(19) DANMARK

(10) **DK/EP 3831065 T3**

(12) Oversættelse af europæisk patentskrift

Patent- og Varemærkestyrelsen

(51) Int.Cl.: ***H 04 N 19/33 (2014.01)*** ***H 04 N 19/91 (2014.01)*** ***H 04 N 19/93 (2014.01)***

(45) Oversættelsen bekendtgjort den: **2024-07-15**

(80) Dato for Den Europæiske Patentmyndigheds bekendtgørelse om meddelelse af patentet: **2024-04-10**

(86) Europæisk ansøgning nr.: **19752544.7**

(86) Europæisk indleveringsdag: **2019-08-01**

(87) Den europæiske ansøgnings publiceringsdag: **2021-06-09**

(86) International ansøgning nr.: **GB2019052166**

(87) Internationalt publikationsnr.: **WO2020025964**

(30) Prioritet: **2018-08-03 GB 201812708**
2018-08-03 GB 201812709
2018-08-03 GB 201812710
2019-03-20 GB 201903844
2019-03-23 GB 201904014
2019-03-29 GB 201904492
2019-04-15 GB 201905325

(84) Designerede stater: **AL AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HR HU IE IS IT LI LT LU LV MC MK MT NL NO PL PT RO RS SE SI SK SM TR**

(73) Patenthaver: **V-NOVA INTERNATIONAL LTD, 1 Sheldon Square, Paddington, London, W2 6TT, Storbritannien**

(72) Opfinder: **MEARDI, Guido, c/o V-Nova International Ltd 8th Floor 1 Sheldon , Square Paddington, London W2 6TT, Storbritannien**

(74) Fuldmægtig i Danmark: **Potter Clarkson A/S, Havnegade 39, 1058 København K, Danmark**

(54) Benævnelse: **ENTROPY CODING FOR SIGNAL ENHANCEMENT CODING**

(56) Fremdragne publikationer:
US-A- 5 402 123
US-A1- 2009 097 548
US-B1- 7 016 547
ANONYMOUS: "H.264 and MPEG-4 Video Compression, chapter 6, "H.264/MPEG4 Part 10", Iain E. Richardson", NOT KNOWN,, 17 October 2003 (2003-10-17), XP030001626,
"Working Draft of Low Complexity Enhancement Video Coding", 126. MPEG MEETING; 20190325 - 20190329; GENEVA; (MOTION PICTURE EXPERT GROUP OR ISO/IEC JTC1/SC29/WG11), , no. n18454 18 April 2019 (2019-04-18), XP030208724, Retrieved from the Internet: URL:http://phenix.int-evry.fr/mpeg/doc_end_user/documents/126_Geneva/wg11/w18454.zip N18454.docx [retrieved on 2019-09-26]
JEONG J ET AL: "Adaptive Huffman coding of 2-D DCT coefficients for image sequence compression", SIGNAL PROCESSING. IMAGE COMMUNICATION, ELSEVIER SCIENCE PUBLISHERS, AMSTERDAM, NL, vol. 7, no. 1, 1 March 1995 (1995-03-01), pages 1-11, XP004047118, ISSN: 0923-5965, DOI: 10.1016/0923-5965(94)00033-F

DESCRIPTION

Description

BACKGROUND

[0001] A hybrid backward-compatible coding technology has been previously proposed, for example in WO 2014/170819 and WO 2018/046940.

[0002] A method is proposed therein which parses a data stream into first portions of encoded data and second portions of encoded data; implements a first decoder to decode the first portions of encoded data into a first rendition of a signal; implements a second decoder to decode the second portions of encoded data into reconstruction data, the reconstruction data specifying how to modify the first rendition of the signal; and applies the reconstruction data to the first rendition of the signal to produce a second rendition of the signal.

[0003] An addition is further proposed therein in which a set of residual elements is useable to reconstruct a rendition of a first time sample of a signal. A set of spatio-temporal correlation elements associated with the first time sample is generated. The set of spatio-temporal correlation elements is indicative of an extent of spatial correlation between a plurality of residual elements and an extent of temporal correlation between first reference data based on the rendition and second reference data based on a rendition of a second time sample of the signal. The set of spatio-temporal correlation elements is used to generate output data.

[0004] Typical video coding technologies involve applying an entropy coding operation to output data. There is a need for a low complexity, simple and fast entropy coding scheme to apply data compression to the reconstruction data or residual elements of the above proposed technologies or other similar residual data.

[0005] The publication titled, "H.264 and MPEG-4 Video Compression, chapter 6, H.264/MPEG4 Part 10", by Iain E. Richardson, mentions that H.264 provides mechanisms for coding video that are optimised for compression efficiency and aim to meet the needs of practical multimedia communication applications.

[0006] US2009097548A1 describes scalable video coding techniques. WO2014170819A1 describes that computer processor hardware parses a data stream into first portions of encoded data and second portions of encoded data; implements a first decoder to decode the first portions of encoded data into a first rendition of a signal; implements a second decoder to decode the second portions of encoded data into reconstruction data, the reconstruction data specifying how to modify the first rendition of the signal; and applies the reconstruction data to

the first rendition of the signal to produce a second rendition of the signal.

SUMMARY OF THE INVENTION

[0007] According to the invention defined by the appended claims, there is provided a method of decoding a video signal, a decoding apparatus, a computer readable medium and an encoded bitstream.

[0008] According to an example not forming part of the claimed invention there is provided a method of encoding a video signal.

[0009] In this way the method of encoding provides for a low complexity, simple and fast solution to data compression of the residual data. The solution takes advantage of the unique characteristics of the residual data such as the relative occurrence of zero values, the likely grouping of the zero values based on a scan order of a transform process of generating the residual data (if a transform is used) and the relative variety of the data values yet their potential relative frequency/infrequency in the residual data.

[0010] The set of symbols may be sequential in the encoded bytestream. Counts of consecutive zero values may also be referred to as a run of zeros. The residual data upon which the run-length coding operation is performed may represent quantised residual data, preferably a quantised set of transform coefficients. The quantised set of transform coefficients may be ordered by layer, that is, by sets of coefficients of the same type, by plane, by level of quality, or by surface. These terms are further described and defined herein. The symbols may be sequential in a corresponding scan order of the transform operation such that the residual data can be easily matched to a reference reconstructed frame.

[0011] Preferably the run-length coding operation comprises: encoding non-zero data values of the residual data into at least a first type of symbol; and, encoding counts of consecutive zero values into a second type of symbol, such that the residual data is encoded as a sequence of symbols of different types. Thus, the residual data may be encoded as a sequence of symbols comprising a data value type and a run of zeros type. The types facilitate speed and ease of decoding at the decoder but also facilitate a further entropy coding operation as elaborated below. Structuring the data into a minimised set of fixed-length codes of different types is beneficial for subsequent steps.

[0012] In certain examples, the run-length coding operation comprises: encoding data values of the residual data into a first type of symbol and a third type of symbol, the first and third types of symbol each comprising a part of a data value, such that the parts can be combined at a decoder to reconstruct the data value. Each type of symbol may have a fixed size and may be a byte. Data values larger than a threshold or larger than a size available in a type of symbol can be easily transmitted or stored using the bytestream. Structuring the data into bytes or symbols not only facilitates decoding but also facilitates entropy coding of fixed length

symbols as set out below. Introducing the third type of symbol allows three types of symbols to be easily distinguished form one another at the decoder side. Where we refer to types of symbols, the terms blocks, bytes (where the symbol is a byte) contexts, or types could be used.

[0013] The run-length coding operation may comprise: comparing a size of each data value of the residual data to be encoded to a threshold; and, encoding each data value into the first type of symbol if the size is below the threshold and encoding a part of each data value into the first type of symbol and a part of each data value into the third type of symbol if the size is above the threshold.

[0014] If the size is above the threshold, the method may comprise setting a flag in the symbol of the first type of symbol indicating a part of the represented data value is encoded into a further symbol of the third type of symbol. The flag may be an overflow flag or overflow bit and may in certain examples be the least significant bit of the symbol or byte. Where the least significant bit of the symbol is a flag, the data value or part of the data value may be comprised in the remaining bits of the byte. Setting the overflow bit as the least significant bit facilitates ease of combination with subsequent symbols.

[0015] The method may further comprise inserting a flag in each symbol indicating a type of symbol encoded next in the run-length encoded bytestream. The flag may be an overflow bit as described above or may further be a run flag or run bit indicating whether or not the next symbol comprises a run of zeros or a data value or a part of a data value. Where the flag represents if the next symbol comprises a run (or a count of consecutive zeros) the flag may be a bit of the symbol, preferably a most significant bit of the symbol. The count of consecutive zeros or the data value may be comprised in the remaining bits of the symbol. Thus, the 'run of zeros' or second type of symbol comprises 7 available bits of the byte for a count and the `data' or first type of symbol comprises 6 available bits for a data value where the overflow bit is not set and 7 available bits for a data value where the overflow bit is set.

[0016] In sum, the flag may be different for each type of symbol and may indicate the type of symbol next in the stream.

[0017] The method further comprises applying a further entropy coding operation to the set of symbols produced by the run-length encoded operation. Thus, fixed length symbols deliberately produced by the structure of the run-length coding operation can be converted into variable length codes to reduce the overall size of the data. The run-length coding structure is designed in order to create fixed length symbols of high frequency to facilitate an improved further entropy coding operation and an overall reduction in data size. The further entropy coding operation takes advantage of the probability or frequency of a symbol occurring in the bytestream created by the run-length coding operation.

[0018] The further entropy coding operation is a Huffman coding operation or an arithmetic coding operation. Preferably the method further comprises applying a Huffman coding

operation to the set of symbols to generate Huffman encoded data comprising a set of codes representing the run-length encoded bytestream. The Huffman coding operation receives as inputs symbols of the run-length coded bytestream and outputs a plurality of variable-length codes in a bitstream. Huffman coding is a coding technique that can effectively reduce fixed length codes. The structure of the run-length coded bytes means that symbols of the two types are highly likely to be replicated frequently which means that only a few bits will be needed for each recurring symbol. Across a plane, the same data value is likely to be replicated (particularly where the residuals are quantised) and thus the variable length code may be small (but repeated).

[0019] Huffman coding is also well optimised for software implementation (as anticipated here) and is computationally efficient. It uses minimal memory and has faster decoding when compared to other entropy coding techniques, as the processing steps involve merely stepping through a tree. Computational benefits arise from the shape and depth of the tree created by the specifically designed run-length coding operation.

[0020] The coding operations may be performed on a group of coefficients, that is, a layer, on a plane of a frame, on a level of quality, or on a whole surface or frame. That is, the statistics or parameters of each operation may be determined based on the group of data values and zero data values to be encoded or the symbols representing those values.

[0021] The Huffman coding operation may be a canonical Huffman coding operation, such that the Huffman encoded data comprises a code length for each unique symbol of the set of symbols, the code length representing a length of a code used to encode a corresponding symbol. Canonical Huffman encoding facilitates a reduction in coding parameters that need to be signalled between the encoder and the decoder in metadata by ensuring that identical code lengths are attached to symbols in a sequential manner. The codebook for the decoder can be inferred and only the code lengths need to be sent. Thus, the canonical Huffman encoder is particularly efficient for shallow trees where there are a large number of different data values.

[0022] The method comprises comparing a data size of at least a portion of the run-length encoded bytestream to a data size of at least a portion of the Huffman encoded data; and, outputting the run-length encoded bytestream or the Huffman encoded data in an output bitstream based on the comparison. Each block or section of the input data is thus selectively sent to reduce the overall data size ensuring the decoder can easily differentiate between the types of encoding. The difference may be within a threshold or tolerance such that although one scheme may result in smaller data, computational efficiencies may mean that one scheme is preferred.

[0023] To distinguish between schemes, the method may further comprise adding a flag to configuration metadata accompanying the output bitstream indicating if the bitstream represents the run-length encoded bytestream or the Huffman encoded data. This is a quick and efficient manner of signalling. Alternatively, the decoder may identify the scheme used from the structure of the data, for example, the Huffman coded data may have a header part

and a data part while the run-length coded data may comprise only a data part and thus the decoder may be able to distinguish between the two schemes.

[0024] The Huffman coding operation may comprise generating a separate frequency table for symbols of the first type of symbol and symbols of the second type of symbol. Further, a separate Huffman coding operation may be applied for each type of symbol. These concepts facilitate particular efficiency of the variable length coding scheme. For example, where the same data value is replicated across a plane, as is likely in video coding where a scene may have similar colours or errors/enhancements, only a few codes may be needed for these values (where the codes are not distorted by the 'run of zeros' type of symbol). Thus, this approach is particularly advantageous for use in conjunction with the run-length coding operation above. As noted above, where the values are quantised, the symbols there may be replicated and may not be too many different values within the same frame or group of coefficients.

[0025] Using a different frequency table for each type of symbol provides a particular benefit in the Huffman coding operation following the specifically designed run-length coding operation.

[0026] The method may thus comprise identifying a type of symbol to be encoded next, selecting a frequency table based on the type of next symbol, decoding the next symbol using the selected frequency table and outputting the encoded symbol in sequence. A frequency table may correspond to a unique codebook.

[0027] The method may comprise generating a stream header which may comprise an indication of the plurality of code lengths, such that a decoder can derive code lengths and respective corresponding symbols for the canonical Huffman decoding operation. The code lengths may for example be the length of each code used to encode a particular symbol. A code length therefore has a corresponding code and a corresponding symbol. The stream header may be a first type of stream header and further comprises an indication of the symbol associated with a respective code length of the plurality of code lengths. Alternatively the stream header may be a second type of stream header and the method may further comprise ordering the plurality of code lengths in the stream header based on a predetermined order of symbols corresponding to each of the code lengths, such that the code lengths can be associated with a corresponding symbol at the decoder. Each type will provide efficient signalling of the coding parameters depending on the symbols and lengths to be decoded. The code lengths may be signalled as a difference between the length and another length, preferably a signalled minimum code length.

[0028] The second type of stream header may further comprise a flag indicating a symbol in the predetermined order of the possible set of symbols does not exist in the set of symbols in the run-length encoded bytestream. In this way only lengths needed need to be included in the header.

[0029] The method may further comprise comparing a number of unique codes in the Huffman

encoded data to a threshold and generating the first type of stream header or the second type of stream header based on the comparison.

[0030] The method may further comprise comparing a number of non-zero symbols or data symbols to a threshold and generating the first type of stream header or the second type of stream header based on the comparison.

[0031] According to an aspect of the invention there is provided a method of decoding a video signal according to appended claim 1.

[0032] The run-length coding operation comprises identifying symbols of a first type of symbol representing non-zero data values of the residual data; identifying symbols of a second type of symbol representing counts of consecutive zero values, such that a sequence of symbols of different types is decoded to generate the residual data; and, parsing the set of symbols according the respective type of each symbol.

[0033] The run-length coding operation comprises identifying symbols of a first type of symbol and symbols of a third type of symbol, the first and third types of symbol each representing a part of a data value; parsing a symbol of the first type of symbol and a symbol of the third type of symbol to derive parts of a data value; and, combining the derived parts of the data value into a data value.

[0034] The method may further comprise: retrieving an overflow flag from a symbol of the first type of symbol indicating if a part of a data value of the symbol of the first type of symbol is comprised in a subsequent third type of symbol.

[0035] The method may further comprise retrieving a flag from each symbol indicating a subsequent type of symbol to be expected in the set of symbols.

[0036] An initial symbol may be assumed to be the first type of symbol, such that the initial symbol is parsed to derive at least a part of a data value.

[0037] The Huffman coding operation may be a canonical Huffman coding operation.

[0038] The method may further comprise iteratively retrieving a flag from a decoded symbol indicating a subsequent type of symbol expected to be derived from the bitstream and applying the Huffman coding operation to the bitstream to derive a subsequent symbol based on a set of coding parameters associated with the subsequent type of symbol expected to be derived from the bitstream.

[0039] The method may further comprise: retrieving a stream header comprising an indication of a plurality of code lengths to be used for the canonical Huffman coding operation; associating each code length with a respective symbol in accordance with the canonical Huffman coding operation; and, identifying a code associated with each symbol based on the

code lengths according to the canonical Huffman coding operation. Thus, the codes may be associated with sequential symbols where the code lengths are identical.

[0040] The stream header may comprise an indication of a symbol associated with a respective code length of the plurality of code lengths and the step of associating each code length with a symbol comprises associating each code length with the corresponding symbol in the stream header.

[0041] The step of associating each code length with a respective symbol comprises associating each code length with a symbol of a set of predetermined symbols in accordance with an order in which each code length was retrieved.

[0042] The method may further comprise not associating a symbol of the set of predetermined symbols with a respective code length where a flag of the stream header indicates no code length exists in the stream header for that symbol.

[0043] According to an example not falling under the scope ofthe claimed invention there is provided a method of encoding a video signal.

[0044] The method comprises: receiving an input frame; processing the input frame to generate residual data, the residual data enabling a decoder to reconstruct the input frame from a reference reconstructed frame; and, applying a Huffman coding operation to the residual data, wherein the Huffman operation comprises generating an encoded bitstream comprising a set of codes encoding a set of symbols representing the residual data.

[0045] According to a further aspect of the invention there is provided a decoding apparatus according to appended claim 2.

[0046] A further aspect of the invention provides a computer readable storage medium according to appended claim 3.

[0047] According to the invention there is provided an encoded bitstream according to appended claim 4.

DETAILED DESCRIPTION

[0048] Examples of systems and methods in accordance with the invention and with examples useful for understanding the invention will now be described with reference to the accompanying drawings, in which:-

Figure 1 shows a high-level schematic of an encoding process;

Figure 2 shows a high-level schematic of a decoding process;

Figure 3 shows a high-level schematic of a process of creating residual data;

Figure 4 shows a high-level schematic of a process of creating further residual data at a different level of quality;

Figure 5 shows a high-level schematic of a process of reconstructing a frame from residual data;

Figure 6 shows a hierarchical data structure;

Figure 7 shows a further hierarchical data structure;

Figure 8 shows an example encoding process;

Figure 9 shows an example decoding process;

Figure 10 shows a structure of a first data symbol;

Figure 11 shows a structure of a second data symbol;

Figure 12 shows a structure of a run symbol;

Figure 13 shows a structure of a first stream header;

Figure 14 shows a structure of a second stream header;

Figure 15 shows a structure of a third stream header;

Figure 16 shows a structure of a fourth stream header;

Figures 17A-17E show Huffman trees; and,

Figure 18 shows a run-length coding state machine.

[0049] The present invention relates to methods. In particular, the present invention relates to methods for decoding signals. Processing data may include, but is not limited to, obtaining, deriving, outputting, receiving and reconstructing data.

[0050] The coding technology discussed herein is a flexible, adaptable, highly efficient and computationally inexpensive coding format which combines a video coding format, a base codec, (e.g. AVC, HEVC, or any other present or future codec) with an enhancement level of coded data, encoded using a different technique.

[0051] The technology uses a down-sampled source signal encoded using a base codec to form a base stream. An enhancement stream is formed using an encoded set of residuals which correct or enhance the base stream for example by increasing resolution or by increasing frame rate. There may be multiple levels of enhancement data in a hierarchical

structure. It is worth noting that typically the base stream is expected to be decodable by a hardware decoder while the enhancement stream is expected to be suitable for software processing implementation with suitable power consumption.

[0052] Methods and systems for efficiently transmitting and storing the enhancement encoded information are needed.

[0053] It is important that any entropy coding operation used in the new coding technology is tailored to the specific requirements or constraints of the enhancement stream and is of low complexity. Such requirements or constraints include: the potential reduction in computational capability resulting from the need for software decoding of the enhancement stream; the need for combination of a decoded set of residuals with a decoded frame; the likely structure of the residual data, i.e. the relatively high proportion of zero values with highly variable data values over a large range; the nuances of the input quantized block of coefficients; and, the structure of the enhancement stream being a set of discrete residual frames separated into planes, layers, etc. The entropy coding must also be suitable for multiple levels of enhancement within the enhancement stream.

[0054] Investigation by the inventors has established that contemporary entropy coding schemes used in video, such as context-based adaptive binary arithmetic coding (CABAC) or Context-adaptive variable-length coding (CAVLC), are unlikely to be appropriate. For example, predictive mechanisms may be unnecessary or may not yield sufficient benefits to compensate for their computational burdens given the structure of the input data. Moreover, arithmetic coding is generally computationally expensive and implementation in software is often undesirable.

[0055] Note that the constraints placed on the enhancement stream mean that a simple and fast entropy coding operation is essential to enable the enhancement stream to effectively correct or enhance individual frames of the base decoded video. Note that in some scenarios the base stream is also being decoded substantially simultaneously before combination, putting a strain on resources.

[0056] This present document preferably fulfils the requirements of the following ISO/IEC documents: "Call for Proposals for Low Complexity Video Coding Enhancements" ISO/IEC JTC1/SC29/WG11 N17944, Macao, CN, Oct. 2018 and "Requirements for Low Complexity Video Coding Enhancements" ISO/IEC JTC1/SC29/WG11 N18098, Macao, CN, Oct. 2018. Moreover, approaches described herein may be incorporated into PERSEUS® products as supplied by V-Nova International Ltd.

[0057] The general structure of the proposed encoding scheme in which the presently described techniques can be applied, uses a down-sampled source signal encoded with a base codec, adds a first level of correction data to the decoded output of the base codec to generate a corrected picture, and then adds a further level of enhancement data to an up-sampled version of the corrected picture.

[0058] Thus, the streams are considered to be a base stream and an enhancement stream. It is worth noting that typically the base stream is expected to be decodable by a hardware decoder while the enhancement stream is expected to be suitable for software processing implementation with suitable power consumption.

[0059] This structure creates a plurality of degrees of freedom that allow great flexibility and adaptability to many situations, thus making the coding format suitable for many use cases including Over-The-Top (OTT) transmission, live streaming, live Ultra High Definition (UHD) broadcast, and so on.

[0060] Although the decoded output of the base codec is not intended for viewing, it is a fully decoded video at a lower resolution, making the output compatible with existing decoders and, where considered suitable, also usable as a lower resolution output.

[0061] In general residuals refer to a difference between a value of a reference array or reference frame and an actual array or frame of data. It should be noted that this generalised example is agnostic as to the encoding operations performed and the nature of the input signal. Reference to "residual data" as used herein refers to data derived from a set of residuals, e.g. a set of residuals themselves or an output of a set of data processing operations that are performed on the set of residuals.

[0062] In certain examples described herein, a number of encoded streams may be generated and sent independently to a decoder. That is, in example methods of encoding a signal, the signal may be encoded using at least two levels of encoding. A first level of encoding may be performed using a first encoding algorithm and a second level may be encoded using a second encoding algorithm. The method may comprise: obtaining a first portion of a bitstream by encoding the first level of the signal; obtaining a second portion of a bitstream by encoding the second level of the signal; and sending the first portion of the bitstream and the second portion of the bytestream as two independent bitstreams.

[0063] It should be noted that the entropy coding technique proposed herein is not limited to the multiple LoQs proposed in the Figures but provides utility in any residual data used to provide enhancement or correction to a frame of a video reconstructed from an encoded stream encoded using a legacy video coding technology such as HEVC. For example, the utility of the coding technique in encoding residual data of different LoQs is particularly beneficial.

[0064] Note, we provide examples of encoding and decoding in Figures 1 to 5 of schemes in which the entropy coding techniques provided herein may provide utility, but it will be understood that the coding technique may be generally utilised to encode residual data.

[0065] Certain examples described herein relate to a generalised encoding and decoding process that provides a hierarchical, scaleable flexible coding technology. The first portion of a

bitstream or first independent bitstream may be decoded using a first decoding algorithm, and the second portion of the bitstream or second or independent bitstream may be decoded using a second decoding algorithm. The first decoding algorithm is capable of being decoded by a legacy decoder using legacy hardware.

[0066] Returning to the initial process described above providing a base stream and two levels of enhancement within an enhancement stream, an example of a generalised encoding process is depicted in the block diagram of Figure 1. An input full resolution video 100 is processed to generate various encoded streams 101, 102, 103. A first encoded stream (encoded base stream) is produced by feeding a base codec (e.g., AVC, HEVC, or any other codec) with a down-sampled version of the input video. The encoded base stream may be referred to as the base layer or base level. A second encoded stream (encoded level 1 stream) is produced by processing the residuals obtained by taking the difference between the reconstructed base codec video and the down-sampled version of the input video. A third encoded stream (encoded level 0 stream) is produced by processing the residuals obtained by taking the difference between an up-sampled version of a corrected version of the reconstructed base coded video and the input video.

[0067] A down-sampling operation may be applied to the input video to produce a down-sampled video to be encoded by a base codec. The down-sampling can be done either in both vertical and horizontal directions, or alternatively only in the horizontal direction.

[0068] Each enhancement stream encoding process may not necessarily include an upsampling step. In Figure 1 for example, the first enhancement stream is conceptually a correction stream while the second enhancement stream is up-sampled to provide a level of enhancement.

[0069] Looking at the process of generating the enhancement streams in more detail, to generate the encoded Level 1 stream, the encoded base stream is decoded 114 (i.e. a decoding operation is applied to the encoded base stream to generate a decoded base stream). The difference between the decoded base stream and the down-sampled input video is then created 110 (i.e. a subtraction operation is applied to the down-sampled input video and the decoded base stream to generate a first set of residuals).

[0070] Here the term residuals is used in the same manner as that known in the art, that is, the error between a reference frame a desired frame. Here the reference frame is the decoded base stream and the desired frame is the down-sampled input video. Thus, the residuals used in the first enhancement level can be considered as a corrected video as they 'correct' the decoded base stream to the down-sampled input video that was used in the base encoding operation.

[0071] Again, the later described entropy coding operation is suitable for any residual data, e.g. any data associated with a set of residuals.

[0072] The difference is then encoded 115 to generate the encoded Level 1 stream 102 (i.e. an encoding operation is applied to the first set of residuals to generate a first enhancement stream).

[0073] As noted above, the enhancement stream may comprise a first level of enhancement 102 and a second level of enhancement 103. The first level of enhancement 102 may be considered to be a corrected stream. The second level of enhancement 103 may be considered to be a further level of enhancement that converts the corrected stream to the original input video.

[0074] The further level of enhancement 103 is created by encoding a further set of residuals which are the difference 119 between an up-sampled 117 version of a decoded 118 level 1 stream and the input video 100.

[0075] As noted, an up-sampled stream is compared to the input video which creates a further set of residuals (i.e. a difference operation is applied to the up-sampled re-created stream to generate a further set of residuals). The further set of residuals are then encoded 121 as the encoded Level 0 enhancement stream (i.e. an encoding operation is then applied to the further set of residuals to generate an encoded further enhancement stream).

[0076] Thus, as illustrated in Figure 1 and described above, the output of the encoding process is a base stream 101 and one or more enhancement streams 102, 103 which preferably comprise a first level of enhancement and a further level of enhancement.

[0077] A corresponding generalised decoding process is depicted in the block diagram of Figure 2. The decoder receives the three streams 101, 102, 103 generated by the encoder together with headers containing further decoding information. The encoded base stream is decoded by a base decoder corresponding to the base codec used in the encoder, and its output is combined with the decoded residuals obtained from the encoded level 1 stream. The combined video is up-sampled and further combined with the decoded residuals obtained from the encoded level 0 stream.

[0078] In the decoding process, the decoder may parse the headers (global config, picture config, data block) and configure the decoder based on those headers. In order to re-create the input video, the decoder may decode each of the base stream, the first enhancement stream and the further enhancement stream. The frames of the stream may be synchronised and then combined to derive the decoded video.

[0079] In each of Figures 1 and 2, the level 0 and level 1 encoding operations may include the steps of transformation, quantization and entropy encoding. Similarly, at the decoding stage, the residuals may be passed through an entropy decoder, a de-quantizer and an inverse transform module. Any suitable encoding and corresponding decoding operation may be used. Preferably however, the level 0 and level 1 encoding steps may be performed in software.

[0080] In summary, the methods and apparatuses herein are based on an overall algorithm which is built over an existing encoding and/or decoding algorithm (such as MPEG standards such as AVC/H.264, HEVC/H.265, etc. as well as non-standard algorithm such as VP9, AV1, and others) which works as a baseline for an enhancement layer which works accordingly to a different encoding and/or decoding algorithm. The idea behind the overall algorithm is to hierarchically encode/decode the video frame as opposed to the use blockbased approaches as used in the MPEG family of algorithms. Hierarchically encoding a frame includes generating residuals for the full frame, and then a decimated frame and so on.

[0081] The video compression residual data for the full-sized video frame may be referred to as LoQ-0 (e.g. 1920 x 1080 for an HD video frame), while that of the decimated frame may be referred to as LoQ-x, where x denotes the number of hierarchical decimations. In the described examples of Figures 1 and 2, the variable x has a maximum value of 1 and hence there are 2 hierarchical levels for which compression residuals will be generated.

[0082] Figure 3 illustrates an example of how LoQ-1 could be generated at an encoding device. In the present figure the overall algorithm and methods are described using an AVC/H.264 encoding/decoding algorithm as baseline algorithm, but it is understood that other encoding/decoding algorithms can be used as baseline algorithms without any impact to the way the overall algorithm works.

[0083] It will of course be understood that the blocks of Figure 3 are merely examples of how the broad concepts could be implemented.

[0084] The diagram of Figure 3 shows the process of generating entropy encoded residual data for the LoQ-1 hierarchy level. In the example, the first step is to decimate the incoming uncompressed video by a factor of 2. This decimated frame is then passed through a base coding algorithm (in this case, an AVC/H.264 coding algorithm) where an entropy encoded reference frame is then generated and stored. A decoded version of the encoded reference frame is then generated, and the difference between the decoded reference frame and the decimated frame (LoQ-1 residuals) will form the input to the transform block.

[0085] The transform (e.g., a Hadamard-based transform in this illustrated example) converts this difference into 4 components (or planes), namely A (average), H (horizontal), V (vertical) and D (diagonal). These components are then quantized via the use of variables called 'step-widths' (e.g. a residual value may be divided by the step-width and the nearest integer value selected). A suitable entropy encoding process is the subject of this disclosure and described in detail below. These quantized residuals are then entropy encoded in order to remove any redundant information. The quantized encoded coefficients or components (Ae, He, Ve and De) are then placed within a serial stream with definition packets inserted at the start of the stream, this final stage is accomplished using a file serialization routine. Packet data may include such information such as the specification of the encoder, type of up-sampling to be employed, whether or not A and D planes are discarded, and other information to enable a decoder to decode the streams.

[0086] Both the reference data (half-sized baseline entropy encoded frame) and the entropy encoded LoQ-1 residual data may be buffered, transmitted or stored away for use by the decoder during the reconstruction process.

[0087] In order to produce LoQ-0 residual data, the quantized output is branched off and reverse quantization and transform processes are performed on it in order to reconstruct the LoQ-1 residuals, which are then added to the decoded reference data (encoded & decoded) in order to get back a video frame closely resembling the originally decimated input frame.

[0088] This process ideally mimics the decoding process and hence the originally decimated frame is not used.

[0089] Figure 4 illustrates an example of how LoQ-0 could be generated at an encoding device. In order to derive the LoQ-0 residuals, the reconstructed LoQ-1 sized frame is derived as described in the previous section.

[0090] The next step is to perform an up-sampling of the reconstructed frame to full size (by 2). At this point various algorithms may be used to enhance the up-sampling process such as nearest, bilinear, sharp or cubic algorithms. This reconstructed full-size frame termed 'predicted' frame is then subtracted from the original uncompressed video input which creates some residuals (LoQ-1 residuals).

[0091] Similar to the LoQ-1 process, the difference is then transformed, quantized, entropy encoded and file serialized which then forms the third and final data. This may be buffered, transmitted or stored away for later use by the decoder. As it can be seen, a component called "predicted average" (described below) can be derived using the upsampling process, and used in place of the A (average) component to further improve the efficiency of the coding algorithm.

[0092] Figure 5 shows schematically how the decoding process could be performed in a particular example. The entropy encoded data, the LOQ-1 entropy encoded residual data and the LOQ-0 entropy encoded residual data (for example as fileserialized encoded data). The entropy encoded data include the reduced-size (e.g., half-size, i.e. with dimensions W/2 and H/2 with respect to the full frame having dimensions Wand H) encoded base.

[0093] The entropy encoded data are then decoded using the decoding algorithm corresponding to the algorithm which has been used to encode those data (in the example, an AVC/H.264 decoding algorithm). At the end of this step, a decoded video frame, having a reduced-size (e.g., half-size) is produced (indicated in the present example as an AVC/H.264 video).

[0094] In parallel, the LoQ-1 entropy encoded residual data are decoded. As discussed above, the LoQ-1 residuals are encoded using four coefficients or components (A, V, H and D) which, as shown in this Figure, have a dimension of one quarter of the full frame dimension, namely

W/4 and H/4. This is because, as discussed in previous patent applications US 13/893,669 and PCT/EP2013/059847, the four components contain all the information relative to the residuals and are generated by applying a 2x2 transform kernel to the residuals whose dimension, for LoQ-1, would be W/2 and H/2, i.e., the same dimension of the reduced-size entropy encoded data. The four components are entropy decoded, then de-quantized and finally transformed back into the original LoQ-1 residuals by using an inverse transform (in this case, a 2x2 Hadamard inverse transform).

[0095] The decoded LoQ-1 residuals are then added to the decoded video frame to produce a reconstructed video frame at reduced-size (in this case, half-size), identified as Half-2D size reconstruction.

[0096] This reconstructed video frame is then up-sampled to bring it up to full resolution (so, in this example, from half width (W/2) and half height (H/2) to full width (W) and full height (H)) using an up-sampling filter such as bilinear, bicubic, sharp, etc. The up-sampled reconstructed video frame will be a predicted frame (full-size, Wx H) to which the LoQ-0 decoded residuals are then added.

[0097] In particular, the LoQ-0 encoded residual data are decoded. As discussed above, the LoQ-0 residuals are encoded using four coefficients or components (A, V, H and D) which, as shown in this figure, have a dimension of half the full frame dimension, namely W/2 and H/2. This is because, as discussed in previous patent applications US 13/893,669 and PCT/EP2013/059847, the four components contain all the information relative to the residuals and are generated by applying a 2x2 transform kernel to the residuals whose dimension, for LoQ-0, would be W and H, i.e., the same dimension of the full frame. The four components are entropy decoded (see process below), then de-quantized and finally transformed back into the original LoQ-0 residuals by using an inverse transform (in this case, a 2x2 Hadamard inverse transform).

[0098] The decoded LoQ-0 residuals are then added to the predicted frame to produce a reconstructed full video frame - the output frame.

[0099] The data structure is represented in an exemplary manner in Figure 6. As discussed, the above description has been made with reference to specific sizes and baseline algorithms, but the above methods apply to other sizes and or baseline algorithms, and the above description is only given by way of example of the more general concepts described.

[0100] In the encoding/decoding algorithm described above, there are typically 3 planes (e.g., YUV or RGB), with two level of qualities LoQs which are described as LoQ-0 (or top level, full resolution) and LoQ-1 (or lower level, reduced-size resolution such as half resolution) in every plane. Each plane may represent a different colour component for a video signal.

[0101] Every LoQ contains four components or layers, namely A, H, V and D. This gives a total of 3x2x4 = 24 surfaces of which 12 are full size (e.g., WxH) and 12 are reduced-size (e.g.,

W/2xH/2). Each layer may comprise coefficient values for a particular one of the components, e.g. layer A may comprise a top-left A coefficient value for every 2x2 block of the input image or frame.

[0102] Figure 7 illustrates an alternative view of a proposed hierarchical data structure.

[0103] The encoded data is separated into chunks. Each payload is ordered hierarchically into chunks. That is, each payload is grouped into planes, then within each plane each level is grouped into layers and each layer comprises a set of chunks for that layer. Level represents each level of enhancement (first or further) and layer represents a set of transform coefficients.

[0104] The method may comprise retrieving chunks for two levels of enhancement for each plane. The method may comprise retrieving 16 layers for each level (e.g. if a 4x4 transform is used). Thus, each payload is ordered into a set of chunks for all layers in each level and then the set of chunks for all layers in the next level of the plane. Then the payload comprises the set of chunks for the layers of the first level of the next plane and so on.

[0105] Thus, the method may decode headers and output entropy encoded coefficients grouped by plane, levels and layers belonging to the picture enhancement being decoded. Thus, the output according to the invention is (nPlanes)x(nLevel)x(nLayer) array surfaces with elements surfaces[nPlanes][nLevel][nLayer].

[0106] Note that the entropy coding techniques provided herein may be performed on each group, that is, may be performed per surface, per plane, per level (LoQ), or per layer. Note that the entropy coding may be performed on any residual data and not necessarily quantised and/or transformed coefficients.

[0107] As shown Figures 4 and 5, the proposed encoding and decoding operations include an entropy coding stage or operation. It is proposed in the following that the entropy coding operation comprises either a run-length coding component or a run length coding component (RLE) and a Huffman coding component. These two components may interrelate to provide additional benefits.

[0108] As noted and as illustrated in Figure 8, the input of the entropy encoder is a surface (e.g. residual data derived from a quantized set of residuals as illustrated in this example) and the output of the process is an entropy encoded version of the residuals (Ae, He, Ve, De). However as above, it should be noted that the entropy coding may be performed on any residual data and not necessarily quantised and/or transformed coefficients. Figure 9 illustrates a corresponding high-level decoder with inverse inputs and outputs. That is, the entropy decoder takes as inputs entropy encoded residuals (Ae, He, Ve, De) and outputs residual data (e.g. quantised residuals in this illustrated example).

[0109] Note that Figure 8 generally introduces RLE encoding before Huffman encoding. In aspects not falling under the scope of the claimed invention, we note that run-length coding

may be provided without Huffman coding and in comparative cases run-length coding and Huffman coding may not be performed (either replaced by alternative lossless entropy coding or no entropy coding performed at all). For example, in a production example with a reduced emphasis on data compression, an encoding pipeline may not comprise entropy coding but the benefits of residuals may lie in layered storage for distribution.

[0110] A comparative run-length encoder (RLE) may compress data by encoding sequences of the same data value as a single data value and a count of that data value. For example, the sequence 555500022 may be encoded as (4,5)(3,0)(2,2). That is, there is a run of four 5s, a run of three 0s followed by a run of two 2s.

[0111] To encode the residual data there is proposed a modified RLE coding operation. To take advantage of the structure of the residual data, it is proposed to encode only runs of zeros. That is, each value is sent as a value with each zero is sent as a run of zeros. The modified RLE coding operations described herein may be used, for example, to provide entropy encoding and/or decoding in one or more LoQ layers, e.g. as shown in Figures 1 to 5.

[0112] Thus, the entropy coding operation comprises parsing the residual data and encoding zero values as a run of consecutive zeros.

[0113] As noted above, to provide an additional level of data compression (which may be lossless), the entropy coding operation further applies a Huffman coding operation to the run-length coded data.

[0114] Huffman coding and run-length coding have been previously paired together, in for example facsimile coding (e.g. ITU Recommendations T.4 and T.45) and the JPEG File Interchange Format, but have been superseded over the last 30 years. The present description proposes techniques and methods to implement Huffman coding in combination with RLE coding, techniques and methods to efficiently exchange data and metadata between an encoder and decoder, and techniques and methods to reduce the overall data size of such a combination when used to code residual data.

[0115] A Huffman code is an optimal prefix code which is used for data compression (e.g. lossless compression). A prefix code is a code system for which there is no code work in the system that is the prefix of any other code. That is, a Huffman coding operation takes a set of input symbols and converts each input symbol into a corresponding code. The code chosen is based on the frequency of which each symbol appears in the original data set. In this way, smaller codes can be associated with symbols which occur more frequently so as to reduce the overall size of the data.

[0116] To facilitate Huffman coding, the input data is preferably structured as symbols. The output of the run-length coding operation is preferably structured to be a stream of bytes of encoded data.

[0117] In one example, the run-length coding operation outputs two types of bytes or symbols. A first type of symbol is a value of a non-zero pixel and a second type of symbol is a run of consecutive zero values. That is, an amount of zero values that occur consecutively in the original data set.

[0118] In a further example, to encode certain data values, two bytes or symbols may be combined. That is, for example, where a pixel value is larger than a threshold value, two bytes or symbols may be used to encode the data value in a symbol. These two symbols can be combined at the decoder to recreate the original data value of the pixel.

[0119] Each symbol or byte may include 6 or 7 bits of data and one or more flags or bits indicating the type of symbol.

[0120] In an implementation example, the run-length coded data may be encoded efficiently by inserting, in each symbol, one or more flags indicating the next symbol to be expected in the stream of bytes.

[0121] To facilitate synchronisation between the encoder and decoder, the first byte of the stream of run-length coded data may be a data value type of symbol. This byte may then indicate the next type of byte that has been encoded.

[0122] In certain embodiments, the flag indicating the next byte may be different depending on the type of symbol. For example, the flag or bit may be located at the beginning or the end of the byte or both. For example, at the beginning of byte, the bit may indicate that the next symbol may be a run of zeros. At the end of the byte, the bite may be an overflow bit, indicating that the next symbol contains a part of the data value to be combined with the data value in the current symbol.

[0123] The following describes a specific implementation example of a run-length coding operation. The RLE has three contexts: RLC_RESIDUAL_LSB, RLC_RESIDUAL_MSB and RLC_ZERO_RUN. The structure of these bytes is illustrated in Figures 10, 11 and 12.

[0124] In a first type of symbol, RLC_RESIDUAL_LSB, illustrated in Figure 10, the 6 least significant bits of a non-zero pixel are encoded. A run bit may be provided that indicates that the next byte is encoding the count of a run of zeros. An overflow bit may be set if the pixel value does not fit within 6 bits of data. When the overflow bit is set, the context of the next byte will be an RLC_RESIDUAL_MSB type. That is, the next symbol will comprise a data byte which can be combined with the bits of the current symbol to encode a data value. Where the overflow bit is set, the next context cannot be a run of zeros and therefore the symbol can be used to encode data.

[0125] Figure 10 indicates an example of this type of symbol. If the data value to be encoded is greater than a threshold value of 64, or the pixel value does not fit within 6 bits of data, the overflow bit may be set by the encoding process. If an overflow bit is set as the least significant

bit of the byte then the remaining bits of the byte may encode data. If the pixel value fits within 6 bits, the overflow bit may not be set and a run bit may be included that indicates whether or not the next symbol is a data value or a run of zeros.

[0126] A second type of symbol, RLC_RESIDUAL_MSB, illustrated in Figure 11, encodes bits 7 to 13 of pixel values that do not fit within 6 bits of data. Bit 7 of this type of symbol encodes whether or not the next byte is a run of zeros.

[0127] A third type of symbol, RLC_ZERO_RUN, illustrated in Figure 12, encodes 7 bits of a zero run count. That is, the symbol comprises the number of consecutive zeros in the residual data. The run bit of the symbol is high if more bits are needed to encode the count. The run bit may be the most significant bit of the byte. That is, where there are a number of consecutive zeros that requires more than the 7 bits available, 178 or 255 say, the run bit indicates that the next bit will indicate a further run of zeros.

[0128] Optionally, the additional symbol may comprise a second run of zeros which can be combined with the first run or may comprise a set of bits of a value which can be combined with the bits of the first symbol at the decoder to indicate the count.

[0129] As indicated above, in order for the decoder to start on a known context, the first symbol in the encoded stream may be a residual data value type of symbol, that is, may be a RLC_RESIDUAL_LSB type.

[0130] In a specific example, the RLE data may be organised in blocks. Each block may have an output capacity of 4096 bytes. The RLE may switch to a new block in the following cases:

- the current block is full;
- the current RLE data is a run and there is less than 5 bytes left in the current block; and/or
- the current RLE data is lead to an LSB/MSB pair and there is less than 2 bytes left in the current block.

[0131] In summary, a run-length coding operation may be performed on the residual data of the new coding technology that comprises encoding into a stream a set of data values and a count of consecutive zero values. In a specific implementation, the output of the run-length coding operation may be a stream of bytes or symbols, where each byte is one of three types or contexts. The byte indicates the next type of byte to be expected in the stream of bytes.

[0132] It should be noted that these structures are provided as an example, and that different bit encoding schemes may be applied while following the functional teachings described herein. For example, the encoding of the least and most significant bits may be exchanged and/or different bit lengths may be enacted. Also, flag bits may be located at different predefined positions within a byte.

[0133] As indicated above, a Huffman coding operation may be applied to the stream of bytes to further reduce the size of the data. The process may create a frequency table of the relative occurrence of each symbol in the stream. From the frequency table, the process may generate a Huffman tree. A Huffman code for each symbol may be generated by traversing the tree from the root until the symbol is reached, assigning a bit of the code for each branch taken.

[0134] In a preferred implementation for use with the specifics of the run-length coded symbols of the quantized residuals (i.e. the residual data), a canonical Huffman coding operation may be applied. Canonical Huffman codes may reduce the storage requirements of a set of codes depending on the structure of the input data. A canonical Huffman procedure provides a way to generate a code that implicitly contains the information of which codeword applies to which symbol. The Huffman codes may be set with a set of conditions, for example, all codes of a given length may have lexicographically consecutive values in the same order as the symbols they represent and shorter codes lexicographically precede longer codes. In a canonical Huffman coding implementation, only the codewords and the lengths of each code need to be transmitted for the decoder to be able to replicate each symbol.

[0135] After the canonical Huffman coding operation is applied, the data size of the output data may be compared to the size of the data after the run-length coding operation. If the data size is smaller, then the smaller data may be transmitted or stored. The comparison of data size may be performed based on the size of a data block, a size of a layer, plane or surface or the overall size of the frame or video. To signal to the decoder how the data was encoded, according to the invention a flag is transmitted in a header of the data that the data is coded using Huffman coding, RLE coding or both. In an alternative implementation not falling under the scope of the claimed invention, the decoder may be able to identify from characteristics of the data that the data has been encoded using a particular coding operation. For example, as indicated below, the data may be split into a header and data part where canonical Huffman coding is used so as to signal the code lengths of encoded symbols.

[0136] The canonical Huffman encoded data comprises a part that signals to the decoder the lengths of the code used for each symbol and a part that comprises a stream of bits representing the encoded data. In a specific implementation proposed herein, the code lengths may be signalled in a header part and the data signalled in a data part. Preferably a header part will be signalled for each surface but may be signalled for each block or other sub-division depending on the configuration.

[0137] In a proposed example, the header may be different depending on the amount of non-zero codes to be encoded. The amount of non-zero codes to be encoded (for each surface or block for example) may be compared to a threshold value and a header used based on the comparison. For example, where there are more than 31 non-zero codes, the header may sequentially indicate all symbols, starting for a predetermined signal such as 0. In order, the length of each symbol may be signalled. Where there are fewer than 31 non-zero codes, each symbol value may be signalled in the header with a corresponding code length for that symbol.

[0138] Figures 13 to 16 illustrate a specific implementation of the header formats and ow the code lengths may be written to a stream header depending on the amount of non-zero codes.

[0139] Figure 13 illustrates a situation where there may be more than 31 non-zero values in the data. The header includes a minimum code length and a maximum code length. The code length for each symbol is sent sequentially. A flag indicates that the length of the symbol is non-zero. The bits of the code length are then sent as a difference between the code length and the minimum signalled length. This reduces the overall size of the header.

[0140] Figure 14 illustrates a header similar to Figure 13 but used where there is are fewer than 31 non-zero codes. The header further includes the number of symbols in the data, followed by a symbol value and the length of the codeword for that symbol, again sent as a difference.

[0141] Figures 15 and 16 illustrate further headers to be sent in outlying cases. For example, where the frequencies are all zero, the stream header may be as illustrated in Figure 14 indicating two values of 31 in the minimum and maximum length fields to indicate a special situation. Where there is only one code in the Huffman tree, the stream header may be as indicated in Figure 16 using a 0 value in the minimum and maximum length fields to indicate a special situation and then followed by the symbol value to be used.

[0142] In this latter example, where there is only one symbol value, this may indicate that there is only one data value in the residual data.

[0143] The encoding process can thus be summarised as follows -

Parse residual data to identify data values and consecutive counts of zero values.

Generate a set of symbols where each symbol is a byte and each byte comprises either a data value or a run of zeros as well as an indication of the next symbol in the set of symbols. The symbols may include an overflow bit indicating that the next symbol includes a part of the data value included in the current symbol or a run bit indicating the next symbol is a run of zeros.

Convert each fixed-length symbol into a variable code using canonical Huffman coding. Canonical Huffman coding parses the symbols to identify the frequency with which each symbol occurs in the set of symbols and assigns a code to the symbol based on the frequency and the value of the symbol (for identical code lengths).

Generate and output a set of code lengths, each associated with a symbol. The code lengths being the length of each code assigned to each symbol.

Combine the variable-length codes into a bitstream.

Output the encoded bitstream.

[0144] In the specific implementation described, it is noted that the maximum code length depends on the number of symbols encoded and the number of samples used to derive the Huffman frequency table. For N symbols the maximum code length is N-1. However, in order for a symbol to have a k-bit Huffman code, the number of samples used to derive the frequency table needs to be at least:

$$S_k = \sum_{i=0}^{k+1} F_i$$

where F_i is the i^{th} Fibonacci number.

[0145] For example, 8-bit symbols, the theoretical maximum code length is 255. However, for an HD 1080p video, the maximum number of samples used to derive the frequency table for an RLE context of a residual is 1920*1080/4 = 518 400, which is between S26 and S27. Hence no symbol can have a Huffman code greater than 26 bits for this example. For 4K this number increases to 29 bits.

[0146] For completeness, using Figure 17, we provide an example of Huffman encoding of a set of symbols, in this example the symbols are alphanumeric. As noted above, a Huffman Code is an optimal prefix code which may be used for lossless data compression. A prefix code is a code system for which there is no code word in the system that is the prefix of any other code.

[0147] In order to find a Huffman Code for a given set of symbols a Huffman tree needs to be created. First the symbols are sorted by frequency, for example:

Symbol	Frequency
A	3
B	8
C	10
D	15
E	20
F	43

[0148] The two lowest elements are removed from the list and made into leaves, with a parent node that has a frequency the sum of the two lower element's frequencies. The partial tree is illustrated in Figure 17a.

[0149] The new sorted frequency list is:

Symbol	Frequency
C	10
*	11

Symbol	Frequency
D	15
E	20
F	43

[0150] Then the loop is repeated, combining the two lowest elements as illustrated in Figure 17b.

[0151] The new list is:

Symbol	Frequency
D	15
E	20
*	21
F	43

[0152] This is repeated until only one element remains in the list as illustrated in Figures 17c, 17d, 17e and the following corresponding tables.

Symbol	Frequency
*	21
*	35
F	43

Symbol	Frequency
F	43
*	56

Symbol	Frequency
*	99

[0153] Once the tree is built, to generate the Huffman code for a symbol the three is traversed from the root to this symbol, outputting a 0 each time a left branch is taken and a 1 each time a right branch is taken. In the example above this gives the following code:

Symbol	Code	Code length
A	1010	3
B	1011	3
C	100	2
D	110	2

Symbol	Code	Code length
E	111	2
F	0	0

[0154] The code length of a symbol is the length of its corresponding code.

[0155] To decode a Huffman code, the tree is traversed beginning at the root, taking a left path if a 0 is read and a right path if a 1 is read. The symbol is found when hitting a leaf.

[0156] As described above, the RLE symbols may be coded using Huffman coding. In a preferred example, canonical Huffman coding is used. In an implementation of canonical Huffman coding, a Huffman coding procedure may be used and the code produced transformed into canonical Huffman codes. That is, the table is rearranged so that identical lengths are sequential in the same order as the symbols they represent and that a later code will always be higher in value than an earlier one. Where the symbols are alphabetical, they may be rearranged in alphabetical order. Where the symbols are values, they may be in sequential value order and the corresponding codewords changed to meet the above constraints. Canonical Huffman coding and/or other Huffman coding approaches described herein may differ from the basic example of Figures 17a to 17e.

[0157] In the examples described above, a Huffman coding operation may be applied to data symbols coded using a modified run-length coding operation. It is further proposed to create a frequency table for each RLE context or state. For each type of symbol encoded by the RLE coding operation, there may be a different set of Huffman codes. In an implementation, there may be a different Huffman encoder for each RLE context or state.

[0158] It is contemplated that one stream header may be sent for each RLE context or type of symbol, that is, a plurality of stream headers may indicate the code lengths for each set of codes, i.e. for each Huffman encoder or each frequency table.

[0159] The following describes an example of specific implementation steps of the encoder. In one implementation, encoding the data outputted by the RLE is done in three main steps:

1. 1. Initialise the encoders (e.g. by an *InitialiseEncode* function):
    1. a. Initialize each encoder (one per RLE state) with the corresponding frequency table generated by the RLE;
    2. b. Create the Huffman tree;
    3. c. Calculate the code length of each symbol; and
    4. d. Determine the minimum and maximum code lengths.
2. 2. Write code length and code table to the stream header (e.g. by a *WriteCodeLengths* function) for each encoder:
    1. a. Assign codes to symbols (e.g. using an *AssignCodes* function). Note that the function used to assign codes to symbol is slightly different from the example of

Figures 19a to 19e above, as it makes sure that all codes of a given length are sequential, i.e. uses canonical Huffman encoding;

2. b. Write the minimum and maximum code lengths to the stream header; and
3. c. Write code lengths of each symbol to the stream.

3. 3. Encode the RLE data:
    1. a. Set RLE current context to RLC_RESIDUAL_LSB;
    2. b. Encode the current symbol in the input stream with the encoder corresponding to the current context;
    3. c. Use the RLE state machine to get next context (see Figure 20). If this is not last symbol of the stream, goto b; and
    4. d. If the RLE data is only one block and the encoded stream is bigger than the RLE, store the RLE encoded stream rather than the Huffman encoded stream.

[0160] For each RLE block the Huffman encoders create a corresponding Huffman encoded part.

[0161] The output of the encoding process described herein is thus either a Huffman encoded bitstream or a run-length encoded bytestream. A corresponding decoder thus employs a corresponding Huffman decoding operation in an inverse manner to the encoding operation. Nevertheless, there are nuances to the decoding operation described below to improve efficiency and to adapt to the specificity of the encoding operation described above.

[0162] Where the encoder signals in a header or other configuration metadata that a RLE or Huffman coding operation has been used, the decoder according to the claimed invention first identifies that a Huffman decoding operation should be applied to the stream. Similarly, in the manner described above, the decoder in an arrangement not falling under the scope of the claimed invention may identify from the data stream whether or not Huffman coding has been used by identifying the number of parts of the stream. If there is a header part and a data part, then a Huffman decoding operation should be applied. If there is only a data part to the stream then only an RLE decoding part may be applied.

[0163] The entropy decoder performs the reverse transformations of the decoder. In the present example, Huffman decoding followed by Run Length Decoding.

[0164] The following example describes a decoding process in which the bitstream has been encoded using a canonical Huffman coding operation and a run-length coding operation as described above in the example of the encoding stream. However, it will be understood that Huffman coding operations and run-length coding operations can be applied separately to decode streams encoded in a different manner.

[0165] The input to the process is an encoded bitstream. The bitstream may be retrieved from a stored file or streamed, either locally or remotely. The encoded bitstream is a series of

sequential bits without an immediately discernible structure. It is only once the Huffman coding operation is applied to the bitstream can a series of symbols be deduced.

[0166] Thus, the process first applies a Huffman coding operation to the bitstream.

[0167] It is assumed that the first encoded symbol is a data value type of symbol. In this manner, where the Huffman coding operation uses different frequency tables or parameters, the Huffman coding operation can use a correct codebook for a symbol type.

[0168] The canonical Huffman coding operation must first retrieve suitable coding parameters or metadata to facilitate decoding the bitstream. In the example here, the parameters are a set of code lengths retrieved from a stream header. For simplicity, we will describe one set of code lengths being exchanged between the encoder and decoder but note that, as above, multiple sets of code lengths may be signalled so that the coding operation may use different parameters for each type of symbol it expects to be decoding.

[0169] The coding operation will assign the code lengths received in the stream header to a corresponding symbol. As noted above, the stream header may be sent in different types. Where the stream header includes set of code lengths with corresponding symbols, each code length may be associated with a corresponding symbol value. Where the code lengths are sent sequentially, the decoder may associate each code length received with a symbol in a predetermined order. For example, the code lengths may be retrieved in an order 3, 4, 6, 2, etc. The coding process may then associate each length with a corresponding symbol. Here the order is sequential - (symbol, length) - (0, 3)(1,4)(3,6)(4,2).

[0170] As illustrated in Figure 15, some symbols may not be sent but a flag may indicate the symbol has a corresponding zero code length, that is, that symbol does not exist in the bitstream to be decoded (or is unused).

[0171] The canonical Huffman coding operation is then able to assign a code to each symbol based on the code length. For example, where the code length is 2, the code for that symbol will be `1x'. Where x indicates that the value does not important. In specific implementations, the code will be `10'.

[0172] Where the lengths are identical, the canonical Huffman coding operation assigns codes based on a sequential order of the symbols. Each code is assigned such that as the bitstream is parsed, the decoder can keep checking the next bit until only one possible code exists. For example, a first sequential symbol of length 4 may be assigned a code 1110 and a second sequential symbol of length 4 may be assigned a code 1111. Thus, parsing the bitstream, if the first bits of the bitstream are 1110x, it is only possible for the bitstream to have encoded the first sequential symbol of length 4.

[0173] Accordingly, once the coding operation has established a set of codes associated with each symbol, the coding operation can move to parsing the bitstream. In certain embodiments,

the coding operation will build a tree so as to establish the symbol associated with the code of the bitstream. That is, the coding operation will take each bit of the bitstream and traverse a tree according to the value of the bit in the bitstream until a leaf is found. Once a leaf is found, the symbol associated with the leaf is output. The process continues at the root of the tree with the next bit of the bitstream. In this manner, the coding operation may output a set of symbols derived from a set of codes stored in the encoded bitstream. In preferred embodiments the symbols are each a byte, as described elsewhere herein.

[0174] The set of symbols may then be passed to a run-length coding operation. The run-length coding operation identifies the type of symbol and parses the symbol to extract a data value or a run of zeros. From the data values and runs of zeros, the operation can recreate the original residual data that was encoded.

[0175] The output of the Huffman coding is, as noted above, likely to be a set of symbols. The challenge for the next decoding step is to extract from those symbols the relevant information, noting that each symbol comprises data in a different format and each symbol will represent different information despite the information not be immediately discernible from the symbol or byte.

[0176] In preferred examples, the coding operation consults a state machine as illustrated in Figure 18. In sum, the coding operation first assumes that the first symbol is a data value type (the data value can of course be 0). From this, the coding operation can identify if the byte includes an overflow flag or a run flag. The overflow and run flag are described above and illustrated in Figure 10.

[0177] The coding operation may first check the least significant bit of the symbol (or byte). Here this is the overflow flag (or bit). If the least significant bit is set, the coding operation identifies that the next symbol will be also be a data value and by of the RLC_RESIDUAL_MSB type. The remaining bits of the symbol will be least significant bits of a data value.

[0178] The coding process goes on to parse the next symbol and extract the least significant bits as the remaining part of the data value. The bits of the first symbol can be combined with the bits of the second symbol to reconstruct the data value.

[0179] In this current symbol, illustrated in Figure 11, there will also be a run flag, here the most significant bit. If this is set, the next symbol will be a run symbol and not a data symbol.

[0180] In the run symbol, illustrated in Figure 12, the coding operation checks the most significant bit to indicate if the next symbol is a run symbol or a data symbol and extracts the remaining bits as the run of zeros. That is, a maximum run of 127 zeros.

[0181] If the data symbol indicates there is no overflow (in the overflow flag, here the least significant bit) then the symbol will also contain in the most significant bit a run bit and the coding operation will be able to identify from this bit that the next symbol in the set is a run

symbol or a data symbol. The coding operation can thus extract the data value from bits 6-1. Here there are 31 data values available without overflow.

[0182] As noted, the state machine of Figure 18 illustrates the process simply.

[0183] Starting at an RLC_RESIDUAL_LSB symbol, if the overflow bit=0 and the run bit=0, then the next symbol will be an RLC_RESIDUAL_LSB symbol. If the overflow bit=1 then the next symbol will be an RLC_RESIDUAL_MSB. If the overflow bit=1 there will not be a run bit. If the run bit=1 and the overflow bit=0, then the next bit will be an RLC_ZERO_RUN symbol.

[0184] In the RLC RLC_RESIDUAL_MSB, if the run bit=0, the next symbol will be an RLC_RESIDUAL_LSB symbol. If the run bit=1, the next symbol will be an RLC_ZERO_RUN symbol.

[0185] In the RLC_ZERO_RUN symbol, if the run bit=0, the next bit will be an RLC_RESIDUAL_LSB symbol. If the run bit=1, the next bit will be an RLC_ZERO_RUN symbol.

[0186] Bits can of course be inverted (0/1, 1/0, etc.) without loss of functionality. Similarly, the locations within the symbols or bytes of the flags is merely illustrative.

[0187] The run-length coding operation can identify the next symbol in the set of symbols and extract either a data value or a run of zeros. The coding operation can then combine these values and zeros to recreate the residual data. The order may be in the order extracted or alternatively in some predetermined order.

[0188] The coding operation can thus output residual data which has been encoded into the bytestream.

[0189] It was described above how multiple metadata or coding parameters may be used in the encoding process. At the decoding side, the coding operation may include a feedback between the operations. That is, the expected next symbol may be extracted from the current symbol (e.g. using the overflow and run bits) and the expected next symbol given to the Huffman coding operation.

[0190] The Huffman coding operation may optionally generate a separate codebook for each type of symbol (and retrieve multiple stream headers or build a table of multiple codes and symbols).

[0191] Assuming the first symbol will be a data value, the Huffman coding operation will decode the first code to match a symbol of that type. From the matched symbol, the coding operation can identify an overflow bit and/or a run bit and identify the next type of symbol. The Huffman coding operation can then use a corresponding codebook for that type of symbol to extract the next code in the bitstream. The process carries in this iterative manner using the currently decoded symbol to extract an indication of the next symbol and change codebooks

accordingly.

[0192] It has been described above how a decoding operation can combine a Huffman coding operation, preferably a canonical Huffman coding operation, and a run-length coding operation to recreate residual data from an encoded bitstream. It will be understood that the techniques of the run-length coding operation can be applied to a bytestream that has not been encoded using a Huffman encoding operation. Similarly, the Huffman coding operation may be applied without the subsequent step of the run-length coding operation.

[0193] The following describes an example of specific implementation steps of the decoder. In the example implementation, if the stream has more than one part, the following steps are performed:

1. 1. Read the code lengths from the stream header (e.g. using a *ReadCodeLengths* function):
    1. a. Set code lengths for each symbol;
    2. b. Assign codes to symbols from the code lengths (e.g. using an *AssignCodes* function); and
    3. c. Generate a table for searching the subsets of codes with identical lengths. Each element of the table records the first index of a given length and the corresponding code (*firstIdx, firstCode*).
2. 2. Decode the RLE data:
    1. a. Set RLE context to RLC_RESIDUAL_LSB.
    2. b. Decode the current code searching for the right code length in the generated table and indexing into code array with: *firstIdx* - (*current_code - firstCode*). This works because all codes for a given length are sequential by construction of the Huffman tree.
    3. c. Use the RLE state machine to get next context (see Figure 18). If the stream is not empty, goto b.

[0194] The Run Length Decoder reads the run length encoded data byte by byte. By construction the context of the first byte of data is guaranteed to be RLC_RESIDUAL_LSB. The decoder uses the state machine shown in Figure 18 to determine the context of the next byte of data. The context tells the decoder how to interpret the current byte of data as described above.

[0195] Note that in this specific implementation example the run length state machine is also used by the Huffman encoding and decoding process to know which Huffman code to use for the current symbol or code word.

[0196] At both the encoder and decoder, for example implemented in a streaming server or client device or client device decoding from a data store, methods and processes described herein can be embodied as code (e.g., software code) and/or data. The encoder and decoder

may be implemented in hardware or software as is well-known in the art of data compression. For example, hardware acceleration using a specifically programmed Graphical Processing Unit (GPU) or a specifically designed Field Programmable Gate Array (FPGA) may provide certain efficiencies. For completeness, such code and data can be stored on one or more computer-readable media, which may include any device or medium that can store code and/or data for use by a computer system. When a computer system reads and executes the code and/or data stored on a computer-readable medium, the computer system performs the methods and processes embodied as data structures and code stored within the computer-readable storage medium. In certain embodiments, one or more of the steps of the methods and processes described herein can be performed by a processor (e.g., a processor of a computer system or data storage system).

[0197] Generally, any of the functionality described in this text or illustrated in the figures can be implemented using software, firmware (e.g., fixed logic circuitry), programmable or nonprogrammable hardware, or a combination of these implementations. The terms "component" or "function" as used herein generally represents software, firmware, hardware or a combination of these. For instance, in the case of a software implementation, the terms "component" or "function" may refer to program code that performs specified tasks when executed on a processing device or devices. The illustrated separation of components and functions into distinct units may reflect any actual or conceptual physical grouping and allocation of such software and/or hardware and tasks.

[0198] In the present application, we have described a method for encoding and decoding a signal, in particular a video signal and/or an image signal.

[0199] In particular, there is described a method of encoding a signal, the method comprising receiving an input frame and processing the input frame to generate at least one first set of residual data, said residual data enabling a decoder to reconstruct the original frame from a reference reconstructed frame.

[0200] In one example, the method comprises obtaining the reconstructed frame from a decoded frame obtained from a decoding module, wherein the decoding module is configured to generate said decoded frame by decoding a first encoded frame which has been encoded according to a first encoding method. The method further comprises down-sampling the input frame to obtain a down-sampled frame, and passing said down-sampled frame to an encoding module configured to encode said down-sampled frame in accordance with the first encoding method in order to generate the first encoded frame. Obtaining the reconstructed frame may further comprise up-sampling the decoded frame to generate the reconstructed frame.

[0201] In another example, the method comprises obtaining the reconstructed frame from a combination of a second set of residual data and a decoded frame obtained from a decoding module, wherein the decoding module is configured to generate said decoded frame by decoding a first encoded frame which has been encoded according to a first encoding method. The method further comprises down-sampling the input frame to obtain a down-sampled frame

and passing said down-sampled frame to an encoding module configured to encode said down-sampled frame in accordance with the first encoding method in order to generate the first encoded frame. The method further comprises generating said second set of residual data by taking a difference between the decoded frame and the down-sampled frame. The method further comprises encoding said second set of residual data to generate a first set of encoded residual data. Encoding said second set of residual data may be performed according to a second encoding method. The second encoding method comprises transforming the second set of residual data into a transformed second set of residual data. Transforming the second set of residual data comprises selecting a subset of the second set of residual data, and applying a transformation on said subset to generate a corresponding subset of transformed second set of residual data.

[0202] One of the subset of transformed second set of residual data may be obtained by averaging the subset of the second set of residual data. Obtaining the reconstructed frame may further comprise up-sampling the combination of the second set of residual data and a decoded frame to generate the reconstructed frame.

[0203] In one example, generating the at least one set of residual data comprises taking a difference between the reference reconstructed frame and the input frame. The method further comprises encoding said first set of residual data to generate a first set of encoded residual data. Encoding said first set of residual data may be performed according to a third encoding method. The third encoding method comprises transforming the first set of residual data into a transformed first set of residual data. Transforming the first set of residual data comprises selecting a subset of the first set of residual data, and applying a transformation on said subset to generate a corresponding subset of transformed first set of residual data. One of the subsets of transformed first set of residual data may be obtained by the difference between an average of a subset of the input frame and a corresponding element of the combination of the second set of residual data and the decoded frame.

[0204] In particular, there is described a method of decoding a signal, the method comprising receiving an encoded frame and at least one set of encoded residual data. The first encoded frame may be encoded using a first encoding method.

[0205] The at least one set of residual data may be encoded using a second and/or a third encoding method.

[0206] The method further comprises passing the first encoded frame to a decoding module, wherein the decoding module is configured to generate a decoded frame by decoding the encoded frame which has been encoded according to a first encoding method.

[0207] The method may further comprise decoding the at least one set of encoded residual data according to the respective encoding method used to encode them.

[0208] In one example, a first set of encoded residual data is decoded by applying a second

decoding method corresponding to said second encoding method to obtain a first set of decoded residual data. The method further comprises combining the first set of residual data with the decoded frame to obtain a combined frame. The method further comprises up-sampling the combined frame to obtain a reference decoded frame.

[0209] The method further comprises decoding a second set of encoded residual data by applying a third decoding method corresponding to said third encoding method to obtain a second set of decoded residual data. The method further comprises combining the second set of decoded residual data with the reference decoded frame to obtain a reconstructed frame.

[0210] In another example, the method comprises up-sampling the decoded frame to obtain a reference decoded frame.

[0211] The method further comprises decoding a set of encoded residual data by applying a second or third decoding method corresponding to said second or third encoding method to obtain a set of decoded residual data. The method further comprises combining the set of decoded residual data with the reference decoded frame to obtain a reconstructed frame.

REFERENCES CITED IN THE DESCRIPTION

Cited references

This list of references cited by the applicant is for the reader's convenience only. It does not form part of the European patent document. Even though great care has been taken in compiling the references, errors or omissions cannot be excluded and the EPO disclaims all liability in this regard.

Patent documents cited in the description

- WO2014170819A [0001]
- WO2018046940A [0001]
- US2009097548A1 [0006]
- WO2014170819A1 [0006]
- US13893669B [0094] [0097]
- EP2013059847W [0094] [0097]

ENTROPIKODNING TIL SIGNALFORSTÆRKNINGSKODNING

Patentkrav

1. En fremgangsmåde til dekodning af et videosignal, hvilken fremgangsmåde omfatter:

   hentning af en kodet bitstrøm (102, 103), der er ordnet hierarkisk i blokke, hvor bitstrømmen er grupperet i planer, og planer er opdelt i niveauer, og inden for hvert plan er hvert niveau grupperet i lag, og hvert lag omfatter et sæt af blokke for det pågældende lag, hvor hvert niveau svarer til et niveau af rumlig forstærkning, og hvert lag omfatter sæt af koefficienter af samme type,
   dekodning af den kodede bitstrøm (102, 103) med henblik på at generere restdata til en forstærkningsstrøm, og
   rekonstruktion af et oprindeligt billede af videosignalet ud fra restdataene og et referencerekonstrueret billede, hvor trinnet til dekodning af den kodede bitstrøm (102, 103) omfatter, for hver blok:

      hentning af et flag ud fra konfigurationsmetadata, der ledsager den kodede bitstrøm (102, 103), som angiver, hvorvidt en Huffman-kodningsoperation skal springes over ved dekodning af blokken,
      selektiv anvendelse af Huffman-kodningsoperationen baseret på flaget med henblik på at generere et sæt af Huffman-dekodede bytes, hvor hver byte svarer til et antal fortløbende nulværdier for restdataene, mindst signifikante bit af en ikke-nuldataværdi for restdataene eller mest signifikante bit af en ikke-nuldataværdi for restdataene,
      anvendelse af en kørselslængdekodningsoperation på sættet af Huffman-dekodede bytes eller en kørselslængdekodet bytestrøm af den kodede bitstrøm, baseret på flaget, med henblik på at generere restdataene for blokken, hvor kørselslængdekodningsoperationen omfatter:

         identifikation af et sæt af bytes, der repræsenterer mindst signifikante bit af ikke-nuldataværdier for restdataene, et sæt af bytes, der repræsenterer mest signifikante bit af ikke-nuldataværdier for restdataene, og et sæt af bytes, der repræsenterer tællinger af fortløbende nulværdier for restdataene,
         parsing af bytene med henblik på at udlede ikke-nuldataværdierne for restdataene og tællingerne af fortløbende nulværdier, og

   generering af restdataene ud fra ikke-nuldataværdierne og tællingerne af fortløbende nulværdier.

2. Et dekodningsapparat, der er konfigureret til at udføre fremgangsmåden ifølge krav 1.

3. Et computerlæsbart medium, der omfatter instrukser, som, når de eksekveres af en processor, får processoren til at udføre fremgangsmåden ifølge krav 1.

4. En kodet bitstrøm (102, 103), der omfatter en kodet version af restdata for en forstærkningsstrøm,

   hvor den kodede bitstrøm er ordnet hierarkisk i blokke, hvor bitstrømmen er grupperet i planer, og planer er opdelt i niveauer, og inden for hvert plan er hvert niveau grupperet i lag, og hvert lag omfatter et sæt af blokke for det pågældende lag, hvor hvert niveau svarer til et niveau af rumlig forstærkning, og hvert lag omfatter sæt af koefficienter af samme type,
   idet restdataene kan anvendes med et referencerekonstrueret billede til at rekonstruere et originalt billede af videosignalet,
   hvor den kodede bitstrøm omfatter, for hver blok:

      konfigurationsmetadata, der omfatter et flag, som angiver, hvorvidt en Huffman-kodningsoperation skal springes over ved dekodning af blokken, og
      baseret på flaget, et sæt af Huffman-kodede bytes, hvor hver byte svarer til et resultat af anvendelse en kørselslængdekodningsoperation på restdataene,
      eller, baseret på flaget, resultatet af anvendelsen af kørselslængdekodningsoperationen på restdataene,

   hvor kørselslængdekodningsoperationen resulterer i et sæt af bytes, der repræsenterer mindst signifikante bit af ikke-nuldataværdier for restdataene, et sæt af bytes, der repræsenterer mest signifikante bit af ikke-nuldataværdier for restdataene, og et sæt af bytes, der repræsenterer tællinger af fortløbende nulværdier for restdataene, idet sættene af bytes kan parses af en dekoder med henblik på at udlede ikke-nuldataværdier for restdataene og tællinger af fortløbende nulværdier, og på at generere restdata ud fra ikke-nuldataværdierne og tællingerne af fortløbende nulværdier.

DRAWINGS

Drawing

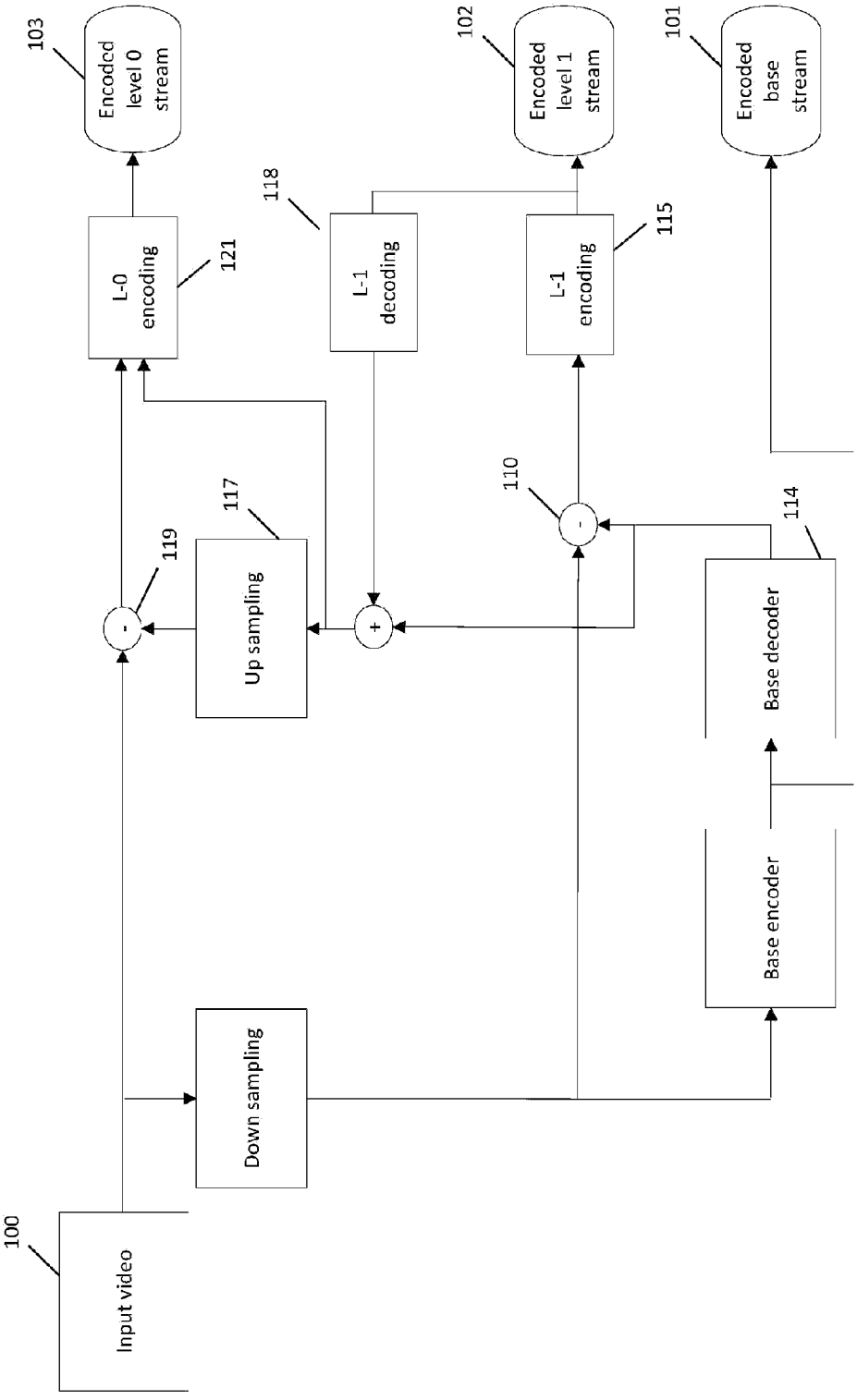

Figure 1

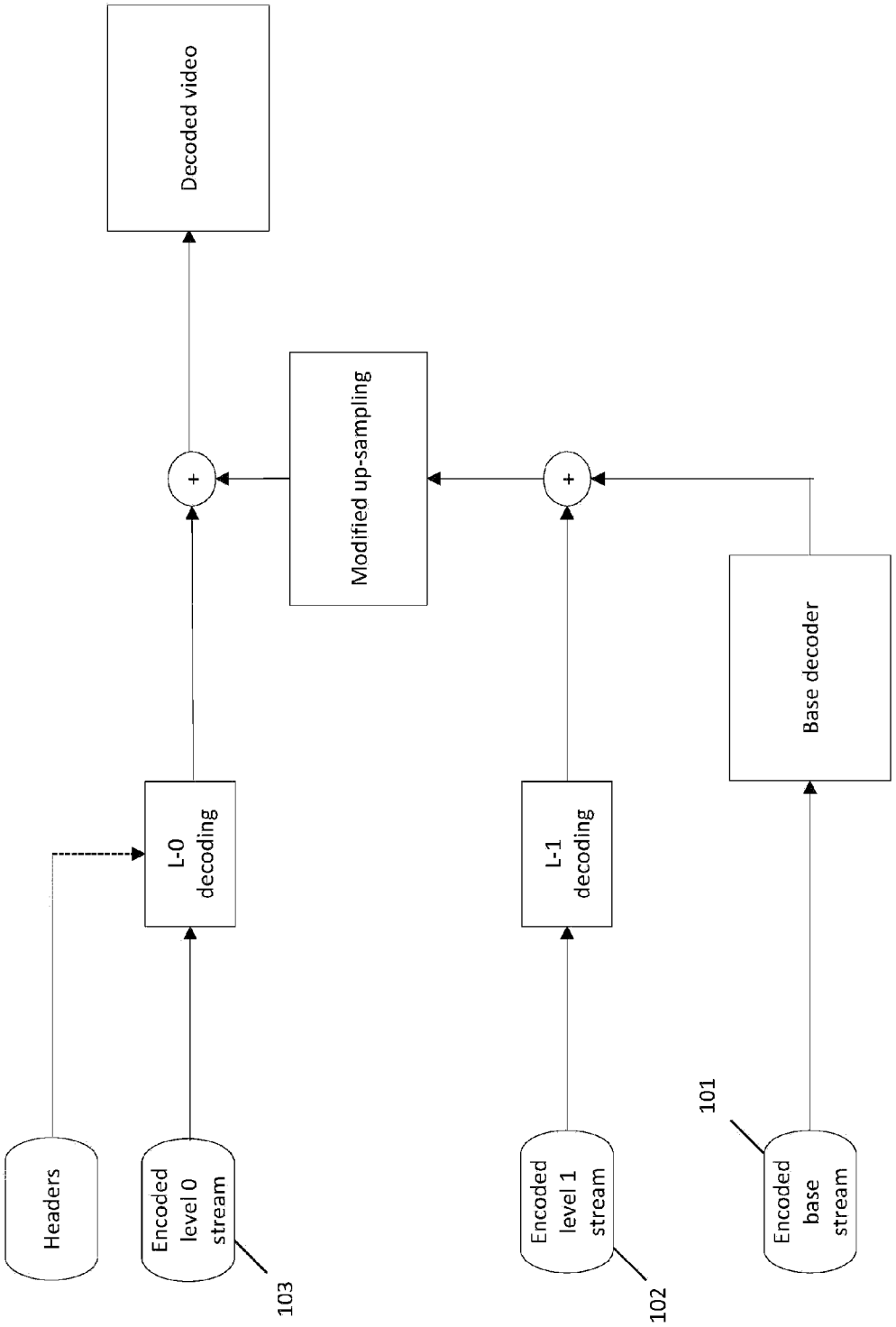
Decoded video
+
Modified up-sampling
+
Base decoder
L-0 decoding
L-1 decoding
Headers
Encoded level 0 stream
Encoded level 1 stream
Encoded base stream
103
102
101

Figure 2

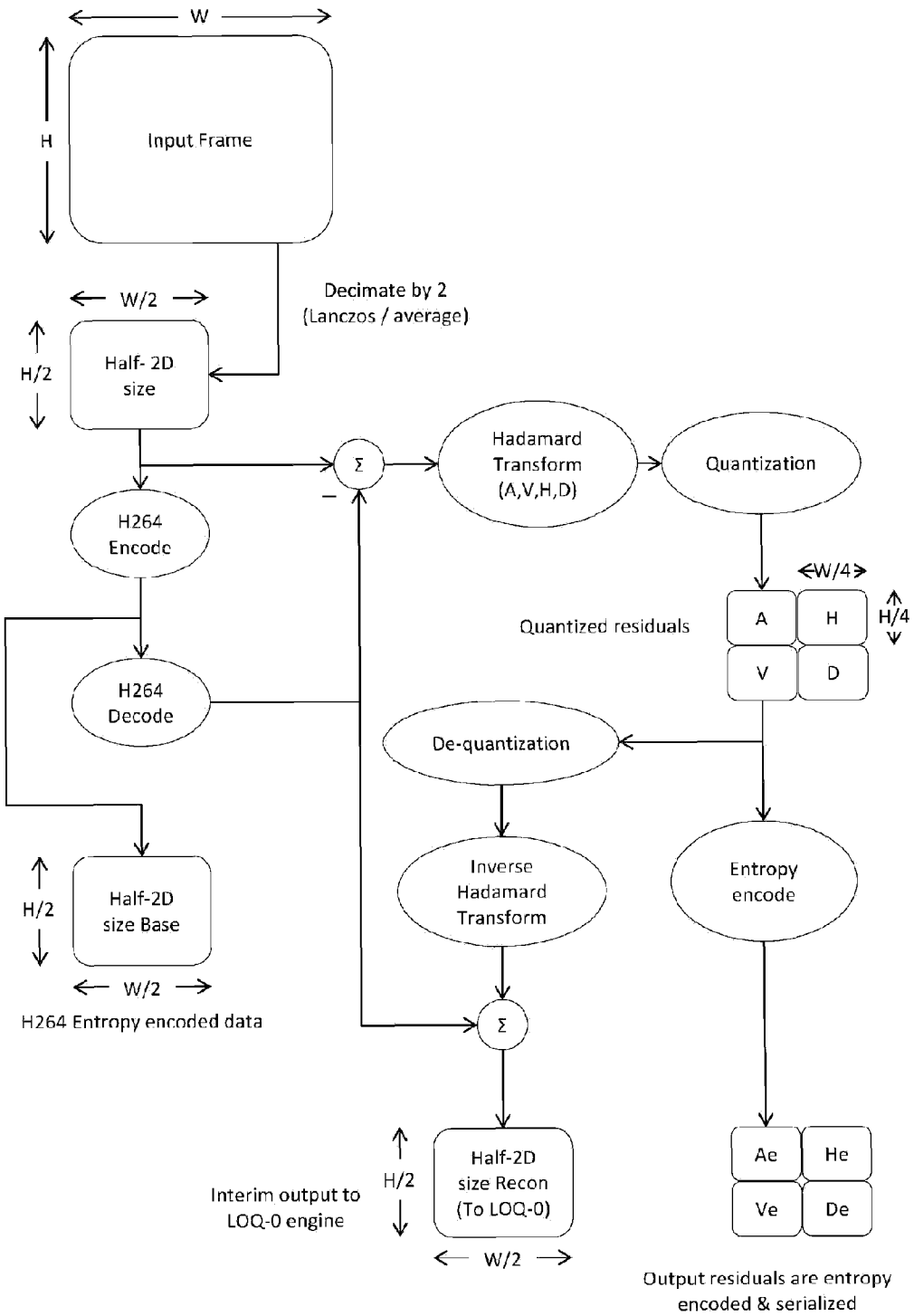
W
H
Input Frame
W/2
H/2
Half- 2D size
Decimate by 2
(Lanczos / average)
Σ
−
Hadamard Transform (A,V,H,D)
Quantization
H264 Encode
W/4
H/4
A
H
V
D
Quantized residuals
H264 Decode
De-quantization
Inverse Hadamard Transform
Entropy encode
H/2
Half-2D size Base
W/2
H264 Entropy encoded data
Σ
Interim output to LOQ-0 engine
H/2
Half-2D size Recon (To LOQ-0)
W/2
Ae
He
Ve
De
Output residuals are entropy encoded & serialized

Figure 3

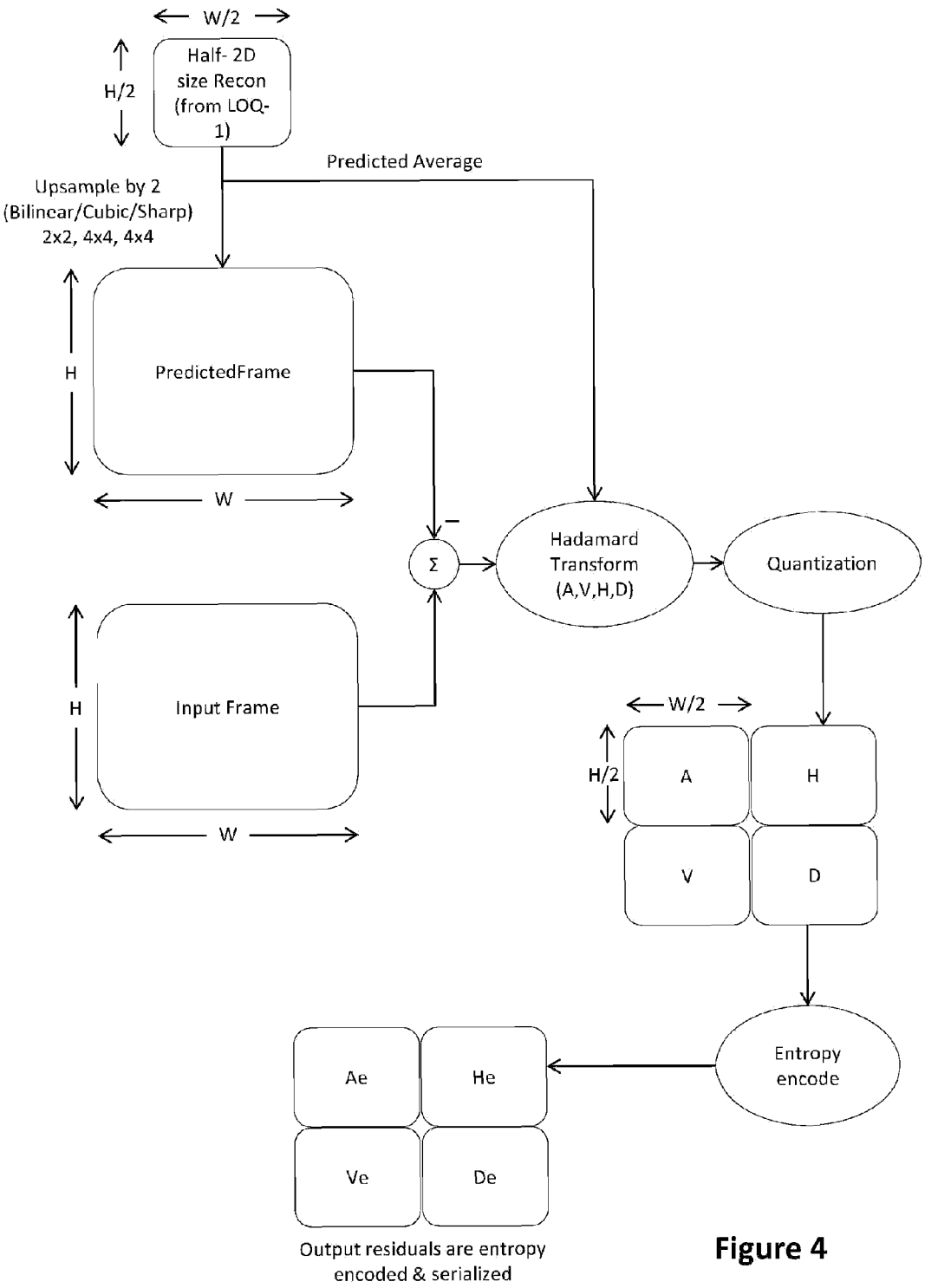
W/2
H/2
Half- 2D size Recon (from LOQ-1)
Predicted Average
Upsample by 2 (Bilinear/Cubic/Sharp) 2x2, 4x4, 4x4
H
PredictedFrame
W
−
Σ
Hadamard Transform (A,V,H,D)
Quantization
H
Input Frame
W
W/2
H/2
A
H
V
D
Entropy encode
Ae
He
Ve
De
Output residuals are entropy encoded & serialized

Figure 4

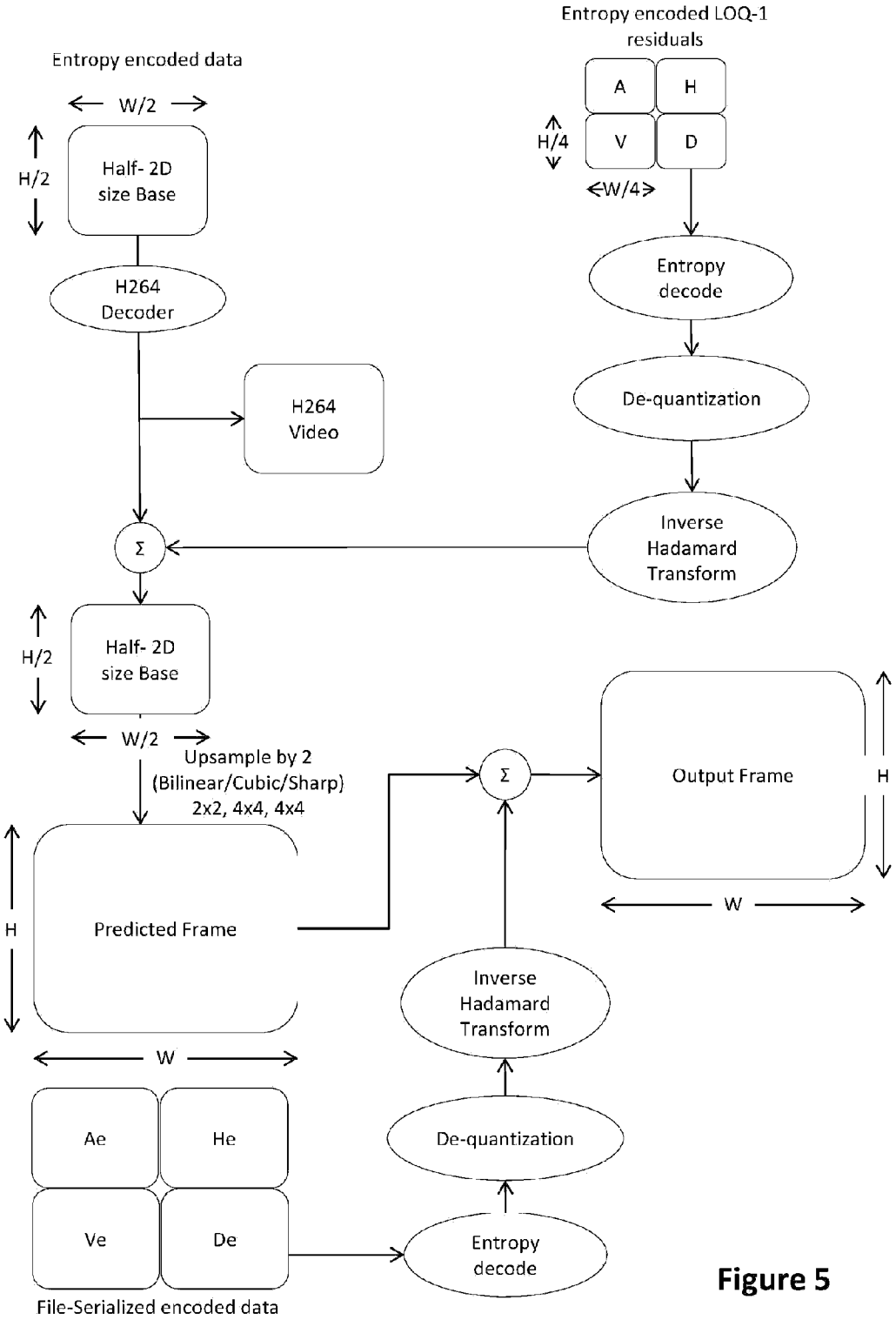
Entropy encoded data
W/2
H/2
Half- 2D size Base
H264 Decoder
H264 Video
Entropy encoded LOQ-1 residuals
A
H
V
D
H/4
W/4
Entropy decode
De-quantization
Inverse Hadamard Transform
Σ
H/2
Half- 2D size Base
W/2
Upsample by 2 (Bilinear/Cubic/Sharp) 2x2, 4x4, 4x4
Σ
Output Frame
H
W
H
Predicted Frame
W
Inverse Hadamard Transform
De-quantization
Entropy decode
Ae
He
Ve
De
File-Serialized encoded data

Figure 5

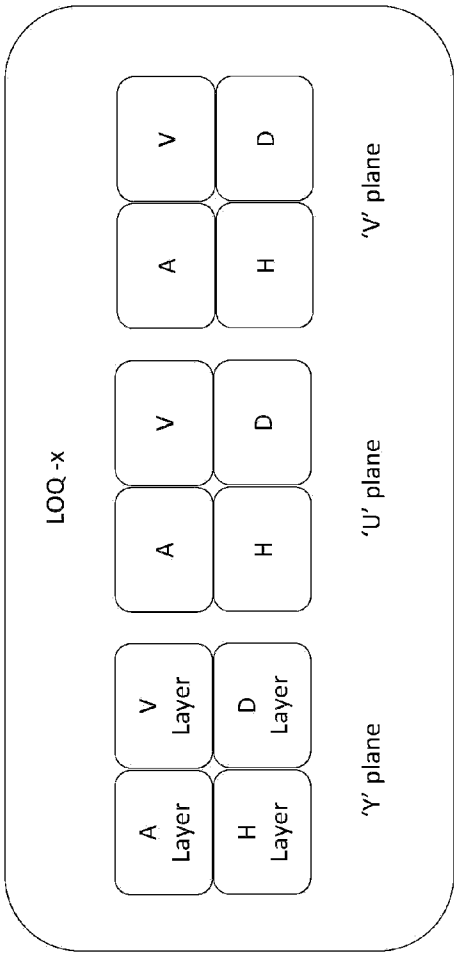
LOQ -x
A Layer
H Layer
V Layer
D Layer
'Y' plane
A
H
V
D
'U' plane
A
H
V
D
'V' plane

Figure 6

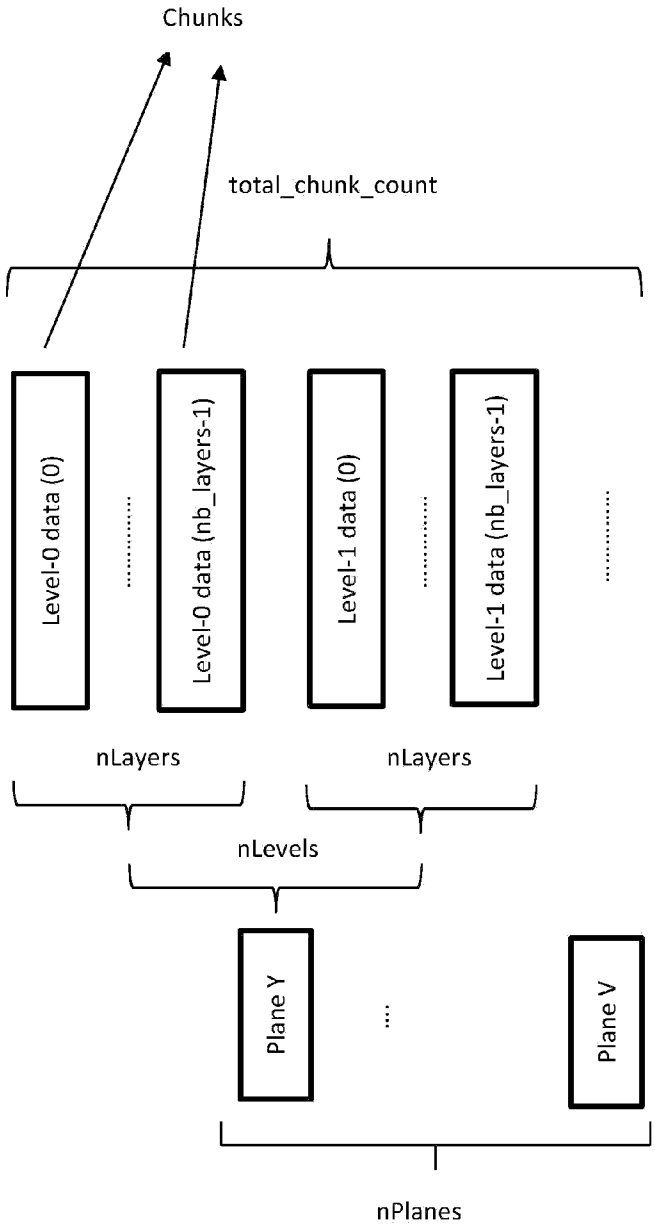
Chunks
total_chunk_count
Level-0 data (0)
Level-0 data (nb_layers-1)
Level-1 data (0)
Level-1 data (nb_layers-1)
nLayers
nLayers
nLevels
Plane Y
Plane V
nPlanes

Figure 7

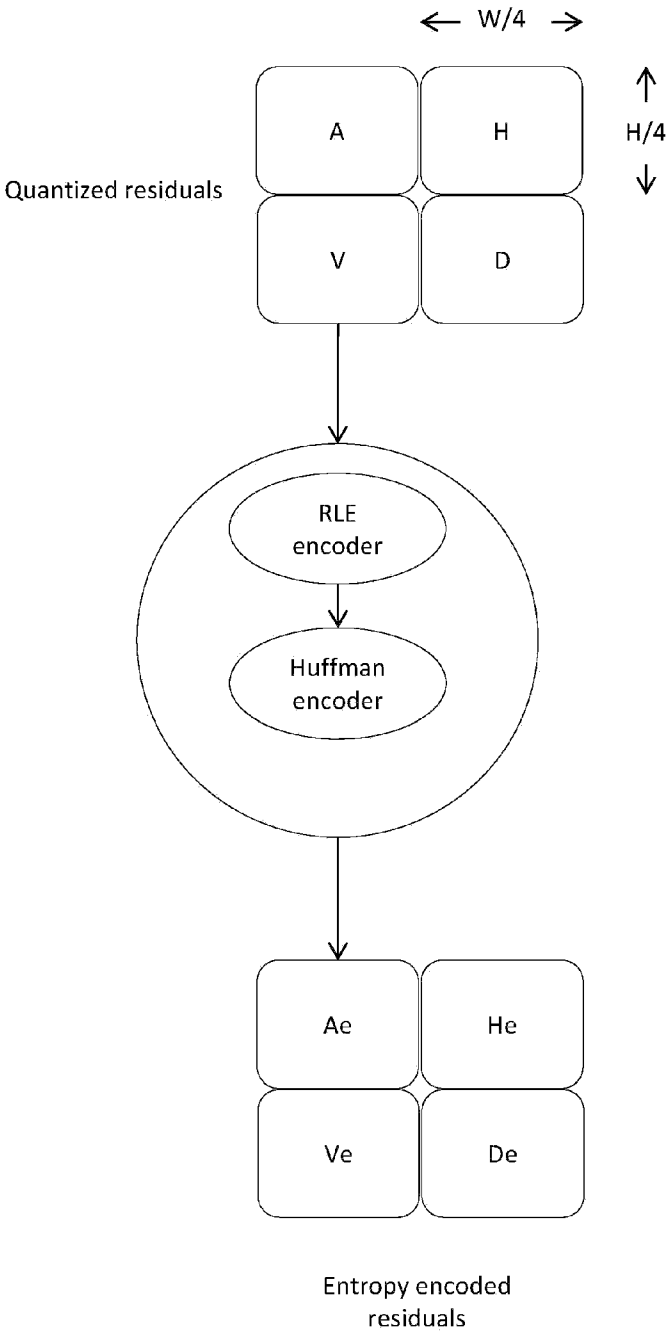
W/4
H/4
A
H
V
D
Quantized residuals
RLE encoder
Huffman encoder
Ae
He
Ve
De
Entropy encoded residuals

Figure 8

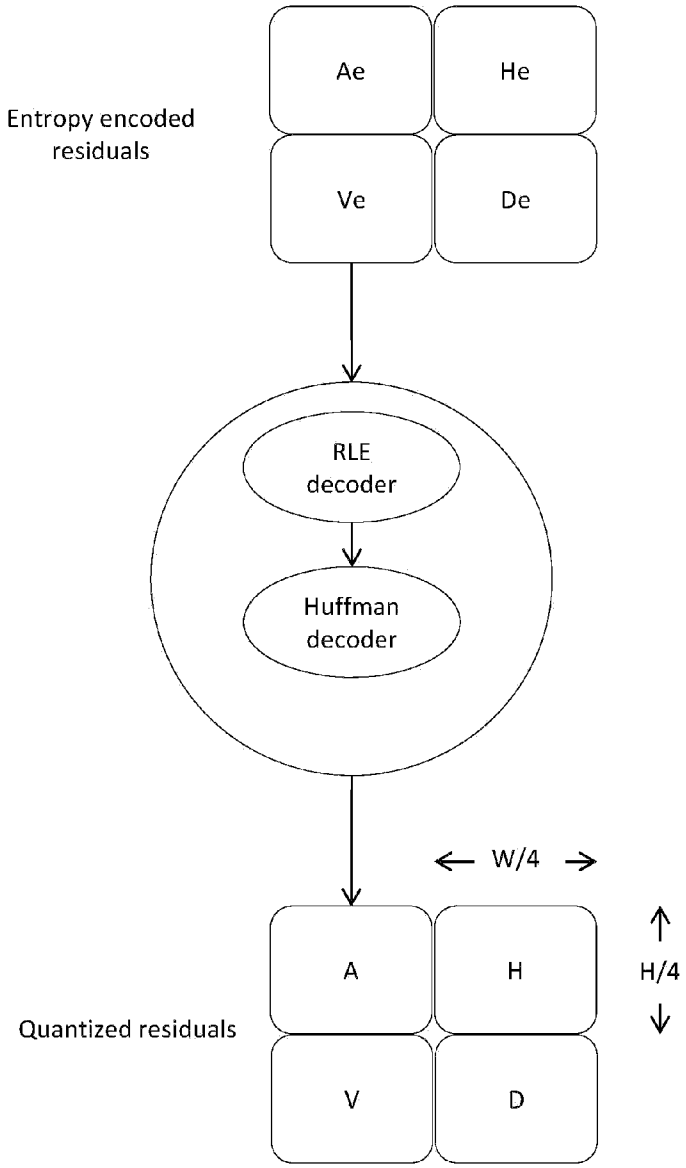
Entropy encoded residuals
Ae
He
Ve
De
RLE decoder
Huffman decoder
W/4
H/4
A
H
V
D
Quantized residuals

Figure 9

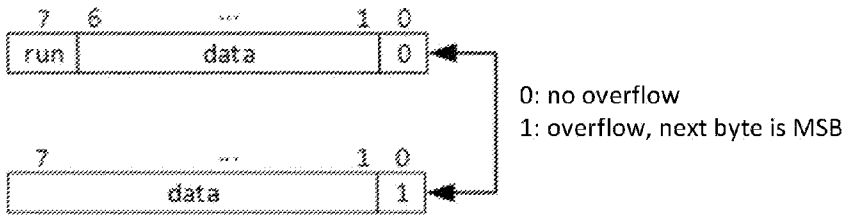

Figure 10

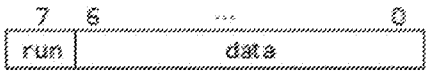

Figure 11

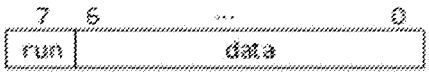

Figure 12

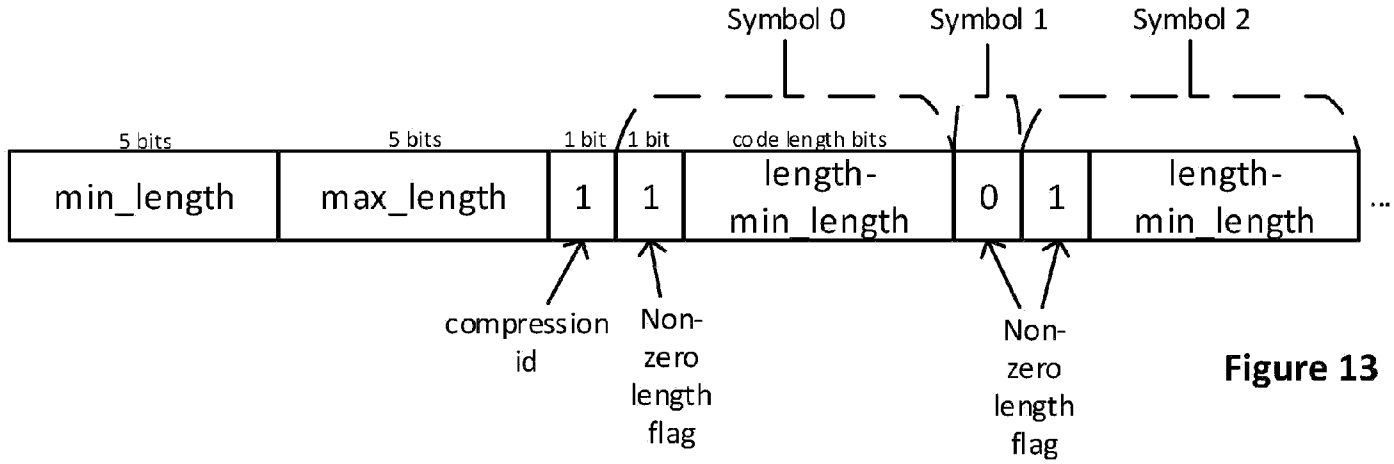

Figure 13

code length bits = log2(max_length-min_length+1)

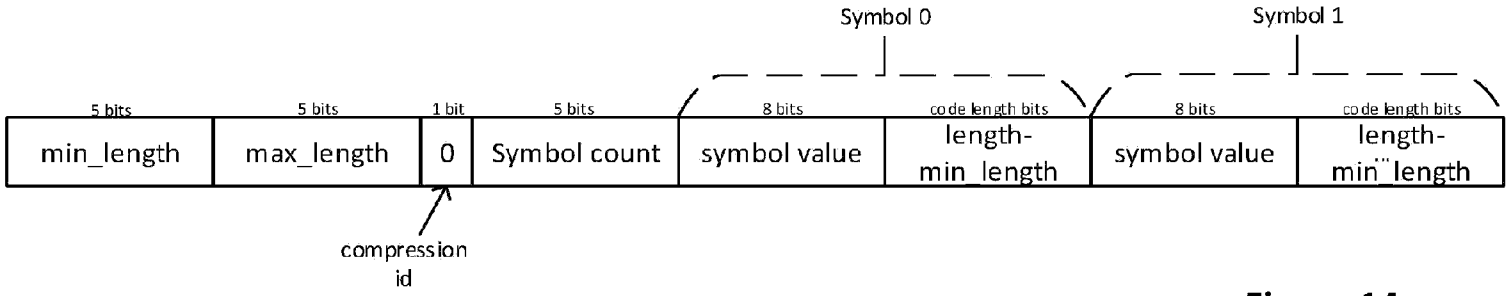

Figure 14

code length bits = log2(max_length-min_length+1)

DK/EP 3831065 T3

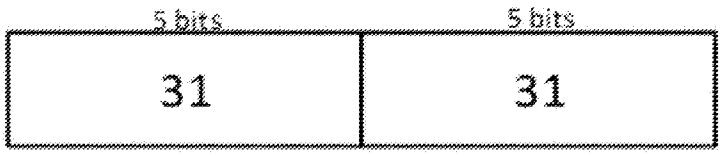

Figure 15

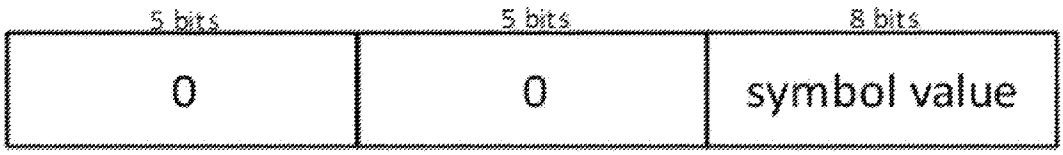

Figure 16

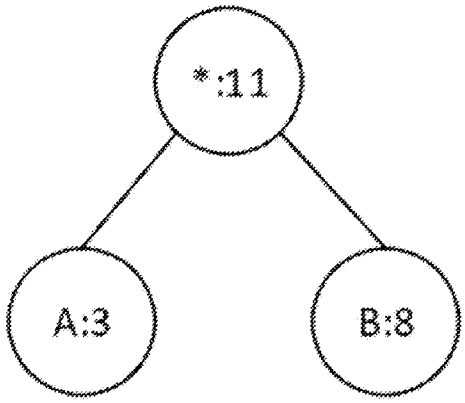

Figure 17a

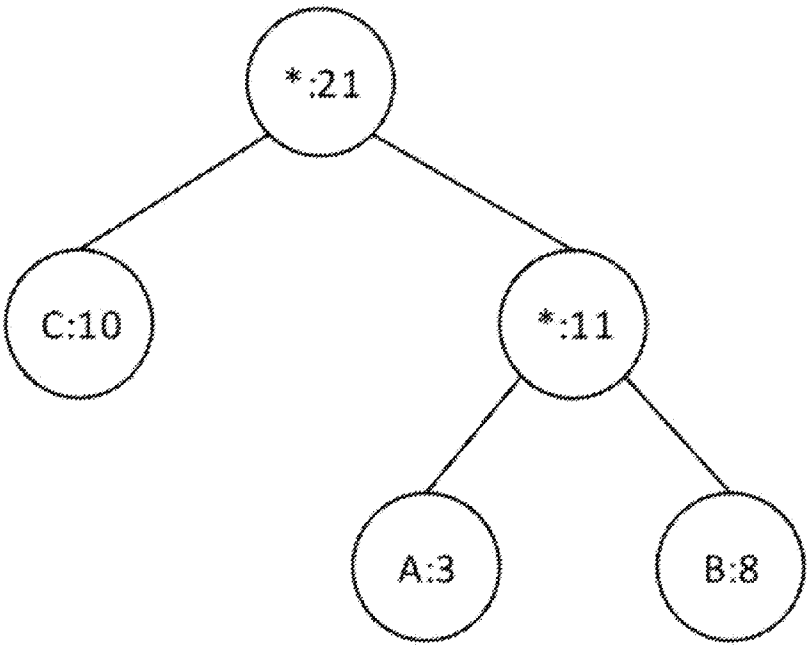

Figure 17b

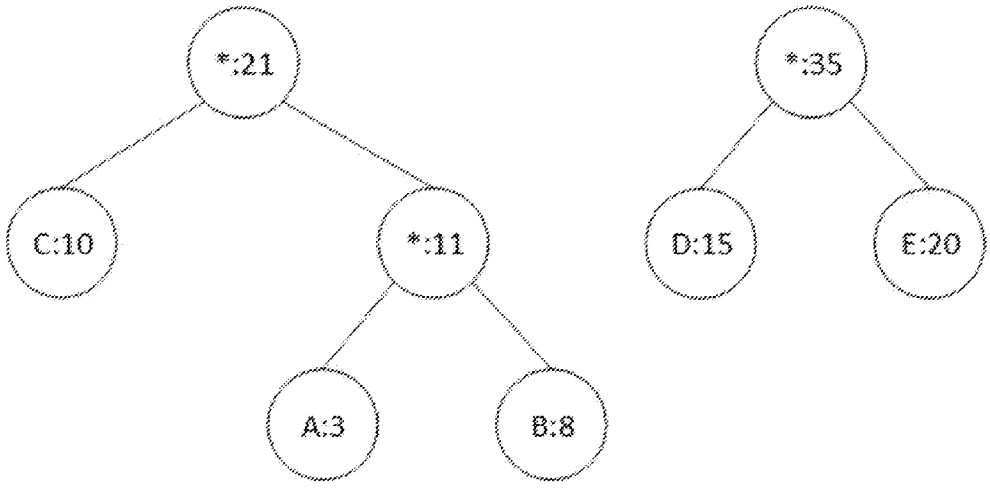

Figure 17c

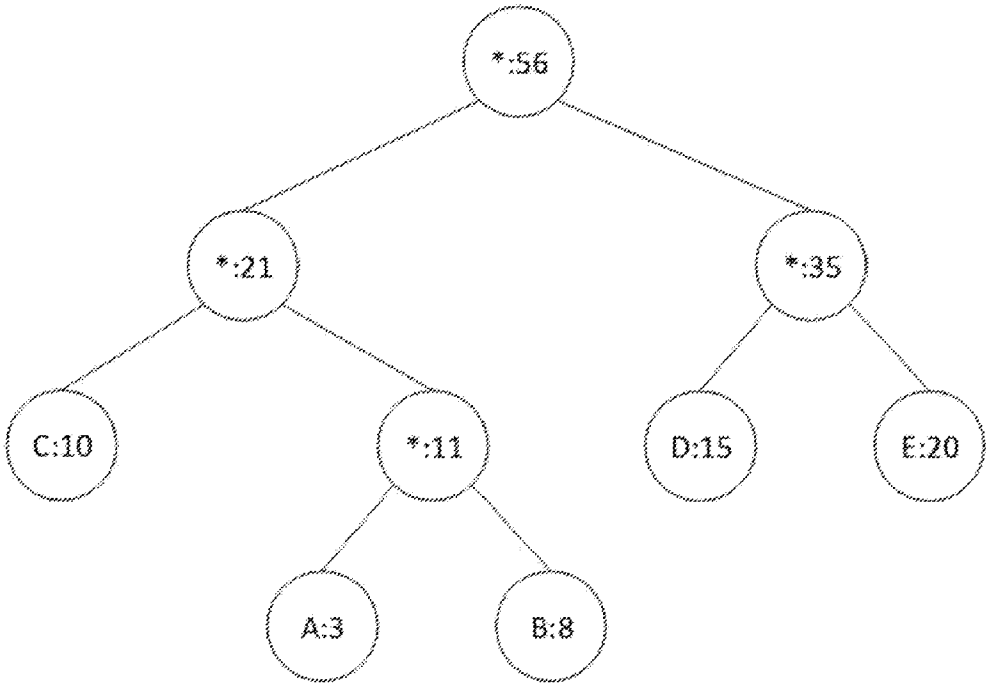

Figure 17d

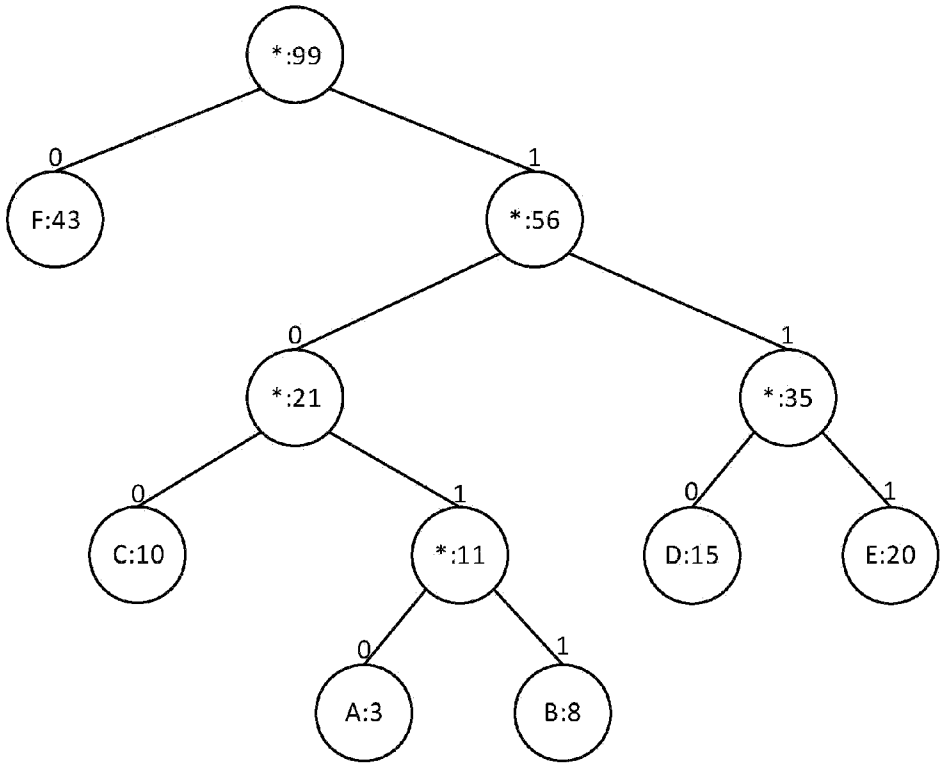

Figure 17e

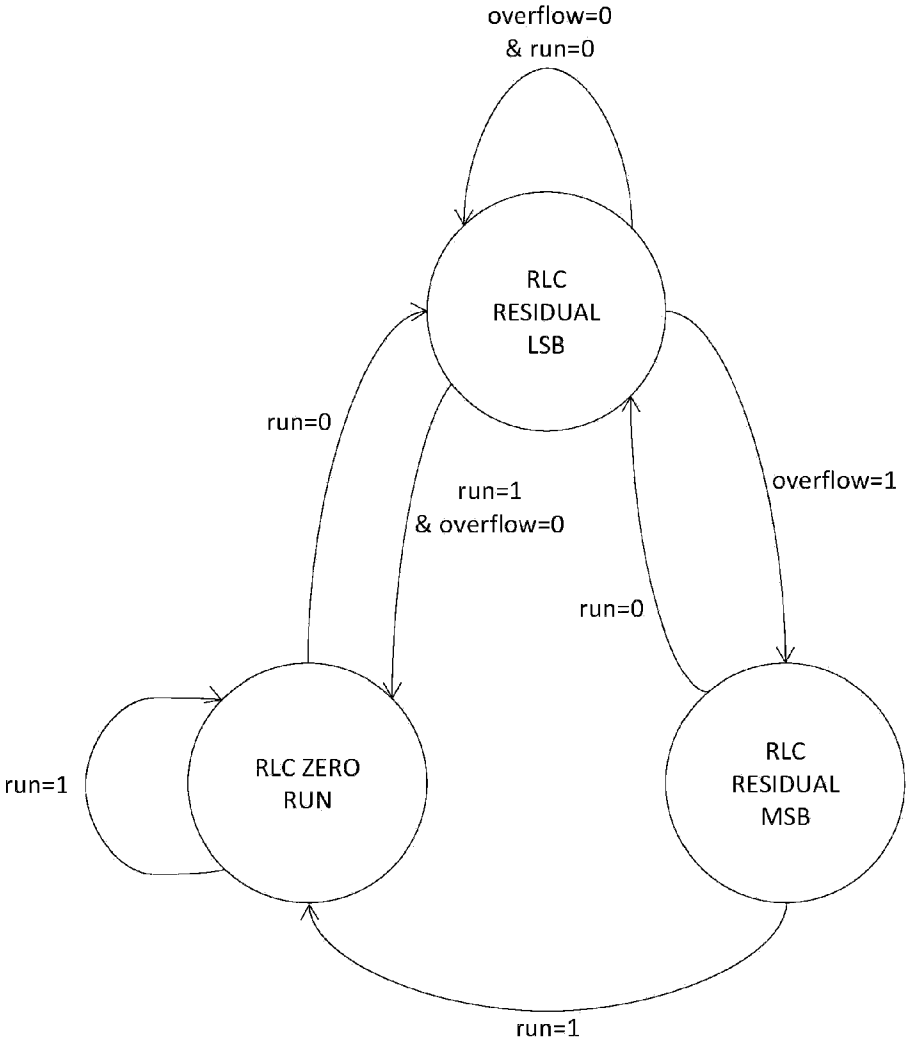
overflow=0
& run=0
RLC
RESIDUAL
LSB
run=0
run=1
& overflow=0
overflow=1
run=0
RLC ZERO
RUN
run=1
RLC
RESIDUAL
MSB
run=1

Figure 18