

(12) STANDARD PATENT
(19) AUSTRALIAN PATENT OFFICE

(11) Application No. **AU 2020239051 B2**

(54) Title
Coefficient coding for transform skip mode

(51) International Patent Classification(s)
H04N 19/13 (2014.01) **H04N 19/182** (2014.01)
H04N 19/157 (2014.01) **H04N 19/70** (2014.01)
H04N 19/18 (2014.01) **H04N 19/91** (2014.01)

(21) Application No: **2020239051** (22) Date of Filing: **2020.03.11**

(87) WIPO No: **WO20/185875**

(30) Priority Data

(31) Number	(32) Date	(33) Country
62/816,745	2019.03.11	US
62/850,453	2019.05.20	US
16/814,654	2020.03.10	US

(43) Publication Date: **2020.09.17**

(44) Accepted Journal Date: **2025.06.05**

(71) Applicant(s)
Qualcomm Incorporated

(72) Inventor(s)
KARCZEWICZ, Marta;COBAN, Muhammed Zeyd;WANG, Hongtao

(74) Agent / Attorney
Madderns Pty Ltd, GPO Box 2752, Adelaide, SA, 5001, AU

(56) Related Art
EP 2777260 A1



(51) International Patent Classification:

H04N 19/13 (2014.01) H04N 19/182 (2014.01)

H04N 19/157 (2014.01) H04N 19/70 (2014.01)

H04N 19/18 (2014.01) H04N 19/91 (2014.01)

(21) International Application Number:

PCT/US2020/022065

(22) International Filing Date:

11 March 2020 (11.03.2020)

(25) Filing Language:

English

(26) Publication Language:

English

(30) Priority Data:

62/816,745 11 March 2019 (11.03.2019) US

62/850,453 20 May 2019 (20.05.2019) US

16/814,654 10 March 2020 (10.03.2020) US

(71) Applicant: QUALCOMM INCORPORATED [US/US];

ATTN: International IP Administration, 5775 Morehouse Drive, San Diego, California 92121-1714 (US).

(72) Inventors: KARCZEWICZ, Marta; 5775 Morehouse

Drive, San Diego, California 92121-1714 (US). COBAN,

Muhammed Zeyd; 5775 Morehouse Drive, San Diego,

California 92121-1714 (US). WANG, Hongtao; 5775

Morehouse Drive, San Diego, California 92121-1714 (US).

(74) Agent: JOSEPH, Jeffrey R.; Shumaker & Sieffert, P.A., 1625 Radio Drive, Suite 100, Woodbury, Minnesota 55125 (US).

(81) Designated States (unless otherwise indicated, for every kind of national protection available):

AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

(84) Designated States (unless otherwise indicated, for every kind of regional protection available):

ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

(54) Title: COEFFICIENT CODING FOR TRANSFORM SKIP MODE

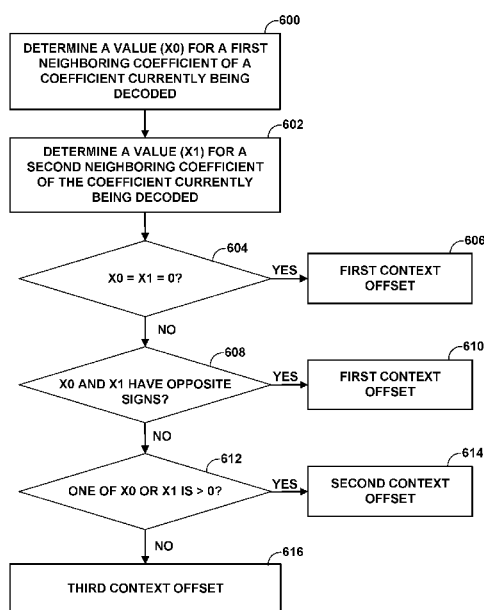


FIG. 12

(57) Abstract: A device for decoding video data determines, for a residual block of video data encoded using a transform skip mode, a value for a first neighboring coefficient of a coefficient currently being decoded; determines a value for a second neighboring coefficient of the coefficient currently being decoded; determines a context offset for the coefficient currently being decoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient; and decodes a value for the coefficient currently being decoded based on the determined context offset.

Published:

— *with international search report (Art. 21(3))*

COEFFICIENT CODING FOR TRANSFORM SKIP MODE

[0001] This application claims the benefit of:

U.S. Patent Application 16/814,654, filed 10 March 2020;
U.S. Provisional Patent Application 62/816,745, filed 11 March 2019; and
U.S. Provisional Patent Application 62/850,453, filed 20 May 2019,
the entire content of each being incorporated herein by reference.

TECHNICAL FIELD

[0002] This disclosure relates to video encoding and video decoding.

BACKGROUND

[0003] Digital video capabilities can be incorporated into a wide range of devices, including digital televisions, digital direct broadcast systems, wireless broadcast systems, personal digital assistants (PDAs), laptop or desktop computers, tablet computers, e-book readers, digital cameras, digital recording devices, digital media players, video gaming devices, video game consoles, cellular or satellite radio telephones, so-called “smart phones,” video teleconferencing devices, video streaming devices, and the like. Digital video devices implement video coding techniques, such as those described in the standards defined by MPEG-2, MPEG-4, ITU-T H.263, ITU-T H.264/MPEG-4, Part 10, Advanced Video Coding (AVC), ITU-T H.265/High Efficiency Video Coding (HEVC), and extensions of such standards. The video devices may transmit, receive, encode, decode, and/or store digital video information more efficiently by implementing such video coding techniques.

[0004] Video coding techniques include spatial (intra-picture) prediction and/or temporal (inter-picture) prediction to reduce or remove redundancy inherent in video sequences. For block-based video coding, a video slice (e.g., a video picture or a portion of a video picture) may be partitioned into video blocks, which may also be referred to as coding tree units (CTUs), coding units (CUs) and/or coding nodes. Video blocks in an intra-coded (I) slice of a picture are encoded using spatial prediction with respect to reference samples in neighboring blocks in the same picture. Video blocks in an inter-coded (P or B) slice of a picture may use spatial prediction with respect to reference samples in neighboring blocks in the same picture or temporal prediction with

respect to reference samples in other reference pictures. Pictures may be referred to as frames, and reference pictures may be referred to as reference frames.

SUMMARY

[0005] In some coding scenarios, a video encoder may encode video data in a transform skip mode in which the transform process is not performed, i.e., the transform process is skipped. Thus, for a block encoded in a transform skip mode, residual data is not transformed. This disclosure describes techniques for a coefficient coding scheme for transform skip mode. The techniques of this disclosure include an entropy decoding process that converts a binary representation of coefficients to a series of non-binary-valued quantized coefficients. The corresponding entropy encoding process, which is generally the reverse process of entropy decoding, is also part of this disclosure.

[0006] In one example, a method of decoding video data includes, for a residual block of video data encoded using a transform skip mode, determining a value for a first neighboring coefficient of a coefficient currently being decoded; determining a value for a second neighboring coefficient of the coefficient currently being decoded; determining a context offset for the coefficient currently being decoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient; and decoding a value for the coefficient currently being decoded based on the determined context offset.

[0007] In another example, a method of encoding video data includes, for a residual block of video data encoded using a transform skip mode, determining a value for a first neighboring coefficient of a coefficient currently being encoded; determining a value for a second neighboring coefficient of the coefficient currently being encoded; determining a context offset for the coefficient currently being encoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient; and encoding a value for the coefficient currently being encoded based on the determined context offset.

[0008] In another example, a device for decoding video data includes a memory configured to store video data and one or more processors implemented in circuitry and configured to: for a residual block of video data encoded using a transform skip mode, determine a value for a first neighboring coefficient of a coefficient currently being decoded; determine a value for a second neighboring coefficient of the coefficient currently being decoded; determine a context offset for the coefficient currently being

decoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient; and decode a value for the coefficient currently being decoded based on the determined context offset.

[0009] In another example, a device for encoding video data includes a memory configured to store video data and one or more processors implemented in circuitry and configured to: for a residual block of video data encoded using a transform skip mode, determine a value for a first neighboring coefficient of a coefficient currently being encoded; determine a value for a second neighboring coefficient of the coefficient currently being encoded; determine a context offset for the coefficient currently being encoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient; and encode a value for the coefficient currently being encoded based on the determined context offset.

[0010] In another example, an apparatus for decoding video data includes means for determining, for a residual block of video data encoded using a transform skip mode, a value for a first neighboring coefficient of a coefficient currently being decoded; means for determining a value for a second neighboring coefficient of the coefficient currently being decoded; means for determining a context offset for the coefficient currently being decoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient; and means for decoding a value for the coefficient currently being decoded based on the determined context offset.

[0011] In another example, an apparatus for encoding video data includes means for determining, for a residual block of video data encoded using a transform skip mode, a value for a first neighboring coefficient of a coefficient currently being encoded; means for determining a value for a second neighboring coefficient of the coefficient currently being encoded; means for determining a context offset for the coefficient currently being encoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient; and means for encoding a value for the coefficient currently being encoded based on the determined context offset.

[0012] In another example, a computer-readable storage medium stores instructions that when executed by one or more processors cause the one or more processors to determine, for a residual block of video data encoded using a transform skip mode, a value for a first neighboring coefficient of a coefficient currently being decoded; determine a value for a second neighboring coefficient of the coefficient currently being decoded; determine a context offset for the coefficient currently being decoded based on

the value for the first neighboring coefficient and the value for the second neighboring coefficient; and decode a value for the coefficient currently being decoded based on the determined context offset.

[0013] In another example, a computer-readable storage medium stores instructions that when executed by one or more processors cause the one or more processors to determine, for a residual block of video data encoded using a transform skip mode, a value for a first neighboring coefficient of a coefficient currently being encoded; determine a value for a second neighboring coefficient of the coefficient currently being encoded; determine a context offset for the coefficient currently being encoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient; and encode a value for the coefficient currently being encoded based on the determined context offset.

[0013A] In another example, a method for performing sign decoding for coefficients in a residual block of video data encoded using a transform skip mode, comprising:

- determining a value for a first neighboring coefficient of a coefficient currently being decoded;

- determining a value for a second neighboring coefficient of the coefficient currently being decoded;

- determining a context offset for the coefficient currently being decoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, comprising:

 - selecting a first context offset in response to the first neighboring coefficient and the second neighboring coefficient are both equal to zero or have opposite signs;

 - otherwise, selecting a second context offset in response to the first neighboring coefficient and the second neighboring coefficient are both positive or one of the first neighboring coefficient and the second neighboring coefficient is equal to zero and the other one of the first neighboring coefficient and the second neighboring coefficient is positive; and

 - otherwise, selecting a third context offset in response to the first neighboring coefficient and the second neighboring coefficient are both negative or one of the first neighboring coefficient and the second neighboring coefficient is equal to zero and the other one of the first neighboring coefficient and the second neighboring coefficient is negative; and

decoding a sign for the coefficient currently being decoded based on the determined context offset.

[0013B] In another example, a method for performing sign encoding for coefficients in a residual block of video data encoded using a transform skip mode, comprising:

- determining a value for a first neighboring coefficient of a coefficient currently being encoded;

- determining a value for a second neighboring coefficient of the coefficient currently being encoded;

- determining a context offset for the coefficient currently being encoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, comprising:

 - selecting a first context offset in response to the first neighboring coefficient and the second neighboring coefficient are both equal to zero or have opposite signs;

 - otherwise, selecting a second context offset in response to the first neighboring coefficient and the second neighboring coefficient are both positive or one of the first neighboring coefficient and the second neighboring coefficient is equal to zero and the other one of the first neighboring coefficient and the second neighboring coefficient is positive; and

 - otherwise, selecting a third context offset in response to the first neighboring coefficient and the second neighboring coefficient are both negative or one of the first neighboring coefficient and the second neighboring coefficient is equal to zero and the other one of the first neighboring coefficient and the second neighboring coefficient is negative; and

- encoding a sign for the coefficient currently being encoded based on the determined context offset.

[0013C] In another example, a device for performing sign decoding for coefficients in a residual block of video data encoded using a transform skip mode, the device comprising:

- a memory configured to store video data; and
- one or more processors configured to:

 - determine a value for a first neighboring coefficient of a coefficient currently being decoded;

determine a value for a second neighboring coefficient of the coefficient currently being decoded;

determine a context offset for the coefficient currently being decoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, wherein to determine the context offset for the coefficient currently being decoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, the one or more processors are configured to:

select a first context offset in response to the first neighboring coefficient and the second neighboring coefficient are both equal to zero or have opposite signs;

otherwise, select a second context offset in response to the first neighboring coefficient and the second neighboring coefficient are both positive or one of the first neighboring coefficient and the second neighboring coefficient is equal to zero and the other one of the first neighboring coefficient and the second neighboring coefficient is positive; and

otherwise, select a third context offset in response to the first neighboring coefficient and the second neighboring coefficient are both negative or one of the first neighboring coefficient and the second neighboring coefficient is equal to zero and the other one of the first neighboring coefficient and the second neighboring coefficient is negative; and

decode a sign for the coefficient currently being decoded based on the determined context offset.

[0013D] In another example, a device for performing sign encoding for coefficients of a residual block of video data encoded using a transform skip mode, the device comprising:

a memory configured to store video data; and
one or more processors configured to:

determine a value for a first neighboring coefficient of a coefficient currently being encoded;

determine a value for a second neighboring coefficient of the coefficient currently being encoded;

determine a context offset for the coefficient currently being encoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, wherein to determine the context offset for the coefficient currently being decoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, the one or more processors are configured to:

select a first context offset in response to the first neighboring coefficient and the second neighboring coefficient are both equal to zero or have opposite signs;

otherwise, select a second context offset in response to the first neighboring coefficient and the second neighboring coefficient are both positive or one of the first neighboring coefficient and the second neighboring coefficient is equal to zero and the other one of the first neighboring coefficient and the second neighboring coefficient is positive; and

otherwise, select a third context offset in response to the first neighboring coefficient and the second neighboring coefficient are both negative or one of the first neighboring coefficient and the second neighboring coefficient is equal to zero and the other one of the first neighboring coefficient and the second neighboring coefficient is negative; and

encode a sign for the coefficient currently being encoded based on the determined context offset.

[0013E] In another example, a non-transitory computer-readable storage medium storing instructions for performing sign decoding for coefficients of a residual block of video data encoded using a transform skip mode, wherein the instruction when executed by one or more processors cause the one or more processors to:

determine a value for a first neighboring coefficient of a coefficient currently being decoded;

determine a value for a second neighboring coefficient of the coefficient currently being decoded;

determine a context offset for the coefficient currently being decoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient; wherein to determine the context offset for the

coefficient currently being decoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, the instructions cause the one or more processors:

select a first context offset in response to the first neighboring coefficient and the second neighboring coefficient are both equal to zero or have opposite signs;

otherwise, select a second context offset in response to the first neighboring coefficient and the second neighboring coefficient are both positive or one of the first neighboring coefficient and the second neighboring coefficient is equal to zero and the other one of the first neighboring coefficient and the second neighboring coefficient is positive; and

otherwise, select a third offset in response to the first neighboring coefficient and the second neighboring coefficient are both negative or one of the first neighboring coefficient and the second neighboring coefficient is equal to zero and the other one of the first neighboring coefficient and the second neighboring coefficient is negative; and

decode a sign for the coefficient currently being decoded based on the determined context offset.

[0014] The details of one or more examples are set forth in the accompanying drawings and the description below. Other features, objects, and advantages will be apparent from the description, drawings, and claims.

BRIEF DESCRIPTION OF DRAWINGS

[0015] FIG. 1 is a block diagram illustrating an example video encoding and decoding system that may perform the techniques of this disclosure.

[0016] FIGS. 2A and 2B are conceptual diagrams illustrating an example quadtree binary tree (QTBT) structure, and a corresponding coding tree unit (CTU).

[0017] FIG. 3 shows an example of neighboring coefficients of a coefficient currently being encoded or decoded.

[0018] FIG. 4 is a block diagram illustrating an example video encoder that may perform the techniques of this disclosure.

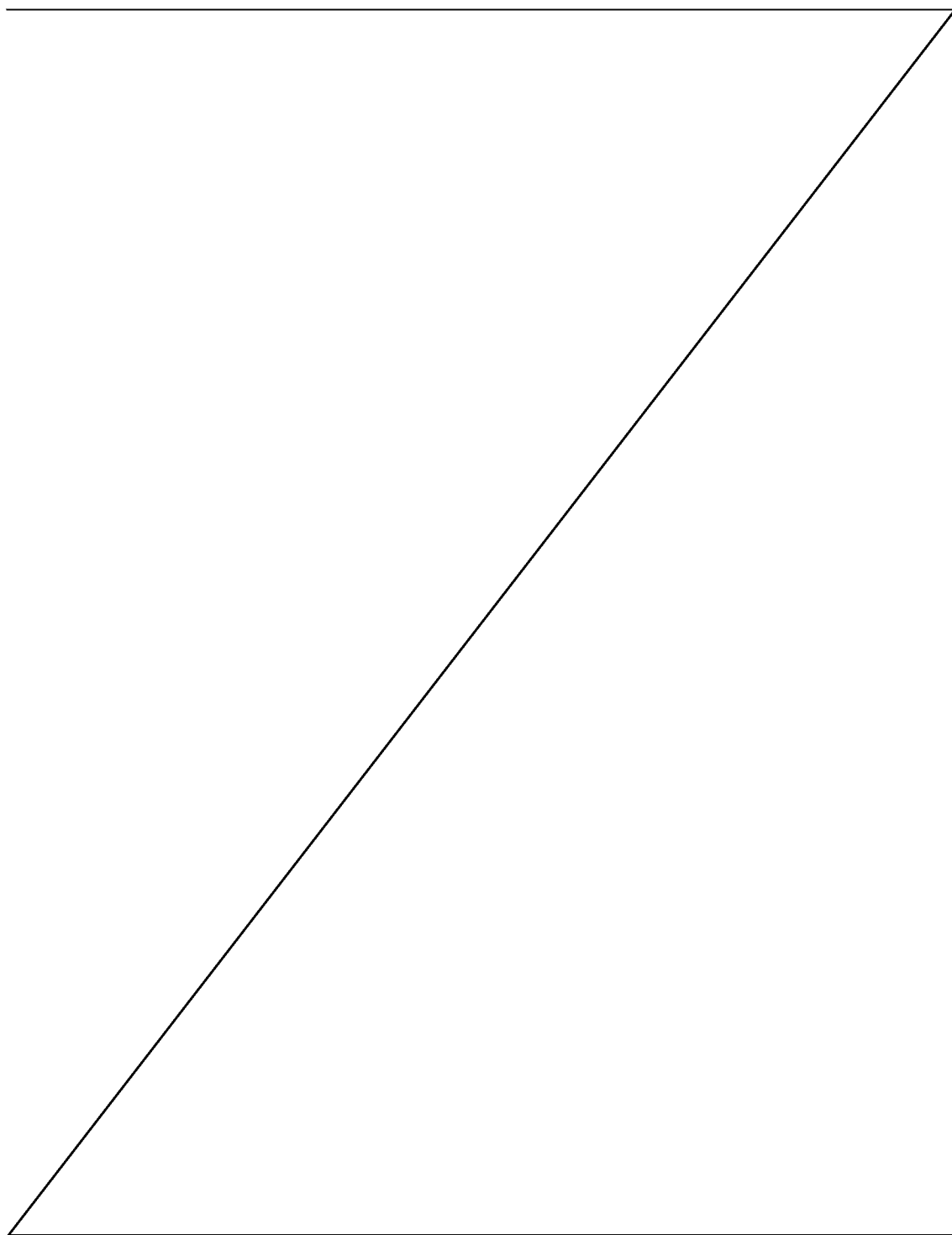
[0019] FIG. 5 is a block diagram illustrating an example video decoder that may perform the techniques of this disclosure.

[0020] FIGS. 6A and 6B are conceptual diagrams illustrating a range update process in binary arithmetic coding.

[0021] FIG. 7 is a conceptual diagram illustrating an output process in binary arithmetic coding.

[0022] FIG. 8 is a block diagram illustrating a context adaptive binary arithmetic coding (CABAC) coder in a video encoder.

[0023] FIG. 9 is a block diagram illustrating a CABAC coder in a video decoder.



[0024] FIG. 10 is a flowchart illustrating a process for encoding video data.

[0025] FIG. 11 is a flowchart illustrating a process for decoding video data.

[0026] FIG. 12 is a flowchart illustrating a process for determining a context for coding a sign of a coefficient of a residual block.

DETAILED DESCRIPTION

[0027] Video coding (e.g., video encoding and/or video decoding) typically involves predicting a block of video data from either an already coded block of video data in the same picture (e.g., intra prediction) or an already coded block of video data in a different picture (e.g., inter prediction). In some instances, the video encoder also calculates residual data by comparing the prediction block to the original block. Thus, the residual data represents a difference between the prediction block and the original block. To reduce the number of bits needed to signal the residual data, the video encoder may transform and quantize the residual data and signal the transformed and quantized residual data in the encoded bitstream. The compression achieved by the transform and quantization processes may be lossy, meaning that transform and quantization processes may introduce distortion into the decoded video data.

[0028] A video decoder decodes and adds the residual data to the prediction block to produce a reconstructed video block that matches the original video block more closely than the prediction block alone. Due to the loss introduced by the transforming and quantizing of the residual data, the first reconstructed block may have distortion or artifacts. One common type of artifact or distortion is referred to as blockiness, where the boundaries of the blocks used to code the video data are visible.

[0029] To further improve the quality of decoded video, a video decoder can perform one or more filtering operations on the reconstructed video blocks. Examples of these filtering operations include deblocking filtering, sample adaptive offset (SAO) filtering, and adaptive loop filtering (ALF). Parameters for these filtering operations may either be determined by a video encoder and explicitly signaled in the encoded video bitstream or may be implicitly determined by a video decoder without needing the parameters to be explicitly signaled in the encoded video bitstream.

[0030] In some coding scenarios, a video encoder may encode video data in a transform skip mode in which the transform process described above is not performed, i.e., the transform process is skipped. Thus, for a block encoded in a transform skip mode, the residual data is not transformed. A residual block of video data encoded using a

transform skip mode may also be referred to as an untransformed residual block. This disclosure describes techniques for a coefficient coding scheme for transform skip mode. The techniques of this disclosure include an entropy decoding process that converts a binary representation to a series of non-binary-valued quantized coefficients. The corresponding entropy encoding process, which is generally the reverse process of entropy decoding, is also part of this disclosure. The techniques of this disclosure may be applied to any of the existing video codecs, such as High Efficiency Video Coding (HEVC), or to a standard currently being developed, such as Versatile Video Coding (VVC), and to other future video coding standards.

[0031] This disclosure proposes techniques that include, for example, determining a context offset for a coefficient currently being decoded based on a value for a first neighboring coefficient and a value for a second neighboring coefficient and decoding a value for the coefficient currently being decoded based on the determined context offset. As coefficient values among neighbor coefficients in residual blocks tend to be more correlated for transform skip blocks than for transformed blocks, the techniques of this disclosure may result in improved entropy coding, which can improve overall coding efficiency, by for example, reducing the bit overhead needed to represent encoded video data without degrading the quality of the decoded video data.

[0032] FIG. 1 is a block diagram illustrating an example video encoding and decoding system 100 that may perform the techniques of this disclosure. The techniques of this disclosure are generally directed to coding (encoding and/or decoding) video data. In general, video data includes any data for processing a video. Thus, video data may include raw, uncoded video, encoded video, decoded (e.g., reconstructed) video, and video metadata, such as signaling data.

[0033] As shown in FIG. 1, system 100 includes a source device 102 that provides encoded video data to be decoded and displayed by a destination device 116, in this example. In particular, source device 102 provides the video data to destination device 116 via a computer-readable medium 110. Source device 102 and destination device 116 may be or include any of a wide range of devices, including a desktop computer, notebook (i.e., laptop) computer, tablet computer or other mobile device, set-top box, telephone handset, smartphone, television, camera, display device, digital media player, video gaming console, video streaming device, broadcast receiver device, or the like. In some cases, source device 102 and destination device 116 may be equipped for wireless communication, and thus may be referred to as wireless communication devices.

[0034] In the example of FIG. 1, source device 102 includes video source 104, memory 106, video encoder 200, and output interface 108. Destination device 116 includes input interface 122, video decoder 300, memory 120, and display device 118. In accordance with this disclosure, video encoder 200 of source device 102 and video decoder 300 of destination device 116 may be configured to apply the techniques for coefficient coding described in this disclosure. Thus, source device 102 represents an example of a video encoding device, while destination device 116 represents an example of a video decoding device. In other examples, a source device and a destination device may include other components or arrangements. For example, source device 102 may receive video data from an external video source, such as an external camera. Likewise, destination device 116 may interface with an external display device, rather than including an integrated display device.

[0035] System 100 as shown in FIG. 1 is merely one example. In general, any digital video encoding and/or decoding device may perform techniques for coefficient coding described in this disclosure. Source device 102 and destination device 116 are merely examples of such coding devices in which source device 102 generates coded video data for transmission to destination device 116. This disclosure refers to a “coding” device as a device that performs coding (encoding and/or decoding) of data. Thus, video encoder 200 and video decoder 300 represent examples of coding devices, in particular, a video encoder and a video decoder, respectively. In some examples, devices 102, 116 may operate in a substantially symmetrical manner such that each of devices 102, 116 include video encoding and decoding components. Hence, system 100 may support one-way or two-way video transmission between video devices 102, 116, e.g., for video streaming, video playback, video broadcasting, or video telephony.

[0036] In general, video source 104 represents a source of video data (i.e., raw, uncoded video data) and provides a sequential series of pictures (also referred to as “frames”) of the video data to video encoder 200, which encodes data for the pictures. Video source 104 of source device 102 may include a video capture device, such as a video camera, a video archive containing previously captured raw video, and/or a video feed interface to receive video from a video content provider. As a further alternative, video source 104 may generate computer graphics-based data as the source video, or a combination of live video, archived video, and computer-generated video. In each case, video encoder 200 encodes the captured, pre-captured, or computer-generated video data. Video encoder 200 may rearrange the pictures from the received order (sometimes referred to

as “display order”) into a coding order for coding. Video encoder 200 may generate a bitstream including encoded video data. Source device 102 may then output the encoded video data via output interface 108 onto computer-readable medium 110 for reception and/or retrieval by, e.g., input interface 122 of destination device 116.

[0037] Memory 106 of source device 102 and memory 120 of destination device 116 represent general purpose memories. In some examples, memories 106, 120 may store raw video data, e.g., raw video from video source 104 and raw, decoded video data from video decoder 300. Additionally or alternatively, memories 106, 120 may store software instructions executable by, e.g., video encoder 200 and video decoder 300, respectively. Although memory 106 and memory 120 are shown separately from video encoder 200 and video decoder 300 in this example, it should be understood that video encoder 200 and video decoder 300 may also include internal memories for functionally similar or equivalent purposes. Furthermore, memories 106, 120 may store encoded video data, e.g., output from video encoder 200 and input to video decoder 300. In some examples, portions of memories 106, 120 may be allocated as one or more video buffers, e.g., to store raw, decoded, and/or encoded video data.

[0038] Computer-readable medium 110 may represent any type of medium or device capable of transporting the encoded video data from source device 102 to destination device 116. In one example, computer-readable medium 110 represents a communication medium to enable source device 102 to transmit encoded video data directly to destination device 116 in real-time, e.g., via a radio frequency network or computer-based network. Output interface 108 may modulate, according to a wireless communication standard, a transmission signal including the encoded video data, and input interface 122 may demodulate the received transmission signal, according to a communication standard, such as a wireless communication protocol. The communication medium may comprise any wireless or wired communication medium, such as a radio frequency (RF) spectrum or one or more physical transmission lines. The communication medium may form part of a packet-based network, such as a local area network, a wide-area network, or a global network such as the Internet. The communication medium may include routers, switches, base stations, or any other equipment that may be useful to facilitate communication from source device 102 to destination device 116.

[0039] In some examples, computer-readable medium 110 may include storage device 112. Source device 102 may output encoded data from output interface 108 to storage

device 112. Similarly, destination device 116 may access encoded data from storage device 112 via input interface 122. Storage device 112 may include any of a variety of distributed or locally accessed data storage media such as a hard drive, Blu-ray discs, DVDs, CD-ROMs, flash memory, volatile or non-volatile memory, or any other suitable digital storage media for storing encoded video data.

[0040] In some examples, computer-readable medium 110 may include file server 114 or another intermediate storage device that may store the encoded video data generated by source device 102. Source device 102 may output encoded video data to file server 114 or another intermediate storage device that may store the encoded video data generated by source device 102. Destination device 116 may access stored video data from file server 114 via streaming or download. File server 114 may be any type of server device capable of storing encoded video data and transmitting that encoded video data to the destination device 116. File server 114 may represent a web server (e.g., for a website), a File Transfer Protocol (FTP) server, a content delivery network device, or a network attached storage (NAS) device. Destination device 116 may access encoded video data from file server 114 through any standard data connection, including an Internet connection. This may include a wireless channel (e.g., a Wi-Fi connection), a wired connection (e.g., digital subscriber line (DSL), cable modem, etc.), or a combination of both that is suitable for accessing encoded video data stored on file server 114. File server 114 and input interface 122 may be configured to operate according to a streaming transmission protocol, a download transmission protocol, or a combination thereof.

[0041] Output interface 108 and input interface 122 may represent wireless transmitters/receivers, modems, wired networking components (e.g., Ethernet cards), wireless communication components that operate according to any of a variety of IEEE 802.11 standards, or other physical components. In examples where output interface 108 and input interface 122 comprise wireless components, output interface 108 and input interface 122 may be configured to transfer data, such as encoded video data, according to a cellular communication standard, such as 4G, 4G-LTE (Long-Term Evolution), LTE Advanced, 5G, or the like. In some examples where output interface 108 comprises a wireless transmitter, output interface 108 and input interface 122 may be configured to transfer data, such as encoded video data, according to other wireless standards, such as an IEEE 802.11 specification, an IEEE 802.15 specification (e.g., ZigBee™), a Bluetooth™ standard, or the like. In some examples, source device 102

and/or destination device 116 may include respective system-on-a-chip (SoC) devices. For example, source device 102 may include an SoC device to perform the functionality attributed to video encoder 200 and/or output interface 108, and destination device 116 may include an SoC device to perform the functionality attributed to video decoder 300 and/or input interface 122.

[0042] The techniques of this disclosure may be applied to video coding in support of any of a variety of multimedia applications, such as over-the-air television broadcasts, cable television transmissions, satellite television transmissions, Internet streaming video transmissions, such as dynamic adaptive streaming over HTTP (DASH), digital video that is encoded onto a data storage medium, decoding of digital video stored on a data storage medium, or other applications.

[0043] Input interface 122 of destination device 116 receives an encoded video bitstream from computer-readable medium 110 (e.g., a communication medium, storage device 112, file server 114, or the like). The encoded video bitstream may include signaling information defined by video encoder 200, which is also used by video decoder 300, such as syntax elements having values that describe characteristics and/or processing of video blocks or other coded units (e.g., slices, pictures, groups of pictures, sequences, or the like). Display device 118 displays decoded pictures of the decoded video data to a user. Display device 118 may represent any of a variety of display devices such as a cathode ray tube (CRT), a liquid crystal display (LCD), a plasma display, an organic light emitting diode (OLED) display, or another type of display device.

[0044] Although not shown in FIG. 1, in some examples, video encoder 200 and video decoder 300 may each be integrated with an audio encoder and/or audio decoder, and may include appropriate MUX-DEMUX units, or other hardware and/or software, to handle multiplexed streams including both audio and video in a common data stream. If applicable, MUX-DEMUX units may conform to the ITU H.223 multiplexer protocol, or other protocols such as the user datagram protocol (UDP).

[0045] Video encoder 200 and video decoder 300 each may be implemented as any of a variety of suitable encoder and/or decoder circuitry, such as one or more microprocessors, digital signal processors (DSPs), application specific integrated circuits (ASICs), field programmable gate arrays (FPGAs), discrete logic, software, hardware, firmware or any combinations thereof. When the techniques are implemented partially in software, a device may store instructions for the software in a suitable, non-

transitory computer-readable medium and execute the instructions in hardware using one or more processors to perform the techniques of this disclosure. Each of video encoder 200 and video decoder 300 may be included in one or more encoders or decoders, either of which may be integrated as part of a combined encoder/decoder (CODEC) in a respective device. A device including video encoder 200 and/or video decoder 300 may comprise an integrated circuit, a microprocessor, and/or a wireless communication device, such as a cellular telephone.

[0046] Video encoder 200 and video decoder 300 may operate according to a video coding standard, such as ITU-T H.265, also referred to as High Efficiency Video Coding (HEVC) or extensions thereto, such as the multi-view and/or scalable video coding extensions. Alternatively, video encoder 200 and video decoder 300 may operate according to other proprietary or industry standards, such as the Joint Exploration Test Model (JEM) or ITU-T H.266, also referred to as Versatile Video Coding (VVC). A draft of the VVC standard is described in Bross, et al. “Versatile Video Coding (Draft 4),” Joint Video Experts Team (JVET) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, 13th Meeting: Marrakech, MA, 9-18 January 2019, JVET-M1001-v5 (hereinafter “VVC Draft 4”). Another draft of the VVC standard is described in Bross, et al. “Versatile Video Coding (Draft 7),” Joint Video Experts Team (JVET) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, 16th Meeting: Geneva, CH, 1–11 October 2019, JVET-P2001-v14 (hereinafter “VVC Draft 7”). The techniques of this disclosure, however, are not limited to any particular coding standard.

[0047] In general, video encoder 200 and video decoder 300 may perform block-based coding of pictures. The term “block” generally refers to a structure including data to be processed (e.g., encoded, decoded, or otherwise used in the encoding and/or decoding process). For example, a block may include a two-dimensional matrix of samples of luminance and/or chrominance data. In general, video encoder 200 and video decoder 300 may code video data represented in a YUV (e.g., Y, Cb, Cr) format. That is, rather than coding red, green, and blue (RGB) data for samples of a picture, video encoder 200 and video decoder 300 may code luminance and chrominance components, where the chrominance components may include both red hue and blue hue chrominance components. In some examples, video encoder 200 converts received RGB formatted data to a YUV representation prior to encoding, and video decoder 300 converts the YUV representation to the RGB format. Alternatively, pre- and post-processing units (not shown) may perform these conversions.

[0048] This disclosure may generally refer to coding (e.g., encoding and decoding) of pictures to include the process of encoding or decoding data of the picture. Similarly, this disclosure may refer to coding of blocks of a picture to include the process of encoding or decoding data for the blocks, e.g., prediction and/or residual coding. An encoded video bitstream generally includes a series of values for syntax elements representative of coding decisions (e.g., coding modes) and partitioning of pictures into blocks. Thus, references to coding a picture or a block should generally be understood as coding values for syntax elements forming the picture or block.

[0049] HEVC defines various blocks, including coding units (CUs), prediction units (PUs), and transform units (TUs). According to HEVC, a video coder (such as video encoder 200) partitions a coding tree unit (CTU) into CUs according to a quadtree structure. That is, the video coder partitions CTUs and CUs into four equal, non-overlapping squares, and each node of the quadtree has either zero or four child nodes. Nodes without child nodes may be referred to as “leaf nodes,” and CUs of such leaf nodes may include one or more PUs and/or one or more TUs. The video coder may further partition PUs and TUs. For example, in HEVC, a residual quadtree (RQT) represents partitioning of TUs. In HEVC, PUs represent inter-prediction data, while TUs represent residual data. CUs that are intra-predicted include intra-prediction information, such as an intra-mode indication.

[0050] As another example, video encoder 200 and video decoder 300 may be configured to operate according to JEM or VVC. According to JEM or VVC, a video coder (such as video encoder 200) partitions a picture into a plurality of coding tree units (CTUs). Video encoder 200 may partition a CTU according to a tree structure, such as a quadtree-binary tree (QTBT) structure or Multi-Type Tree (MTT) structure. The QTBT structure removes the concepts of multiple partition types, such as the separation between CUs, PUs, and TUs of HEVC. A QTBT structure includes two levels: a first level partitioned according to quadtree partitioning, and a second level partitioned according to binary tree partitioning. A root node of the QTBT structure corresponds to a CTU. Leaf nodes of the binary trees correspond to coding units (CUs).

[0051] In an MTT partitioning structure, blocks may be partitioned using a quadtree (QT) partition, a binary tree (BT) partition, and one or more types of triple tree (TT) (also called ternary tree (TT)) partitions. A triple or ternary tree partition is a partition where a block is split into three sub-blocks. In some examples, a triple or ternary tree partition divides a block into three sub-blocks without dividing the original block

through the center. The partitioning types in MTT (e.g., QT, BT, and TT), may be symmetrical or asymmetrical.

[0052] In some examples, video encoder 200 and video decoder 300 may use a single QTBT or MTT structure to represent each of the luminance and chrominance components, while in other examples, video encoder 200 and video decoder 300 may use two or more QTBT or MTT structures, such as one QTBT/MTT structure for the luminance component and another QTBT/MTT structure for both chrominance components (or two QTBT/MTT structures for respective chrominance components).

[0053] Video encoder 200 and video decoder 300 may be configured to use quadtree partitioning per HEVC, QTBT partitioning, MTT partitioning, or other partitioning structures. For purposes of explanation, the description of the techniques of this disclosure is presented with respect to QTBT partitioning. However, it should be understood that the techniques of this disclosure may also be applied to video coders configured to use quadtree partitioning, or other types of partitioning as well.

[0054] The blocks (e.g., CTUs or CUs) may be grouped in various ways in a picture. As one example, a brick may refer to a rectangular region of CTU rows within a particular tile in a picture. A tile may be a rectangular region of CTUs within a particular tile column and a particular tile row in a picture. A tile column refers to a rectangular region of CTUs having a height equal to the height of the picture and a width specified by syntax elements (e.g., such as in a picture parameter set). A tile row refers to a rectangular region of CTUs having a height specified by syntax elements (e.g., such as in a picture parameter set) and a width equal to the width of the picture.

[0055] In some examples, a tile may be partitioned into multiple bricks, each of which may include one or more CTU rows within the tile. A tile that is not partitioned into multiple bricks may also be referred to as a brick. However, a brick that is a true subset of a tile may not be referred to as a tile.

[0056] The bricks in a picture may also be arranged in a slice. A slice may be an integer number of bricks of a picture that may be exclusively contained in a single network abstraction layer (NAL) unit. In some examples, a slice includes either a number of complete tiles or only a consecutive sequence of complete bricks of one tile.

[0057] This disclosure may use “N×N” and “N by N” interchangeably to refer to the sample dimensions of a block (such as a CU or other video block) in terms of vertical and horizontal dimensions, e.g., 16x16 samples or 16 by 16 samples. In general, a 16x16 CU will have 16 samples in a vertical direction ($y = 16$) and 16 samples in a

horizontal direction ($x = 16$). Likewise, an $N \times N$ CU generally has N samples in a vertical direction and N samples in a horizontal direction, where N represents a nonnegative integer value. The samples in a CU may be arranged in rows and columns. Moreover, CUs need not necessarily have the same number of samples in the horizontal direction as in the vertical direction. For example, CUs may comprise $N \times M$ samples, where M is not necessarily equal to N .

[0058] Video encoder 200 encodes video data for CUs representing prediction and/or residual information, and other information. The prediction information indicates how the CU is to be predicted in order to form a prediction block for the CU. The residual information generally represents sample-by-sample differences between samples of the CU prior to encoding and the prediction block.

[0059] To predict a CU, video encoder 200 may generally form a prediction block for the CU through inter-prediction or intra-prediction. Inter-prediction generally refers to predicting the CU from data of a previously coded picture, whereas intra-prediction generally refers to predicting the CU from previously coded data of the same picture. To perform inter-prediction, video encoder 200 may generate the prediction block using one or more motion vectors. Video encoder 200 may generally perform a motion search to identify a reference block that closely matches the CU, e.g., in terms of differences between the CU and the reference block. Video encoder 200 may calculate a difference metric using a sum of absolute difference (SAD), sum of squared differences (SSD), mean absolute difference (MAD), mean squared differences (MSD), or other such difference calculations to determine whether a reference block closely matches the current CU. In some examples, video encoder 200 may predict the current CU using uni-directional prediction or bi-directional prediction.

[0060] Some examples of JEM and VVC also provide an affine motion compensation mode, which may be considered an inter-prediction mode. In affine motion compensation mode, video encoder 200 may determine two or more motion vectors that represent non-translational motion, such as zoom in or out, rotation, perspective motion, or other irregular motion types.

[0061] To perform intra-prediction, video encoder 200 may select an intra-prediction mode to generate the prediction block. Some examples of JEM and VVC provide sixty-seven intra-prediction modes, including various directional modes, as well as planar mode and DC mode. In general, video encoder 200 selects an intra-prediction mode that describes neighboring samples to a current block (e.g., a block of a CU) from which

to predict samples of the current block. Such samples may generally be above, above and to the left, or to the left of the current block in the same picture as the current block, assuming video encoder 200 codes CTUs and CUs in raster scan order (left to right, top to bottom).

[0062] Video encoder 200 encodes data representing the prediction mode for a current block. For example, for inter-prediction modes, video encoder 200 may encode data representing which of the various available inter-prediction modes is used, as well as motion information for the corresponding mode. For uni-directional or bi-directional inter-prediction, for example, video encoder 200 may encode motion vectors using advanced motion vector prediction (AMVP) or merge mode. Video encoder 200 may use similar modes to encode motion vectors for affine motion compensation mode.

[0063] Following prediction, such as intra-prediction or inter-prediction of a block, video encoder 200 may calculate residual data for the block. The residual data, such as a residual block, represents sample by sample differences between the block and a prediction block for the block, formed using the corresponding prediction mode. Video encoder 200 may apply one or more transforms to the residual block, to produce transformed data in a transform domain instead of the sample domain. For example, video encoder 200 may apply a discrete cosine transform (DCT), an integer transform, a wavelet transform, or a conceptually similar transform to residual video data. Additionally, video encoder 200 may apply a secondary transform following the first transform, such as a mode-dependent non-separable secondary transform (MDNSST), a signal dependent transform, a Karhunen-Loeve transform (KLT), or the like. Video encoder 200 produces transform coefficients following application of the one or more transforms.

[0064] Although the above describes examples where transforms are preformed, in some examples, the transform may be skipped. For instance, video encoder 200 may implement transform skip mode in which the transform operation is skipped. In examples where transform is skipped, video encoder 200 may output coefficients corresponding to residual values instead of transform coefficients. In the following description, the term “coefficient” should be interpreted to include either coefficients corresponding to residual values or transform coefficients generated from the result of a transform.

[0065] As noted above, following any transforms or where transform is skipped to produce coefficients, video encoder 200 may perform quantization of the coefficients.

In some examples, quantization may also be skipped when transform is skipped. Quantization generally refers to a process in which coefficients are quantized to possibly reduce the amount of data used to represent the coefficients, providing further compression. By performing the quantization process, video encoder 200 may reduce the bit depth associated with some or all of the coefficients. For example, video encoder 200 may round an n -bit value down to an m -bit value during quantization, where n is greater than m . In some examples, to perform quantization, video encoder 200 may perform a bitwise right-shift of the value to be quantized.

[0066] Following quantization, video encoder 200 may scan the coefficients (e.g., generated from the result of the transform or due to transform skip), producing a one-dimensional vector from the two-dimensional matrix including the quantized coefficients. The scan may be designed to place higher energy (and therefore lower frequency) coefficients at the front of the vector and to place lower energy (and therefore higher frequency) coefficients at the back of the vector. In examples where transform is skipped, the result of the scan may not be that higher energy coefficients are at the front of the vector and lower energy coefficients are at the back of the vector. In some examples, video encoder 200 may utilize a predefined scan order to scan the quantized coefficients to produce a serialized vector, and then entropy encode the quantized coefficients of the vector. In other examples, video encoder 200 may perform an adaptive scan. After scanning the quantized coefficients to form the one-dimensional vector, video encoder 200 may entropy encode the one-dimensional vector, e.g., according to context-adaptive binary arithmetic coding (CABAC). Video encoder 200 may also entropy encode values for syntax elements describing metadata associated with the encoded video data for use by video decoder 300 in decoding the video data.

[0067] As introduced above, video encoder 200 encodes residual data in TUs. Depending on the expected characteristics of the residual data in a TU, video encoder 200 may encode TUs in different modes, such as a transform mode or a transform skip mode, with different modes utilizing different coefficient coding schemes. Some coefficient coding schemes utilize coefficient groups to encode a TU. A coefficient group generally is a subset of the coefficients in a TU. For example, video encoder 200 may encode a 16x16 TU as four 4x4 coefficient groups.

[0068] To perform CABAC, video encoder 200 may assign a context within a context model to a symbol to be transmitted. The context may relate to, for example, whether

neighboring values of the symbol are zero-valued or not. The probability determination may be based on a context assigned to the symbol.

[0069] Video encoder 200 may further generate syntax data, such as block-based syntax data, picture-based syntax data, and sequence-based syntax data, to video decoder 300, e.g., in a picture header, a block header, a slice header, or other syntax data, such as a sequence parameter set (SPS), picture parameter set (PPS), or video parameter set (VPS). Video decoder 300 may likewise decode such syntax data to determine how to decode corresponding video data.

[0070] In this manner, video encoder 200 may generate a bitstream including encoded video data, e.g., syntax elements describing partitioning of a picture into blocks (e.g., CUs) and prediction and/or residual information for the blocks. Ultimately, video decoder 300 may receive the bitstream and decode the encoded video data.

[0071] In general, video decoder 300 performs a reciprocal process to that performed by video encoder 200 to decode the encoded video data of the bitstream. For example, video decoder 300 may decode values for syntax elements of the bitstream using CABAC in a manner substantially similar to, albeit reciprocal to, the CABAC encoding process of video encoder 200. The syntax elements may define partitioning information of a picture into CTUs and partitioning of each CTU according to a corresponding partition structure, such as a QTBT structure, to define CUs of the CTU. The syntax elements may further define prediction and residual information for blocks (e.g., CUs) of video data.

[0072] The residual information may be represented by, for example, quantized coefficients that represent either residual values or transform coefficients. Video decoder 300 may inverse quantize and inverse transform the quantized transform coefficients of a block to reproduce a residual block for the block. In examples where video encoder 200 skipped the transform operation (e.g., transform skip mode), video decoder 300 may skip the inverse transform operation. Video decoder 300 uses a signaled prediction mode (intra- or inter-prediction) and related prediction information (e.g., motion information for inter-prediction) to form a prediction block (i.e., predictive block) for the block. Video decoder 300 may then combine the prediction block and the residual block (on a sample-by-sample basis) to reproduce the original block. Video decoder 300 may perform additional processing, such as performing a deblocking process to reduce visual artifacts along boundaries of the block.

[0073] According to the techniques of this disclosure, video encoder 200 and video decoder 300 may be configured to determine a context offset for a coefficient currently being decoded based on a value for a first neighboring coefficient and a value for a second neighboring coefficient and decoding a value for the coefficient currently being decoded based on the determined context offset. As coefficient values among neighbor coefficients in residual blocks tend to be more correlated for transform skip blocks than for transformed blocks, the techniques of this disclosure may result in improved entropy coding, which can improve overall coding efficiency, by for example, reducing the bit overhead needed to represent encoded video data without degrading the quality of the decoded video data.

[0074] This disclosure may generally refer to “signaling” certain information, such as syntax elements. The term “signaling” may generally refer to the communication of values for syntax elements and/or other data used to decode encoded video data. That is, video encoder 200 may signal values for syntax elements in the bitstream. In general, signaling refers to generating a value in the bitstream. As noted above, source device 102 may transport the bitstream to destination device 116 substantially in real time, or not in real time, such as might occur when storing syntax elements to storage device 112 for later retrieval by destination device 116.

[0075] FIGS. 2A and 2B are conceptual diagrams illustrating an example quadtree binary tree (QTBT) structure 130, and a corresponding coding tree unit (CTU) 132. The solid lines represent quadtree splitting, and dotted lines indicate binary tree splitting. In each split (i.e., non-leaf) node of the binary tree, one flag is signaled to indicate which splitting type (i.e., horizontal or vertical) is used, where 0 indicates horizontal splitting and 1 indicates vertical splitting in this example. For the quadtree splitting, there is no need to indicate the splitting type, because quadtree nodes split a block horizontally and vertically into 4 sub-blocks with equal size. Accordingly, video encoder 200 may encode, and video decoder 300 may decode, syntax elements (such as splitting information) for a region tree level (i.e., the first level) of QTBT structure 130 (i.e., the solid lines) and syntax elements (such as splitting information) for a prediction tree level (i.e., the second level) of QTBT structure 130 (i.e., the dashed lines). Video encoder 200 may encode, and video decoder 300 may decode, video data, such as prediction and transform data, for CUs represented by terminal leaf nodes of QTBT structure 130.

[0076] In general, CTU 132 of FIG. 2B may be associated with parameters defining sizes of blocks corresponding to nodes of QTBT structure 130 at the first and second levels. These parameters may include a CTU size (representing a size of CTU 132 in samples), a minimum quadtree size (MinQTSIZE, representing a minimum allowed quadtree leaf node size), a maximum binary tree size (MaxBTSIZE, representing a maximum allowed binary tree root node size), a maximum binary tree depth (MaxBTDepth, representing a maximum allowed binary tree depth), and a minimum binary tree size (MinBTSIZE, representing the minimum allowed binary tree leaf node size).

[0077] The root node of a QTBT structure corresponding to a CTU may have four child nodes at the first level of the QTBT structure, each of which may be partitioned according to quadtree partitioning. That is, nodes of the first level are either leaf nodes (having no child nodes) or have four child nodes. The example of QTBT structure 130 represents such nodes as including the parent node and child nodes having solid lines for branches. If nodes of the first level are not larger than the maximum allowed binary tree root node size (MaxBTSIZE), then the nodes can be further partitioned by respective binary trees. The binary tree splitting of one node can be iterated until the nodes resulting from the split reach the minimum allowed binary tree leaf node size (MinBTSIZE) or the maximum allowed binary tree depth (MaxBTDepth). The example of QTBT structure 130 represents such nodes as having dashed lines for branches. The binary tree leaf node is referred to as a coding unit (CU), which is used for prediction (e.g., intra-picture or inter-picture prediction) and transform, without any further partitioning. As discussed above, CUs may also be referred to as “video blocks” or “blocks.”

[0078] In one example of the QTBT partitioning structure, the CTU size is set as 128x128 (luma samples and two corresponding 64x64 chroma samples), the MinQTSIZE is set as 16x16, the MaxBTSIZE is set as 64x64, the MinBTSIZE (for both width and height) is set as 4, and the MaxBTDepth is set as 4. The quadtree partitioning is applied to the CTU first to generate quad-tree leaf nodes. The quadtree leaf nodes may have a size from 16x16 (i.e., the MinQTSIZE) to 128x128 (i.e., the CTU size). If the quadtree leaf node is 128x128, the quadtree leaf node will not be further split by the binary tree, because the size exceeds the MaxBTSIZE (i.e., 64x64, in this example). Otherwise, the quadtree leaf node will be further partitioned by the binary tree. Therefore, the quadtree leaf node is also the root node for the binary tree and has the

binary tree depth as 0. When the binary tree depth reaches MaxBTDepth (4, in this example), no further splitting is permitted. The binary tree node having a width equal to MinBTSize (4, in this example) implies that no further horizontal splitting is permitted. Similarly, a binary tree node having a height equal to MinBTSize implies no further vertical splitting is permitted for that binary tree node. As noted above, leaf nodes of the binary tree are referred to as CUs and are further processed according to prediction and transform without further partitioning.

[0079] When a block of video data is coded in a transform skip mode, video encoder 200 skips the transform process for residual signals before performing a quantization process. Video decoder 300 likewise skips the inverse transform process step after performing a dequantization process. The characteristics of a not-transformed residual signal are typically quite different than those of transformed signals. For example, coefficients for a transform skipped block tend to be more correlated with their neighboring coefficients when compared to coefficients for a transformed block. As a result, the level values and sign information for neighboring coefficients in a transform skipped block of residual data tend to be more correlated when compared to level values and sign information for a transformed block of residual data.

[0080] B. Bross, T. Nguyen, P. Keydel, H. Schwarz, D. Marpe, T. Wiegand, “Non-CE8: Unified Transform Type Signalling and Residual Coding for Transform Skip,” JVET document JVET-M0464, Marrackech, MA, Jan 2019 sets forth a proposed process for performing residual coding for blocks coded in a transform skip mode. For efficient coding of the levels and the sign information in the transform skip mode, the coefficient coding proposed in JVET-M0464 may be modified to exploit the signal characteristics for more efficient coding.

[0081] FIG. 3 shows an example of three coefficients from a transform skipped block of residual data. The transform skipped block would also include additional coefficients not shown in FIG. 3. In the example of FIG. 3, coefficient value X represents the value of coefficient 140, which represents a coefficient currently being coded. Coefficient value X0 represents the value of coefficient 142, which is the left neighboring coefficient of coefficient 140. Coefficient value X1 represents the value of coefficient 144, which is the top neighboring coefficient of coefficient 140. In this disclosure, the top neighbor may also be referred to as an above neighbor.

[0082] Video encoder 200 and video decoder 300 may be configured to perform sign coding for coefficients in a transform skip block. In the techniques for transform skip

residual coding described in JVET-M0464, the coefficients are coded from first (i.e., top-left) to last (i.e., bottom-right) instead of last to first as is done for transformed blocks. If implementing the techniques of JVET-M0464, video encoder 200 and video decoder 300 may be configured to context code, using CABAC, the sign information using a channel type of residual as a context. That is, video encoder 200 and video decoder 300 may use one context for luma residual and another context for chroma residual. This disclosure describes sign coding techniques that may be used in lieu of or in addition to the techniques described in JVET-M0464. The following techniques utilize the sign information of an above neighboring coefficient (e.g., coefficient 144 in FIG. 3) and a left neighboring coefficient (e.g., coefficient 142 in FIG. 3) to derive the sign coding context offset for a coefficient currently being coded (e.g., coefficient 140 in FIG. 3).

[0083] Referring to FIG. 3, X0 is the left neighboring coefficient value, and X1 is the above neighboring coefficient value. If both neighboring coefficients are zero or both nonzero but with opposite signs, then video encoder 200 and video decoder 300 may use a context offset 0 (ctxOffset = 0). Otherwise, if both are non-negative, then video encoder 200 and video decoder 300 may use context offset 1 (ctxOffset = 1). For all other cases, video encoder 200 and video decoder 300 may use context offset 2 (ctxOffset = 2). This can be described by the following pseudo code:

```

if ((X0 == 0 && X1 == 0) || ((X0 * X1) < 0))
{
    ctxOffset = 0;
}
else if (X0 >= 0 && X1 >= 0)
{
    ctxOffset = 1;
}
else
{
    ctxOffset = 2;
}

```

[0084] In some examples, if both neighboring coefficients are zero or both nonzero but with opposite signs, then video encoder 200 and video decoder 300 may use context offset 0. Otherwise (when both are positive, or both are negative, or one is zero, and the

other one is non zero), video encoder 200 and video decoder 300 may use context 1. If context 1 is used, then the signaled sign value of 0 or 1, depending on the convention, would mean the sign of the coefficient currently being coded is the same as the sign of one of the non-zero neighbors. This technique can also be extended to a coding scenario where only a previous coded nonzero coefficient's value may be used for context derivation, where a sign value of 0 or 1 indicates the sign of the coefficient is the same as the sign of the previous coded nonzero coefficient sign with single context.

[0085] The separate context sets for luma and chroma components can be used in combination with the above described context offset derivation.

[0086] Video encoder 200 and video decoder 300 may also be configured to perform level mapping. In the transform skip residual coding of JVET-M0464, coefficient absolute levels *absCoeffLevel* are coded using *sig_coeff_flag*, *abs_level_gtX_flag*, *par_level_flag*, and *abs_remainder* value to form the final absolute transform coefficient value, where *X* can be 1,...,5 (or some other cutoff value *C*). So, the *absCoeffLevel* value may be constructed by:

$$absCoeffLevel = 1 + abs_level_gt1_flag + par_level_flag + 2 * (abs_level_gt2_flag + abs_level_gt3_flag + \dots + abs_level_gtC_flag) + 2 * abs_remainder$$

[0087] Instead of representing the *absCoeffLevel* directly as in JVET-M0464, video encoder 200 may be configured to map the *absCoeffLevel* to a modified level. Video decoder 300 may be configured to perform an inverse mapping.

[0088] Similar to the sign coding context offset derivation techniques described above, video encoder 200 and video decoder 300 may use the *absCoeffLevel* information of left neighboring and above neighboring coefficients for encoding and decoding absolute coefficient level values. In this example, let *X*₀ denote the absolute coefficient level of a left neighboring coefficient (e.g., coefficient 142 in FIG. 3) to the coefficient currently being coded (e.g., coefficient 140 in FIG. 3), and let *X*₁ denote the absolute coefficient level of an above neighboring coefficient (e.g., coefficient 144 in FIG. 3) to the coefficient currently being coded (e.g., coefficient 140 in FIG. 3). For representing a coefficient with an absolute coefficient level *absCoeff*, a mapped *absCoeffMod* may be coded.

[0089] The operation of video encoder 200 for deriving a value for *absCoeffMod* can be shown with the following pseudocode:

```
pred = max(X0, X1);
if (absCoeff == pred
```

```

{
    absCoeffMod = 1;
}
else
{
    absCoeffMod = (absCoeff < pred) ? absCoeff + 1 : absCoeff;
}

```

[0090] In some examples, if the absolute value of a coefficient (*absCoeff*) to be coded is equal to the maximum neighboring predictor, *pred*, then video encoder 200 sets the modified level *absCoeffMod* to 1. Otherwise, if *absCoeff* is less than the predictor, then video encoder 200 increments the value to be coded by 1. Otherwise, video encoder 200 does not modify the *absCoeff* value.

[0091] Video encoder 200 may, for example, determine a predicted level value for the coefficient currently being encoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient and, in response to a level value of the coefficient currently being encoded being equal to the predicted level value, encode a syntax element with a value equal to one. In other instances, video encoder 200 may determine a predicted level value for the coefficient currently being encoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient and, in response to a level value of the coefficient currently being encoded being less than the predicted level value, encode a syntax element with a value equal to the level value of the coefficient currently being encoded. In other instances, video encoder 200 may determine a predicted level value for the coefficient currently being encoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient and, in response to a level value of the coefficient currently being encoded being greater than the predicted level value, encoding a syntax element with a value equal to the level value of the coefficient currently being encoded minus one.

[0092] The operation of video decoder 300 for deriving a value of *absCoeff* can be shown with the following pseudocode:

```

pred = max(X0, X1);
if (absCoeffMod == 1 && pred > 0)
{
    absCoeff = pred;
}

```

```
    }  
    else  
    {  
        absCoeff = absCoeffMod – (absCoeffMod <= pred);  
    }
```

[0093] Video decoder 300 may, for example, determine a predicted level value for the coefficient currently being decoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, receive a syntax element indicating, and in response to the syntax element having a value equal to one, determine that a level value of the coefficient currently being decoded is equal to the predicted level value. In other instances, video decoder 300 may determine a predicted level value for the coefficient currently being decoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, receive a syntax element indicating, and in response to a value for the syntax element being greater than the predicted level value, determine that a level value of the coefficient currently being decoded is equal to the value for the syntax element plus one. In other instances, video decoder 300 may determine a predicted level value for the coefficient currently being decoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, receive a syntax element indicating, and in response to a value for the syntax element being less than the predicted level value, determine that a level value of the coefficient currently being decoded is equal to the value for the syntax element.

[0094] In some examples, video encoder 200 and video decoder 300 determine or derive the context of the syntax element identified herein as `abs_level_gt1_flag` based on whether the left neighboring and above neighboring coefficient values are zero. The syntax element `abs_level_gt1_flag` is a syntax element used to code coefficient values. The value of `abs_level_gt1_flag` equal to 1 may, for example, mean that the absolute level of a coefficient is greater than 1. The value of `abs_level_gt1_flag` equal to 0 may, for example, mean that the absolute level of a coefficient is not greater than 1.

[0095] In one example, the context may be one of a number, e.g., three, of different contexts. One context may be derived for the case where both a left neighboring coefficient and an above neighboring coefficient have non-zero values. Another context may be derived for the case where only one of a left neighboring coefficient or an above neighboring coefficient having a value that is non-zero. A third context may be derived

for the case where both a left neighboring coefficient and an above neighboring coefficient have values of zero. In one example, this context derivation is applied to non-BDPCM (block differential pulse code modulation) modes only.

[0096] In some examples, for coding scenarios with a non-existing or unavailable neighbor value, such as when a coefficient being coded is on the left boundary of a block and a left neighbor is not present), video encoder 200 and video decoder 300 may be configured to use a zero value when deriving the context.

[0097] In some examples, the context derivation can be described as follows:

```
ctxOffset = 0;
if (Exist(left_neighbor) && non-zero(left_neighbor))
{
    ctxOffset += 1;
}
else if (Exist(above_neighbor) && non-zero(above_neighbor))
{
    ctxOffset += 1;
}
```

In such examples, for a non-existing / unavailable neighbor value (e.g., left neighbor of a value on the left boundary of a block), video encoder 200 and video decoder 300 may be configured to use a zero value for the unavailable value when deriving the context.

[0098] FIG. 4 is a block diagram illustrating an example video encoder 200 that may perform the techniques of this disclosure. FIG. 4 is provided for purposes of explanation and should not be considered limiting of the techniques as broadly exemplified and described in this disclosure. For purposes of explanation, this disclosure describes video encoder 200 in the context of video coding standards such as the HEVC (H.265) video coding standard and the VVC (H.266) video coding standard in development. However, the techniques of this disclosure are not limited to these video coding standards and are applicable generally to video encoding and decoding.

[0099] In the example of FIG. 4, video encoder 200 includes video data memory 230, mode selection unit 202, residual generation unit 204, transform processing unit 206, quantization unit 208, inverse quantization unit 210, inverse transform processing unit 212, reconstruction unit 214, filter unit 216, decoded picture buffer (DPB) 218, and entropy encoding unit 220. Any or all of video data memory 230, mode selection unit 202, residual generation unit 204, transform processing unit 206, quantization unit 208,

inverse quantization unit 210, inverse transform processing unit 212, reconstruction unit 214, filter unit 216, DPB 218, and entropy encoding unit 220 may be implemented in one or more processors or in processing circuitry. Moreover, video encoder 200 may include additional or alternative processors or processing circuitry to perform these and other functions.

[0100] Video data memory 230 may store video data to be encoded by the components of video encoder 200. Video encoder 200 may receive the video data stored in video data memory 230 from, for example, video source 104 (FIG. 1). DPB 218 may act as a reference picture memory that stores reference video data for use in prediction of subsequent video data by video encoder 200. Video data memory 230 and DPB 218 may be formed by any of a variety of memory devices, such as dynamic random access memory (DRAM), including synchronous DRAM (SDRAM), magnetoresistive RAM (MRAM), resistive RAM (RRAM), or other types of memory devices. Video data memory 230 and DPB 218 may be provided by the same memory device or separate memory devices. In various examples, video data memory 230 may be on-chip with other components of video encoder 200, as illustrated, or off-chip relative to those components.

[0101] In this disclosure, reference to video data memory 230 should not be interpreted as being limited to memory internal to video encoder 200, unless specifically described as such, or memory external to video encoder 200, unless specifically described as such. Rather, reference to video data memory 230 should be understood as reference memory that stores video data that video encoder 200 receives for encoding (e.g., video data for a current block that is to be encoded). Memory 106 of FIG. 1 may also provide temporary storage of outputs from the various units of video encoder 200.

[0102] The various units of FIG. 4 are illustrated to assist with understanding the operations performed by video encoder 200. The units may be implemented as fixed-function circuits, programmable circuits, or a combination thereof. Fixed-function circuits refer to circuits that provide particular functionality and are preset on the operations that can be performed. Programmable circuits refer to circuits that can be programmed to perform various tasks and provide flexible functionality in the operations that can be performed. For instance, programmable circuits may execute software or firmware that cause the programmable circuits to operate in the manner defined by instructions of the software or firmware. Fixed-function circuits may execute software instructions (e.g., to receive parameters or output parameters), but the

types of operations that the fixed-function circuits perform are generally immutable. In some examples, one or more of the units may be distinct circuit blocks (fixed-function or programmable), and in some examples, the one or more units may be integrated circuits.

[0103] Video encoder 200 may include arithmetic logic units (ALUs), elementary function units (EFUs), digital circuits, analog circuits, and/or programmable cores, formed from programmable circuits. In examples where the operations of video encoder 200 are performed using software executed by the programmable circuits, memory 106 (FIG. 1) may store the object code of the software that video encoder 200 receives and executes, or another memory within video encoder 200 (not shown) may store such instructions.

[0104] Video data memory 230 is configured to store received video data. Video encoder 200 may retrieve a picture of the video data from video data memory 230 and provide the video data to residual generation unit 204 and mode selection unit 202. Video data in video data memory 230 may be raw video data that is to be encoded.

[0105] Mode selection unit 202 includes a motion estimation unit 222, motion compensation unit 224, and an intra-prediction unit 226. Mode selection unit 202 may include additional functional units to perform video prediction in accordance with other prediction modes. As examples, mode selection unit 202 may include a palette unit, an intra-block copy unit (which may be part of motion estimation unit 222 and/or motion compensation unit 224), an affine unit, a linear model (LM) unit, or the like.

[0106] Mode selection unit 202 generally coordinates multiple encoding passes to test combinations of encoding parameters and resulting rate-distortion values for such combinations. The encoding parameters may include partitioning of CTUs into CUs, prediction modes for the CUs, transform types for residual data of the CUs, quantization parameters for residual data of the CUs, and so on. Mode selection unit 202 may ultimately select the combination of encoding parameters having rate-distortion values that are better than the other tested combinations.

[0107] Video encoder 200 may partition a picture retrieved from video data memory 230 into a series of CTUs and encapsulate one or more CTUs within a slice. Mode selection unit 202 may partition a CTU of the picture in accordance with a tree structure, such as the QTBT structure or the quad-tree structure of HEVC described above. As described above, video encoder 200 may form one or more CUs from

partitioning a CTU according to the tree structure. Such a CU may also be referred to generally as a “video block” or “block.”

[0108] In general, mode selection unit 202 also controls the components thereof (e.g., motion estimation unit 222, motion compensation unit 224, and intra-prediction unit 226) to generate a prediction block for a current block (e.g., a current CU, or in HEVC, the overlapping portion of a PU and a TU). For inter-prediction of a current block, motion estimation unit 222 may perform a motion search to identify one or more closely matching reference blocks in one or more reference pictures (e.g., one or more previously coded pictures stored in DPB 218). In particular, motion estimation unit 222 may calculate a value representative of how similar a potential reference block is to the current block, e.g., according to sum of absolute difference (SAD), sum of squared differences (SSD), mean absolute difference (MAD), mean squared differences (MSD), or the like. Motion estimation unit 222 may generally perform these calculations using sample-by-sample differences between the current block and the reference block being considered. Motion estimation unit 222 may identify a reference block having a lowest value resulting from these calculations, indicating a reference block that most closely matches the current block.

[0109] Motion estimation unit 222 may form one or more motion vectors (MVs) that defines the positions of the reference blocks in the reference pictures relative to the position of the current block in a current picture. Motion estimation unit 222 may then provide the motion vectors to motion compensation unit 224. For example, for uni-directional inter-prediction, motion estimation unit 222 may provide a single motion vector, whereas for bi-directional inter-prediction, motion estimation unit 222 may provide two motion vectors. Motion compensation unit 224 may then generate a prediction block using the motion vectors. For example, motion compensation unit 224 may retrieve data of the reference block using the motion vector. As another example, if the motion vector has fractional sample precision, motion compensation unit 224 may interpolate values for the prediction block according to one or more interpolation filters. Moreover, for bi-directional inter-prediction, motion compensation unit 224 may retrieve data for two reference blocks identified by respective motion vectors and combine the retrieved data, e.g., through sample-by-sample averaging or weighted averaging.

[0110] As another example, for intra-prediction, or intra-prediction coding, intra-prediction unit 226 may generate the prediction block from samples neighboring the

current block. For example, for directional modes, intra-prediction unit 226 may generally mathematically combine values of neighboring samples and populate these calculated values in the defined direction across the current block to produce the prediction block. As another example, for DC mode, intra-prediction unit 226 may calculate an average of the neighboring samples to the current block and generate the prediction block to include this resulting average for each sample of the prediction block.

[0111] Mode selection unit 202 provides the prediction block to residual generation unit 204. Residual generation unit 204 receives a raw, uncoded version of the current block from video data memory 230 and the prediction block from mode selection unit 202. Residual generation unit 204 calculates sample-by-sample differences between the current block and the prediction block. The resulting sample-by-sample differences define a residual block for the current block. In some examples, residual generation unit 204 may also determine differences between sample values in the residual block to generate a residual block using residual differential pulse code modulation (RDPCM). In some examples, residual generation unit 204 may be formed using one or more subtractor circuits that perform binary subtraction.

[0112] In examples where mode selection unit 202 partitions CUs into PUs, each PU may be associated with a luma prediction unit and corresponding chroma prediction units. Video encoder 200 and video decoder 300 may support PUs having various sizes. As indicated above, the size of a CU may refer to the size of the luma coding block of the CU and the size of a PU may refer to the size of a luma prediction unit of the PU. Assuming that the size of a particular CU is $2N \times 2N$, video encoder 200 may support PU sizes of $2N \times 2N$ or $N \times N$ for intra prediction, and symmetric PU sizes of $2N \times 2N$, $2N \times N$, $N \times 2N$, $N \times N$, or similar for inter prediction. Video encoder 200 and video decoder 300 may also support asymmetric partitioning for PU sizes of $2N \times nU$, $2N \times nD$, $nL \times 2N$, and $nR \times 2N$ for inter prediction.

[0113] In examples where mode selection unit does not further partition a CU into PUs, each CU may be associated with a luma coding block and corresponding chroma coding blocks. As above, the size of a CU may refer to the size of the luma coding block of the CU. The video encoder 200 and video decoder 300 may support CU sizes of $2N \times 2N$, $2N \times N$, or $N \times 2N$.

[0114] For other video coding techniques such as intra-block copy mode coding, affine-mode coding, and linear model (LM) mode coding, as a few examples, mode selection

unit 202, via respective units associated with the coding techniques, generates a prediction block for the current block being encoded. In some examples, such as palette mode coding, mode selection unit 202 may not generate a prediction block, and instead generate syntax elements that indicate the manner in which to reconstruct the block based on a selected palette. In such modes, mode selection unit 202 may provide these syntax elements to entropy encoding unit 220 to be encoded.

[0115] As described above, residual generation unit 204 receives the video data for the current block and the corresponding prediction block. Residual generation unit 204 then generates a residual block for the current block. To generate the residual block, residual generation unit 204 calculates sample-by-sample differences between the prediction block and the current block.

[0116] Transform processing unit 206 applies one or more transforms to the residual block to generate a block of transform coefficients (referred to herein as a “transform coefficient block”). Transform processing unit 206 may apply various transforms to a residual block to form the transform coefficient block. For example, transform processing unit 206 may apply a discrete cosine transform (DCT), a directional transform, a Karhunen-Loeve transform (KLT), or a conceptually similar transform to a residual block. In some examples, transform processing unit 206 may perform multiple transforms to a residual block, e.g., a primary transform and a secondary transform, such as a rotational transform. In some examples, transform processing unit 206 does not apply transforms to a residual block. In such instances, transform processing unit 206 may output a block of coefficients, where the coefficients correspond to residual values instead of transform coefficients.

[0117] Quantization unit 208 may quantize transform coefficients in a transform coefficient block, to produce a quantized transform coefficient block. For a block coded in transform skip mode, quantization unit 208 may quantize coefficients in a coefficient block to produce a quantized coefficient block. Quantization unit 208 may quantize coefficients or transform coefficients according to a quantization parameter (QP) value associated with the current block. Video encoder 200 (e.g., via mode selection unit 202) may adjust the degree of quantization applied by adjusting the QP value associated with the CU. Quantization may introduce loss of information, and thus, quantized coefficients or transform coefficients may have lower precision than the original coefficients or transform coefficients produced by transform processing unit 206.

[0118] Inverse quantization unit 210 and inverse transform processing unit 212 may apply inverse quantization and inverse transforms to a quantized coefficient block, respectively, to reconstruct a residual block from the coefficient block. Reconstruction unit 214 may produce a reconstructed block corresponding to the current block (albeit potentially with some degree of distortion) based on the reconstructed residual block and a prediction block generated by mode selection unit 202. For example, reconstruction unit 214 may add samples of the reconstructed residual block to corresponding samples from the prediction block generated by mode selection unit 202 to produce the reconstructed block.

[0119] Filter unit 216 may perform one or more filter operations on reconstructed blocks. For example, filter unit 216 may perform deblocking operations to reduce blockiness artifacts along edges of CUs. Operations of filter unit 216 may be skipped, in some examples.

[0120] Video encoder 200 stores reconstructed blocks in DPB 218. For instance, in examples where operations of filter unit 216 are not performed, reconstruction unit 214 may store reconstructed blocks to DPB 218. In examples where operations of filter unit 216 are performed, filter unit 216 may store the filtered reconstructed blocks to DPB 218. Motion estimation unit 222 and motion compensation unit 224 may retrieve a reference picture from DPB 218, formed from the reconstructed (and potentially filtered) blocks, to inter-predict blocks of subsequently encoded pictures. In addition, intra-prediction unit 226 may use reconstructed blocks in DPB 218 of a current picture to intra-predict other blocks in the current picture.

[0121] In general, entropy encoding unit 220 may entropy encode syntax elements received from other functional components of video encoder 200. For example, entropy encoding unit 220 may entropy encode quantized coefficient blocks from quantization unit 208. As another example, entropy encoding unit 220 may entropy encode prediction syntax elements (e.g., motion information for inter-prediction or intra-mode information for intra-prediction) from mode selection unit 202. Entropy encoding unit 220 may perform one or more entropy encoding operations on the syntax elements, which are another example of video data, to generate entropy-encoded data. For example, entropy encoding unit 220 may perform a context-adaptive variable length coding (CAVLC) operation, a CABAC operation, a variable-to-variable (V2V) length coding operation, a syntax-based context-adaptive binary arithmetic coding (SBAC) operation, a Probability Interval Partitioning Entropy (PIPE) coding operation, an

Exponential-Golomb encoding operation, or another type of entropy encoding operation on the data. In some examples, entropy encoding unit 220 may operate in bypass mode where syntax elements are not entropy encoded.

[0122] Video encoder 200 may output a bitstream that includes the entropy encoded syntax elements needed to reconstruct blocks of a slice or picture. In particular, entropy encoding unit 220 may output the bitstream.

[0123] The operations described above are described with respect to a block. Such description should be understood as being operations for a luma coding block and/or chroma coding blocks. As described above, in some examples, the luma coding block and chroma coding blocks are luma and chroma components of a CU. In some examples, the luma coding block and the chroma coding blocks are luma and chroma components of a PU.

[0124] In some examples, operations performed with respect to a luma coding block need not be repeated for the chroma coding blocks. As one example, operations to identify a motion vector (MV) and reference picture for a luma coding block need not be repeated for identifying a MV and reference picture for the chroma blocks. Rather, the MV for the luma coding block may be scaled to determine the MV for the chroma blocks, and the reference picture may be the same. As another example, the intra-prediction process may be the same for the luma coding blocks and the chroma coding blocks.

[0125] Video encoder 200 represents an example of a device for encoding video data that includes a memory configured to store video data and one or more processing units implemented in circuitry and configured to determine, for a residual block of video data encoded using a transform skip mode, a value for a first neighboring coefficient of a coefficient currently being encoded; determine a value for a second neighboring coefficient of the coefficient currently being encoded; determine a context offset for the coefficient currently being encoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient; and encode a value for the coefficient currently being encoded based on the determined context offset. The first neighboring coefficient may, for example, be one of a top neighboring coefficient or a left neighboring coefficient, and the second neighboring coefficient may be the other of the top neighboring coefficient or the left neighboring coefficient.

[0126] To encode the value for the coefficient currently being encoded based on the determined context offset, video encoder 200 may be configured to determine a context

based on the determined context offset, determine a sign for the coefficient currently being encoded, and context encode one or more bins of data based on the determined context to represent the sign for the coefficient currently being encoded. Video encoder 200 may be configured to quantize a residual value for the residual block of video data to determine the value for the coefficient currently being encoded. Video encoder 200 may be configured to determine a prediction block and compare the prediction block to an original block of video data to determine the residual block of video data.

[0127] To determine the context offset for the coefficient currently being encoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, video encoder 200 may be configured to select the context offset from three available context offsets based on the value for the first neighboring coefficient and the value for the second neighboring coefficient. The three available context offsets may include a first context offset for when the first neighboring coefficient and the second neighboring coefficient are both equal to zero or have opposite signs; a second context offset for when the first neighboring coefficient and the second neighboring coefficient are both positive or one of the first neighboring coefficient and the second neighboring coefficient is equal to zero and the other one of the first neighboring coefficient and the second neighboring coefficient is positive; and a third context offset for when the first neighboring coefficient and the second neighboring coefficient are both negative or one of the first neighboring coefficient and the second neighboring coefficient is equal to zero and the other one of the first neighboring coefficient and the second neighboring coefficient is negative.

[0128] To determine the context offset for the coefficient currently being encoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, video encoder 200 may be configured to set the context offset value to a first offset value in response to the value for the first neighboring coefficient being equal to zero and the value for the second neighboring coefficient being equal to zero. To determine the context offset for the coefficient currently being encoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, video encoder 200 may be configured to set the context offset value to a first offset value in response to the value for the first neighboring coefficient being one of greater than zero or less than zero and the value for the second neighboring coefficient being the other of greater than zero or less than zero.

[0129] To determine the context offset for the coefficient currently being encoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, video encoder 200 may be configured to set the context offset value to a first offset value in response to the value for the first neighboring coefficient and the value for the second neighboring coefficient both being non-zero values and having opposite signs. To determine the context offset for the coefficient currently being encoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, video encoder 200 may be configured to set the context offset value to a second offset value in response to the value for the first neighboring coefficient being greater than or equal to zero and the value for the second neighboring coefficient being greater than or equal to zero, wherein at least one of the value for the first neighboring coefficient or the value for the second neighboring coefficient is greater than or equal to one.

[0130] To determine the context offset for the coefficient currently being encoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, video encoder 200 may be configured to set the context offset value to a second offset value in response to the value for the first neighboring coefficient and the value for the second neighboring coefficient both being non-negative values. To determine the context offset for the coefficient currently being encoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, video encoder 200 may be configured to set the context offset value to a third offset value in response to the value for the first neighboring coefficient being less than or equal to zero and the value for the second neighboring coefficient being less than or equal to zero, wherein the value for the first neighboring coefficient or the value for the second neighboring coefficient is less than or equal to negative one (-1). The first, second, and third offset values may, for example, be different offset values. A context offset is a value for determining a context. Thus, the first, second, and third offset values may be considered to identify or refer to three different contexts.

[0131] Video encoder 200 also represents an example of a device configured to encode video data including a memory configured to store video data, and one or more processing units implemented in circuitry and configured to determine, for a residual block of video data encoded using a transform skip mode, an absolute coefficient level for a first neighboring coefficient of a coefficient currently being encoded; determine an absolute coefficient level for a second neighboring coefficient of the coefficient

currently being encoded; determine an absolute coefficient level for the coefficient currently being encoded; and based on the absolute coefficient level for the first neighboring coefficient and the absolute coefficient level for the second neighboring coefficient, encode one or more syntax elements indicating the absolute coefficient level for the coefficient currently being encoded. Video encoder 200 may, for example, quantize a residual value for the residual block of video data to determine the value for the coefficient currently being encoded.

[0132] In some examples, video encoder 200 may be configured to determine a predictor level based on the absolute coefficient level for the first neighboring coefficient and the absolute coefficient level for the second neighboring coefficient and encode a syntax element, with a first value for the syntax element indicating the predictor level is equal to the absolute coefficient level for the coefficient currently being encoded and a second value for the syntax element indicating the predictor level is not equal to the absolute coefficient level for the coefficient currently being encoded. In some examples, video encoder 200 may, for example, be configured to determine a predictor level based on the absolute coefficient level for the first neighboring coefficient and the absolute coefficient level for the second neighboring coefficient, determine a value for a syntax element based on the predictor level and the absolute coefficient level for the coefficient currently being encoded, and encode the syntax element. To determine the predictor level, video encoder 200 may be configured to set the predictor level equal to the greater of the absolute coefficient level for the first neighboring coefficient or the absolute coefficient level for the second neighboring coefficient.

[0133] FIG. 5 is a block diagram illustrating an example video decoder 300 that may perform the techniques of this disclosure. FIG. 5 is provided for purposes of explanation and is not limiting on the techniques as broadly exemplified and described in this disclosure. For purposes of explanation, this disclosure describes video decoder 300 according to the techniques of JEM, VVC, and HEVC. However, the techniques of this disclosure may be performed by video coding devices that are configured to other video coding standards.

[0134] In the example of FIG. 5, video decoder 300 includes coded picture buffer (CPB) memory 320, entropy decoding unit 302, prediction processing unit 304, inverse quantization unit 306, inverse transform processing unit 308, reconstruction unit 310, filter unit 312, and decoded picture buffer (DPB) 314. Any or all of CPB memory 320,

entropy decoding unit 302, prediction processing unit 304, inverse quantization unit 306, inverse transform processing unit 308, reconstruction unit 310, filter unit 312, and DPB 314 may be implemented in one or more processors or in processing circuitry. Moreover, video decoder 300 may include additional or alternative processors or processing circuitry to perform these and other functions.

[0135] Prediction processing unit 304 includes motion compensation unit 316 and intra-prediction unit 318. Prediction processing unit 304 may include additional units to perform prediction in accordance with other prediction modes. As examples, prediction processing unit 304 may include a palette unit, an intra-block copy unit (which may form part of motion compensation unit 316), an affine unit, a linear model (LM) unit, or the like. In other examples, video decoder 300 may include more, fewer, or different functional components.

[0136] CPB memory 320 may store video data, such as an encoded video bitstream, to be decoded by the components of video decoder 300. The video data stored in CPB memory 320 may be obtained, for example, from computer-readable medium 110 (FIG. 1). CPB memory 320 may include a CPB that stores encoded video data (e.g., syntax elements) from an encoded video bitstream. Also, CPB memory 320 may store video data other than syntax elements of a coded picture, such as temporary data representing outputs from the various units of video decoder 300. DPB 314 generally stores decoded pictures, which video decoder 300 may output and/or use as reference video data when decoding subsequent data or pictures of the encoded video bitstream. CPB memory 320 and DPB 314 may be formed by any of a variety of memory devices, such as DRAM, SDRAM, MRAM, RRAM, or other types of memory devices. CPB memory 320 and DPB 314 may be provided by the same memory device or separate memory devices. In various examples, CPB memory 320 may be on-chip with other components of video decoder 300, or off-chip relative to those components.

[0137] Additionally or alternatively, in some examples, video decoder 300 may retrieve coded video data from memory 120 (FIG. 1). That is, memory 120 may store data as discussed above with CPB memory 320. Likewise, memory 120 may store instructions to be executed by video decoder 300, when some or all of the functionality of video decoder 300 is implemented in software to be executed by processing circuitry of video decoder 300.

[0138] The various units shown in FIG. 5 are illustrated to assist with understanding the operations performed by video decoder 300. The units may be implemented as fixed-

function circuits, programmable circuits, or a combination thereof. Similar to FIG. 4, fixed-function circuits refer to circuits that provide particular functionality and are preset on the operations that can be performed. Programmable circuits refer to circuits that can be programmed to perform various tasks and provide flexible functionality in the operations that can be performed. For instance, programmable circuits may execute software or firmware that cause the programmable circuits to operate in the manner defined by instructions of the software or firmware. Fixed-function circuits may execute software instructions (e.g., to receive parameters or output parameters), but the types of operations that the fixed-function circuits perform are generally immutable. In some examples, one or more of the units may be distinct circuit blocks (fixed-function or programmable), and in some examples, the one or more units may be integrated circuits.

[0139] Video decoder 300 may include ALUs, EFUs, digital circuits, analog circuits, and/or programmable cores formed from programmable circuits. In examples where the operations of video decoder 300 are performed by software executing on the programmable circuits, on-chip or off-chip memory may store instructions (e.g., object code) of the software that video decoder 300 receives and executes.

[0140] Entropy decoding unit 302 may receive encoded video data from the CPB and entropy decode the video data to reproduce syntax elements. Prediction processing unit 304, inverse quantization unit 306, inverse transform processing unit 308, reconstruction unit 310, and filter unit 312 may generate decoded video data based on the syntax elements extracted from the bitstream.

[0141] In general, video decoder 300 reconstructs a picture on a block-by-block basis. Video decoder 300 may perform a reconstruction operation on each block individually (where the block currently being reconstructed, i.e., decoded, may be referred to as a “current block”).

[0142] Entropy decoding unit 302 may entropy decode syntax elements defining quantized coefficients of a quantized coefficient block, as well as transform information, such as a quantization parameter (QP) and/or transform mode indication(s). Inverse quantization unit 306 may use the QP associated with the quantized coefficient block to determine a degree of quantization and, likewise, a degree of inverse quantization for inverse quantization unit 306 to apply. Inverse quantization unit 306 may, for example, perform a bitwise left-shift operation to inverse quantize the transform coefficients.

Inverse quantization unit 306 may thereby form a coefficient block including coefficients.

[0143] After inverse quantization unit 306 forms the coefficient block for a block that is transformed, inverse transform processing unit 308 may apply one or more inverse transforms to the transform coefficient block to generate a residual block associated with the current block. For example, inverse transform processing unit 308 may apply an inverse DCT, an inverse integer transform, an inverse Karhunen-Loeve transform (KLT), an inverse rotational transform, an inverse directional transform, or another inverse transform to the transform coefficient block. For blocks that are coded in a transform skip mode, inverse transform processing unit 308 may not perform an inverse transform, and in these coding scenarios, may be viewed as a pass-through unit that does not process or alter the block of coefficients.

[0144] Furthermore, prediction processing unit 304 generates a prediction block according to prediction information syntax elements that were entropy decoded by entropy decoding unit 302. For example, if the prediction information syntax elements indicate that the current block is inter-predicted, motion compensation unit 316 may generate the prediction block. In this case, the prediction information syntax elements may indicate a reference picture in DPB 314 from which to retrieve a reference block, as well as a motion vector identifying a location of the reference block in the reference picture relative to the location of the current block in the current picture. Motion compensation unit 316 may generally perform the inter-prediction process in a manner that is substantially similar to that described with respect to motion compensation unit 224 (FIG. 4).

[0145] As another example, if the prediction information syntax elements indicate that the current block is intra-predicted, intra-prediction unit 318 may generate the prediction block according to an intra-prediction mode indicated by the prediction information syntax elements. Again, intra-prediction unit 318 may generally perform the intra-prediction process in a manner that is substantially similar to that described with respect to intra-prediction unit 226 (FIG. 4). Intra-prediction unit 318 may retrieve data of neighboring samples to the current block from DPB 314.

[0146] Reconstruction unit 310 may reconstruct the current block using the prediction block and the residual block. For example, reconstruction unit 310 may add samples of the residual block to corresponding samples of the prediction block to reconstruct the current block.

[0147] Filter unit 312 may perform one or more filter operations on reconstructed blocks. For example, filter unit 312 may perform deblocking operations to reduce blockiness artifacts along edges of the reconstructed blocks. Operations of filter unit 312 are not necessarily performed in all examples.

[0148] Video decoder 300 may store the reconstructed blocks in DPB 314. For instance, in examples where operations of filter unit 312 are not performed, reconstruction unit 310 may store reconstructed blocks to DPB 314. In examples where operations of filter unit 312 are performed, filter unit 312 may store the filtered reconstructed blocks to DPB 314. As discussed above, DPB 314 may provide reference information, such as samples of a current picture for intra-prediction and previously decoded pictures for subsequent motion compensation, to prediction processing unit 304. Moreover, video decoder 300 may output decoded pictures from DPB for subsequent presentation on a display device, such as display device 118 of FIG. 1.

[0149] Video decoder 300 represents an example of a video decoding device including a memory configured to store video data and one or more processing units implemented in circuitry and configured to determine, for a residual block of video data encoded using a transform skip mode, a value for a first neighboring coefficient of a coefficient currently being decoded; determine a value for a second neighboring coefficient of the coefficient currently being decoded; determine a context offset for the coefficient currently being decoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient; and decode a value for the coefficient currently being decoded based on the determined context offset. The first neighboring coefficient may, for example, be one of a top neighboring coefficient or a left neighboring coefficient, and the second neighboring coefficient may be the other of the top neighboring coefficient or the left neighboring coefficient. As described above, for residual blocks coded in a transform skip mode, the coefficient value may correspond to a residual value, such as a quantized residual value or a non-quantized residual value.

[0150] To decode the value for the coefficient currently being decoded based on the determined context offset, video decoder 300 may be configured to determine a context based on the determined context offset; receive one or more bins of data; and context decode the one or more bins of data based on the determined context to determine a sign for the coefficient currently being decoded. Video decoder 300 may be configured to inverse quantize the value for the coefficient currently being decoded to determine a residual value for the residual block of video data. Video decoder 300 may be

configured to determine a decoded residual block based on the value for the coefficient currently being decoded; add the decoded residual block to a prediction block to determine a reconstructed block; perform one or more filtering operations on the reconstructed block to determine a decoded block of video data; and output a decoded picture of video data that includes the decoded block of video data.

[0151] To determine the context offset for the coefficient currently being decoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, video decoder 300 may be configured to select the context offset from three available context offsets based on the value for the first neighboring coefficient and the value for the second neighboring coefficient. Three available context offsets may include a first context offset for when the first neighboring coefficient and the second neighboring coefficient are both equal to zero or have opposite signs, a second context offset for when the first neighboring coefficient and the second neighboring coefficient are both positive or one of the first neighboring coefficient and the second neighboring coefficient is equal to zero and the other one of the first neighboring coefficient and the second neighboring coefficient is positive, and a third context offset for when the first neighboring coefficient and the second neighboring coefficient are both negative or one of the first neighboring coefficient and the second neighboring coefficient is equal to zero and the other one of the first neighboring coefficient and the second neighboring coefficient is negative.

[0152] To determine the context offset for the coefficient currently being decoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, video decoder 300 may be configured to set the context offset value to a first offset value in response to the value for the first neighboring coefficient being equal to zero and the value for the second neighboring coefficient being equal to zero. To determine the context offset for the coefficient currently being decoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, video decoder 300 may be configured to set the context offset value to a first offset value in response to the value for the first neighboring coefficient being one of greater than zero or less than zero and the value for the second neighboring coefficient being the other of greater than zero or less than zero. To determine the context offset for the coefficient currently being decoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, video decoder 300 may be configured to set the context offset value to a first offset value in

response to the value for the first neighboring coefficient and the value for the second neighboring coefficient both being non-zero values and having opposite signs.

[0153] To determine the context offset for the coefficient currently being decoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, video decoder 300 may be configured to set the context offset value to a second offset value in response to the value for the first neighboring coefficient being greater than or equal to zero and the value for the second neighboring coefficient being greater than or equal to zero, where at least one of the value for the first neighboring coefficient or the value for the second neighboring coefficient is greater than or equal to one. To determine the context offset for the coefficient currently being decoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, video decoder 300 may be configured to set the context offset value to a second offset value in response to the value for the first neighboring coefficient and the value for the second neighboring coefficient both being non-negative values. To determine the context offset for the coefficient currently being decoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, video decoder 300 may be configured to set the context offset value to a third offset value in response to the value for the first neighboring coefficient being less than or equal to zero and the value for the second neighboring coefficient being less than or equal to zero, wherein the value for the first neighboring coefficient or the value for the second neighboring coefficient is less than or equal to negative one (-1).

[0154] Video decoder 300 also represents an example of a video decoding device including a memory configured to store video data, and one or more processing units implemented in circuitry and configured to determine, for a residual block of video data encoded using a transform skip mode, an absolute coefficient level for a first neighboring coefficient of a coefficient currently being decoded; determine an absolute coefficient level for a second neighboring coefficient of the coefficient currently being decoded; and determine an absolute coefficient level for the coefficient currently being decoded based on the absolute coefficient level for the first neighboring coefficient and the absolute coefficient level for the second neighboring coefficient. Video decoder 300 may, for example, inverse quantize the coefficient currently being decoded to determine a residual value for the residual block of video data.

[0155] To determine the absolute coefficient level for the coefficient currently being decoded, video decoder 300 may be configured to determine a predictor level based on the absolute coefficient level for the first neighboring coefficient and the absolute coefficient level for the second neighboring coefficient and receive a syntax element, with a first value for the syntax element indicating the predictor level is equal to the absolute coefficient level for the coefficient currently being decoded and a second value for the syntax element indicating the predictor level is not equal to the absolute coefficient level for the coefficient currently being decoded. To determine the absolute coefficient level for the coefficient currently being decoded, video decoder 300 may be configured to determine a predictor level based on the absolute coefficient level for the first neighboring coefficient and the absolute coefficient level for the second neighboring coefficient, receive a syntax element, and determine the absolute coefficient level for the coefficient currently being decoded based on the predictor level and the syntax element. To determine the predictor level, video decoder 300 may set the predictor level equal to the greater of the absolute coefficient level for the first neighboring coefficient or the absolute coefficient level for the second neighboring coefficient.

[0156] FIGS. 6A and 6B show examples of a CABAC process at a bin n . In example 400 of FIG. 6A, the range at bin n includes the RangeMPS and RangeLPS given by the probability of the least probable symbol (LPS) (p_σ) given a certain context state (σ). Example 400 shows the update of the range at bin $n+1$ when the value of bin n is equal to the most probable symbol (MPS). In this example, the low value of the range stays the same, but the value of the range at bin $n+1$ is reduced to the value of RangeMPS at bin n . Example 402 of FIG. 6B shows the update of the range at bin $n+1$ when the value of bin n is not equal to the MPS (i.e., equal to the LPS). In this example, the low value of the range is moved to the lower range value of RangeLPS at bin n . In addition, the value of the range at bin $n+1$ is reduced to the value of RangeLPS at bin n .

[0157] In one example of the HEVC video coding process, the range is expressed with 9 bits and the low value with 10 bits. There is a renormalization process to maintain the range and low values at sufficient precision. The renormalization occurs whenever the range is less than 256. Therefore, the range is always equal to or larger than 256 after renormalization. Depending on the values of range and the low value, the binary arithmetic coder (BAC) outputs to the bitstream, a '0,' or a '1,' or updates an internal variable (called BO: bits-outstanding) to keep for future outputs. FIG. 7 shows

examples of BAC output depending on the range. For example, a '1' is output to the bitstream when the range and low value are above a certain threshold (e.g., 512). A '0' is output to the bitstream when the range and low value are below a certain threshold (e.g., 512). Nothing is output to the bitstream when the range and low are between certain thresholds. Instead, the BO value is incremented, and the next bin is encoded.

[0158] In the CABAC context model of H.264/AVC and in some examples of HEVC, there are 128 states. There are 64 possible LPS probabilities (denoted by state σ) that can be from 0 to 63. Each MPS can be zero or one. As such, the 128 states are 64 state probabilities times the 2 possible values for MPS (0 or 1). Therefore, the state can be indexed with 7 bits.

[0159] To reduce the computation of deriving LPS ranges ($rangeLPS_{\sigma}$), results for all cases may be pre-calculated and stored as approximations in a look-up table. Therefore, the LPS range can be obtained without any multiplication by using a simple table lookup. Avoiding multiplication can be important for some devices or applications, since this operation may cause significant latency in many hardware architectures.

[0160] A 4-column pre-calculated LPS range table may be used instead of the multiplication. The range is divided into four segments. The segment index can be derived by the question $(range \gg 6) \& 3$. In effect, the segment index is derived by shifting and dropping bits from the actual range. The following Table 1 shows the possible ranges and their corresponding indexes.

TABLE 1 – Range Index

Range	256-319	320-383	384-447	448-511
$(range \gg 6) \& 3$	0	1	2	3

[0161] The LPS range table has then 64 entries (one for each probability state) times 4 (one for each range index). Each entry is the Range LPS, that is, the value of multiplying the range times the LPS probability. An example of part of this table is shown in the following Table 2. Table 2 depicts probability states 9-12. In one proposal for HEVC, the probability states may range from 0-63.

TABLE 2 – RangeLPS

Prob State (σ)	RangeLPS			
	Index 0	Index 1	Index 2	Index 3
...	...	...	...	...
9	90	110	130	150
10	85	104	123	142
11	81	99	117	135
12	77	94	111	128
...	...	...	...	...

[0162] In each segment (i.e., range value), the LPS range of each probability state σ is pre-defined. In other words, the LPS range of a probability state σ is quantized into four values (i.e., one value for each range index). The specific LPS range used at a given point depends on the range to which the segment belongs. The number of possible LPS ranges used in the table is a trade-off between the number of table columns (i.e., the number of possible LPS range values) and the LPS range precision. Generally speaking, more columns results in smaller quantization errors of LPS range values, but also increases the need for more memory to store the table. Fewer columns increases quantization errors, but also reduces the memory needed to store the table.

[0163] As described above, each LPS probability state has a corresponding probability. The probability p for each state is derived as follows:

$$p_{\sigma} = \alpha p_{\sigma-1}$$

where the state σ is from 0 to 63. The constant α represents the amount of probability change between each context state. In one example, $\alpha=0.9493$, or, more precisely, $\alpha=(0.01875/0.5)^{1/63}$. The probability at state $\sigma = 0$ is equal to 0.5 (i.e., $p_0=1/2$). That is, at context state 0, the LPS and MPS are equally probable. The probability at each successive state is derived by multiplying the previous state by α . As such, the probability of the LPS occurring at context state $\alpha = 1$ is $p_0 * 0.9493$ ($0.5 * 0.9493 = .47465$). As such, as the index of state α increases, the probability of the LPS occurring goes down.

[0164] CABAC is adaptive because the probability states are updated in order to follow the signal statistics (i.e., the values of previously coded bins). The update process is as follows. For a given probability state, the update depends on the state index and the

value of the encoded symbol identified either as an LPS or an MPS. As a result of the updating process, a new probability state is derived, which consists of a potentially modified LPS probability estimate and, if necessary, a modified MPS value.

[0165] In the event of a bin value equaling the MPS, a given state index may be incremented by 1. This is for all states except when an MPS occurs at state index 62, where the LPS probability is already at its minimum (or equivalently, the maximum MPS probability is reached). In this case, the state index 62 remains fixed until an LPS is seen, or the last bin value is encoded (state 63 is used for the special case of the last bin value). When an LPS occurs, the state index is changed by decrementing the state index by a certain amount, as shown in the equation below. This rule applies in general to each occurrence of an LPS with the following exception. Assuming an LPS has been encoded at the state with index $\sigma=0$, which corresponds to the equi-probable case, the state index remains fixed, but the MPS value will be toggled such that the value of the LPS and MPS will be interchanged. In all other cases, no matter which symbol has been encoded, the MPS value will not be altered. The derivation of the transition rules for the LPS probability is based on the following relation between a given LPS probability p_{old} and its updated counterpart p_{new} :

$$p_{new} = \max(\alpha p_{old}, p_{62}) \quad \text{if an MPS occurs}$$

$$p_{new} = (1 - \alpha) + \alpha p_{old} \quad \text{if an LPS occurs}$$

[0166] With regard to a practical implementation of the probability estimation process in CABAC, it is important to note that all transition rules may be realized by at most two tables each having 63 entries of 6-bit unsigned integer values. In some examples, state transitions may be determined with a single table $TransIdxLPS$, which determines, for a given state index σ , the new updated state index $TransIdxLPS[\sigma]$ in case an LPS has been observed. The MPS-driven transitions can be obtained by a simple (saturated) increment of the state index by the fixed value of 1, resulting in an updated state index $\min(\sigma+1, 62)$. Table 3 below is an example of a partial $TransIdxLPS$ table.

TABLE 3 - TransIdxLPS

Prob State (σ)	New State TransIdxLPS [σ]
...	...
9	6
10	8
11	8
12	8
...	...

[0167] The techniques described above with respect to FIGS. 6A, 6B, and 7 merely represent one example implementation of CABAC. It should be understood that the techniques of this disclosure are not limited only to this described implementation of CABAC. For example, in older BAC approaches (e.g., the BAC approach used in H.264/AVC), the tables RangeLPS and TransIdxLPS were tuned for low resolution videos, (i.e., common intermediate format (CIF) and quarter-CIF (QCIF) videos). With HEVC and future codecs such as VVC, a large amount of video content is high definition (HD) and, in some cases, greater than HD. Video content that is HD or greater than HD resolution tends to have different statistics than the 10-year-old QCIF sequences used to develop H.264/AVC. As such, the tables RangeLPS and TransIdxLPS from H.264/AVC may cause adaptation between states in a manner that is too quick. That is, the transitions between probability states, especially when an LPS occurs, can be too great for the smoother, higher resolution content of HD video. Thus, the probability models used according to conventional techniques may not be as accurate for HD and extra-HD content. In addition, as HD video content includes a greater range of pixel values, the H.264/AVC tables do not include enough entries to account for the more extreme values that may be present in HD content.

[0168] As such, for HEVC and for future coding standards such as VVC, the RangeLPS and TransIdxLPS tables may be modified to account for the characteristics of this new content. In particular, the BAC processes for HEVC and future coding standards may use tables that allow for a slower adaptation process and may account for more extreme cases (i.e., skewed probabilities). Thus, as one example, the RangeLPS and TransIdxLPS tables may be modified to achieve these goals by including more probability states and ranges than used in BAC with H.264/AVC or HEVC.

[0169] FIG. 8 is a block diagram of an example entropy encoding unit 220, e.g., forming part of video encoder 200 as shown in FIG. 4, that may be configured to perform CABAC in accordance with the techniques of this disclosure. A syntax element 418 is input into the entropy encoding unit 220. If the syntax element is already a binary-value syntax element (i.e., a syntax element that only has a value of 0 and 1), the step of binarization may be skipped. If the syntax element is a non-binary valued syntax element (e.g., a syntax element represented by multiple bits, such as coefficient levels), the non-binary valued syntax element is binarized by binarizer 420. Binarizer 420 performs a mapping of the non-binary valued syntax element into a sequence of binary decisions. These binary decisions are often called “bins.” For example, for coefficient levels, the value of the level may be broken down into successive bins, each bin indicating whether or not the absolute value of coefficient level is greater than some value. For example, bin 0 (sometimes called a significance flag) indicates if the absolute value of the coefficient level is greater than 0 or not. Bin 1 indicates if the absolute value of the coefficient level is greater than 1 or not, and so on. A unique mapping may be developed for each non-binary valued syntax element.

[0170] Each bin produced by binarizer 420 is fed to the binary arithmetic coding side of entropy encoding unit 220. That is, for a predetermined set of non-binary valued syntax elements, each bin type (e.g., bin 0) is coded before the next bin type (e.g., bin 1). Coding may be performed in either regular mode or bypass mode. In bypass mode, bypass encoding engine 426 performs arithmetic coding using a fixed probability model, for example, using Golomb-Rice or exponential Golomb coding. Bypass mode is generally used for more predictable syntax elements.

[0171] Coding in regular mode involves performing CABAC. Regular mode CABAC is for coding bin values where the probability of a value of a bin is predictable given the values of previously coded bins. The probability of a bin being an LPS is determined by context modeler 422. Context modeler 422 outputs the bin value and the context model (e.g., the probability state σ). The context model may be an initial context model for a series of bins or may be determined based on the coded values of previously coded bins. As described above, the context modeler may update the state based on whether or not the previously-coded bin was an MPS or an LPS.

[0172] After the context model and probability state σ are determined by context modeler 422, regular encoding engine 424 performs BAC on the bin value. According to the techniques of this disclosure, regular encoding engine 424 performs BAC using

TransIdxLPS table 430 that includes more than 64 probability states σ . In one example, the number of probability states is 128. TransIdxLPS is used to determine which probability state is used for a next bin (bin $n+1$) when the previous bin (bin n) is an LPS. Regular encoding engine 424 may also use a RangeLPS table 428 to determine the range value for an LPS given a particular probability state σ . However, according to the techniques of this disclosure, rather than using all possible probability states σ of the TransIdxLPS table 430, the probability state indexes σ are mapped to grouped indexes for use in the RangeLPS table. That is, each index into the RangeLPS table 428 may represent two or more of the total number of probability states. The mapping of probability state index σ to grouped indexes may be linear (e.g., by dividing by two), or may be non-linear (e.g., a logarithmic function or mapping table).

[0173] In other examples of the disclosure, the difference between successive probability states may be made smaller by setting the parameter α to be greater than 0.9493. In one example, $\alpha = 0.9689$. In another example of the disclosure, the highest probability (p_0) of an LPS occurring may be set to be lower than 0.5. In one example, p_0 may be equal to 0.493.

[0174] In accordance with one or more techniques of this disclosure, as opposed to using the same value of a variable used to update a probability state in a binary arithmetic coding process (e.g., one or more of a window size, a scaling factor (α), and a probability updating speed), entropy encoding unit 220 may use different values of the variable for different context models and/or different syntax elements. For instance, entropy encoding unit 220 may determine, for a context model of a plurality of context models, a value of a variable used to update a probability state in a binary arithmetic coding process and update the probability state based on the determined value.

[0175] FIG. 9 is a block diagram of an example entropy decoding unit 302, e.g., forming part of video decoder 300 as shown in FIG. 5, that may be configured to perform CABAC in accordance with the techniques of this disclosure. The entropy decoding unit 302 of FIG. 9 performs CABAC in an inverse manner as that of entropy encoding unit 220 described in FIG. 8. Coded bits from bitstream 448 are input into entropy decoding unit 302. The coded bits are fed to either context modeler 450 or bypass decoding engine 452 based on whether or not the coded bits were entropy coded using bypass mode or regular mode. If the coded bits were coded in bypass mode, bypass decoding engine 452 may, for example, use Golomb-Rice or exponential

Golomb decoding to retrieve the binary-valued syntax elements or bins of non-binary syntax elements.

[0176] If the coded bits were coded in regular mode, context modeler 450 may determine a probability model for the coded bits and regular decoding engine 454 may decode the coded bits to produce bins of non-binary valued syntax elements (or the syntax elements themselves if binary-valued). After the context model and probability state σ is determined by context modeler 450, regular decoding engine 454 performs BAC on the bin value. According to the techniques of this disclosure, regular decoding engine 454 performs BAC using TransIdxLPS table 458 that includes more than 64 probability states σ . In one example, the number of probability states is 128, although other numbers of probability states could be defined, consistent with the techniques of this disclosure. TransIdxLPS table 458 is used to determine which probability state is used for a next bin (bin $n+1$) when the previous bin (bin n) is an LPS. Regular decoding engine 454 may also use a RangeLPS table 456 to determine the range value for an LPS given a particular probability state σ . However, according to the techniques of this disclosure, rather than using all possible probability states σ of the TransIdxLPS table 458, the probability state indexes σ are mapped to grouped indexes for use in RangeLPS table 456. That is, each index into RangeLPS table 456 may represent two or more of the total number of probability states. The mapping of probability state index σ to grouped indexes may be linear (e.g., by dividing by two), or may be non-linear (e.g., a logarithmic function or mapping table).

[0177] In other examples of the disclosure, the difference between successive probability states may be made smaller by setting the parameter α to be greater than 0.9493. In one example, $\alpha = 0.9689$. In another example of the disclosure, the highest probability (p_0) of an LPS occurring may be set to be lower than 0.5. In one example, p_0 may be equal to 0.493.

[0178] After the bins are decoded by regular decoding engine 454, a reverse binarizer 460 may perform a reverse mapping to convert the bins back into the values of the non-binary valued syntax elements.

[0179] FIG. 10 is a flowchart illustrating an example method for encoding a current block. The current block may comprise a current CU. Although described with respect to video encoder 200 (FIGS. 1 and 4), it should be understood that other devices may be configured to perform a method similar to that of FIG. 10.

[0180] In this example, video encoder 200 initially predicts the current block (550). For example, video encoder 200 may form a prediction block for the current block. Video encoder 200 may then calculate a residual block for the current block (552). To calculate the residual block, video encoder 200 may calculate a difference between the original, uncoded block and the prediction block for the current block. Video encoder 200 may then transform and quantize coefficients of the residual block (554). In some coding modes, such as a transform skip mode, video encoder 200 may skip the transform and only quantize the residual data. Next, video encoder 200 may scan the quantized coefficients of the residual block (556). During the scan, or following the scan, video encoder 200 may entropy encode the coefficients (558). For example, video encoder 200 may encode the coefficients using CAVLC or CABAC. Video encoder 200 may then output the entropy coded data of the coefficients (560).

[0181] FIG. 11 is a flowchart illustrating an example method for decoding a current block of video data. The current block may comprise a current CU. Although described with respect to video decoder 300 (FIGS. 1 and 5), it should be understood that other devices may be configured to perform a method similar to that of FIG. 11.

[0182] Video decoder 300 may receive entropy coded data for the current block, such as entropy coded prediction information and entropy coded data for coefficients of a residual block corresponding to the current block (570). Video decoder 300 may entropy decode the entropy coded data to determine prediction information for the current block and to reproduce coefficients of the residual block (572). Video decoder 300 may predict the current block (574), e.g., using an intra- or inter-prediction mode as indicated by the prediction information for the current block, to calculate a prediction block for the current block. Video decoder 300 may then inverse scan the reproduced coefficients (576), to create a block of quantized coefficients. Video decoder 300 may then inverse quantize and inverse transform the coefficients to produce a residual block (578). In some coding modes, such as a transform skip mode, video decoder 300 may skip the inverse transform and only inverse quantize the coefficients. Video decoder 300 may ultimately decode the current block by combining the prediction block and the residual block (580).

[0183] FIG. 12 is a flowchart illustrating an example method for determining a context for coding (e.g., encoding or decoding) a sign of a coefficient of a residual block. The residual block may, for example, be a transform skipped residual block. The techniques of FIG. 12 will be described with respect to a generic video coder, which may

correspond to either a video encoder such as video encoder 200 or a video decoder such as video decoder 300. It should be understood, however, that other devices may be configured to perform a method similar to that of FIG. 12.

[0184] The video coder determines a value (X0) for a first neighboring coefficient of a coefficient currently being coded (600). The video coder determines a value (X1) for a second neighboring coefficient of the coefficient currently being coded (602). In response to both X0 and X1 being equal to zero (604, yes), the video coder sets a context for coding the sign of the coefficient currently being coded to a first context (606). In response to at least one of X0 or X1 not being equal to zero (604, no) but X0 and X1 having opposite signs (608, yes), the video coder also sets a context for coding the sign of the coefficient currently being coded to the first context (610).

[0185]). In response to at least one of X0 or X1 not being equal to zero (604, no), X0 and X1 not having opposite signs (608, no), and one of X0 or X1 being greater than zero (612, yes), the video coder sets a context for coding the sign of the coefficient currently being coded to a second context, different than the first context (614). In response to at least one of X0 or X1 not being equal to zero (604, no), X0 and X1 not having opposite signs (608, no), and neither X0 nor X1 being greater than zero (612, no), the video coder sets a context for coding the sign of the coefficient currently being coded to a third context, different than the first or second context (616).

[0186] It is to be recognized that depending on the example, certain acts or events of any of the techniques described herein can be performed in a different sequence, may be added, merged, or left out altogether (e.g., not all described acts or events are necessary for the practice of the techniques). Moreover, in certain examples, acts or events may be performed concurrently, e.g., through multi-threaded processing, interrupt processing, or multiple processors, rather than sequentially.

[0187] In one or more examples, the functions described may be implemented in hardware, software, firmware, or any combination thereof. If implemented in software, the functions may be stored on or transmitted over as one or more instructions or code on a computer-readable medium and executed by a hardware-based processing unit. Computer-readable media may include computer-readable storage media, which corresponds to a tangible medium such as data storage media, or communication media including any medium that facilitates transfer of a computer program from one place to another, e.g., according to a communication protocol. In this manner, computer-readable media generally may correspond to (1) tangible computer-readable storage

media which is non-transitory or (2) a communication medium such as a signal or carrier wave. Data storage media may be any available media that can be accessed by one or more computers or one or more processors to retrieve instructions, code and/or data structures for implementation of the techniques described in this disclosure. A computer program product may include a computer-readable medium.

[0188] By way of example, and not limitation, such computer-readable storage media can comprise RAM, ROM, EEPROM, CD-ROM or other optical disk storage, magnetic disk storage, or other magnetic storage devices, flash memory, or any other medium that can be used to store desired program code in the form of instructions or data structures and that can be accessed by a computer. Also, any connection is properly termed a computer-readable medium. For example, if instructions are transmitted from a website, server, or other remote source using a coaxial cable, fiber optic cable, twisted pair, digital subscriber line (DSL), or wireless technologies such as infrared, radio, and microwave, then the coaxial cable, fiber optic cable, twisted pair, DSL, or wireless technologies such as infrared, radio, and microwave are included in the definition of medium. It should be understood, however, that computer-readable storage media and data storage media do not include connections, carrier waves, signals, or other transitory media, but are instead directed to non-transitory, tangible storage media. Disk and disc, as used herein, includes compact disc (CD), laser disc, optical disc, digital versatile disc (DVD), floppy disk and Blu-ray disc, where disks usually reproduce data magnetically, while discs reproduce data optically with lasers. Combinations of the above should also be included within the scope of computer-readable media.

[0189] Instructions may be executed by one or more processors, such as one or more digital signal processors (DSPs), general purpose microprocessors, application specific integrated circuits (ASICs), field programmable gate arrays (FPGAs), or other equivalent integrated or discrete logic circuitry. Accordingly, the terms “processor” and “processing circuitry,” as used herein may refer to any of the foregoing structures or any other structure suitable for implementation of the techniques described herein. In addition, in some aspects, the functionality described herein may be provided within dedicated hardware and/or software modules configured for encoding and decoding, or incorporated in a combined codec. Also, the techniques could be fully implemented in one or more circuits or logic elements.

[0190] The techniques of this disclosure may be implemented in a wide variety of devices or apparatuses, including a wireless handset, an integrated circuit (IC) or a set of

ICs (e.g., a chip set). Various components, modules, or units are described in this disclosure to emphasize functional aspects of devices configured to perform the disclosed techniques, but do not necessarily require realization by different hardware units. Rather, as described above, various units may be combined in a codec hardware unit or provided by a collection of interoperative hardware units, including one or more processors as described above, in conjunction with suitable software and/or firmware.

[0191] Various examples have been described. These and other examples are within the scope of the following claims.

[0192] The reference to any prior art in this specification is not, and should not be taken as, an acknowledgement or any form of suggestion that such prior art forms part of the common general knowledge.

[0193] It will be understood that the terms “comprise” and “include” and any of their derivatives (e.g. comprises, comprising, includes, including) as used in this specification, and the claims that follow, is to be taken to be inclusive of features to which the term refers, and is not meant to exclude the presence of any additional features unless otherwise stated or implied.

WHAT IS CLAIMED IS:

1. A method for performing sign decoding for coefficients in a residual block of video data encoded using a transform skip mode, comprising:
 - determining a value for a first neighboring coefficient of a coefficient currently being decoded;
 - determining a value for a second neighboring coefficient of the coefficient currently being decoded;
 - determining a context offset for the coefficient currently being decoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, comprising:
 - selecting a first context offset in response to the first neighboring coefficient and the second neighboring coefficient are both equal to zero or have opposite signs;
 - otherwise, selecting a second context offset in response to the first neighboring coefficient and the second neighboring coefficient are both positive or one of the first neighboring coefficient and the second neighboring coefficient is equal to zero and the other one of the first neighboring coefficient and the second neighboring coefficient is positive; and
 - otherwise, selecting a third context offset in response to the first neighboring coefficient and the second neighboring coefficient are both negative or one of the first neighboring coefficient and the second neighboring coefficient is equal to zero and the other one of the first neighboring coefficient and the second neighboring coefficient is negative; and
 - decoding a sign for the coefficient currently being decoded based on the determined context offset.
2. The method of claim 1, wherein the first neighboring coefficient comprises one of a top neighboring coefficient or a left neighboring coefficient to the coefficient currently being decoded, and the second neighboring coefficient comprises the other of the top neighboring coefficient or the left neighboring coefficient.

3. The method of claim 1, wherein decoding the sign for the coefficient currently being decoded based on the determined context offset comprises:

determining a context based on the determined context offset;

receiving one or more bins of data; and

context decoding the one or more bins of data based on the determined context to determine the sign for the coefficient currently being decoded.

4. The method of claim 3, wherein decoding the sign for the coefficient currently being decoded based on the determined context offset further comprises:

determining a predicted level value for the coefficient currently being decoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient;

receiving a syntax element; and

in response to the syntax element having a value equal to one, determining that a level value of the coefficient currently being decoded is equal to the predicted level value; or

in response to a value for the syntax element being greater than the predicted level value, determining that a level value of the coefficient currently being decoded is equal to the value for the syntax element plus one; or

in response to a value for the syntax element being less than the predicted level value, determining that a level value of the coefficient currently being decoded is equal to the value for the syntax element.

5. The method of claim 1, further comprising:

inverse quantizing, without inverse transforming, the value for the coefficient currently being decoded to determine a residual value for the residual block of video data.

6. The method of claim 1, further comprising:

determining a decoded residual block based on the value for the coefficient currently being decoded;

adding the decoded residual block to a prediction block to determine a reconstructed block;

performing one or more filtering operations on the reconstructed block to determine a decoded block of video data; and
outputting a decoded picture of video data that includes the decoded block of video data.

7. A method for performing sign encoding for coefficients in a residual block of video data encoded using a transform skip mode, comprising:

determining a value for a first neighboring coefficient of a coefficient currently being encoded;

determining a value for a second neighboring coefficient of the coefficient currently being encoded;

determining a context offset for the coefficient currently being encoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, comprising:

selecting a first context offset in response to the first neighboring coefficient and the second neighboring coefficient are both equal to zero or have opposite signs;

otherwise, selecting a second context offset in response to the first neighboring coefficient and the second neighboring coefficient are both positive or one of the first neighboring coefficient and the second neighboring coefficient is equal to zero and the other one of the first neighboring coefficient and the second neighboring coefficient is positive; and

otherwise, selecting a third context offset in response to the first neighboring coefficient and the second neighboring coefficient are both negative or one of the first neighboring coefficient and the second neighboring coefficient is equal to zero and the other one of the first neighboring coefficient and the second neighboring coefficient is negative; and

encoding a sign for the coefficient currently being encoded based on the determined context offset.

8. The method of claim 7, wherein the first neighboring coefficient comprises one of a top neighboring coefficient or a left neighboring coefficient, and the second neighboring coefficient comprises the other of the top neighboring coefficient or the left neighboring coefficient.

9. The method of claim 7, wherein encoding the sign for the coefficient currently being encoded based on the determined context offset comprises:

determining a context based on the determined context offset;

determining the sign for the coefficient currently being encoded; and

context encoding one or more bins of data based on the determined context to represent the sign for the coefficient currently being encoded.

10. The method of claim 9, wherein encoding the sign for the coefficient currently being encoded based on the determined context offset further comprises:

determining a predicted level value for the coefficient currently being encoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient;

in response to a level value of the coefficient currently being encoded being equal to the predicted level value, encoding a syntax element with a value equal to one; or

in response to a level value of the coefficient currently being encoded being less than the predicted level value, encoding a syntax element with a value equal to the level value of the coefficient currently being encoded; or

in response to a level value of the coefficient currently being encoded being greater than the predicted level value, encoding a syntax element with a value equal to the level value of the coefficient currently being encoded minus one.

11. The method of claim 7, further comprising:

quantizing a residual value for the residual block of video data to determine the value for the coefficient currently being encoded.

12. The method of claim 7, further comprising:

determining a prediction block; and

comparing the prediction block to an original block of video data to determine the residual block of video data.

13. A device for performing sign decoding for coefficients in a residual block of video data encoded using a transform skip mode, the device comprising:

a memory configured to store video data; and
one or more processors configured to:

determine a value for a first neighboring coefficient of a coefficient currently being decoded;

determine a value for a second neighboring coefficient of the coefficient currently being decoded;

determine a context offset for the coefficient currently being decoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, wherein to determine the context offset for the coefficient currently being decoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, the one or more processors are configured to:

select a first context offset in response to the first neighboring coefficient and the second neighboring coefficient are both equal to zero or have opposite signs;

otherwise, select a second context offset in response to the first neighboring coefficient and the second neighboring coefficient are both positive or one of the first neighboring coefficient and the second neighboring coefficient is equal to zero and the other one of the first neighboring coefficient and the second neighboring coefficient is positive; and

otherwise, select a third context offset in response to the first neighboring coefficient and the second neighboring coefficient are both negative or one of the first neighboring coefficient and the second neighboring coefficient is equal to zero and the other one of the first neighboring coefficient and the second neighboring coefficient is negative; and

decode a sign for the coefficient currently being decoded based on the determined context offset.

14. The device of claim 13, wherein the one or more processors are further configured to carry out the method of any one of claims 2 to 6.

15. A device for performing sign encoding for coefficients of a residual block of video data encoded using a transform skip mode, the device comprising:
- a memory configured to store video data; and
 - one or more processors configured to:
 - determine a value for a first neighboring coefficient of a coefficient currently being encoded;
 - determine a value for a second neighboring coefficient of the coefficient currently being encoded;
 - determine a context offset for the coefficient currently being encoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, wherein to determine the context offset for the coefficient currently being decoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, the one or more processors are configured to:
 - select a first context offset in response to the first neighboring coefficient and the second neighboring coefficient are both equal to zero or have opposite signs;
 - otherwise, select a second context offset in response to the first neighboring coefficient and the second neighboring coefficient are both positive or one of the first neighboring coefficient and the second neighboring coefficient is equal to zero and the other one of the first neighboring coefficient and the second neighboring coefficient is positive; and
 - otherwise, select a third context offset in response to the first neighboring coefficient and the second neighboring coefficient are both negative or one of the first neighboring coefficient and the second neighboring coefficient is equal to zero and the other one of the first neighboring coefficient and the second neighboring coefficient is negative; and
 - encode a sign for the coefficient currently being encoded based on the determined context offset.
16. The device of claim 15, wherein the one or more processors are further configured to carry out the method of any one of claims 8 to 12.

17. A non-transitory computer-readable storage medium storing instructions for performing sign decoding for coefficients of a residual block of video data encoded using a transform skip mode, wherein the instruction when executed by one or more processors cause the one or more processors to:

- determine a value for a first neighboring coefficient of a coefficient currently being decoded;

- determine a value for a second neighboring coefficient of the coefficient currently being decoded;

- determine a context offset for the coefficient currently being decoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient; wherein to determine the context offset for the coefficient currently being decoded based on the value for the first neighboring coefficient and the value for the second neighboring coefficient, the instructions cause the one or more processors:

- select a first context offset in response to the first neighboring coefficient and the second neighboring coefficient are both equal to zero or have opposite signs;

- otherwise, select a second context offset in response to the first neighboring coefficient and the second neighboring coefficient are both positive or one of the first neighboring coefficient and the second neighboring coefficient is equal to zero and the other one of the first neighboring coefficient and the second neighboring coefficient is positive; and

- otherwise, select a third offset in response to the first neighboring coefficient and the second neighboring coefficient are both negative or one of the first neighboring coefficient and the second neighboring coefficient is equal to zero and the other one of the first neighboring coefficient and the second neighboring coefficient is negative; and

- decode a sign for the coefficient currently being decoded based on the determined context offset.

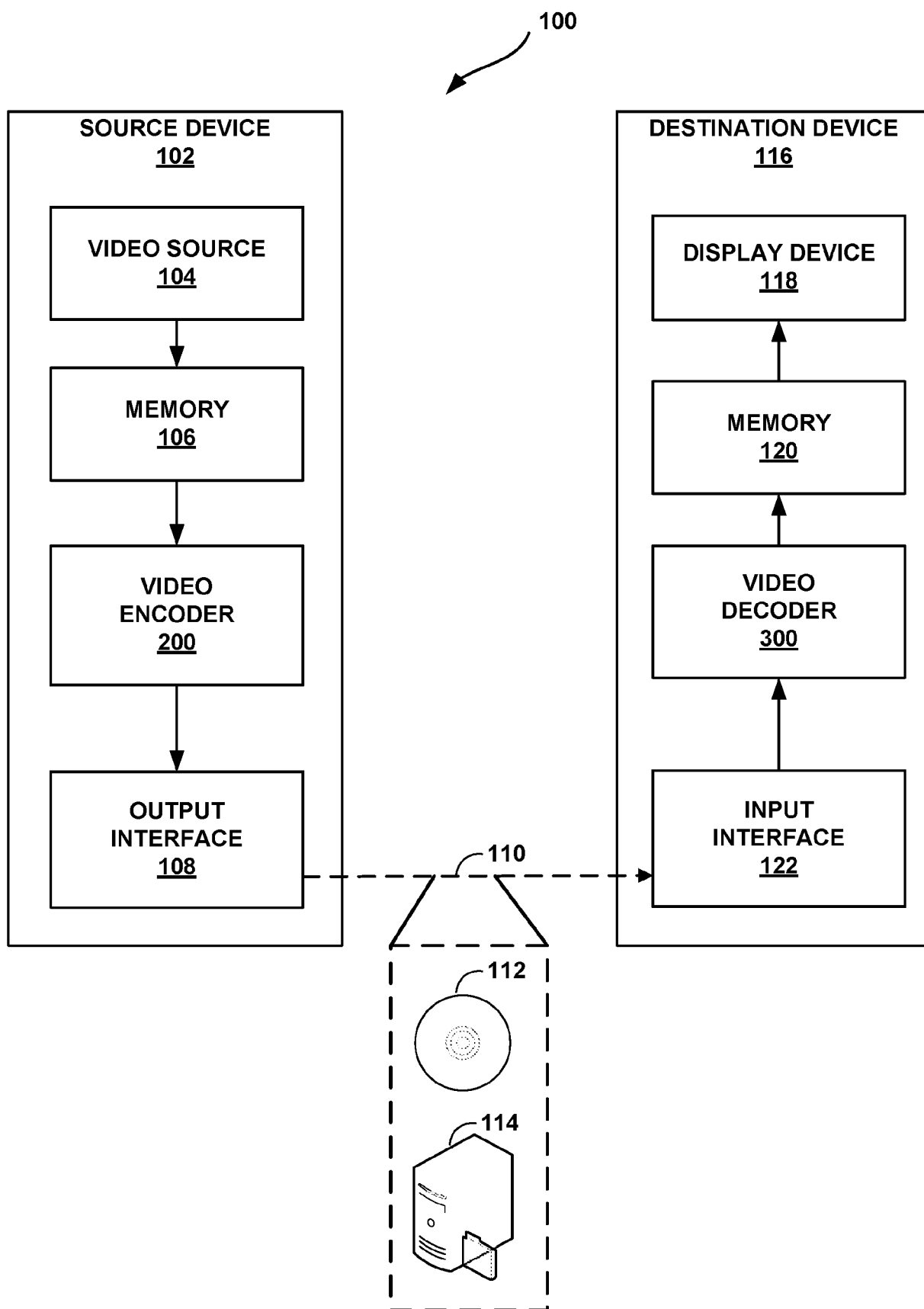


FIG. 1

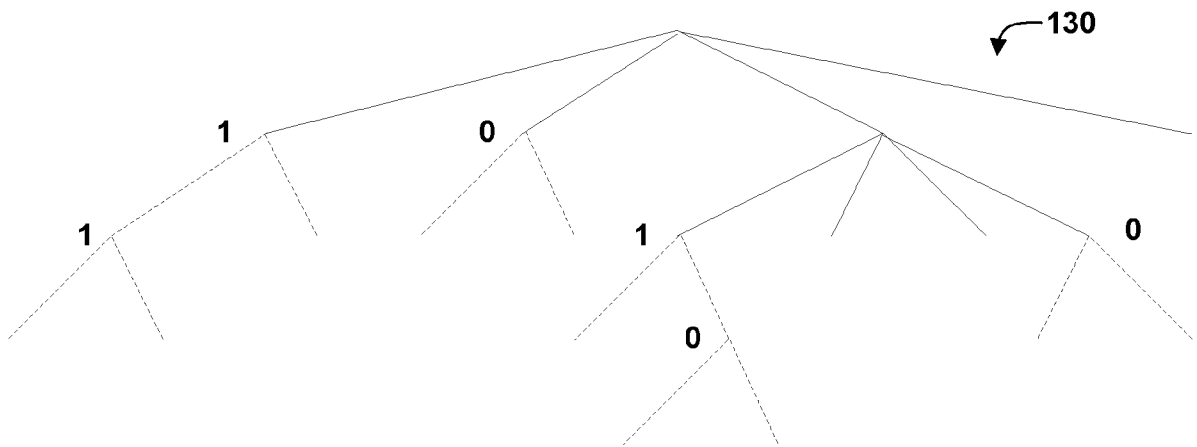


FIG. 2A

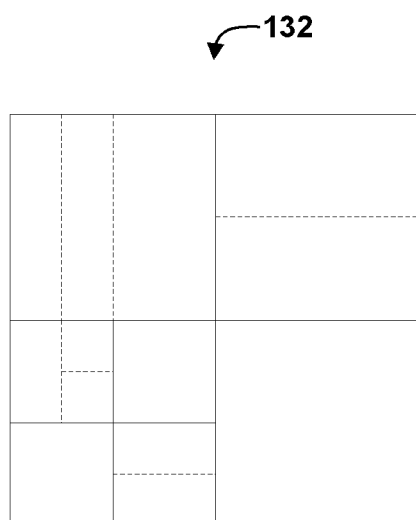


FIG. 2B

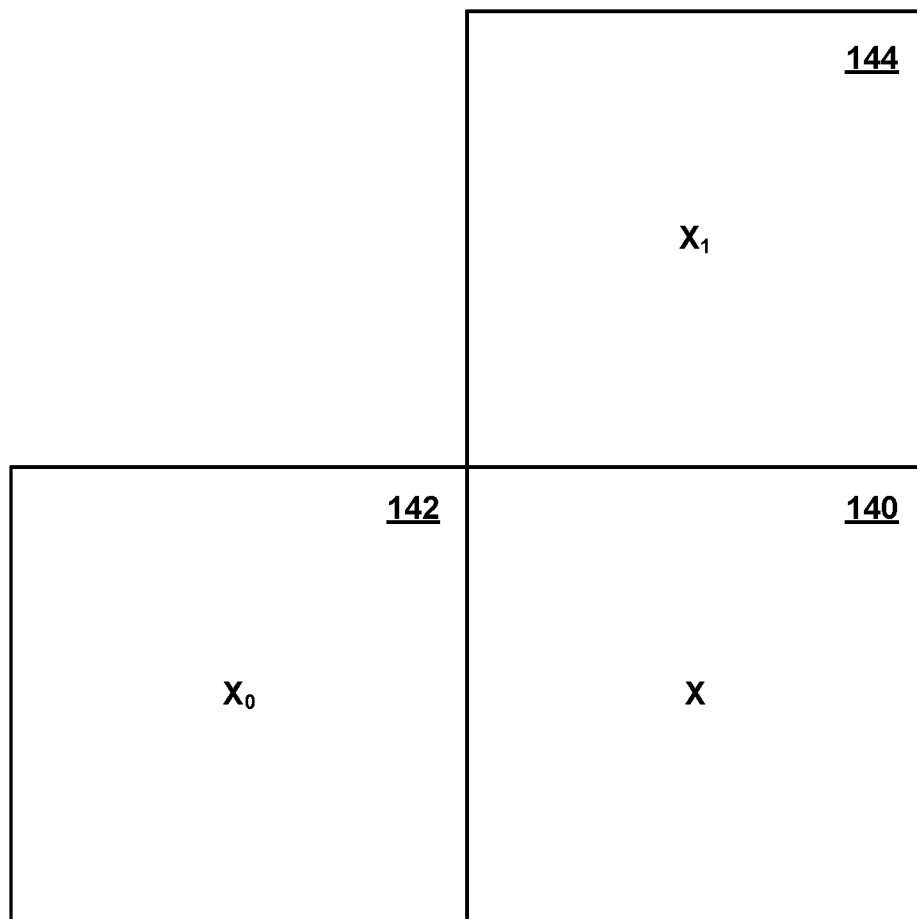


FIG. 3

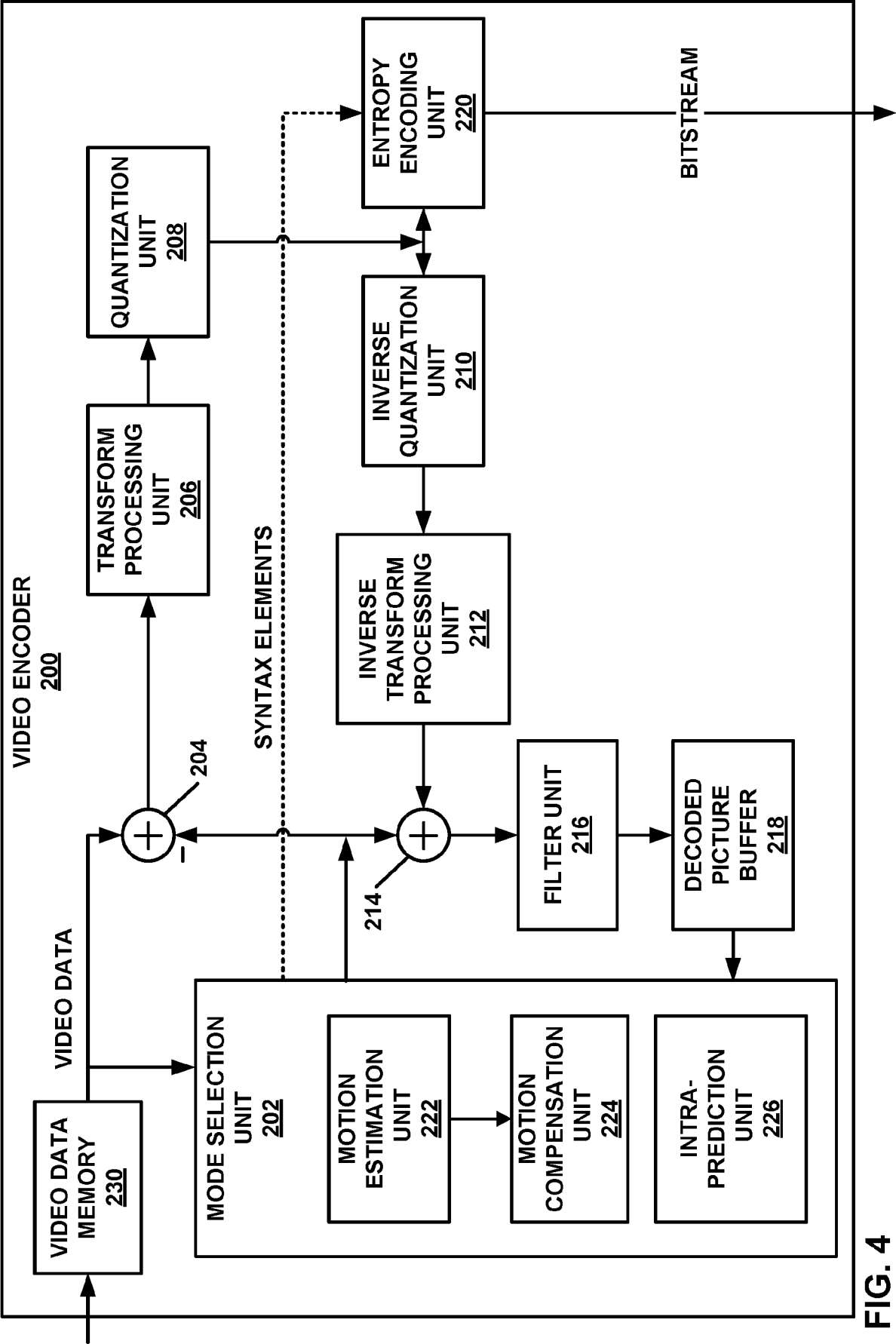


FIG. 4

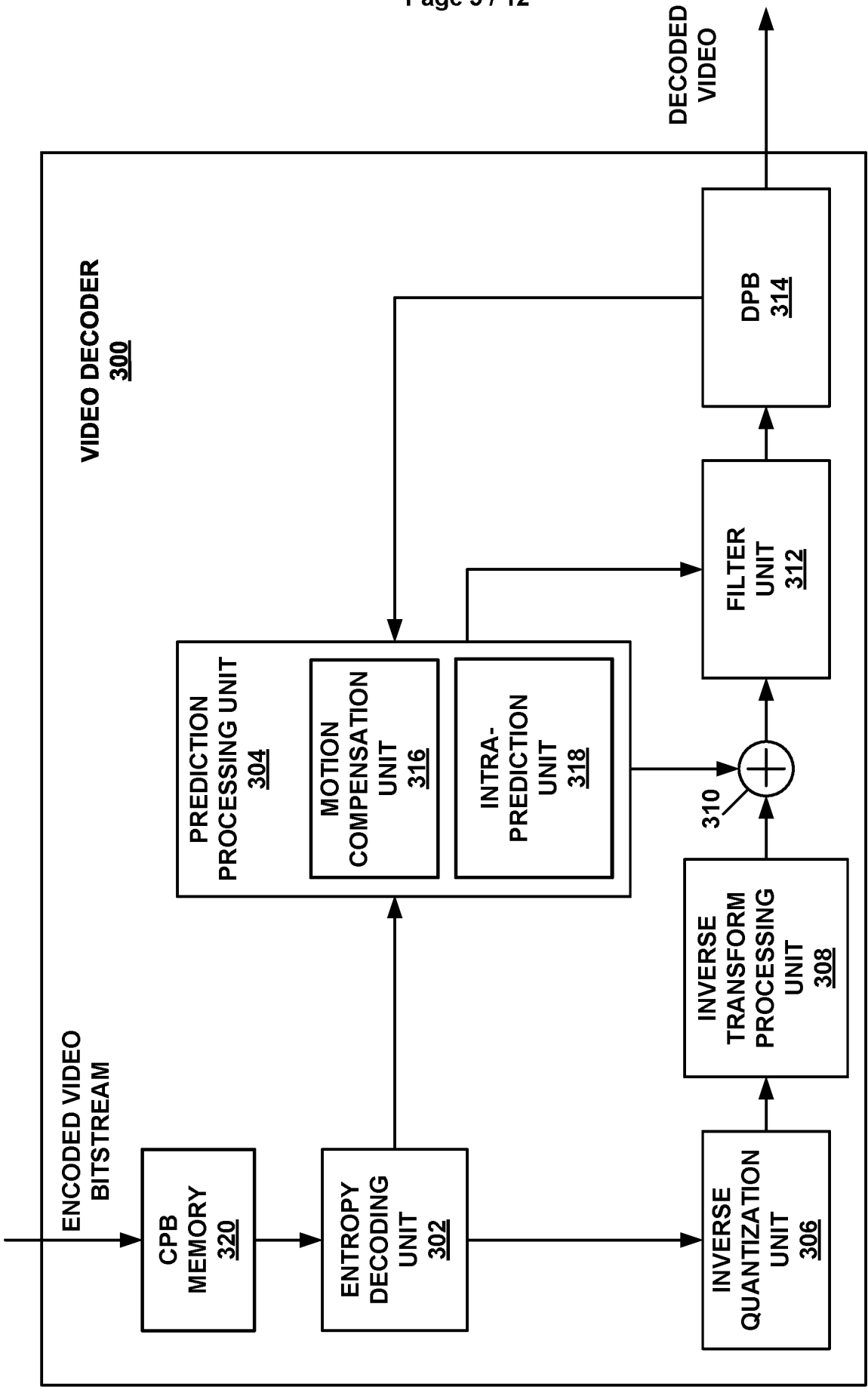


FIG. 5

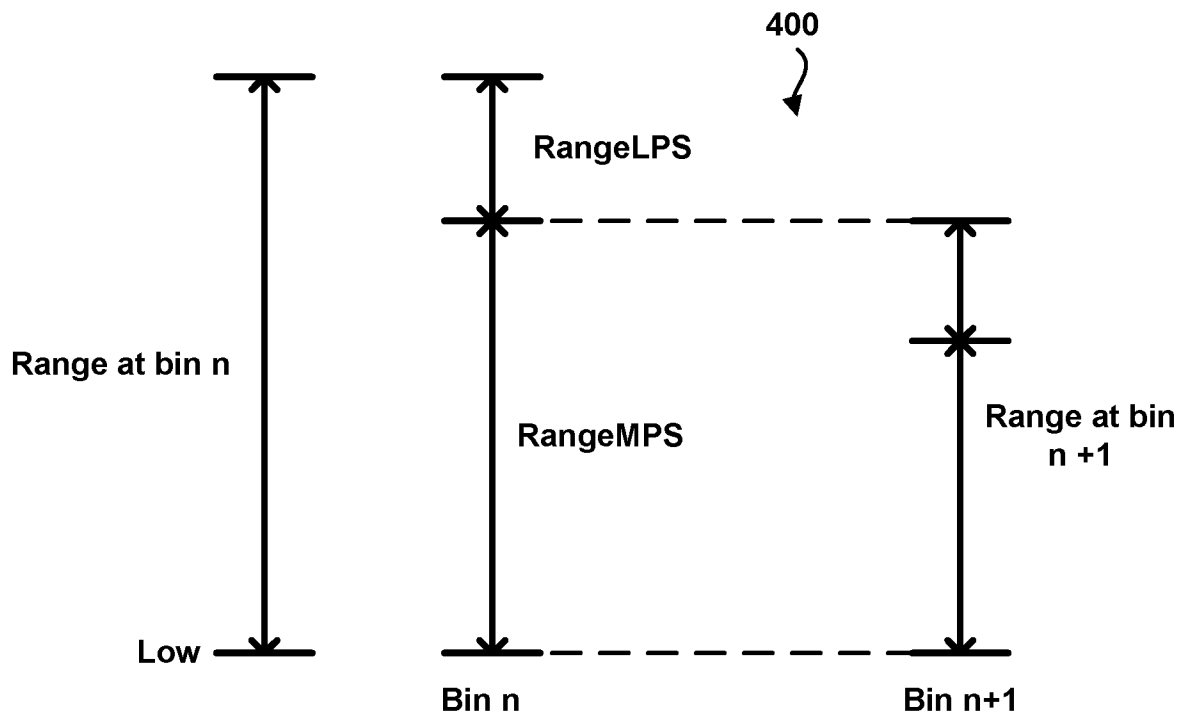


FIG. 6A

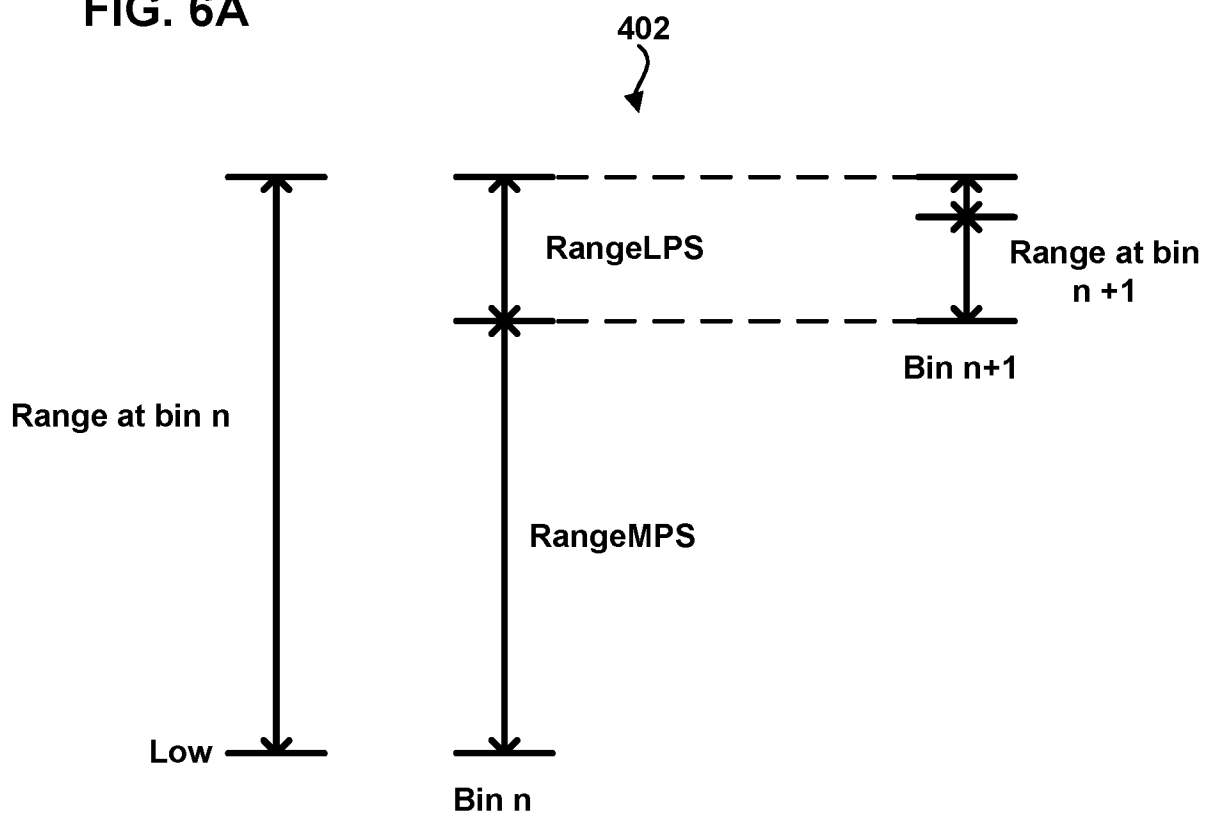


FIG. 6B

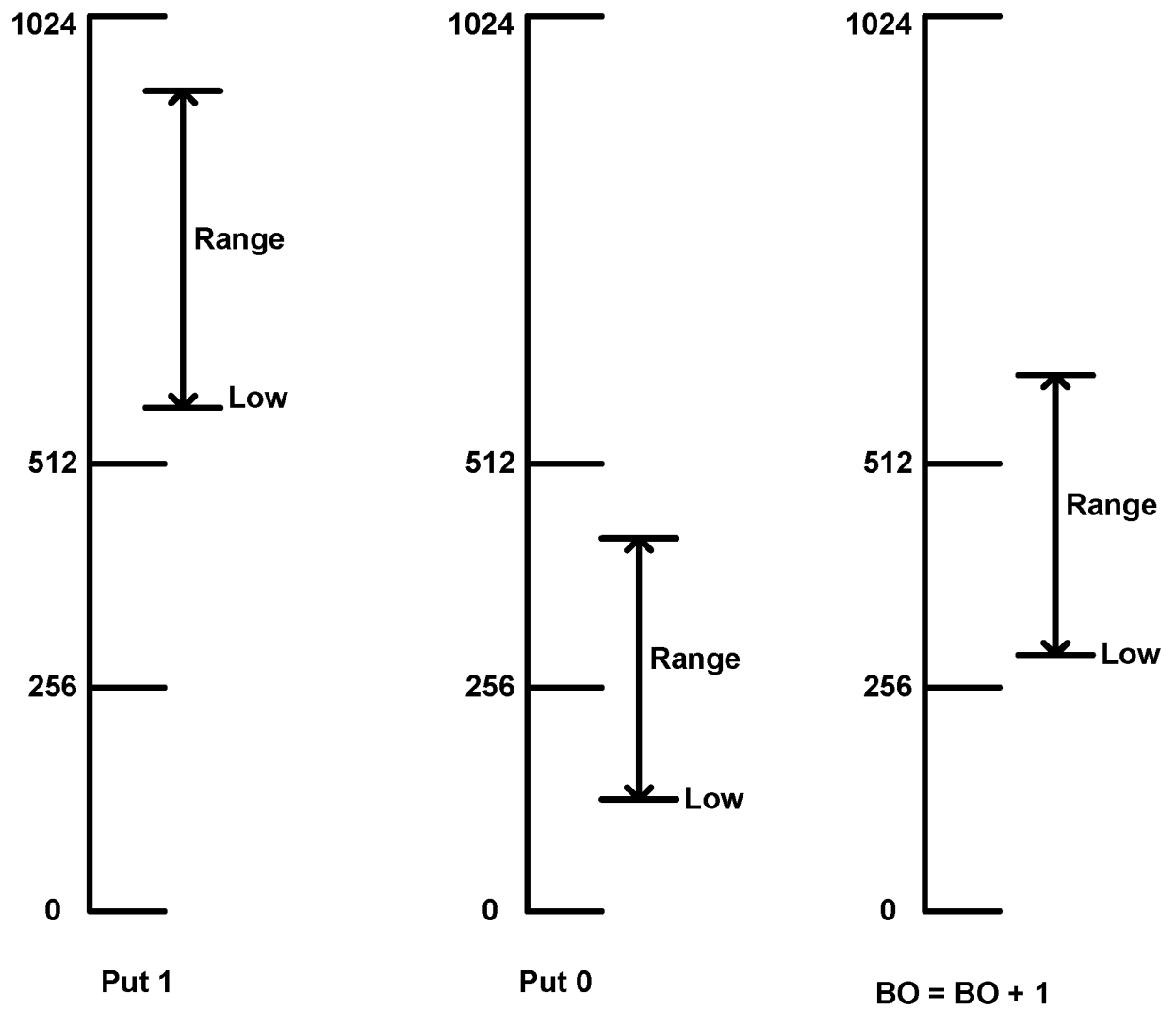


FIG. 7

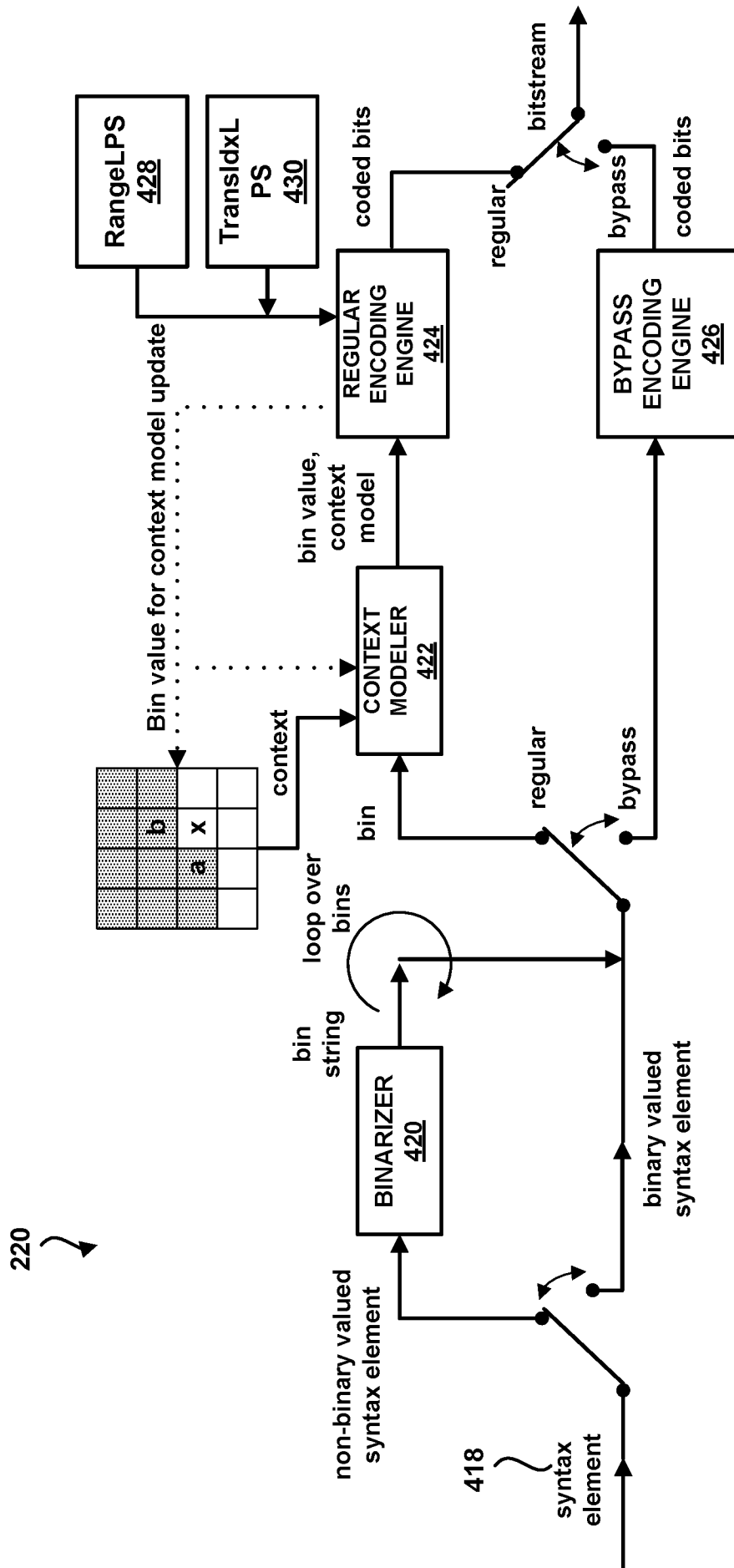


FIG. 8

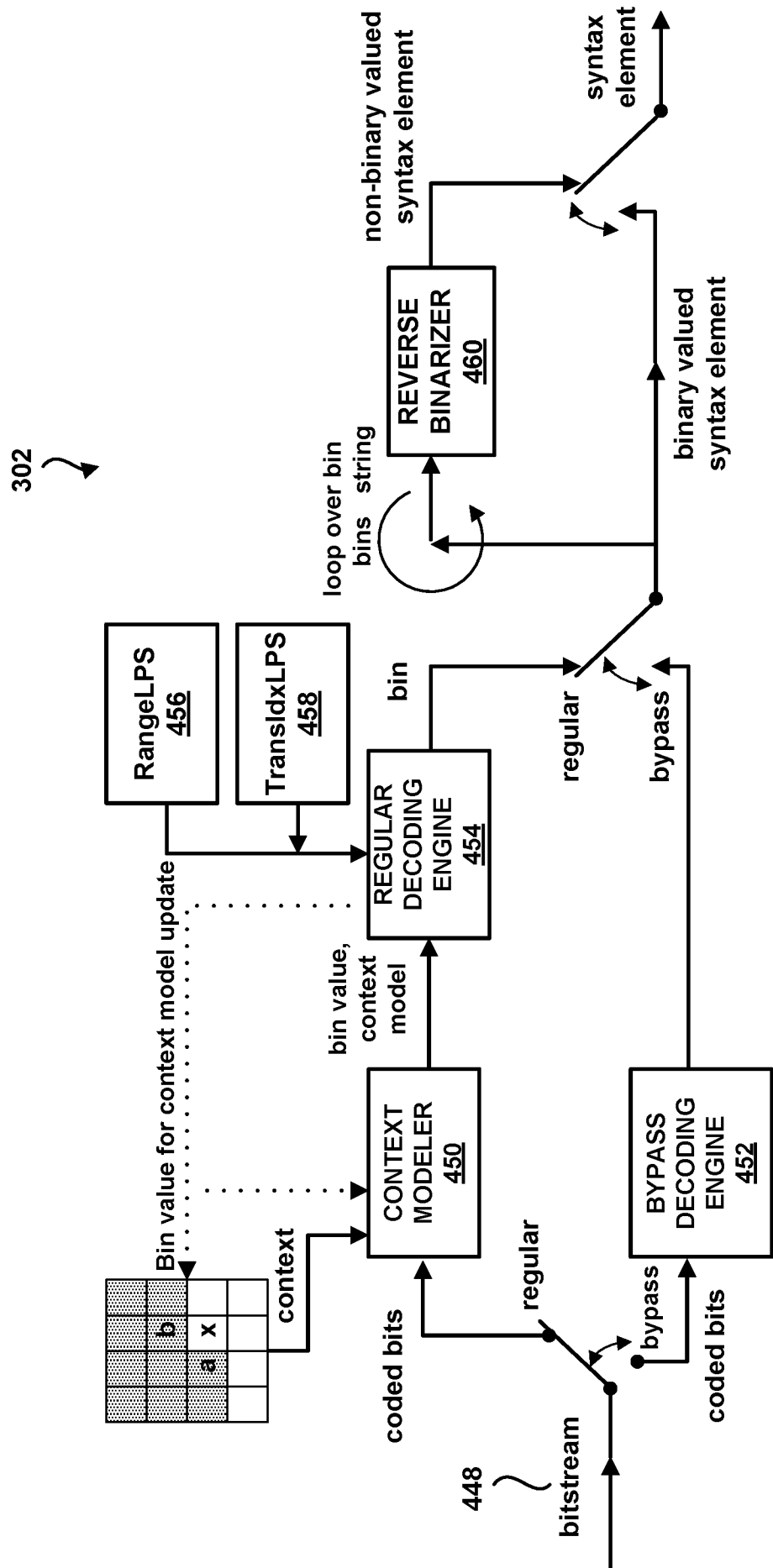


FIG. 9

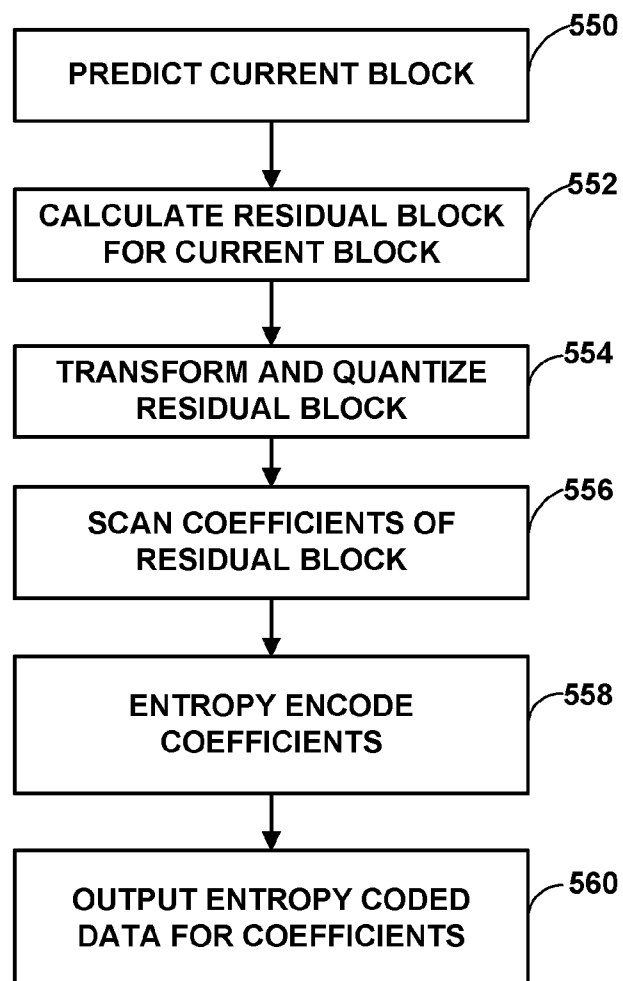


FIG. 10

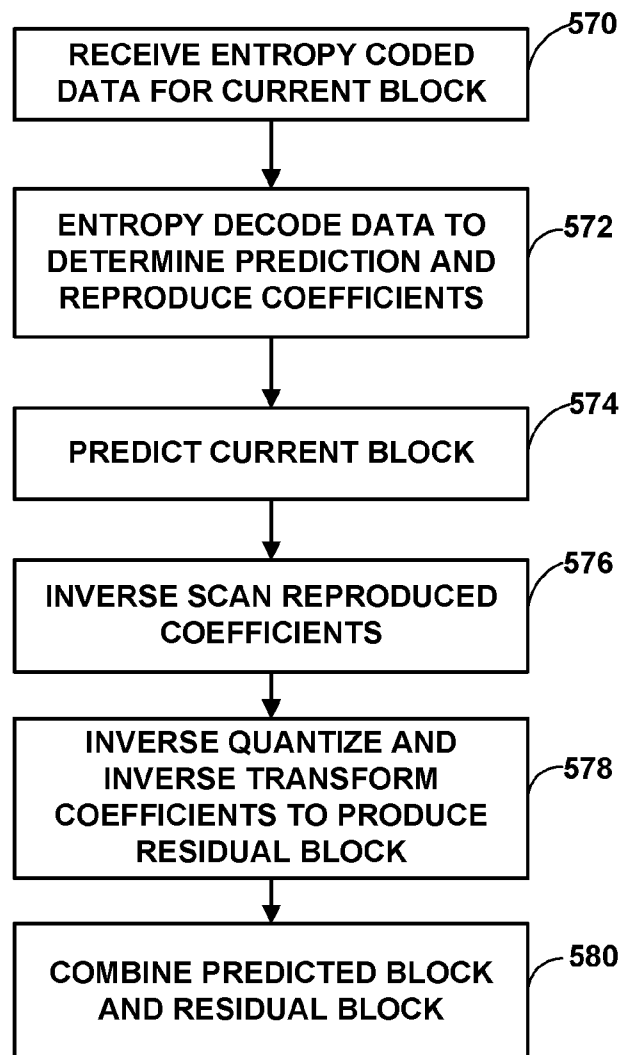


FIG. 11

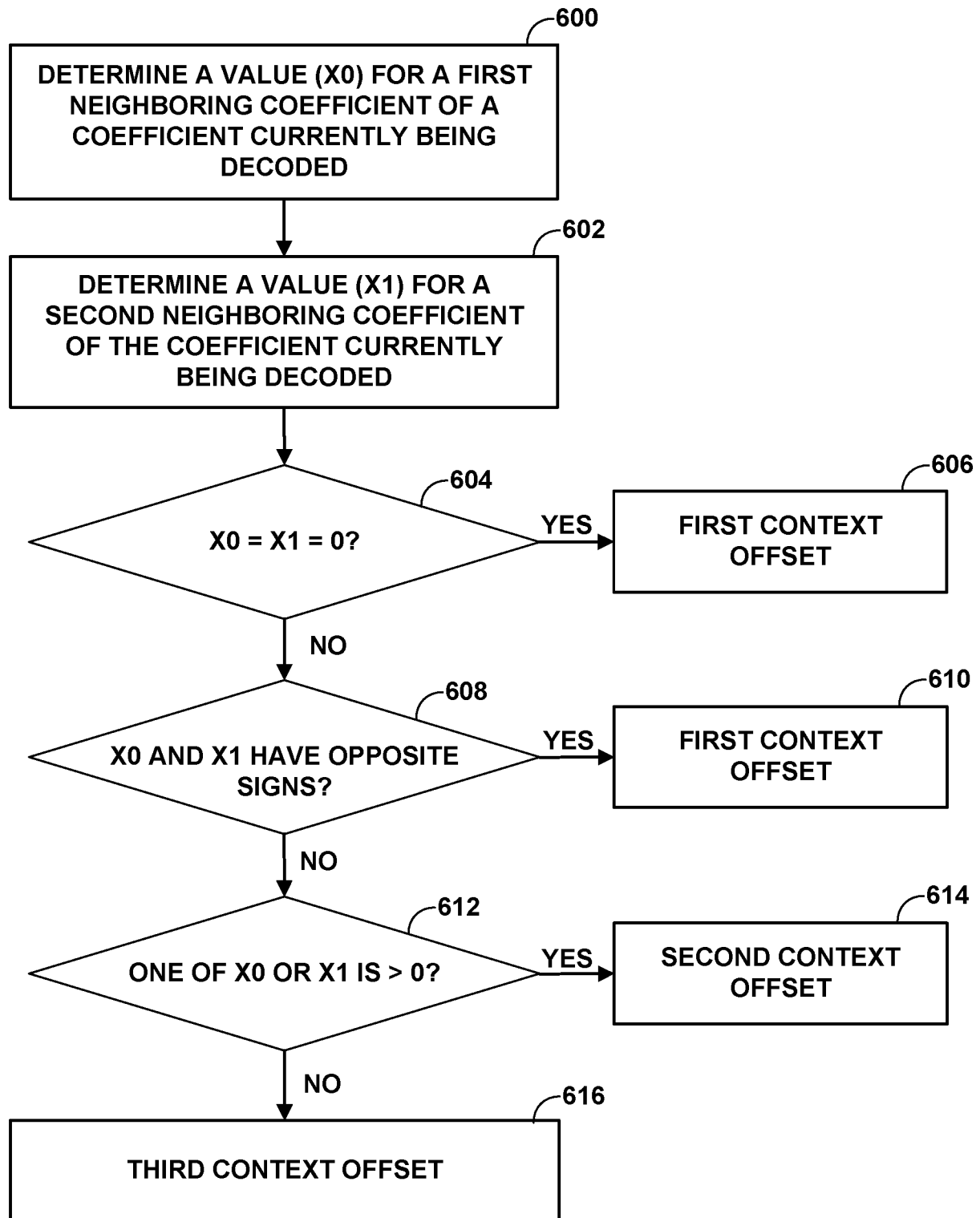


FIG. 12