3).

(3) Explicit values provided by tag-value pairs in the bitstream (Section 9.4).

(4) Implicit values from codec state changes that occur while decoding the bitstream (Section 9.5).

A tag-value pair is redundant if the value in the tag-value pair is identical to the value of the corresponding parameter in the codec state at the bitstream position of the tag-value pair. An encoder should not insert redundant tag-value pairs in the bitstream.

9.2 Codec State Initialization

The codec state at the beginning of the decoding process shall be initialized using the default values listed in Table B.2.

9.3 External Parameter Values

An external application may pass parameter values to the decoder. External parameter values shall be used to update the codec state after codec state initialization and before the first syntax element in the bitstream is processed.

9.4 Explicit Parameter Changes

Tag-value pairs correspond to codec state parameters. The tag identifies the codec state parameter. If the tag-value pair is required, then the decoding process shall assign the value in the tag-value pair to the corresponding parameter in the codec state.

9.5 Codec State Changes

9.5.1 Subband State Change

The value of **SubbandNumber** in effect immediately after a codeblock shall be the value of **SubbandNumber** in effect immediately before the codeblock incremented by 1.

9.5.2 Channel State Change

Immediately after the last codeblock for a channel:

(1) The value of **ChannelNumber** in effect immediately after the last codeblock for a channel shall be the value of **ChannelNumber** in effect immediately before the last codeblock incremented by 1.

(2) The value of **SubbandNumber** in effect immediately after the last codeblock for a channel shall be 0.

The last codeblock for a channel shall be the codeblock that begins at a bitstream position that is larger than the bitstream position at the beginning of any other codeblock in the set of codeblocks for that channel.

10 Subband Coding

10.1 Coefficient Order

Coefficients in a wavelet band that is present in the bitstream shall be in raster-scan order beginning with the first coefficient in the first row of coefficients and ending with the last coefficient in the last row of coefficients.

10.2 Subband Codeblocks

For each channel, the bitstream shall contain one codeblock for each subband in the wavelet sequence for the channel (Table 1).

A codeblock:

(1) Shall be the payload of a codeblock element (Section 8.8);

(2) Shall only contain coefficients for a single subband;

(3) Shall not include padding before any coefficient or codeword in the codeblock;

(4) Shall be padded at the end with up to 31 zero bits to the nearest segment boundary.

10.3 Lowpass Band Codeblocks

The codeblock for a lowpass band shall contain only the contiguous (bit packed) sequence of coefficients for the lowpass band in coefficient order:

(1) The lowpass coefficients shall not be quantized.

(2) The lowpass coefficients shall not be companded.

(3) The number of bits used to represent each lowpass coefficient shall be the value of **LowpassPrecision** in effect immediately before the codeblock.

10.4 Highpass Band Codeblocks

10.4.1 Highpass Band Codeblock Syntax

The codeblock for a highpass band shall contain only the contiguous (bit packed) sequence of codewords for the entropy coded highpass band in coefficient order followed by the band end marker (Section 10.4.4).

10.4.2 Run-Length Coding

A contiguous sequence of highpass coefficients with the value zero may be represented using one or more codewords for runs of zeros. A codeword for a run of zeros may represent a contiguous sequence of zeros from more than one row in the highpass band.

10.4.3 Non-Zero Coded Values

Each non-zero highpass coefficient shall be represented by the codeword for the coefficient magnitude followed by the sign bit for the coefficient value:

(1) The sign bit for a positive value shall be 0.

(2) The sign bit for a negative value shall be 1.

10.4.4 Band End Marker

The last codeword in the codeblock for a highpass band shall be the band end marker (Table C.2). The band end marker shall immediately follow the codeword for the last coefficient in the highpass band.

11 Wavelet Transforms

11.1 Spatial Wavelet Transform (Informative)

The encoding process uses the spatial wavelet transform to decompose coefficient arrays and lowpass bands into spatial wavelets.

A wavelet transform is a pair of convolution filters, a lowpass transform and a highpass transform. The wavelet transforms are 2/6 reversible wavelets. The lowpass transform is a 2-tap convolution filter followed by 2:1 decimation and the highpass transform is a 6-tap convolution filter followed by 2:1 decimation.

The number of coefficients in the lowpass result is half the number of input values and the number of coefficients in the highpass result is half the number of input values. The wavelet transforms can be implemented with integer arithmetic using only shift and add operations.

The spatial wavelet transform is a separable transform computed by applying the wavelet transform to each entire row in the input array, then applying the wavelet transform to each entire column in the lowpass and highpass results from the horizontal wavelet transform.

The result of the horizontal wavelet transform is an array of lowpass coefficients and an array of highpass coefficients, each array equal in height to the input array and half the width of the input array (Figure 3, Part A). The vertical wavelet transform is applied to the lowpass horizontal result to compute two output arrays (the LL and HL bands), each half the width and half the height of the input array, and the vertical wavelet transform is applied to the highpass horizontal result to compute two output arrays (the LH and HH bands), each half the width and half the height of the input array.

For purposes of illustration, the four wavelet bands are arranged in quadrants with the lowlow (LL) band in the upper left quadrant, the lowhigh (LH) band the upper right quadrant, the highlow (HL) band in the lower left quadrant, and the highhigh band (HH) in the lower right quadrant (Figure 3, Part B).

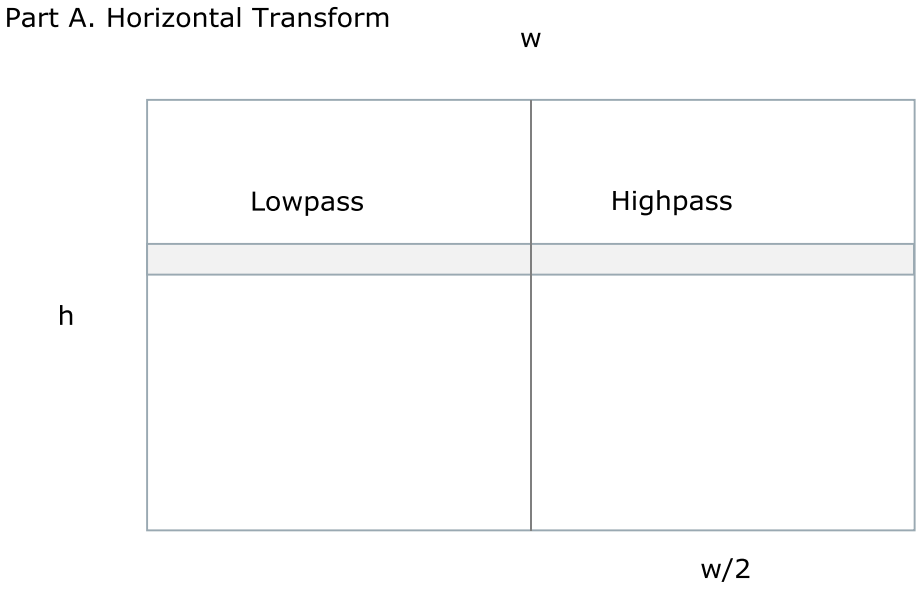

Figure 3 – Two-dimensional spatial wavelet transform as a horizontal wavelet transform followed by a vertical wavelet transform

11.2 Spatial Wavelet Sequence

This section describes the wavelet sequence that is represented in the bitstream.

Wavelet transforms shall be performed on the entire input array without blocking or partitioning. A spatial wavelet transform shall be applied to each component array.

The spatial wavelet sequence is the set of wavelets that result from recursively applying the spatial wavelet transform to a component array:

Level 1. Spatial wavelet from the application of the spatial wavelet transform to the component array.

Level 2. Spatial wavelet from the application of the spatial wavelet transform to the lowpass band in the spatial wavelet at level 1.

Level 3. Spatial wavelet from the application of the spatial wavelet transform to the lowpass band in the spatial wavelet at level 2.

The spatial wavelet sequence is shown in Figure 4.

The lowpass and highpass bands from the wavelet sequence that are present in the bitstream are called subbands. Subbands shall be numbered as:

(1) Lowpass band in the wavelet at level 3 shall be subband number 0.

(2) Highpass bands in the wavelet at level 3 shall be subbands 1 through 3 in the order in which the bands are numbered within a wavelet.

(3) Highpass bands in the wavelet at level 2 shall be subbands 4 through 6 in the order in which the bands are numbered within a wavelet.

(4) Highpass bands in the wavelet at level 1 shall be subbands 7 through 9 in the order in which the bands are numbered within a wavelet.

Subband numbers and the corresponding wavelet levels and bands shall be numbered as stated in Table 1.

Table 1 – Subband numbering for the sequence of wavelets in each channel

Subband Number	*Wavelet Level*	*Wavelet Band*
0	3	LL
1	3	LH
2	3	HL
3	3	HH
4	2	LH
5	2	HL
6	2	HH
7	1	LH
8	1	HL
9	1	HH

An arithmetic right shift specified by the **PrescaleShift** parameter shall be applied to the array input to the wavelet transform at each level in the wavelet sequence. The prescale shifts and the lowpass and highpass wavelet transforms used to compute the wavelets at each level in the sequence are diagrammed in Figure 5 using the notation for the prescale shifts and wavelet transforms described in Table 2.

Table 2 – Notation used in Figure 5

Symbol	**Description**
Sn	Prescale shift applied to the array input to the wavelet at level n
Ln	Lowpass wavelet transform applied to the prescaled input to the wavelet at level n
Hn	Highpass wavelet transform applied to the prescaled input to the wavelet at level n
LLn	Lowpass transform applied to the result of the lowpass transform at wavelet level n
LHn	Lowpass transform applied to the result of the highpass transform at wavelet level n
HLn	Highpass transform applied to the result of the lowpass transform at wavelet level n
HHn	Highpass transform applied to the result of the highpass transform at wavelet level n

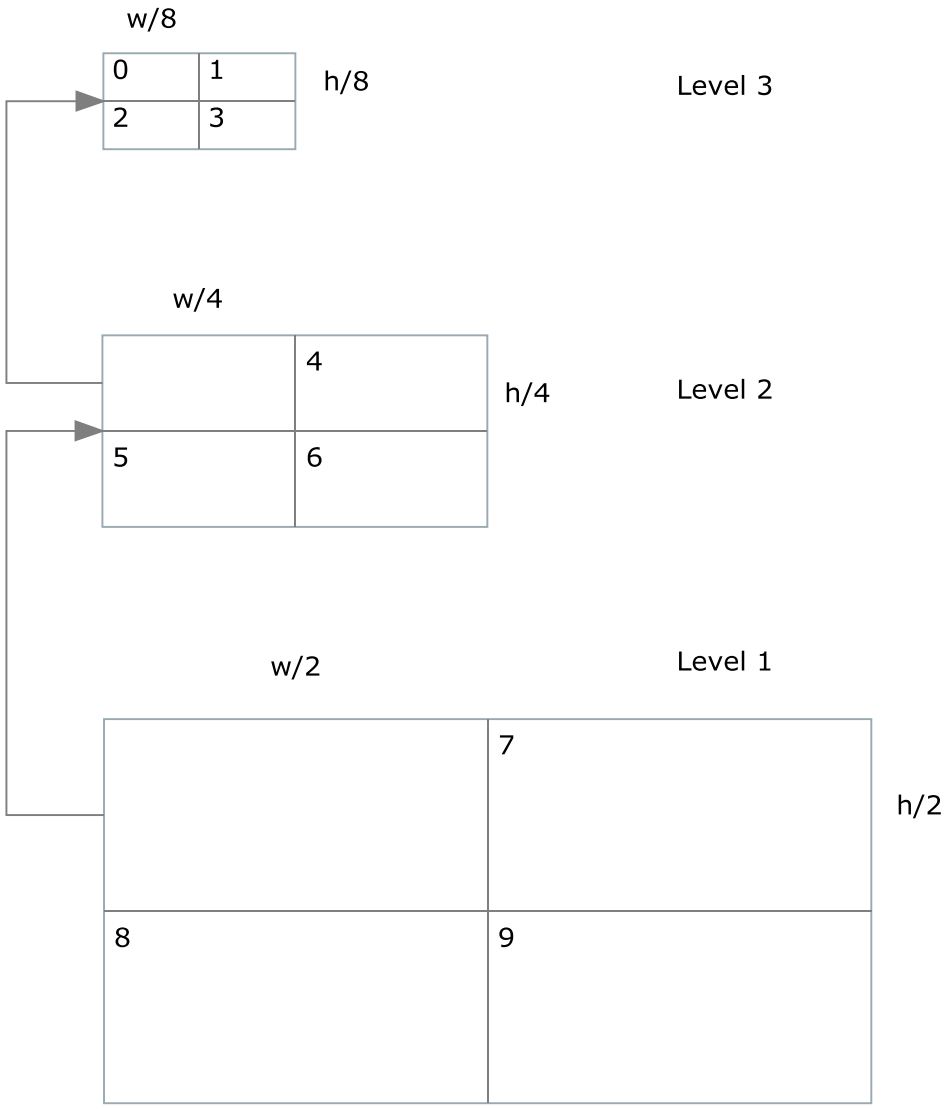

Figure 4 – Three levels of spatial wavelet transform showing the lowpass bands used as inputs to the next wavelet level and the numbering of subbands. Each channel is encoded using this sequence of spatial wavelet transforms.

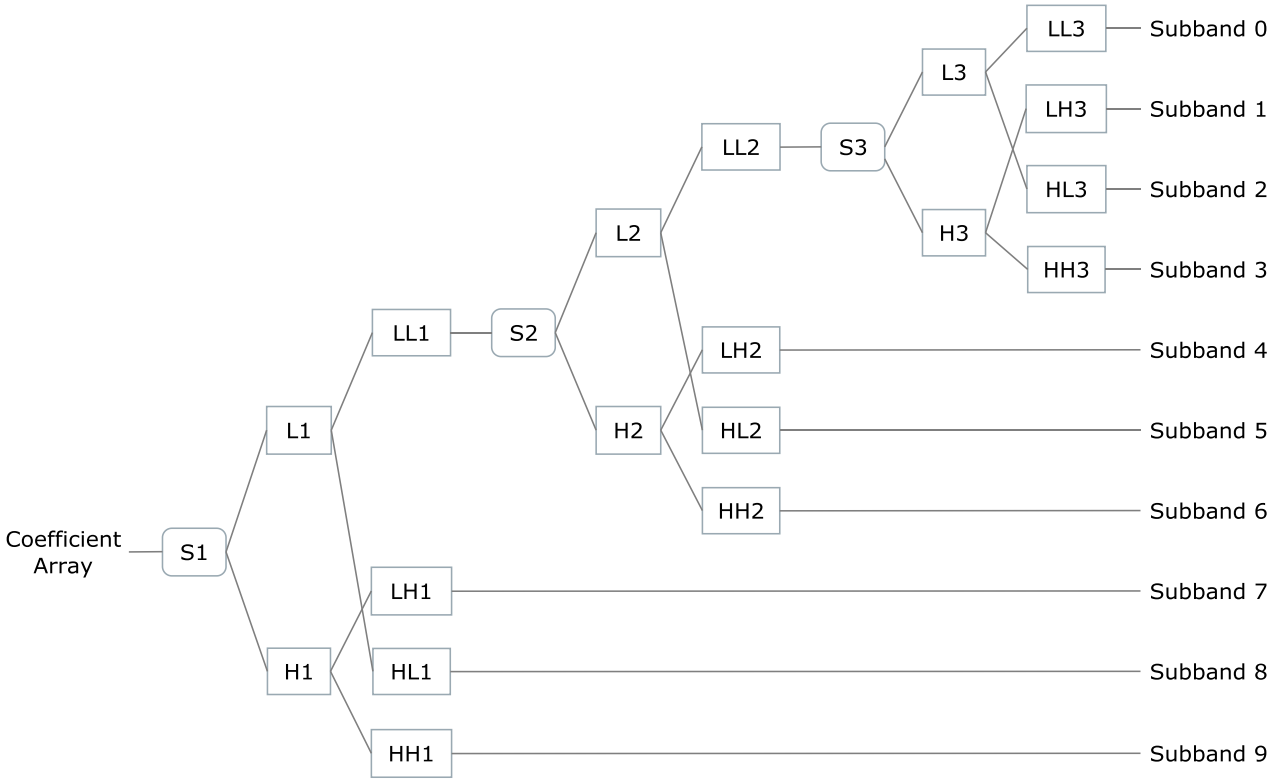

Figure 5 – Sequence of operations used to encode a component array into the bitstream

11.3 Inverse Wavelet Transforms

The decoding process shall apply the inverse spatial wavelet transform to the coefficients in the wavelet bands after inverse companding and dequantization.

The inverse spatial wavelet transform shall be applied to the wavelet sequence for each channel independently and in the following steps:

(1) Apply the inverse spatial wavelet transform to the wavelet in level 3 to compute the lowpass band in the wavelet at level 2.

(2) Left shift the coefficients in the lowpass band at level 2 by **PrescaleShift[2]**.

(3) Apply the inverse spatial wavelet transform to the wavelet in level 2 to compute the lowpass band in the wavelet at level 1.

(4) Left shift the coefficients in the lowpass band at level 1 by **PrescaleShift[1]**.

(5) Apply the inverse spatial wavelet transform to the wavelet in level 1 to compute the component array.

(6) Left shift the values in the component array by **PrescaleShift[0]**.

The inverse spatial wavelet transform at each level shall be computed in a manner that is functionally equivalent to the following steps:

(1) The inverse wavelet transform shall be applied vertically using the LL band as the lowpass input to the inverse wavelet transform and using the HL band as the highpass input to the inverse wavelet transform.

(2) The inverse wavelet transform shall be applied vertically using the LH band as the lowpass input to the inverse wavelet transform and using the HH band as the highpass input to the inverse wavelet transform.

(3) The inverse wavelet transform shall be applied horizontally using the result of step (1) as the lowpass input to the inverse wavelet transform and the result of step (2) as the highpass input to the inverse wavelet transform.

The result of the first step is an array of lowpass coefficients with the same width as the bands in the wavelet input to the inverse spatial wavelet transform and twice the height of the bands in the wavelet input to the inverse spatial wavelet transform, the result of the second step is an array of highpass coefficients with the same width as the bands in the wavelet input to the inverse spatial wavelet transform and twice the height of the bands in the wavelet input to the inverse spatial wavelet transform (Figure 3).

Note: The application of the inverse spatial wavelet transform produces two rows in the output lowpass band for each row in the input lowpass band; and each row is twice the width of the bands in the input spatial wavelet.

The inverse wavelet transforms shall be implemented by a method that is mathematically equivalent to the formulas in Annex A.

11.4 Wavelet Transform Requirements

11.4.1 Wavelet Height Requirement

11.4.1.1 Wavelet Height Computation

The height h_{i+1} of the wavelet at level $i+1$ in the sequence shall be computed from the height h_i of the wavelet at level i in the sequence using the following formula:

$$h_{i+1} = \left\lceil \frac{h_i}{2} \right\rceil$$

11.4.1.2 Encoding Process

If the height of the array input to the spatial wavelet transform at any level is an odd number, then the spatial wavelet transform shall be performed as if the input array includes one additional row that duplicates the last row of the input array.

11.4.1.3 Decoding Process

If the output array for the inverse wavelet transform contains an odd number of rows, then the last row produced by the inverse wavelet transform shall be discarded.

11.4.2 Wavelet Width Requirement

11.4.2.1 Wavelet Width Computation

The width w_{i+1} of the wavelet at level $i+1$ in the sequence shall be computed from the width w_i of the wavelet at level i in the sequence using the following formula:

$$w_{i+1} = \left\lceil \frac{w_i}{2} \right\rceil$$

11.4.2.2 Encoding Process

If the width of the array input to the spatial wavelet transform at any level is an odd number, then the spatial wavelet transform shall be performed as if the input array includes one additional column that duplicates the last column of the input array.

11.4.2.3 Decoding Process

If the output array for the inverse wavelet transform contains an odd number of columns, then the last column produced by the inverse wavelet transform shall be discarded.

12 Decoding Process

12.1 Subband Decoding

All coefficients in each codeblock corresponding to a lowpass band shall be decoded into the corresponding array for the wavelet band.

All codewords in each codeblock for a highpass band shall be decoded into coefficients by the following steps in order:

(1) Entropy decoding (Section 12.2);

(2) Inverse companding (Section 12.3);

(3) Inverse quantization (Section 12.4).

12.2 Entropy Decoding

The codewords in a highpass band shall be entropy decoded (Section 10.4).

12.3 Inverse Companding

The inverse of the companding curve shall be applied to the absolute value of the decoded highpass coefficients before inverse quantization. The sign of the inverse companded coefficient shall be the same as the sign of the decoded highpass coefficient before inverse companding. The decoder shall use the inverse of the companding curve defined by the formula in Annex F.

12.4 Inverse Quantization

Every highpass coefficient value shall be multiplied by the value of the **Quantization** parameter.

12.5 Inverse Wavelet Processing

The inverse spatial wavelet transform shall be applied to each spatial wavelet after all bands in the wavelet have been computed by subband decoding or application of the inverse wavelet transform to the wavelet transform at the next higher level in the wavelet sequence.

12.6 Component Array Order

The decoder shall output the decoded component arrays in increasing numerical order by channel number starting with the component array corresponding to channel number 0.

Annex A Inverse Spatial Wavelet Transform (Normative)

The inverse spatial wavelet transform is a separable filter that is the composition of an inverse vertical wavelet transform and an inverse horizontal wavelet transform.

The inverse vertical wavelet transform expands every row into two rows; the inverse horizontal transform expands every column into two columns.

The inverse vertical wavelet transform shall be applied to every column in the LL and HL wavelet bands to yield an array L of lowpass results that is twice the height and the same width as the LL and HL wavelet bands.

The inverse vertical wavelet transform shall be applied to every column in the LH and HH wavelet bands to yield an array H of highpass results that is twice the height and the same width as the LH and HH wavelet bands.

The inverse horizontal wavelet transform shall be applied to every row in the array L of lowpass results and the corresponding row in the array H of highpass results to produce an output array that is twice the width and twice the height of the wavelet bands in the input to the inverse spatial wavelet transform.

Refer to the diagram for the spatial wavelet transform (Figure 3).

In the formulas described in this section, Y_i refers to the inverse filter output, L_i refers to the lowpass filter input, and H_i refers to the highpass filter input, for $i = 0, 1, \cdots, n-1$, where n is the number of rows (columns) in the input array.

Special formulas are used for the first and last output indices.

The value at output index 0 shall be calculated as follows:

$$Y_0 = \text{ash}(\text{ash}(11L_0 - 4L_1 + L_2 + 4, 3) + H_0, 1)$$

The value at output index 1 shall be calculated as follows:

$$Y_1 = \text{ash}(\text{ash}(5L_0 + 4L_1 - L_2 + 4, 3) - H_0, 1)$$

The value at output index $2i$ for interior points $0 < i < n - 1$ shall be calculated as follows:

$$Y_{2i} = \text{ash}(\text{ash}(L_{i-1} - L_{i+1} + 4, 3) + L_i + H_i, 1)$$

The value at output index $2i + 1$ for interior points $0 < i < n - 1$ shall be calculated as follows:

$$Y_{2i+1} = \text{ash}(\text{ash}(L_{i+1} - L_{i-1} + 4, 3) + L_i - H_i, 1)$$

The value at output index $2n - 2$ shall be calculated as follows:

$$Y_{2n-2} = \text{ash}(\text{ash}(5L_{n-1} + 4L_{n-2} - L_{n-3} + 4, 3) + H_{n-1}, 1)$$

The value at output index $2n - 1$ shall be calculated as follows:

$$Y_{2n-1} = \text{ash}(\text{ash}(11L_{n-1} - 4L_{n-2} + L_{n-3} + 4, 3) - H_{n-1}, 1)$$

Annex B Codec State Parameters (Normative)

B.1 Parameter Descriptions

B.1.1 ImageWidth

The **ImageWidth** parameter shall be no less than the number of component values in the longest row of all component arrays.

B.1.2 ImageHeight

The **ImageHeight** parameter shall be no less than the number of component values in the longest column of all component arrays.

B.1.3 BitsPerComponent

The **BitsPerComponent** parameter shall specify the number of bits per component value in the component array. The value for **BitsPerComponent** shall be in the range [8, 24].

B.1.4 ChannelCount

The **ChannelCount** parameter shall specify the number of component arrays. The value shall be at least 1.

B.1.5 SubbandCount

The **SubbandCount** parameter shall specify the number of subbands in the channel. The value shall be 10.

B.1.6 ChannelNumber

The ChannelNumber parameter shall specify the number of the channel. The value shall be in the range [0, ChannelCount − 1].

B.1.7 SubbandNumber

The SubbandNumber parameter shall specify the number of the subband. The value shall be in the range [0, SubbandCount – 1].

B.1.8 LowpassPrecision

The **LowpassPrecision** parameter shall specify the number of bits per coefficient in the codeblock for the lowpass band. The value shall be in the range [8, 32].

B.1.9 Quantization

The **Quantization** parameter shall specify the multiplier used to dequantize the coefficients in the codeblock for a highpass band. The value shall be 1 or greater.

B.1.10 PrescaleShift

The **PrescaleShift** parameter shall specify the vector of prescale shifts to apply to the output of each inverse spatial wavelet transform in the wavelet sequence for the channel.

PrescaleShift[i] shall be the prescale shift applied to the output of the inverse wavelet transform at wavelet level $i+1$ after computing the inverse wavelet transform.

The value of **PrescaleShift[i]** shall be in the range [0, 15].

The prescale shifts shall be represented in the tag-value pair for **PrescaleShift** using a packed representation listed in Table B.1 with the bit positions numbered from right to left in the 16-bit unsigned value.

Table B.1 – Packed representation of the prescale shifts

Bit Positions	Prescale Shift
3-0	**PrescaleShift[0]**
7-4	**PrescaleShift[1]**
11-8	**PrescaleShift[2]**
15-12	**0**

B.1.11 ChannelWidth

The **ChannelWidth** parameter shall specify the number of component values per row in the channel component array. The value shall be in the range [24, **ImageWidth**].

Note: The minimum value of **ChannelWidth** is determined by the minimum width of 3 for the lowpass band in the spatial wavelet at the highest level.

The default value of the **ChannelWidth** parameter is the value of **ImageWidth**.

B.1.12 ChannelHeight

The **ChannelHeight** parameter shall specify the number of rows in the channel component array. The value shall be in the range [24, **ImageHeight**].

Note: The minimum value of **ChannelHeight** is determined by the minimum height of 3 for the lowpass band in the spatial wavelet at the highest level.

The default value of the **ChannelHeight** parameter is the value of **ImageHeight**.

B.2 Tag-Value Pairs

Parameters in the codec state shall be represented in the bitstream using tag-value pairs listed in Table B.2. Except as noted in Annex B.1, the value of the codec state parameter corresponds to the integer value in the tag-value pair.

The values for any tag-value pair that corresponds to a codec state parameter shall be as specified for the corresponding parameter in Annex B.1.

Required tag-value pairs shall have a tag number listed in the table of codec state parameters (Table B.2).

The range of tag numbers from 0x2000 to 0x7FFF shall not be used for tag-value pairs.

The default parameter values used to initialize the codec state shall be as listed in Table B.2.

If the default parameter value listed in Table B.2 is N/A, then the value for the parameter shall be provided as an external parameter (Section 9.3) or the value for the parameter shall be provided explicitly as a tag-value pair in the bitstream (Section 9.4).

Parameters that are designated as mandatory parameters in Table B.2 shall be either provided as an external parameter (Section 9.3) or present in the bitstream (Section 9.4).

Table B.2 – Codec state parameters

Parameter Name	Tag Number	Default Value	Header Parameter	Mandatory Parameter	Reference
ImageWidth	20	N/A	Yes	Yes	B.1.1
ImageHeight	21	N/A	Yes	Yes	B.1.2
BitsPerComponent	101	12	No	No	B.1.3
ChannelCount	12	4	Yes	No	B.1.4
SubbandCount	14	10	Yes	No	B.1.5
ChannelNumber	62	0	No	No	B.1.6
SubbandNumber	48	0	No	No	B.1.7
LowpassPrecision	35	16	No	No	B.1.8
Quantization	53	1	No	No	B.1.9
PrescaleShift	109	(0,0,0)	No	No	B.1.10
ChannelWidth	104	**ImageWidth**	No	No	B.1.11
ChannelHeight	105	**ImageHeight**	No	No	B.1.12

B.3 Chunk Headers

B.3.1 Small Chunk Headers

The absolute value of the 16-bit tag in the header for a small chunk element shall be in the range from 0x4000 to 0x40FF inclusive.

B.3.2 Large Chunk Headers

The absolute value of the chunk tag in the header of a large chunk element shall be in the range from 0x60 to 0x6F inclusive and shall be as listed in Table B.3.

Table B.3 – Symbolic names and numbers for tags that identify large chunk elements

Tag Name	Number	Description
LargeCodeblockTag	0x60	Large chunk containing a codeblock
	0x61 to 0x6F	Reserved for large chunk elements

B.4 Image Dimensions (Informative)

The **ImageWidth** and **ImageHeight** parameters (Annexes B.1.1 and B.1.2) are defined as upper bounds on the maximum dimensions of the component arrays in the bitstream to conform to industry standard practice for Bayer and Color Filter Array (CFA) images.

For example, a Bayer image with dimensions 1920 by 1080 is unpacked by the image unpacking process into four component arrays, each with dimensions 960 by 540. The **ChannelWidth** is 960 component values per row and the **ChannelHeight** is 540 rows. The value of the **ImageWidth** parameter is 1920 and the value of the **ImageHeight** parameter is 1080.

Annex C Codebook (Normative)

C.1 Coefficient Magnitudes and Zero Runs

All coefficients in highpass codeblocks shall be represented using codewords from the codebook listed in Table C.1.

Each entry in the codebook specifies:

(1) The hexadecimal representation for the codeword;

(2) The length of the codeword in bits;

(3) Number of times that the value is repeated (run length);

(4) The magnitude represented by the codeword.

Note: All non-zero values have a run length of one. The codebook entry for a non-zero value specifies the magnitude of the coded value.

Table C.1 – Codebook for non-negative coefficients (codewords listed in hexadecimal)

Codeword	Length (bits)	Run Count	Run Value	Codeword	Length (bits)	Run Count	Run Value
00000000	1	1	0	00001BB5	13	1	35
00000002	2	1	1	0000188B	13	1	30
00000007	3	1	2	000018BB	13	1	31
00000019	5	1	3	000018BF	13	180	0
00000030	6	1	4	00001AA8	13	1	32
00000036	6	1	5	00001BA0	13	1	33
0000006F	7	1	8	00001BA5	13	320	0
00000063	7	1	6	00001BA4	13	1	34
00000069	7	12	0	00003115	14	1	36
0000006B	7	1	7	00003175	14	1	37
000000D1	8	20	0	0000317D	14	1	38
000000D4	8	1	9	00003553	14	1	39
000000DC	8	1	10	00003768	14	1	40
00000189	9	1	11	00006E87	15	1	46
0000018A	9	32	0	00006ED3	15	1	47
000001A0	9	1	12	000062E8	15	1	42
000001AB	9	1	13	000062F8	15	1	43
00000377	10	1	18	00006228	15	1	41
00000310	10	1	14	00006AA4	15	1	44
00000316	10	1	15	00006E85	15	1	45
00000343	10	60	0	0000C453	16	1	48
00000354	10	1	16	0000C5D3	16	1	49
00000375	10	1	17	0000C5F3	16	1	50
00000623	11	1	19	0000DDA4	16	1	53
00000684	11	1	20	0000DD08	16	1	51
00000685	11	100	0	0000DD0C	16	1	52
000006AB	11	1	21	0001BB4B	17	1	61
000006EC	11	1	22	0001BB4A	17	1	60
00000DDB	12	1	29	00018BA5	17	1	55
00000C5C	12	1	24	00018BE5	17	1	56
00000C5E	12	1	25	0001AA95	17	1	57
00000C44	12	1	23	0001AA97	17	1	58
00000D55	12	1	26	000188A4	17	1	54
00000DD1	12	1	27	0001BA13	17	1	59
00000DD3	12	1	28	00031748	18	1	62

Codeword	Length (bits)	Run Count	Run Value	Codeword	Length (bits)	Run Count	Run Value
000317C8	18	1	63	003552DF	22	1	98
00035528	18	1	64	0062E933	23	1	102
0003552C	18	1	65	0062295D	23	1	101
00037424	18	1	66	006AA53D	23	1	103
00037434	18	1	67	006AA53F	23	1	105
00037436	18	1	68	006AA53E	23	1	104
00062294	19	1	69	006E86B9	23	1	106
00062E92	19	1	70	006E86F8	23	1	107
00062F92	19	1	71	00D54A79	24	1	111
0006AA52	19	1	72	00C5D265	24	1	109
0006AA5A	19	1	73	00C452B8	24	1	108
0006E86A	19	1	75	00DD0D71	24	1	113
0006E86E	19	1	76	00D54A78	24	1	110
0006E84A	19	1	74	00DD0D70	24	1	112
000C452A	20	1	77	00DD0DF2	24	1	114
000C5D27	20	1	78	00DD0DF3	24	1	115
000C5F26	20	1	79	0188A5F6	25	1	225
000D54A6	20	1	80	0188A5F5	25	1	189
000D54B6	20	1	81	0188A5F4	25	1	188
000DD096	20	1	82	0188A5F3	25	1	203
000DD0D6	20	1	83	0188A5F2	25	1	202
000DD0DE	20	1	84	0188A5F1	25	1	197
00188A56	21	1	85	0188A5F0	25	1	207
0018BA4D	21	1	86	0188A5EF	25	1	169
0018BE4E	21	1	87	0188A5EE	25	1	223
0018BE4F	21	1	88	0188A5ED	25	1	159
001AA96E	21	1	89	0188A5AA	25	1	235
001BA12E	21	1	90	0188A5E3	25	1	152
001BA12F	21	1	91	0188A5DF	25	1	192
001BA1AF	21	1	92	0188A589	25	1	179
001BA1BF	21	1	93	0188A5DD	25	1	201
0037435D	22	1	99	0188A578	25	1	172
0037437D	22	1	100	0188A5E0	25	1	149
00317498	22	1	94	0188A588	25	1	178
0035529C	22	1	95	0188A5D6	25	1	120
0035529D	22	1	96	0188A5DB	25	1	219
003552DE	22	1	97	0188A5E1	25	1	150

Codeword	Length (bits)	Run Length	Run Value	Codeword	Length (bits)	Run Length	Run Value
0188A587	25	1	127	0188A57D	25	1	162
0188A59A	25	1	211	0188A5A3	25	1	213
0188A5C4	25	1	125	0188A5E8	25	1	165
0188A5EC	25	1	158	0188A5A2	25	1	212
0188A586	25	1	247	0188A57C	25	1	227
0188A573	25	1	238	0188A58E	25	1	198
0188A59C	25	1	163	0188A5B3	25	1	236
0188A5C8	25	1	228	0188A5B2	25	1	234
0188A5FB	25	1	183	0188A5B1	25	1	117
0188A5A1	25	1	217	0188A5B0	25	1	215
0188A5EB	25	1	168	0188A5AF	25	1	124
0188A5A8	25	1	122	0188A5AE	25	1	123
0188A584	25	1	128	0188A5AD	25	1	254
0188A5D2	25	1	249	0188A5AC	25	1	253
0188A599	25	1	187	0188A5AB	25	1	148
0188A598	25	1	186	0188A5DA	25	1	218
0188A583	25	1	136	0188A5E4	25	1	146
018BA4C9	25	1	181	0188A5E5	25	1	147
0188A5D0	25	1	255	0188A5D9	25	1	224
0188A594	25	1	230	0188A5B5	25	1	143
0188A582	25	1	135	0188A5BC	25	1	184
0188A5CB	25	1	233	0188A5BD	25	1	185
0188A5D8	25	1	222	0188A5E9	25	1	166
0188A5E7	25	1	145	0188A5CC	25	1	132
0188A581	25	1	134	0188A585	25	1	129
0188A5EA	25	1	167	0188A5D3	25	1	250
0188A5A9	25	1	248	0188A5E2	25	1	151
0188A5A6	25	1	209	0188A595	25	1	119
0188A580	25	1	243	0188A596	25	1	193
0188A5A0	25	1	216	0188A5B8	25	1	176
0188A59D	25	1	164	0188A590	25	1	245
0188A5C3	25	1	140	0188A5C9	25	1	229
0188A57F	25	1	157	0188A5A4	25	1	206
0188A5C0	25	1	239	0188A5E6	25	1	144
0188A5DE	25	1	191	0188A5A5	25	1	208
0188A5D4	25	1	251	0188A5CE	25	1	137
0188A57E	25	1	156	0188A5BF	25	1	241
0188A5C2	25	1	139	0188A572	25	1	237
0188A592	25	1	242	0188A59B	25	1	190

Codeword	Length (bits)	Run Length	Run Value
0188A5CD	25	1	133
0188A5BE	25	1	240
0188A5C7	25	1	131
0188A5CA	25	1	232
0188A5D5	25	1	252
0188A57B	25	1	171
0188A58D	25	1	205
0188A58C	25	1	204
0188A58B	25	1	118
0188A58A	25	1	214
018BA4C8	25	1	180
0188A5C5	25	1	126
0188A5FA	25	1	182
0188A5BB	25	1	175
0188A5C1	25	1	141
0188A5CF	25	1	138
0188A5B9	25	1	177
0188A5B6	25	1	153
0188A597	25	1	194
0188A5FE	25	1	160
0188A5D7	25	1	121
0188A5BA	25	1	174
0188A591	25	1	246
0188A5C6	25	1	130
0188A5DC	25	1	200
0188A57A	25	1	170
0188A59F	25	1	221
0188A5F9	25	1	196
0188A5B4	25	1	142
0188A5A7	25	1	210
0188A58F	25	1	199
0188A5FD	25	1	155
0188A5B7	25	1	154
0188A593	25	1	244
0188A59E	25	1	220
0188A5F8	25	1	195
0188A5FF	25	1	161
0188A5FC	25	1	231
0188A579	25	1	173
0188A5F7	25	1	226
03114BA2	26	1	116

C.2 Special Codewords

Special codewords shall be as listed in Table C.2. The band end marker shall be the last codeword in the codeblock for each highpass band.

Table C.2 – Special codewords

Description	Codeword	Length (bits)	Value
Band end marker	03114BA3	26	256

Annex D Decoding Process (Informative)

D.1 Decoding Overview

The decoding loop reads segments from the bitstream and extracts the tag and value from the segment:

1. If the tag corresponds to a codec state parameter, then the value is used to update the corresponding parameter in the codec state,

2. If the tag corresponds to a chunk, then:

    2.1. If the chunk is optional, then the payload is skipped.

    2.2. If the chunk is required, then the payload is processed.

        2.2.1. If the most significant byte of the tag corresponds to a large chunk and the chunk payload is a codeblock, then:

            2.2.1.1. If **SubbandNumber** is zero, then the lowpass band is decoded.

            2.2.1.2. If **SubbandNumber** is positive, then the highpass band is decoded.

3. After all bands in a wavelet have been decoded or computed, the inverse wavelet transform is used to compute the array of lowpass coefficients or component values at the next lower level in the wavelet sequence.

D.2 Initial Codec State

The decoder initializes the codec state with information from two sources:

(1) Initial values for the parameters in the codec state (Table B.2);

(2) Parameters passed to the decoder by the calling application.

Parameters passed to the decoder by the calling application override the initial values.

Note: Parameters passed to the decoder by the calling application can depend on information from the media container in which the VC-5 bitstream is embedded.

D.3 Memory Allocation

The parameters in the bitstream header provide sufficient information to allocate memory for decoding all channels (Section 8.7). The restriction that header parameters occur before the first codeblock allows a decoder to allocate memory before processing any codeblock. The **ImageWidth** and **ImageHeight** parameters bound the dimensions of the largest component array represented in the bitstream. A decoder can allocate one wavelet sequence based on the image dimensions in the bitstream header and reuse the memory allocated to the wavelet sequence for decoding each channel in the bitstream.

D.4 Entropy Decoding

An algorithm for decoding a highpass band is provided in Table D.1. The algorithm assumes that the two-dimensional highpass band array is allocated without padding at the end of each row so that the array can be accessed as a one-dimensional vector.

Table D.1 – Highpass band codeblock decoding algorithm

```
DecodeHighpassBand(array, width, height)
    index = 0
    count = width * height
    while (index < count)
        (length, value) = getrun()
        while (length > 0)
            array[index] = value
            index = index + 1
            length = length - 1
    (length, value) = getrun()
    assert(length == 0 && value = 0x03114BA3)
```

Entropy coding uses a single codebook for runs of zeros and non-zero coefficient magnitudes to eliminate branch instructions in the body of the decoding loop.

D.5 Lower Resolution Decoding

A decoder can omit one or more inverse horizontal and vertical wavelet transforms to yield a lower resolution component array if all of the following conditions are satisfied:

(1) The inverse horizontal wavelet transform that reconstructs the component array from the wavelet in level one is omitted,

(2) The inverse horizontal and vertical transforms that are omitted are a contiguous subset of the sequence of inverse horizontal and vertical transforms that reconstruct the component array,

(3) The values in the lower resolution proxy for the component array are adjusted to account for prescale shifts that were omitted.

D.6 Image Repacking Process

A decoder implementation can include an image repacking process that packs the decoded component arrays into an image format specified by the application.

D.7 Decoding Resource Requirements

The amount of memory required to decode a bitstream can be determined from the image dimensions and the number of channels in the bitstream. A decoder implementation can characterize its decoding capacity by specifying the maximum image width, image height, and number of channels (or the maximum product of these parameters).

The arithmetic precision required by the inverse wavelet transforms for each channel can be determined from the **LowpassPrecision**, **BitsPerComponent**, and **PrescaleShift** parameters.

Annex E Encoding Process (Informative)

E.1 Codec State Synchronization

The encoder inserts tag-value pairs into the bitstream so that the codec state contains the correct parameters for decoding each codeblock and reconstructing the component array for each channel.

The decoder initializes the codec state with default values (Section 9.2), modifies the default values with external parameters provided by the application (Section 9.3), and updates the parameters in the codec state with the values in tag-value pairs (Section 9.4) and codec state changes (Section 9.5).

The encoder initializes the codec state with the initial values used by the decoder and mimics the codec state changes performed by the decoder to keep the encoding and decoding state synchronized (Figure E.1).

The encoder can explicitly specify all parameters required for decoding a codeblock by inserting tag-value pairs for all parameters into the bitstream before the codeblock.

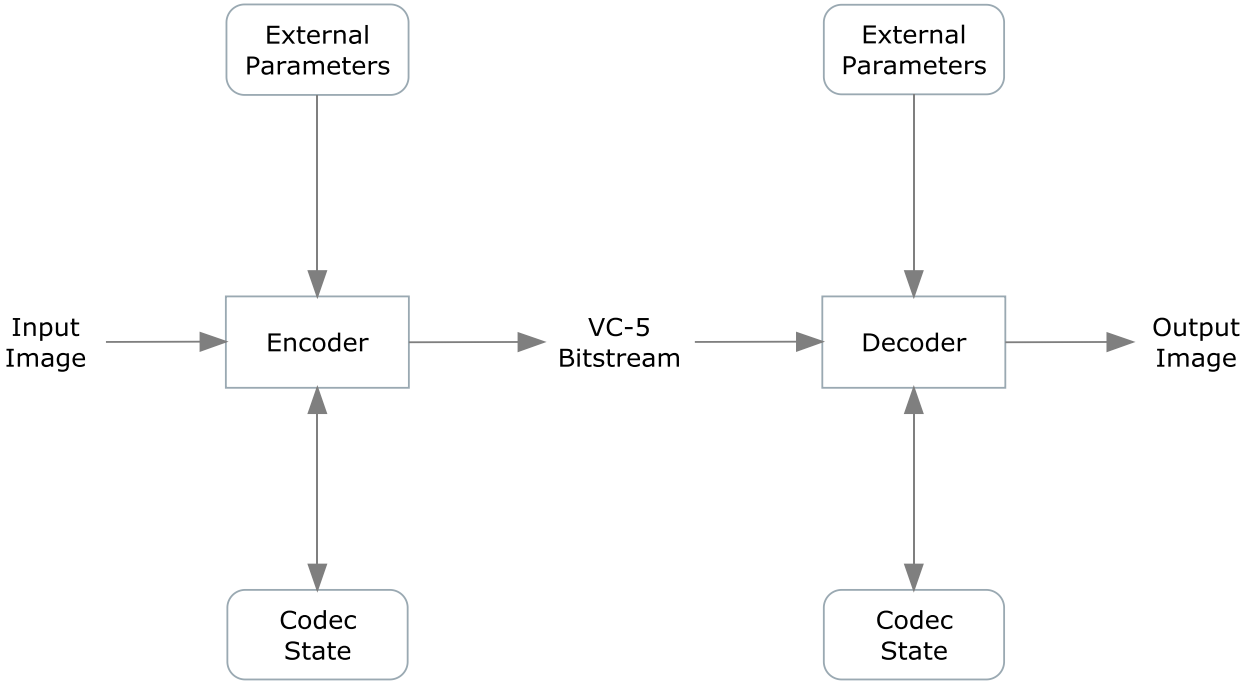

Figure E.1 – Synchronizing the codec state between the encoder and decoder

E.2 Image Processing

Processing performed prior to encoding the VC-5 bitstream is out of scope for this standard. The VC-5 bitstream represents the image data presented to the encoder regardless of the color space of the image data or processing applied to the image data prior to encoding.

E.3 Image Unpacking

Any image can be encoded as a VC-5 bitstream if an image unpacking process can output a set of component arrays that satisfy the requirements of this standard.

E.4 Prescale Shifts

If the value of **BitsPerComponent** is 12 bits or less, then the encoded quality can benefit from using the prescale shifts listed in Table E.1. These prescale shifts allow computation of the inverse wavelet transforms using 16-bit arithmetic.

The prescale shift at wavelet level k is applied to the values in the array input to the spatial wavelet transform at level k.

Table E.1 – Prescale shifts versus the number of bits per component value

	BitsPerComponent		
Wavelet Level	**8 bits**	**10 bits**	**12 bits**
1	0	0	0
2	0	0	2
3	0	2	2

E.5 Lowpass Transform

The lowpass transform is a 2-tap filter that computes the sum of each pair of input values:

$$L_i = X_{2i} + X_{2i+1}, \qquad i = 0, 1, \ldots, \frac{n}{2} - 1$$

where n is the number of values in the input. The lowpass transform does not require any special formulas along the border of the input image. The vertical lowpass wavelet filter is identical to the horizontal lowpass wavelet filter.

E.6 Highpass Transform

The highpass transform is a 6-tap filter that is applied to every contiguous sequence of six values at every other position in the filter input. It can be implemented using shift and add instructions.

In applying the highpass transform to a row (horizontal wavelet transform) or to a column (vertical wavelet transform), the first and last columns or rows, respectively, of the output can be computed using special formulas.

The value at output index 0 is calculated as follows:

$$H_0 = \mathrm{ash}(5X_0 - 11X_1 + 4X_2 + 4X_3 - X_4 - X_5 + 4, 3)$$

The values at output indices *i* = 1 to $n/2 - 2$ are calculated as follows:

$$H_i = X_{2i} - X_{2i+1} + \text{ash}(X_{2i+2} + X_{2i+3} - (X_{2i-2} + X_{2i-1}) + 4, 3), \qquad i = 1, 2, \ldots, \frac{n}{2} - 2$$

where n is the number of values in the result.

The value at output index $n/2 - 1$ is calculated as follows:

$$H_{n/2-1} = \text{ash}(-5X_{n-1} + 11X_{n-2} - 4X_{n-3} - 4X_{n-4} + X_{n-5} + X_{n-6} + 4, 3)$$

E.7 Quantization

The highpass wavelet coefficients are quantized by:

(1) Adding a prequant correction to the absolute value of each non-zero coefficient;

(2) Dividing the absolute value by the quantization value; and

(3) Restoring the sign.

Each highpass subband can be quantized by a different value for the **Quantization** parameter.

Quantization maps the coefficient c into the quantized value c' using the **Quantization** divisor q and prequant midpoint correction p and is described by the formula:

$$c' = \text{sgn}(c) \times \left\lfloor \frac{|c| + p}{q} \right\rfloor$$

Encoded quality can benefit from using a prequant midpoint correction p computed using the formula:

$$p = \max\left(\left\lfloor \frac{q}{2} \right\rfloor - 1, 0\right)$$

E.8 Companding

Quantization and companding reduce the range of non-zero coefficients to fit that range of coefficient magnitudes in the codebook.

The codebook has a limited number of entries for non-zero coefficients and it is possible that a quantized highpass coefficient can exceed the limit of the codebook. The maximum non-zero coefficient in the codebook is 255 (8 bits of precision). The value of a quantized coefficient magnitude should not exceed the maximum value of 1023 (10 bits of precision).

The companding curve reduces the range of non-zero coefficients to the range of values in the codebook.

The inverse companding curve is defined for the decoding process in Annex F. An encoder can compute a table that implements companding by using the inverse companding curve to determine the un-companded magnitude corresponding to an index in the companding table. Table entries can be interpolated by copying

the companded values into the interval of unfilled entries immediately after a table entry, creating a table that truncates companded values towards zero.

E.9 Entropy Coding

E.9.1 Non-Zero Coefficients

Negative coefficients are encoded using the codeword for the magnitude of the coefficient followed by the sign bit (Section 10.4.3). This is equivalent to extending the codebook for negative values and appending the sign bit.

E.9.2 Runs of Zeros

The encoding process can use the following algorithm for encoding runs of zeros:

(1) Set the remaining run count to the length of the run of zeros.

(2) Find the codeword for the longest encodable run of zeros that is less than or equal to the remaining run count.

(3) Output the codeword and decrease the remaining run count by the number of zeros represented by the codeword.

(4) Loop back to step (2) if the remaining run count is greater than zero.

E.10 Subband Encoding Order

For each channel, if the bitstream contains the wavelet subbands in order of wavelet level from highest to lowest (smallest wavelet to largest wavelet), then the decoder can invert the wavelet transforms as early as possible during decoding and this can reduce the amount of memory required for decoding.

Annex F Inverse Companding and Dequantization (Normative)

This section specifies how to calculate inverse companding and dequantization.

The decoder shall calculate the inverse companded value c' from the coefficient c as follows:

$$c' = \left\lfloor \frac{768 \times |c|^3}{255^3} \right\rfloor + |c|$$

The decoder shall multiply the inverse companded value c' by q and restore the sign:

$$c'' = \mathrm{sgn}(c) \times q \times c'$$

where q is the value of the **Quantization** parameter in the codec state.

The value c'' shall be used as the input to the inverse wavelet transform.

Annex G Additional Notation (Informative)

G.1 Arithmetic Operators

The arithmetic operators used in this document are described in Table G.1.

Table G.1 – Elementary arithmetic operators used in this document

a + b	Signed addition
a – b	Signed subtraction
a * b	Multiplication returning the low part of the result

G.2 Logical Operators

The result of evaluating a variable or expression used as an argument of a logical operator has Boolean type with the value:

(1) True if the variable or expression evaluates to a non-zero value,

(2) False if the variable or expression evaluates to a zero value.

The logical operators used in this document are described in Table G.2.

Table G.2 – Logical operators used in this document

a && b	True if both a and b are true, otherwise false

G.3 Relational Operators

Relational operators evaluate to a Boolean value. The relational operators used in this document are described in Table G.3.

Table G.3– Relational operators used in the document

a > b	True if a is greater than b
a < b	True if a is less than b
a == b	True if a is equal to b

G.4 Bitwise Operators

The bitwise operators are applied to an integer value and return an integer that is the same precision as the input. The bitwise operators used in this document are described in Table G.4

Table G.4 – Bitwise operators used in this document

a >> b	Arithmetic right shift of a by b bits with sign extension

G.5 Assignment Operator

The assignment operator replaces the value of the variable on the left side of the operator with the value of the expression on the right side of the operator after evaluation of the expression. The assignment operator is described in Table G.5.

Table G.5 – Assignment operators used in this document

a = b	Replace the value of a with the value of b

G.6 Operator Precedence

The precedence order of operators is defined in Table G.6.

Operators are listed in descending order of precedence. If several operators appear in the same line, they have equal precedence. When several operators of equal precedence appear at the same level in an expression, evaluation proceeds according to the associativity of the operator either from right to left or from left to right.

Table G.6 – Operator precedence

Operator	Operation	Associativity
()	Expression	Left to right
-	Unary minus	Left to right
+ , -	Addition, subtraction	Left to right
*, /	Multiplication, division	Left to right
>, <	Relational operator	Left to right
==	Equality	Left to right
=	Assignment	Right to left

G.7 Compound Statements

The following compound statements are used in the algorithm descriptions to define the decoding process.

An indented group of statements is a compound statement that is functionally equivalent to a single statement.

The looping compound statement

```
while (condition) statement
```

specifies repeated execution of `statement` while `condition` is true.

G.8 Elementary Functions

Each of the functions listed Table G.7 accepts a numerical input argument with unlimited precision, uses unlimited precision for internal calculations, and returns a result with unlimited precision.

Table G.7– Elementary functions operate on the argument x with unlimited precision

sgn(x)	Return 1 if x is greater than zero, -1 if x is less than zero, otherwise 0
floor(x)	Largest integer value that is less than or equal to x

G.9 Bitstream Functions

Functions for reading the bitstream that are used in this document are listed in Table G.8.

Table G.8 – Functions for sequentially reading the bitstream

getrun()	Return a tuple of two numbers: the value and the number of times that the value is repeated.

All bitstream functions that read bits begin reading at the current position in the bitstream and advance the bitstream position to the position immediately after the bits that were read.

Bibliography (Informative)

1. Robert Sedgewick, *Algorithms in C* (Addison-Wesley, 1990).

2. A. Zandi, J. D. Allen, E. L. Schwartz, and M. Boliek, CREW: Compression with Reversible Embedded Wavelets, *Data Compression Conference*, pp. 212-221, March 1995.