

Table 8-68-58-5 – Specification of intraFilterType[nS][IntraPredMode] for various prediction unit sizes

IntraPredMode	intraFilterType for nS = 4	intraFilterType for nS = 8	intraFilterType for nS = 16	intraFilterType for nS = 32	intraFilterType for nS = 64
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	1	1	<u>2</u>	<u>2</u>	<u>2</u>
4	0	1	<u>2</u>	<u>2</u>	<u>2</u>
5	0	<u>2</u>	<u>2</u>	<u>2</u>	0
6	1	<u>2</u>	<u>2</u>	<u>2</u>	0
7	0	<u>2</u>	<u>2</u>	<u>2</u>	0
8	0	1	<u>2</u>	<u>2</u>	0
9	1	1	<u>2</u>	<u>2</u>	0
10	0	0	<u>2</u>	<u>2</u>	0
11	0	0	<u>2</u>	<u>2</u>	0
12	0	0	<u>2</u>	<u>2</u>	0
13	0	0	<u>2</u>	<u>2</u>	0
14	0	0	<u>2</u>	<u>2</u>	0
15	0	0	<u>2</u>	<u>2</u>	0
16	0	0	<u>2</u>	<u>2</u>	0
17	0	0	<u>2</u>	<u>2</u>	0
18-33	0	0	0	<u>2</u>	0
<u>34</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>
<u>35</u>	<u>n/a</u>	<u>n/a</u>	<u>n/a</u>	<u>n/a</u>	<u>n/a</u>

Filtered sample array pF[x, y] with x = -1..nS*2-1 and y = -1..nS*2-1 are derived as follows:

– If When intraFilterType[nS][IntraPredMode] is equal to 1, the following applies:

$$pF[-1, nS*2-1] = p[-1, nS*2-1] \quad (8-191917)$$

$$pF[nS*2-1, -1] = p[nS*2-1, -1] \quad (8-202018)$$

$$pF[-1, y] = (p[-1, y+1] + 2*p[-1, y] + p[-1, y-1] + 2) >> 2 \text{ for } y = nS*2-2..0 \quad (8-212119)$$

$$pF[-1, -1] = (p[-1, 0] + 2*p[-1, -1] + p[0, -1] + 2) >> 2 \quad (8-222220)$$

$$pF[x, -1] = (p[x-1, -1] + 2*p[x, -1] + p[x+1, -1] + 2) >> 2 \text{ for } x = 0..nS*2-2 \quad (8-232321)$$

— Otherwise, intraFilterType[nS][IntraPredMode] is equal to 2, the following applies:

$$pF[-1, nS*2-1] = p[-1, nS*2-1] \quad (8-22)$$

$$pF[nS*2-1, -1] = p[nS*2-1, -1] \quad (8-23)$$

$$pF[-1, y] = (p[-1, y+1] + 2*p[-1, y] + p[-1, y-1] + 2) >> 2 \text{ for } y = nS*2-2..0 \quad (8-24)$$

$$pF[-1, -1] = (p[-1, 0] + 2*p[-1, -1] + p[0, -1] + 2) >> 2 \quad (8-25)$$

$$pF[x, -1] = (p[x-1, -1] + 2*p[x, -1] + p[x+1, -1] + 2) >> 2 \text{ for } x = 0..nS*2-2 \quad (8-26)$$

$$pF[-1, y] = (pF[-1, y+1] + 2*pF[-1, y] + pF[-1, y-1] + 2) >> 2 \text{ for } y = nS*2-2..0 \quad (8-27)$$

Formatted: Normal, Indent: Left: 0", Hanging: 0.2", Tab stops: 0.2", Left + Not at 0.59" + 0.79" + 0.98"

HEVC

$$pF[-1, -1] = (pF1[-1, 0] + 2 * pF1[-1, -1] + pF1[0, -1] + 2) \gg 2 \quad (8-28)$$

$$pF[x, -1] = (pF1[x-1, -1] + 2 * pF1[x, -1] + pF1[x+1, -1] + 2) \gg 2 \text{ for } x=0..nS-2 \quad (8-29)$$

8.3.3.1.48.3.3.1.3 Specification of Intra_Vertical prediction mode

Inputs to this process are:

- neighbouring samples $p[x, y]$, with $x, y = -1..2*nS-1$,
- a variable nS specifying the prediction size.

Output of this process is:

- predicted samples $predSamples[x, y]$, with $x, y = 0..nS-1$.

This intra prediction mode is invoked when $intraPredMode$ is equal to 0.

The values of the prediction samples $predSamples[x, y]$, with $x, y = 0..nS-1$, are derived by

$$predSamples[x, y] = p[x, -1], \text{ with } x, y = 0..nS-1 \quad (8-242430)$$

8.3.3.1.48.3.3.1.4 Specification of Intra_Horizontal prediction mode

Inputs to this process are:

- neighbouring samples $p[x, y]$, with $x, y = -1..2*nS-1$,
- a variable nS specifying the prediction size.

Output of this process is:

- predicted samples $predSamples[x, y]$, with $x, y = 0..nS-1$.

This intra prediction mode is invoked when $intraPredMode$ is equal to 1.

The values of the prediction samples $predSamples[x, y]$, with $x, y = 0..nS-1$, are derived by

$$predSamples[x, y] = p[-1, y], \text{ with } x, y = 0..nS-1 \quad (8-252534)$$

8.3.3.1.48.3.3.1.5 Specification of Intra_DC prediction mode

Inputs to this process are:

- neighbouring samples $p[x, y]$, with $x, y = -1..2*nS-1$,
- a variable nS specifying the prediction block size,
- ~~the bit depth of the chroma component, $bitDepth$~~

Output of this process is:

- predicted samples $predSamples[x, y]$, with $x, y = 0..nS-1$.

This intra prediction mode is invoked when $intraPredMode$ is equal to 2.

The values of the prediction samples $predSamples[x, y]$, with $x, y = 0..nS-1$, are derived as follows: the following ordered steps:

~~If all samples $p[x, -1]$, with $x=0..nS-1$ and $p[-1, y]$, with $y=0..nS-1$ are marked as "available for intra prediction," the values of the prediction samples $predSamples[x, y]$, with $x, y = 0..nS-1$ are derived as the following ordered steps:~~

1. A variable $DCVal$ is derived as:

by

$$predSamples[x, y]DCVal = \left(\sum_{x'=0}^{nS-1} p[x', -1] + \sum_{y'=0}^{nS-1} p[-1, y'] + nS \right) \gg (k+1), \text{ with } x, y = 0..nS-1 \quad (8-262632)$$

where $k = \log_2(nS)$

Formatted: Font: 10 pt

HEVC

2. Depending on the prediction block size nS, the following applies.

– If DCPredFilterFlag is equal to 1 and nS is less than 32, the following applies.

– If nS is equal to 4, the prediction samples predSamples[x, y] are derived as

$$\begin{aligned} \text{predSamples}[0, 0] &= (3 * p[-1, 0] + 2 * DCVal + 3 * p[0, -1] + 4) \gg 3 && (8-272732) \\ \text{predSamples}[x, 0] &= (3 * p[x, -1] + 5 * DCVal + 4) \gg 3, \text{ with } x = 1..nS-1 && (8-282832) \\ \text{predSamples}[0, y] &= (3 * p[-1, y] + 5 * DCVal + 4) \gg 3, \text{ with } y = 1..nS-1 && (8-292932) \\ \text{predSamples}[x, y] &= DCVal, \text{ with } x, y = 1..nS-1 && (8-303032) \end{aligned}$$

– Otherwise, if nS is equal to 8, the prediction samples predSamples[x, y] are derived as

$$\begin{aligned} \text{predSamples}[0, 0] &= (1 * p[-1, 0] + 2 * DCVal + 1 * p[0, -1] + 2) \gg 2 && (8-313132) \\ \text{predSamples}[x, 0] &= (1 * p[x, -1] + 3 * DCVal + 2) \gg 2, \text{ with } x = 1..nS-1 && (8-32) \\ \text{predSamples}[0, y] &= (1 * p[-1, y] + 3 * DCVal + 2) \gg 2, \text{ with } y = 1..nS-1 && (8-333332) \\ \text{predSamples}[x, y] &= DCVal, \text{ with } x, y = 1..nS-1 && (8-343432) \end{aligned}$$

– Otherwise, if nS is equal to 16, the prediction samples predSamples[x, y] are derived as

$$\begin{aligned} \text{predSamples}[0, 0] &= (1 * p[-1, 0] + 6 * DCVal + 1 * p[0, -1] + 4) \gg 3 && (8-353532) \\ \text{predSamples}[x, 0] &= (1 * p[x, -1] + 7 * DCVal + 4) \gg 3, \text{ with } x = 1..nS-1 && (8-363632) \\ \text{predSamples}[0, y] &= (1 * p[-1, y] + 7 * DCVal + 4) \gg 3, \text{ with } y = 1..nS-1 && (8-373732) \\ \text{predSamples}[x, y] &= DCVal, \text{ with } x, y = 1..nS-1 && (8-383832) \end{aligned}$$

– Otherwise, the prediction samples predSamples[x, y] are derived as

$$\text{predSamples}[x, y] = DCVal, \text{ with } x, y = 0..nS-1 \quad (8-393932)$$

– Otherwise, If all samples p[x, -1], with x=0..nS-1 are marked as “not available for intra prediction” and all samples p[-1, y], with y=0..nS-1 are marked as “available for intra prediction,” the values of the prediction samples predSamples[x, y], with x, y=0..nS-1 are derived by

$$\text{predSamples}[x, y] = \left(\sum_{y'=0}^{nS-1} p[-1, y'] + (nS \gg 1) \right) \gg k, \text{ with } x, y = 0..nS-1 \quad (8-33)$$

where $k = \log_2(nS)$

– Otherwise, If all samples p[x, -1], with x=0..nS-1 are marked as “available for intra prediction” and all samples p[-1, y], with y=0..nS-1 are marked as “not available for intra prediction,” the values of the prediction samples predSamples[x, y], with x, y=0..nS-1 are derived by

$$\text{predSamples}[x, y] = \left(\sum_{x'=0}^{nS-1} p[x', -1] + (nS \gg 1) \right) \gg k, \text{ with } x, y = 0..nS-1 \quad (8-34)$$

where $k = \log_2(nS)$

– Otherwise, the values of the prediction samples predSamples[x, y], with x, y=0..nS-1 are derived by

$$\text{predSamples}[x, y] = 1 \ll \left((\text{bitDepth} - \text{BitDepth}_y + \text{increased_bit_depth_luma}) - 11 \right) \quad (8-35)$$

[Ed: (WJ) current software implementation seems different. It uses division operation and may have a bug when both above and left samples are not available]

8.3.3.1.58.3.3.1.6 Specification of Intra_Angular prediction mode

[Ed: (WJ) provided text from JCTVC-C042 was modified. It still needs to be improved]

Inputs to this process are:

- neighbouring samples p[x, y], with x, y = -1..2*nS-1,
- a variable nS specifying the prediction size.

Formatted: Normal, Indent: Left: 0.21", Numbered + Level: 1 + Numbering Style: 1, 2, 3, ... + Start at: 1 + Alignment: Left + Aligned at: 0" + Tab after: 0.28" + Indent at: 0.28", Tab stops: 0.5", Left + 0.75", Left + 1", Left + 1.18", Left + 1.38", List tab + Not at 0.28" + 0.59" + 0.79" + 0.98"

Formatted: English (United Kingdom)

Formatted: Indent: Left: 0.98", Tab stops: Not at 0.59" + 0.79"

Formatted: Font: 10 pt, English (United Kingdom)

Formatted: English (United Kingdom)

Formatted: English (United Kingdom)

Formatted: Normal, Indent: Left: 0.73", Bulleted + Level: 1 + Aligned at: 0.49" + Indent at: 0.74", Tab stops: 0.5", Left + 1", Left + 1.18", Left + Not at 0.59" + 0.79" + 0.98"

Formatted ... [1]

Formatted: Indent: Left: 0.98", Tab stops: Not at 0.59" + 0.79"

Formatted: English (United Kingdom)

Formatted

... [2]

Formatted: Normal, Indent: Left: 0.73", Bulleted + Level: 1 + Aligned at: 0.49" + Indent at: 0.74", Tab stops: 0.5", Left + 1", Left + 1.18", Left + Not at 0.59" + 0.79" + 0.98"

Formatted ... [3]

Formatted: Indent: Left: 0.98", Tab stops: Not at 0.59" + 0.79"

Formatted: English (United Kingdom)

Formatted: Normal, Bulleted + Level: 1 + Aligned at: 0.49" + Indent at: 0.74", Tab stops: 0.5", Left + 0.75", Left + 1", Left + 1.18", Left + Not at 0.59" + 0.79" + 0.98"

Formatted: Normal, Indent: Left: 0", Tab stops: 0.5", Left + 0.75", Left + 1", Left + 1.18", Left + Not at 0.59" + 0.79" + 0.98"

Formatted: Indent: Left: 0.59"

Output of this process is:

- predicted samples $\text{predSamples}[x, y]$, with $x, y = 0..nS-1$.

This intra prediction mode is invoked when intraPredMode is [in the range of 3..33 not equal to 0, 1, and 2](#).

[Table 8-7](#)[Table 8-6](#) specifies the mapping table between intraPredMode and the rearranged intra prediction order intraPredOrder .

Table 8-7[Table 8-6](#) – Specification of intraPredOrder

intraPredMode	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
intraPredOrder	-	-	-	1	5	13	17	21	29	33	3	7	11	15	19	23	27
intraPredMode	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
intraPredOrder	31	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32

[Figure 8-2](#)[Figure 8-2](#) illustrates the total 33 intra angles and [Table 8-8](#)[Table 8-7](#) specifies the mapping table between intraPredOrder and the angle parameter intraPredAngle .

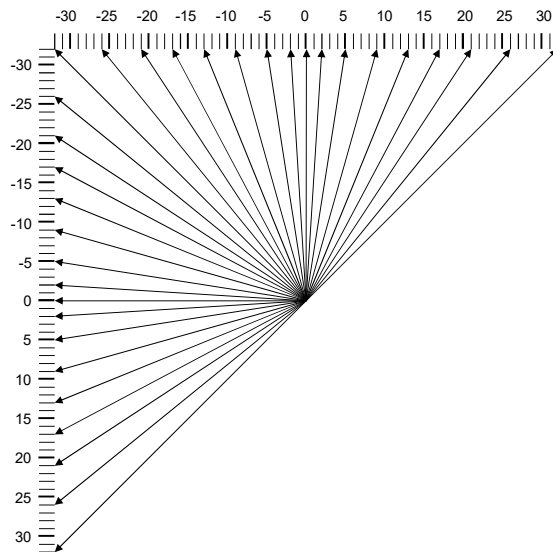


Figure 8-2 – Intra prediction angle definition (informative)

Table 8-8[Table 8-7](#) – Specification of intraPredAngle

intraPredOrder	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
intraPredAngle	-	-32	-26	-21	-17	-13	-9	-5	-2	-	2	5	9	13	17	21	26
intraPredOrder	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
intraPredAngle	32	-26	-21	-17	-13	-9	-5	-2	-	2	5	9	13	17	21	26	32

HEVC

Table 8-9 further specifies the mapping table between `intraPredOrder` and the inverse angle parameter `invAngle`.

Table 8-9 – Specification of `invAngle`

intraPredOrder	1	2	3	4	5	6	7	8
invAngle	-256	-315	-390	-482	-630	-910	-1638	-4096
intraPredOrder	18	19	20	21	22	23	24	25
invAngle	-315	-390	-482	-630	-910	-1638	-4096	-

The reference pixel array `refMain[x]`, with $x = -nS..2*nS$ is specified as follows.

- If `intraPredOrder` is less than 18,

$$\text{refMain}[x] = p[-1+x, -1], \text{ with } x=0..nS \quad (8-404036)$$

- If `intraPredAngle` is less than 0,

$$\text{refMain}[x] = p[-1, -1+((x*\text{invAngle}+128) >> 8)], \text{ with } x=(nS*\text{intraPredAngle}) >> 5..-1 \quad (8-414137)$$

- Otherwise,

$$\text{refMain}[x] = p[-1+x, -1], \text{ with } x=nS+1..2*nS \quad (8-424238)$$

Otherwise,

$$\text{refMain}[x] = p[-1, -1+x], \text{ with } x=0..nS \quad (8-434339)$$

- If `intraPredAngle` is less than 0,

$$\text{refMain}[x] = p[-1+((x*\text{invAngle}+128) >> 8), -1], \text{ with } x=(nS*\text{intraPredAngle}) >> 5..-1 \quad (8-444440)$$

- Otherwise,

$$\text{refMain}[x] = p[-1, -1+x], \text{ with } x=nS+1..2*nS \quad (8-454541)$$

The values of the prediction samples `predSamples[x, y]`, with $x, y = 0..nS-1$ are derived by the following procedures.

- The index variable `iIdx` and the multiplication factor `iFact` are derived by

$$\text{iIdx} = ((y + 1) * \text{intraPredAngle}) >> 5 \quad (8-464642)$$

$$\text{iFact} = ((y + 1) * \text{intraPredAngle}) \&\& 31 \quad (8-474743)$$

- Depending on the value of `iFact`, the following applies.

- If `iFact` is not equal to 0, the value of the prediction samples `predSamples[x, y]` is derived by

$$\text{predSamples}[x, y] = ((32 - \text{iFact}) * \text{refMain}[x+\text{iIdx}+1] + \text{iFact} * \text{refMain}[x+\text{iIdx}+2] + 16) >> 5 \quad (8-484844)$$

- Otherwise, the value of the prediction samples `predSamples[x, y]` is derived by

$$\text{predSamples}[x, y] = \text{refMain}[x+\text{iIdx}+1] \quad (8-494945)$$

8.3.3.1.7 Specification of Intra Planar prediction mode

Inputs to this process are:

- neighbouring samples `p[x, y]`, with $x, y = -1..2*nS-1$,
- a variable `nS` specifying the prediction size,
- the bit-depth of the chroma component, `bitDepth`.

Formatted: Heading 5, Indent: Left: 0", Tab stops: Not at 0.59" + 0.79" + 0.98"

HEVC

Output of this process is:

– predicted samples $\text{predSamples}[x, y]$, with $x, y = 0..nS-1$.

This intra prediction mode is invoked when intraPredMode is equal to 34.

The $\text{seedHorp}[y]$ and $\text{seedVerp}[x]$ arrays with $x, y = 0..nS-1$ are initialized as follows:

– If all samples $p[x, -1]$, with $x = 0..nS-1$ and $p[-1, y]$, with $y = 0..nS-1$ are marked as “available for intra prediction,” the initialization is done as $\text{seedHorp}[y] = p[-1, y]$ and $\text{seedVerp}[x] = p[x, -1]$.

– Otherwise, if all samples $p[x, -1]$, with $x = 0..nS-1$ are marked as “not available for intra prediction” and all samples $p[-1, y]$, with $y = 0..nS-1$ are marked as “available for intra prediction,” the initialization is done as $\text{seedHorp}[y] = p[-1, y]$ and $\text{seedVerp}[x] = p[-1, 0]$.

– Otherwise, if all samples $p[x, -1]$, with $x = 0..nS-1$ are marked as “available for intra prediction” and all samples $p[-1, y]$, with $y = 0..nS-1$ are marked as “not available for intra prediction,” the initialization is done as $\text{seedHorp}[y] = p[0, -1]$ and $\text{seedVerp}[x] = p[x, -1]$.

– Otherwise, the initialization is done as $\text{seedHorp}[y] = 1 \ll (\text{bitDepth} - 1)$ and $\text{seedVerp}[x] = 1 \ll (\text{bitDepth} - 1)$.

The values of the prediction samples $\text{predSamples}[x, y]$, with $x, y = 0..nS-1$, are derived by

$$\text{predSamples}[x, y] = \left(\frac{(nS-1-x) * \text{seedHorp}[-1, y] + (x+1) * \text{pseedVerp}[nS-1, -1] + (nS-1-y) * \text{seedVerp}[x, -1] + (y+1) * \text{seedHorp}[-1, nS-1] + nS \right) \gg (k+1) \quad (8-505045)$$

with $x, y = 0..nS-1$ where $k = \log_2(nS)$

8.3.3.1.8 Specification of Intra FromLuma prediction mode

Inputs to this process are:

– a sample location (x_B, y_B) specifying the top-left sample of the current block relative to the top-left sample of the current picture,

– neighbouring samples $p[x, y]$, with $x, y = -1..2*nS-1$,

– a variable nS specifying the prediction size.

Output of this process is:

– predicted samples $\text{predSamples}[x, y]$, with $x, y = 0..nS-1$.

This intra prediction mode is invoked when intraPredMode is equal to 345.

The values of the prediction samples $\text{predSamples}[x, y]$, with $x, y = 0..nS-1$, are derived as the following ordered steps:

1. Variable $k3$ and the sample array p_y' are derived as:

$$k3 = \text{Max}(0, \text{BitDepth}_c + \log_2(nS) - 14) \quad (8-515130)$$

$$p_y'[x, y] = (\text{recSamples}_l[2x, 2y] + \text{recSamples}_l[2x, 2y+1]) \gg 1, \text{ with } x, y = -1..2*nS \quad (8-525230)$$

2. Variables L, C, LL, LC and $k2$ are derived as follows:

– If all samples $p[x, -1]$, with $x = 0..nS-1$ and $p[-1, y]$, with $y = 0..nS-1$ are marked as “available for intra prediction” the variables L, C, LL and LC are derived as:

$$L = \left(\sum_{y=0}^{nS-1} p_y'[-1, y] + \sum_{x=0}^{nS-1} p_y'[x, -1] \right) \gg k3 \quad (8-535330)$$

$$C = \left(\sum_{y=0}^{nS-1} p[-1, y] + \sum_{x=0}^{nS-1} p[x, -1] \right) \gg k3 \quad (8-545430)$$

$$LL = \left(\sum_{y=0}^{nS-1} p_y'[-1, y]^2 + \sum_{x=0}^{nS-1} p_y'[x, -1]^2 \right) \gg k3 \quad (8-555530)$$

Formatted: Tab stops: Not at 0.2"

Formatted: Subscript

Formatted: Heading 5, Indent: Left: 0", Tab stops: Not at 0.59" + 0.79" + 0.98"

Formatted: Subscript

Formatted: Indent: Left: 0.21", Numbered + Level: 1 + Numbering Style: 1, 2, 3, ... + Start at: 1 + Alignment: Left + Aligned at: 0" + Tab after: 0.28" + Indent at: 0.28", Tab stops: 0.5", Left + 0.75", Left + 1", Left + 1.18", Left + Not at 0.55" + 0.83" + 1.1" + 1.38"

Formatted: Subscript

Formatted: Subscript

Formatted: Subscript

Formatted: Subscript

Formatted: Subscript

Formatted: Indent: Left: 0.21", Numbered + Level: 1 + Numbering Style: 1, 2, 3, ... + Start at: 1 + Alignment: Left + Aligned at: 0" + Tab after: 0.28" + Indent at: 0.28", Tab stops: 0.5", Left + 0.75", Left + 1", Left + 1.18", Left + Not at 0.55" + 0.83" + 1.1" + 1.38"

Formatted: Indent: Left: 0.51", Bulleted + Level: 1 + Aligned at: 0" + Tab after: 0.28" + Indent at: 0.28", Tab stops: 0.79", List tab + 1", Left + 2.07", Left + Not at 0.28" + 0.55" + 0.83" + 1.1" + 1.38"

Formatted: Font: 10 pt

Formatted: Font: 10 pt

Formatted: Font: 10 pt

HEVC

$$\underline{LC} = \left(\sum_{y=0}^{nS-1} p_y'[-1, y] * p[-1, y] + \sum_{y=0}^{nS-1} p_y'[x, -1] * p[x, -1] \right) \gg k3 \quad (8-565630)$$

Formatted: Font: 10 pt

$$\underline{k2} = \log_2(2 * nS) \gg k3 \quad (8-575730)$$

Formatted: Font: 10 pt

Otherwise, if all samples $p[x, -1]$, with $x = 0..nS-1$ are marked as "not available for intra prediction" and all samples $p[-1, y]$, with $y = 0..nS-1$ are marked as "available for intra prediction" the variables L , C , LL and LC are derived as:

Formatted: Font: 10 pt, Subscript

Formatted: Font: 10 pt

Formatted: Font: 10 pt

Formatted: Font: 10 pt

$$\underline{L} = \left(\sum_{y=0}^{nS-1} p_y'[-1, y] \right) \gg k3 \quad (8-30)$$

Formatted: Normal, Indent: Left: 0.51", Bulleted + Level: 1 + Aligned at: 0" + Tab after: 0.28" + Indent at: 0.28", Tab stops: 0.79", List tab + 1", Left + 2.07", Left + Not at 0.28" + 0.59" + 0.98"

$$\underline{C} = \left(\sum_{y=0}^{nS-1} p_c[-1, y] \right) \gg k3 \quad (8-30)$$

$$\underline{LL} = \left(\sum_{y=0}^{nS-1} p_y'[-1, y]^2 \right) \gg k3 \quad (8-30)$$

$$\underline{LC} = \left(\sum_{y=0}^{nS-1} p_y'[-1, y] * p_c[-1, y] \right) \gg k3 \quad (8-30)$$

$$\underline{k2} = \log_2(nS) \gg k3 \quad (8-30)$$

Otherwise, if all samples $p[x, -1]$, with $x = 0..nS-1$ are marked as "available for intra prediction" and all samples $p[-1, y]$, with $y = 0..nS-1$ are marked as "not available for intra prediction" the variables L , C , LL and LC are derived as:

Formatted: Normal, Indent: Left: 0.51", Bulleted + Level: 1 + Aligned at: 0" + Tab after: 0.28" + Indent at: 0.28", Tab stops: 0.79", List tab + 1", Left + 2.07", Left + Not at 0.28" + 0.59" + 0.98"

$$\underline{L} = \left(\sum_{x=0}^{nS-1} p_y'[x, -1] \right) \gg k3 \quad (8-30)$$

$$\underline{C} = \left(\sum_{x=0}^{nS-1} p_c[x, -1] \right) \gg k3 \quad (8-30)$$

$$\underline{LL} = \left(\sum_{x=0}^{nS-1} p_y'[x, -1]^2 \right) \gg k3 \quad (8-30)$$

$$\underline{LC} = \left(\sum_{x=0}^{nS-1} p_y'[x, -1] * p_c[x, -1] \right) \gg k3 \quad (8-30)$$

$$\underline{k2} = \log_2(nS) \gg k3 \quad (8-30)$$

3. Variables a and b are derived as:

$$a1 = (LC \ll k2) - L * C \quad (8-585830)$$

$$a2 = (LL \ll k2) - L * L \quad (8-595930)$$

$$k = \text{Max}(0, \log_2(\text{abs}(a2)) - 5) - \text{Max}(0, \log_2(\text{abs}(a1)) - 14) + 2 \quad (8-606030)$$

$$a1s = a1 \gg \text{Max}(0, \log_2(\text{abs}(a1)) - 14) \quad (8-616130)$$

$$a2s = \text{abs}(a2 \gg \text{Max}(0, \log_2(\text{abs}(a2)) - 5)) + 1 \quad (8-626230)$$

Formatted: Indent: Left: 0.21", Numbered + Level: 1 + Numbering Style: 1, 2, 3, ... + Start at: 1 + Alignment: Left + Aligned at: 0" + Tab after: 0.28" + Indent at: 0.28", Tab stops: 0.5", Left + 0.75", Left + 1", Left + 1.18", Left + Not at 0.55" + 0.83" + 1.1" + 1.38"

Formatted: Equation, Indent: Left: 0.59", Tab stops: 0.59", Left + 0.79", Left + 0.98", Left

Formatted: Font: 10 pt, Subscript

Formatted: Font: 10 pt, Subscript

Formatted: Font: 10 pt, Subscript

Formatted: Font: 10 pt, Subscript

[Ed. (WJ): bug in software – a2s should be $\text{abs}(a2 \gg \text{Max}(0, \log_2(\text{abs}(a2)) - 5))$ – without '+1']

Formatted: Font: 10 pt, English (United States)

HEVC

$$a = \text{Clip3}(-2^{15}, 2^{15}-1, a1s * \text{ImDiv} + (1 \ll (k1 - 1)) \gg k1) \quad (8-636330)$$

$$b = (L - ((a * C) \gg k1) + (1 \ll (k2 - 1))) \gg k2 \quad (8-646430)$$

where ImDiv is specified in Table 8-10 with the input a2s.

4. The values of the prediction samples predSamples[x, y] are derived as:

$$\text{predSamples}[x, y] = \text{Clip1}_c(((pv'[x, y] * a) \gg 13) + b), \text{ with } x, y = 0..nS-1 \quad (8-656530)$$

Table 8-10 – Specification of ImDiv

a2s	1	2	3	4	5	6	7	8	9	10	11	12	13
ImDiv	32768	16384	10923	8192	6554	5461	4681	4096	3641	3277	2979	2731	2521
a2s	14	15	16	17	18	19	20	21	22	23	24	25	26
ImDiv	2341	2185	2048	1928	1820	1725	1638	1560	1489	1425	1365	1311	1260
a2s	27	28	29	30	31	32	33	34	35	36	37	38	39
ImDiv	1214	1170	1130	1092	1057	1024	993	964	936	910	886	862	840
a2s	40	41	42	43	44	45	46	47	48	49	50	51	52
ImDiv	819	799	780	762	745	728	712	697	683	669	655	643	630
a2s	53	54	55	56	57	58	59	60	61	62	63	64	
ImDiv	618	607	596	585	575	565	555	546	537	529	520	512	

- Formatted: Font: 10 pt, Superscript
- Formatted: Font: 10 pt, Superscript
- Formatted: Font: 10 pt
- Formatted: Indent: Left: 0.21", Numbered + Level: 1 + Numbering Style: 1, 2, 3, ... + Start at: 1 + Alignment: Left + Aligned at: 0" + Tab after: 0.28" + Indent at: 0.28", Tab stops: 0.5", Left + 0.75", Left + 1", Left + 1.18", Left + Not at 0.55" + 0.83" + 1.1" + 1.38"
- Formatted: Subscript
- Formatted: Subscript
- Formatted: Normal, Indent: Left: 0", Tab stops: Not at 0.59" + 0.79" + 0.98"
- Formatted: Caption

- Formatted: (Asian) Korean
- Formatted: Normal, Indent: Left: 0", Tab stops: Not at 0.59" + 0.79" + 0.98"
- Formatted: (Asian) Korean, (Other) (none)
- Formatted: Normal, Indent: Left: 0", Tab stops: Not at 0.59" + 0.79" + 0.98"

- Formatted: Indent: Left: 0.21", Numbered + Level: 1 + Numbering Style: 1, 2, 3, ... + Start at: 1 + Alignment: Left + Aligned at: 0" + Tab after: 0.28" + Indent at: 0.28"

8.4 Decoding process for coding units coded in inter prediction mode

Inputs to this process are:

- a luma location (xC, yC) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture,
- a variable log2CUSize specifying the size of the current coding unit.

Output of this process is a modified reconstructed picture before deblocking filtering.

The variable nCS_L is set equal to 1 << log2CUSize and the variable nCS_C is set equal to (1 << log2CUSize) >> 1.

Decoding process for coding units coded in inter prediction mode consists of following ordered steps:

1. The inter prediction process as specified in subclause 8.4.1 is invoked with the luma location (xC, yC), coding unit size log2CUSize, inter prediction mode **PredMode**, and prediction partition mode **PartMode** as the inputs and the outputs are 3 arrays predSamples_L, predSamples_{Cb}, predSamples_{Cr}.
2. The decoding process for the residual signal of coding units coded in inter prediction mode specified in subclause 8.4.3 is invoked with the luma location (xC, yC), the size of the current coding unit log2CUSize as inputs and the outputs are 3 arrays resSamples_L, resSamples_{Cb}, resSamples_{Cr}.
3. The residual signal accumulation process as specified in subclause **XXXX** is invoked with arrays predSamples_L, predSamples_{Cb}, predSamples_{Cr}, and the arrays resSamples_L, resSamples_{Cb}, resSamples_{Cr} as the inputs and the outputs are 3 arrays recSamples_L, recSamples_{Cb}, recSamples_{Cr}. [Ed: (WJ) it may be split per each color component. Revisit after writing residual signal accumulation process]
4. The picture reconstruction process for a component before deblocking filtering as specified in subclause **XXXX** is invoked with the luma location (xB, yB), and 3 arrays recSamples_L, recSamples_{Cb}, recSamples_{Cr} as the inputs and the output is a modified reconstructed picture before deblocking filtering. [Ed: (WJ) it may be split per each color component. Revisit after writing picture reconstruction process]

HEVC

8.4.1 Inter prediction process

This process is invoked when decoding coding unit whose PredMode is not equal to MODE_INTRA.

Inputs to this process are:

- a luma location (x_C, y_C) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture,
- a variable $\log_2\text{CUSize}$ specifying the size of the current coding unit,
- a variable PredMode specifying prediction mode of current coding unit,
- a variable PartMode specifying prediction partition mode of current coding unit.

Outputs of this process are:

- a $(n\text{CS}_L) \times (n\text{CS}_L)$ array predSamples_L of luma prediction samples, where $n\text{CS}_L$ is derived as specified below,
- a $(n\text{CS}_C) \times (n\text{CS}_C)$ array predSamples_{Cb} of chroma prediction samples for the component Cb, where $n\text{CS}_C$ is derived as specified below,
- a $(n\text{CS}_C) \times (n\text{CS}_C)$ array predSamples_{Cr} of chroma prediction samples for the component Cr, where $n\text{CS}_C$ is derived as specified below.

The variable $n\text{CS}_L$ is set equal to $1 \ll \log_2\text{CUSize}$ and the variable $n\text{CS}_C$ is set equal to $(1 \ll \log_2\text{CUSize}) \gg 1$. [Ed: (WJ) revisit for supporting other chroma formats]

The variable $n\text{CS}_{1L}$ is set equal to $n\text{CS}_L \gg 1$.

Depending on PartMode, the following applies: [Ed: (WJ) is PUType better?]

- If PartMode is equal to PART_2Nx2N, the following ordered steps apply:
 1. The decoding process for prediction units in inter prediction mode as specified in subclause 8.4.2 is invoked with the luma location (x_C, y_C), the luma location (x_B, y_B) set equal to (0, 0), the size of the coding unit $n\text{CS}_L$, the width of the luma prediction samples $n\text{PSW}$ set equal to $n\text{CS}_L$, the height of the luma prediction samples $n\text{PSH}$ set equal to $n\text{CS}_L$ and a partition index PartIdx set equal to 0 as inputs, and the outputs are a $(n\text{CS}_L) \times (n\text{CS}_L)$ array predSamples_L and two $(n\text{CS}_C) \times (n\text{CS}_C)$ arrays predSamples_{Cb} and predSamples_{Cr} .
- Otherwise, if PartMode is equal to PART_2NxN, the following ordered steps apply:
 1. The decoding process for prediction units in inter prediction mode as specified in subclause 8.4.2 is invoked with the luma location (x_C, y_C), the luma location (x_B, y_B) set equal to (0, 0), the size of the coding unit $n\text{CS}_L$, the width of the luma prediction samples $n\text{PSW}$ set equal to $n\text{CS}_L$, the height of the luma prediction samples $n\text{PSH}$ set equal to $n\text{CS}_{1L}$ and a partition index PartIdx set equal to 0 as inputs, and the outputs are a $(n\text{CS}_L) \times (n\text{CS}_L)$ array predSamples_L and two $(n\text{CS}_C) \times (n\text{CS}_C)$ arrays predSamples_{Cb} and predSamples_{Cr} .
 2. The decoding process for prediction units in inter prediction mode as specified in subclause 8.4.2 is invoked with the luma location (x_C, y_C), the luma location (x_B, y_B) set equal to (0, $n\text{CS}_{1L}$), the size of the coding unit $n\text{CS}_L$, the width of the luma prediction samples $n\text{PSW}$ set equal to $n\text{CS}_L$, the height of the luma prediction samples $n\text{PSH}$ set equal to $n\text{CS}_{1L}$ and a partition index PartIdx set equal to 1 as inputs, and the outputs are a $(n\text{CS}_L) \times (n\text{CS}_L)$ array predSamples_L and two $(n\text{CS}_C) \times (n\text{CS}_C)$ arrays predSamples_{Cb} and predSamples_{Cr} .

Otherwise, if PartMode is equal to PART_Nx2N, the following ordered steps apply:

1. The decoding process for prediction units in inter prediction mode as specified in subclause 8.4.2 is invoked with the luma location (x_C, y_C), the luma location (x_B, y_B) set equal to (0, 0), the size of the coding unit $n\text{CS}_L$, the width of the luma prediction samples $n\text{PSW}$ set equal to $n\text{CS}_{1L}$, the height of the luma prediction samples $n\text{PSH}$ set equal to $n\text{CS}_L$ and a partition index PartIdx set equal to 0 as inputs, and the outputs are a $(n\text{CS}_L) \times (n\text{CS}_L)$ array predSamples_L and two $(n\text{CS}_C) \times (n\text{CS}_C)$ arrays predSamples_{Cb} and predSamples_{Cr} .
2. The decoding process for prediction units in inter prediction mode as specified in subclause 8.4.2 is invoked with the luma location (x_C, y_C), the luma location (x_B, y_B) set equal to ($n\text{CS}_{1L}, 0$), the size of the coding unit $n\text{CS}_L$, the width of the luma prediction samples $n\text{PSW}$ set equal to $n\text{CS}_{1L}$, the height of the luma prediction samples $n\text{PSH}$ set equal to $n\text{CS}_L$ and a partition index PartIdx set equal to 1 as inputs, and the outputs are a $(n\text{CS}_L) \times (n\text{CS}_L)$ array predSamples_L and two $(n\text{CS}_C) \times (n\text{CS}_C)$ arrays predSamples_{Cb} and predSamples_{Cr} .

Otherwise, if PartMode is equal to PART_NxN, the following ordered steps apply:

1. The decoding process for prediction units in inter prediction mode as specified in subclause 8.4.2 is invoked with the luma location (x_C, y_C), the luma location (x_B, y_B) set equal to (0, 0), the size of the coding unit $n\text{CS}_L$, the width of the luma prediction samples $n\text{PSW}$ set equal to $n\text{CS}_{1L}$, the height of the luma prediction

HEVC

samples $nPSH$ set equal to nCS_{1L} as and a partition index $PartIdx$ set equal to 0 inputs, and the outputs are a $(nCS_L) \times (nCS_L)$ array $predSamples_L$ and two $(nCS_C) \times (nCS_C)$ arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.

2. The decoding process for prediction units in inter prediction mode as specified in subclause 8.4.2 is invoked with the luma location (x_C, y_C) , the luma location (x_B, y_B) set equal to $(nCS_{1L}, 0)$, the size of the coding unit nCS_L , the width of the luma prediction samples $nPSW$ set equal to nCS_{1L} , the height of the luma prediction samples $nPSH$ set equal to nCS_{1L} and a partition index $PartIdx$ set equal to 1 as inputs, and the outputs are a $(nCS_L) \times (nCS_L)$ array $predSamples_L$ and two $(nCS_C) \times (nCS_C)$ arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.
3. The decoding process for prediction units in inter prediction mode as specified in subclause 8.4.2 is invoked with the luma location (x_C, y_C) , the luma location (x_B, y_B) set equal to $(0, nCS_{1L})$, the size of the coding unit nCS_L , the width of the luma prediction samples $nPSW$ set equal to nCS_{1L} , the height of the luma prediction samples $nPSH$ set equal to nCS_{1L} and a partition index $PartIdx$ set equal to 2 as inputs, and the outputs are a $(nCS_L) \times (nCS_L)$ array $predSamples_L$ and two $(nCS_C) \times (nCS_C)$ arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.
4. The decoding process for inter prediction units as specified in subclause 8.4.2 is invoked with the luma location (x_C, y_C) , the luma location (x_B, y_B) set equal to (nCS_{1L}, nCS_{1L}) , the size of the coding unit nCS_L , the width of the luma prediction samples $nPSW$ set equal to nCS_{1L} , the height of the luma prediction samples $nPSH$ set equal to nCS_{1L} and a partition index $PartIdx$ set equal to 3 as inputs, and the outputs are a $(nCS_L) \times (nCS_L)$ array $predSamples_L$ and two $(nCS_C) \times (nCS_C)$ arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.

8.4.2 Decoding process for prediction units in inter prediction mode

Inputs to this process are:

- a luma location (x_C, y_C) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture,
- a luma location (x_B, y_B) specifying the top-left luma sample of the current prediction unit relative to the top left luma sample of the current coding unit,
- a variable nCS specifying the size of the current coding unit,
- a variable $nPSW$ specifying the width of the current prediction unit,
- a variable $nPSH$ specifying the height of the current prediction unit,
- a variable $PartIdx$ specifying the index of the current prediction unit within the current coding unit.

Outputs of this process are:

- a $(nCS_L) \times (nCS_L)$ array $predSamples_L$ of luma prediction samples, where nCS_L is derived as specified below,
- a $(nCS_C) \times (nCS_C)$ array $predSamples_{Cb}$ of chroma prediction samples for the component C_b , where nCS_C is derived as specified below,
- a $(nCS_C) \times (nCS_C)$ array $predSamples_{Cr}$ of chroma prediction samples for the component C_r , where nCS_C is derived as specified below.

The variable nCS_L is set equal to nCS and the variable nCS_C is set equal to $nCS \gg 1$. [Ed: (WJ) revisit for supporting other chroma formats]

The decoding process for prediction units in inter prediction mode consists of the following ordered steps:

1. Derivation process for motion vector components and reference indices as specified in subclause 8.4.2.1.

Inputs to this process are

- a luma location (x_C, y_C) of the top-left luma sample of the current coding unit relative to the top-left luma sample of the current picture,
- a luma location (x_B, y_B) of the top-left luma sample of the current prediction unit relative to the top-left luma sample of the current coding unit,
- variables specifying the width and the height of the prediction unit for luma, $nPSW$ and $nPSH$,
- a variable $PartIdx$ specifying the index of the current prediction unit within the current coding unit.

Outputs of this process are

- luma motion vectors $mvL0$ and $mvL1$, and chroma motion vectors $mvCL0$ and $mvCL1$,
- reference indices $refIdxL0$ and $refIdxL1$,

HEVC

- prediction list utilization flags predFlagL0 and predFlagL1.

2. Decoding process for inter sample prediction as specified in subclause 8.4.2.2.

Inputs to this process are

- a luma location (xC, yC) of the top-left luma sample of the current coding unit relative to the top-left luma sample of the current picture,
- a luma location (xB, yB) of the top-left luma sample of the current prediction unit relative to the top-left luma sample of the current coding unit,
- a variable nCS specifying the size of the current coding unit,
- variables specifying the width and the height of the prediction unit for luma, nPSW and nPSH.
- luma motion vectors mvL0 and mvL1, and chroma motion vectors mvCL0 and mvCL1,
- reference indices refldxL0 and refldxL1,
- prediction list utilization flags predFlagL0 and predFlagL1.

Outputs of this process are

- inter prediction samples (predSamples); which are a (nCS_L)x(nCS_L) array predSamples_L of prediction luma samples and two (nCS_C)x(nCS_C) arrays predSamples_{Cb} and predSamples_{Cr} of prediction chroma samples, one for each of the chroma components Cb and Cr.

For use in derivation processes of variables invoked later in the decoding process, the following assignments are made:

MvL0[xB, yB] = mvL0	(8-666646)
MvL1[xB, yB] = mvL1	(8-676747)
ReflDxL0[xB, yB] = reflDxL0	(8-686848)
ReflDxL1[xB, yB] = reflDxL1	(8-696949)
PredFlagL0[xB, yB] = predFlagL0	(8-707050)
PredFlagL1[xB, yB] = predFlagL1	(8-717151)

8.4.2.1 Derivation process for motion vector components and reference indices

Input to this process are

- a luma location (xC, yC) of the top-left luma sample of the current coding unit relative to the top-left luma sample of the current picture,
- a luma location (xB, yB) of the top-left luma sample of the current prediction unit relative to the top-left luma sample of the current coding unit,
- variables specifying the width and the height of the prediction unit for luma, nPSW and nPSH,
- a variable PartIdx specifying the index of the current prediction unit within the current coding unit.

Outputs of this process are

- luma motion vectors mvL0 and mvL1 and chroma motion vectors mvCL0 and mvCL1,
- reference indices refldxL0 and refldxL1,
- prediction list utilization flags predFlagL0 and predFlagL1.

Let (xP, yP) specify the top-left luma sample of the current prediction unit relative to the top-left luma sample of the current picture where xP = xC + xB and yP = yC + yB.

For the derivation of the variables mvL0 and mvL1, refldxL0 and refldxL1 as well as PredFlagL0 and PredFlagL1, the following applies.

- If PredMode is equal to MODE_SKIP, the derivation process for luma motion vectors for merge mode as specified in subclause 8.4.2.1.18.4.2.1.3 is invoked with the luma location (xP, yP), variables nPSW, nPSH and the partition index PartIdx as inputs and the output being the luma motion vectors mvL0, mvL1, the reference indices refldxL0, refldxL1, and the prediction list utilization flags predFlagL0 and predFlagL1.

HEVC

- Otherwise, if PredMode is equal to MODE_INTER and merge_flag[xP][yP] is equal to 1, the derivation process for luma motion vectors for merge mode as specified in subclause 8.4.2.1.18.4.2.1.3 is invoked with the luma location (xP, yP), variables nPSW and nPSH and the partition index PartIdx as inputs and the outputs being the luma motion vectors mvL0 and mvL1, the reference indices refIdxL0 and refIdxL1, the prediction utilization flags predFlagL0 and predFlagL1.
- Otherwise, for X being replaced by either 0 or 1 in the variables predFlagLX, mvLX, refIdxLX and in Pred_LX and in the syntax elements ref_idx_IX and mvd_IX, the following applies.

1. The variables LcToLx, refIdxLX and predFlagLX are derived as follows.

- If inter_pred_flag[xP][yP] is equal to Pred_LC and PredLCToPredLx[ref_idx_lc[xP][yP]] is equal to Pred_LX,

```

refIdxLX = RefIdxLCToRefIdxLx[ ref_idx_lc[ xP ][ yP ] ] (8-727252)
predFlagLX = 1 (8-737353)
mvd_IX[ xP ][ yP ][ 0 ] = mvd_lc[ xP ][ yP ][ 0 ] (8-747454)
mvd_IX[ xP ][ yP ][ 1 ] = mvd_lc[ xP ][ yP ][ 1 ] (8-757555)
mvp_idx_IX[ xP ][ yP ][ 0 ] = mvp_idx_lc[ xP ][ yP ] (8-767656)
LcToLx = LX (8-777757)

```

- Otherwise, if inter_pred_flag[xP][yP] is equal to Pred_LX or Pred_BI,

```

refIdxLX = ref_idx_IX[ xP ][ yP ] (8-787858)
predFlagLX = 1 (8-797959)

```

- Otherwise, the variables refIdxLX and predFlagLX are specified by

```

refIdxLX = -1 (8-808060)
predFlagLX = 0 (8-818161)

```

2. The variable mvdLX is derived as follows.

```

mvdLX[ 0 ] = mvd_IX[ xP ][ yP ][ 0 ] (8-828262)
mvdLX[ 1 ] = mvd_IX[ xP ][ yP ][ 1 ] (8-838363)

```

3. When predFlagLX is equal to 1, the variable mvpLX is derived as follows.

- The derivation process for luma motion vector prediction in subclause 8.4.2.1.38.4.2.1.5 is invoked with the luma location (xP, yP), variables nPSW and nPSH and refIdxLX as the inputs and the output being mvpLX.

4. When predFlagLX is equal to 1, the luma motion vector mvLX is derived as

```

mvLX[ 0 ] = mvpLX[ 0 ] + mvdLX[ 0 ] (8-848464)
mvLX[ 1 ] = mvpLX[ 1 ] + mvdLX[ 1 ] (8-858565)

```

When ChromaArrayType is not equal to 0 and predFlagLX (with X being either 0 or 1) is equal to 1, the derivation process for chroma motion vectors in subclause 8.4.2.1.78.4.2.1.8 is invoked with mvLX and refIdxLX as inputs and the output being mvCLX.

8.4.2.1.1 Derivation process for luma motion vectors for merge mode

This process is only invoked when PredMode is equal to MODE_INTER and merge_flag [xP][yP] is equal to 1, where (xP, yP) specify the top-left luma sample of the current prediction unit relative to the top-left luma sample of the current picture.

Inputs of this process are

- a luma location (xP, yP) of the top-left luma sample of the current prediction unit relative to the top-left luma sample of the current picture,
- variables specifying the width and the height of the prediction unit for luma, nPSW and nPSH,
- a variable PartIdx specifying the index of the current prediction unit within the current coding unit.

Outputs of this process are

- the luma motion vectors mvL0 and mvL1,

Formatted: Heading 5, Indent: Left: 0", Tab stops: Not at 0.59" + 0.79" + 0.98"

HEVC

- the reference indices `refIdxL0` and `refIdxL1`,
- the prediction list utilization flags `predFlagL0` and `predFlagL1`.

The motion vectors `mvL0` and `mvL1`, the reference indices `refIdxL0` and `refIdxL1`, and the prediction utilization flags `predFlagL0` and `predFlagL1` are derived as specified by the following ordered steps:

1. The derivation process for merging candidates from neighboring prediction unit partitions in subclause 8.4.2.1.4 is invoked with luma location (`xP`, `yP`), the width and the height of the prediction unit `nPSW` and `nPSH` and the partition index `PartIdx` as inputs and the output is assigned to the availability flags `availableFlagN`, the motion vectors `mvL0N` and `mvL1N`, the reference indices `refIdxL0N` and `refIdxL1N` and the prediction list utilization flags `predFlagL0N` and `predFlagL1N` with `N` being replaced by `A`, `B`, `C` or `D`.
2. The derivation process of reference indices for temporal merging candidate in subclause 8.4.2.1.3 is invoked with luma location (`xP`, `yP`), `nPSW`, `nPSH` as the inputs and the output is directly assigned to `refIdxLX`.
3. The derivation process for temporal luma motion vector prediction in subclause 4.8.4.2.1.7 is invoked with luma location (`xP`, `yP`), `refIdxLX` as the inputs and with the output being the availability flag `availableFlagLXCol` and the temporal motion vector `mvLXCol`. The variables `availableFlagCol` and `predFlagLXCol` (with `X` being 0 or 1, respectively) are derived as specified below.

$$\text{availableFlagCol} = \text{availableFlagL0Col} \parallel \text{availableFlagL1Col} \quad (8-868671)$$

$$\text{predFlagLXCol} = \text{availableFlagLXCol} \quad (8-878772)$$

4. The merging candidate list, `mergeCandList`, is constructed of which elements are given as specified order:
 1. `A`, if `availableFlagA` is equal to 1
 2. `B`, if `availableFlagB` is equal to 1
 3. `Col`, if `availableFlagCol` is equal to 1
 4. `C`, if `availableFlagC` is equal to 1
 5. `D`, if `availableFlagD` is equal to 1
5. If several merging candidates have the motion vectors and the same reference indices, the merging candidates are removed from the list except the merging candidate which has the smallest order in the `mergeCandList`.
6. If the number of elements `NumMergeCand` within the `mergeCandList` is equal to 1, `mergeIdx` is set equal to 0, otherwise, `mergeIdx` is set equal to `merge_idx[xP][yP]`.
7. The following assignments are made with `N` being the candidate at position `mergeIdx` in the merging candidate list `mergeCandList` (`N = mergeCandList[mergeIdx]`) and `X` being replaced by 0 or 1:

$$\text{mvLX}[0] = \text{mvLXN}[0] \quad (8-888873)$$

$$\text{mvLX}[1] = \text{mvLXN}[1] \quad (8-898974)$$

$$\text{refIdxLX} = \text{refIdxLXN} \quad (8-909075)$$

$$\text{predFlagLX} = \text{predFlagLXN} \quad (8-919176)$$

8. If all availability flags `availableFlagN` (with `N` being replaced by `A`, `B`, `Col`, `C`, or `D`) are equal to 0, `mergeIdx` is set equal to 0 and the variables `mvLX`, `refIdxLX` and `predFlagLX` (with `X` being replaced by 0 or 1) are inferred as follows.

If `slice_type` is equal to `P`, the following applies.

$$\text{mvLX}[0] = 0 \quad (8-929277)$$

$$\text{mvLX}[1] = 0 \quad (8-939378)$$

HEVC

refIdxL0 = 0
(8-949479)
refIdxL1 = -1
(8-959580)
predFlagL0 = 1
(8-969681)
prefFlagL1 = 0
(8-979782)

Otherwise (slice_type is equal to B), the following applies.

mvLX[0] = 0
(8-989883)
mvLX[1] = 0
(8-999984)
refIdxL0 = 0
(8-10010085)
refIdxL1 = 0
(8-10110186)
predFlagL0 = 1
(8-10210287)
prefFlagL1 = 1
(8-10310388)

8.4.2.1.18.4.2.1.2 Derivation process for spatial merging candidates

Inputs to this process are

- a luma location (xP, yP) specifying the top-left luma sample of the current prediction unit relative to the top-left sample of the current picture,
- variables specifying the width and the height of the prediction unit for luma, nPSW and nPSH,
- a variable PartIdx specifying the index of the current prediction unit within the current coding unit.

Outputs of this process are (with N being replaced by A, B, C, or D and with X being replaced by 0 or 1)

- the availability flags availableFlagN of the neighbouring prediction units,
- the reference indices refIdxLXN of the neighbouring prediction units,
- the prediction list utilization flags predFlagLXN of the neighbouring prediction units,
- the motion vectors mvLXN of the neighbouring prediction units.

HEVC

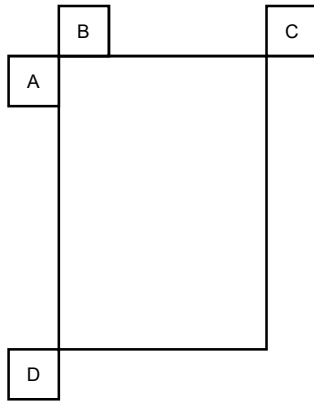


Figure 8-3 – Spatial neighbours that can be used as merging candidates (informative) illustrates the position of the spatial neighbours A, B, C and D relative to the current prediction unit.

For the derivation of availableFlagN, with N being A, B, C or D and (xN, yN) being (xP - 1, yP), (xP, yP - 1), (xP + nPSW, yP - 1) or (xP - 1, yP + nPSH) the following applies.

- If one of the following conditions is true, the availableFlagN is set equal to 0, both components mvLXN are set equal to 0, refldxLXN and predFlagLX[xN, yN] of the prediction unit covering luma location (xN, yN) are assigned respectively to mvLXN, refldxLXN and predFlagLXN.
 - The prediction unit covering luma location (xN, yN) is not available or PredMode is MODE_INTRA.
 - PartMode of the current prediction unit is PART_2NxN and PartIdx is equal to 1 and the prediction units covering luma location (xP, yP - 1) (PartIdx = 0) and luma location (xN, yN) (Cand. N) have identical motion parameters:
 - $mvLX[xP, yP - 1] == mvLX[xN, yN]$
 - $refldxLX[xP, yP - 1] == refldxLX[xN, yN]$
 - $predFlagLX[xP, yP - 1] == predFlagLX[xN, yN]$
 - PartMode of the current prediction unit is PART_Nx2N and PartIdx is equal to 1 and the prediction units covering luma location (xP - 1, yP) (PartIdx = 0) and luma location (xN, yN) (Cand. N) have identical motion parameters:
 - $mvLX[xP - 1, yP] == mvLX[xN, yN]$
 - $refldxLX[xP - 1, yP] == refldxLX[xN, yN]$
 - $predFlagLX[xP - 1, yP] == predFlagLX[xN, yN]$
 - PartMode of the current prediction unit is PART_NxN and PartIdx is equal to 3 and the prediction units covering luma location (xP - 1, yP) (PartIdx = 2) and luma location (xP - 1, yP - 1) (PartIdx = 0) have identical motion parameters:
 - $mvLX[xP - 1, yP] == mvLX[xP - 1, yP - 1]$
 - $refldxLX[xP - 1, yP] == refldxLX[xP - 1, yP - 1]$
 - $predFlagLX[xP - 1, yP] == predFlagLX[xP - 1, yP - 1]$
 and the prediction units covering luma location (xP, yP - 1) (PartIdx = 1) and luma location (xN, yN) (Cand. N) have identical motion parameters:
 - $mvLX[xP, yP - 1] == mvLX[xN, yN]$
 - $refldxLX[xP, yP - 1] == refldxLX[xN, yN]$
 - $predFlagLX[xP, yP - 1] == predFlagLX[xN, yN]$

HEVC

- PartMode of the current prediction unit is PART_NxN and PartIdx is equal to 3 and the prediction units covering luma location (xP, yP - 1) (PartIdx = 1) and luma location (xP - 1, yP - 1) (PartIdx = 0) have identical motion parameters:
 - $mvLX[xP, yP - 1] == mvLX[xP - 1, yP - 1]$
 - $refIdxLX[xP, yP - 1] == refIdxLX[xP - 1, yP - 1]$
 - $predFlagLX[xP, yP - 1] == predFlagLX[xP - 1, yP - 1]$
 and the prediction units covering luma location (xP - 1, yP) (PartIdx = 2) and luma location (xN, yN) (Cand. N) have identical motion parameters:
 - $mvLX[xP - 1, yP] == mvLX[xN, yN]$
 - $refIdxLX[xP - 1, yP] == refIdxLX[xN, yN]$
 - $predFlagLX[xP - 1, yP] == predFlagLX[xN, yN]$
- Otherwise, availableFlagN is set equal to 1 and the variables $mvLX[xN, yN]$, $refIdxLX[xN, yN]$ and $predFlagLX[xN, yN]$ of the prediction unit covering luma location (xN, yN) are assigned respectively to $mvLXN$, $refIdxLXN$ and $predFlagLXN$.

8.4.2.1.28.4.2.1.3 Derivation process of reference indices for temporal merging candidate

Inputs to this process are

- a luma location (xP, yP) specifying the top-left luma sample of the current prediction unit relative to the top-left sample of the current picture,
- variables specifying the width and the height of the prediction unit for luma, nPSW and nPSH.

Outputs of this process are (with X being replaced by 0 or 1)

- the reference indices $refIdxLX$ of the current prediction units.

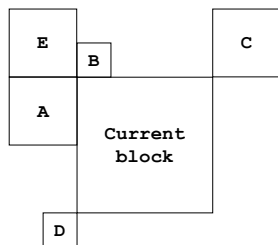


Figure 8-4 – Spatial neighbours, that can be used to derive reference indices for temporal merging candidates, (informative) illustrates the position of the spatial neighbours A, B, C, D and E relative to the current prediction unit.

For the derivation of $refIdxLXA$, the following applies.

- If the prediction unit covering luma location (xP-1, yP) is available and PredMode is not MODE_INTRA, $refIdxLXA$ is assigned to $refIdxLX[xP-1, yP]$.
- Otherwise, $refIdxLXA$ is assigned to -1.

For the derivation of $refIdxLXB$, the following applies.

- If the prediction unit covering luma location (xP, yP-1) is available and PredMode is not MODE_INTRA, $refIdxLXB$ is assigned to $refIdxLX[xP, yP-1]$.
- Otherwise, $refIdxLXB$ is assigned to -1.

For the derivation of $refIdxLXC$, the following applies.

- If the prediction unit covering luma location (xP+nPSW, yP-1) is available and PredMode is not MODE_INTRA, $refIdxLXC$ is assigned to $refIdxLX[xP+nPSW, yP-1]$.
- Otherwise if the prediction unit covering luma location (xP-1, yP+nPSH) is available and PredMode is not MODE_INTRA, $refIdxLXC$ is assigned to $refIdxLX[xP-1, yP+nPSH]$.

HEVC

- Otherwise if the prediction unit covering luma location (xP-1, yP-1) is available and PredMode is not MODE_INTRA reflDxLXC is assigned to reflDxLX[xP-1, yP-1].
- Otherwise, reflDxLXC is assigned to -1.

The variable reflDxLX is derived as follows.

- If reflDxLXA is equal to reflDxLXB and reflDxLXB is equal to reflDxLXC, the following applies.
 - If reflDxLXA is equal to -1 then $\text{reflDxLX} = 0$
 - Otherwise $\text{reflDxLX} = \text{reflDxLXA}$
- Otherwise if reflDxLXA is equal to reflDxLXB, the following applies.
 - If reflDxLXA is equal to -1 then $\text{reflDxLX} = \text{reflDxLXC}$
 - Otherwise $\text{reflDxLX} = \text{reflDxLXA}$
- Otherwise if reflDxLXB is equal to reflDxLXC, the following applies.
 - If reflDxLXB is equal to -1 then $\text{reflDxLX} = \text{reflDxLXA}$
 - Otherwise $\text{reflDxLX} = \text{reflDxLXB}$
- Otherwise if reflDxLXA is equal to reflDxLXC, the following applies.
 - If reflDxLXA is equal to -1 then $\text{reflDxLX} = \text{reflDxLXB}$
 - Otherwise $\text{reflDxLX} = \text{reflDxLXA}$
- Otherwise if reflDxLXA is equal to -1, the following applies.
 - $\text{reflDxLX} = \min(\text{reflDxLXB}, \text{reflDxLXC})$
- Otherwise if reflDxLXB is equal to -1, the following applies.
 - $\text{reflDxLX} = \text{Min}(\text{reflDxLXA}, \text{reflDxLXC})$
- Otherwise if reflDxLXC is equal to -1, the following applies.
 - $\text{reflDxLX} = \text{Min}(\text{reflDxLXA}, \text{reflDxLXB})$
- Otherwise, the following applies.
 - $\text{reflDxLX} = \text{Min}(\text{Min}(\text{reflDxLXA}, \text{reflDxLXB}), \text{reflDxLXC})$

8.4.2.1.38, 4.2.1.4 Derivation process for luma motion vector prediction

Inputs to this process are

- a luma location (xP, yP) specifying the top-left luma sample of the current prediction unit relative to the top-left sample of the current picture,
- variables specifying the width and the height of the prediction unit for luma, nPSW and nPSH.
- the reference index of the current prediction unit partition reflDxLX (with X being 0 or 1).

Output of this process is

- the prediction mvPLX of the motion vector mvLX (with X being 0 or 1).

The motion vector predictor mvPLX is derived in the following ordered steps.

1. The derivation process for motion vector predictor candidates from neighboring prediction unit partitions in subclause [8.4.2.1.58, 4.2.1.6](#) is invoked with luma location (xP, yP), the width and the height of the prediction unit nPSW and nPSH, and reflDxLX (with X being 0 or 1, respectively) as inputs and the availability flags availableFlagLXN and the motion vectors mvLXN with N being replaced by A, B as the output.
2. The derivation process for temporal luma motion vector prediction in subclause [48.4.2.1.7](#) is invoked with luma location (xP, yP), the width and the height of the prediction unit nPSW and nPSH, and reflDxLX (with X being 0 or 1, respectively) as the inputs and with the output being the availability flag availableFlagLXCcol and the temporal motion vector predictor mvLXCcol.
3. The motion vector predictor list, mvPLXList, is constructed of which elements are given as specified order:
 1. mvLXCcol, if availableFlagLXCcol is equal to 1

HEVC

2. mvLXA, if availableFlagLXA is equal to 1
3. mvLXB, if availableFlagLXB is equal to 1
4. If several motion vectors have the same value, the motion vectors are removed from the list except the motion vector which has the smallest order in the mvListLX.
5. If the number of elements NumMVPCand(LX) within the mvListLX is equal to 1, mvIdx is set equal to 0, otherwise, mvIdx is set equal to mv_idx_IX[xP, yP].
6. The motion vector of mvListLX[mvIdx] is assigned to mvpLX.

8.4.2.1.48.4.2.1.5 Derivation process for motion vector predictor candidates

Inputs to this process are

- a luma location (xP, yP) specifying the top-left luma sample of the current prediction unit relative to the top-left sample of the current picture,
- variables specifying the width and the height of the prediction unit for luma, nPSW and nPSH,
- the reference index of the current prediction unit partition refIdxLX (with X being 0 or 1).

Outputs of this process are (with N being replaced by A, or B)

- the motion vectors mvLXN of the neighbouring prediction units,
- the availability flags availableFlagLXN of the neighbouring prediction units.

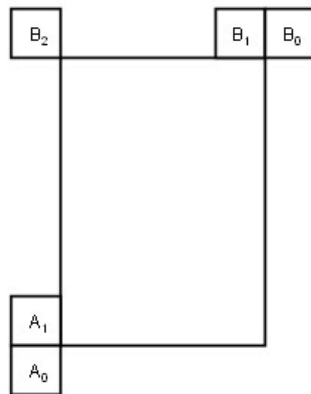


Figure 8-5 – Spatial motion vector neighbours

The function RefPicOrderCnt(pic, refIdx, LX) is specified by the value of PicOrderCnt of the picture that is the reference picture RefPicListX[refIdx] of pic with X being 0 or 1. PicOrderCnt of the reference picture shall be maintained until the picture is marked as “non-existing.”

The motion vector mvLXA and the availability flag availableFlagLXA are derived in the following ordered steps:

1. Let a set of two sample locations be (xA_k, yA_k) , with $k = 0 \dots 1$, specifies sample locations with $xA_k = xP - 1$, $yA_0 = yP + nPSH$ and $yA_1 = yA_0 - \text{MinPuSize}$. The set of sample locations (xA_k, yA_k) represent the sample locations immediately to the left side of the left partition boundary and its extended line. [Ed.: (WJ) MinPuSize should be defined somewhere]
2. Let the availability flag availableFlagLXA be initially set equal to 0 and the both components of mvLXA are set equal to 0.
3. For (xA_k, yA_k) from (xA_0, yA_0) to (xA_1, yA_1) where $yA_1 = yA_0 - \text{MinPuSize}$, the following applies repeatedly until availableFlagLXA is equal to 1:

HEVC

- When the prediction unit covering luma location (x_{A_k}, y_{A_k}) is available, $PredMode$ is not $MODE_INTRA$, $predFlagLX[x_{A_k}][y_{A_k}]$ is equal to 1 and the reference index $refIdxLX[x_{A_k}][y_{A_k}]$ is equal to the reference index of the current prediction unit $refIdxLX$, $availableFlagLXA$ is set equal to 1 and the motion vector $mvLXA$ is set equal to the motion vector $mvLX[x_{A_k}][y_{A_k}]$, $refIdxA$ is set equal to $refIdxLX[x_{A_k}][y_{A_k}]$ and $ListA$ is set equal to LX .
4. When $availableFlagLXA$ is equal to 0, for (x_{A_k}, y_{A_k}) from (x_{A_0}, y_{A_0}) to (x_{A_1}, y_{A_1}) where $y_{A_1} = y_{A_0} - MinPuSize$, the following applies repeatedly until $availableFlagLXA$ is equal to 1:
- If the prediction unit covering luma location (x_{A_k}, y_{A_k}) is available, $PredMode$ is not $MODE_INTRA$, $predFlagLY[x_{A_k}][y_{A_k}]$ (with $Y = !X$) is equal to 1 and $RefPicOrderCnt(currPic, refIdxLY[x_{A_k}][y_{A_k}], LY)$ is equal to $RefPicOrderCnt(currPic, refIdxLX, LX)$, $availableFlagLXA$ is set equal to 1, the motion vector $mvLXA$ is set equal to the motion vector $mvLY[x_{A_k}][y_{A_k}]$, $refIdxA$ is set equal to $refIdxLY[x_{A_k}][y_{A_k}]$ and $ListA$ is set equal to LY .
 - Otherwise if the prediction unit covering luma location (x_{A_k}, y_{A_k}) is available, $PredMode$ is not $MODE_INTRA$, $predFlagLX[x_{A_k}][y_{A_k}]$ is equal to 1, $availableFlagLXA$ is set equal to 1, the motion vector $mvLXA$ is set equal to the motion vector $mvLX[x_{A_k}][y_{A_k}]$, $refIdxA$ is set equal to $refIdxLX[x_{A_k}][y_{A_k}]$, $ListA$ is set equal to LX .
 - Otherwise if the prediction unit covering luma location (x_{A_k}, y_{A_k}) is available, $PredMode$ is not $MODE_INTRA$, $predFlagLY[x_{A_k}][y_{A_k}]$ (with $Y = !X$) is equal to 1, $availableFlagLXA$ is set equal to 1, the motion vector $mvLXA$ is set equal to the motion vector $mvLY[x_{A_k}][y_{A_k}]$, $refIdxA$ is set equal to $refIdxLY[x_{A_k}][y_{A_k}]$, $ListA$ is set equal to LY .
5. When $availableFlagLXA$ is equal to 1, the following applies.
- If $RefPicOrderCnt(currPic, refIdxA, ListA)$ is equal to $RefPicOrderCnt(currPic, refIdxLX, LX)$, $mvLXA$ is set equal to $mvLXA$
 - Otherwise, $mvLXA$ is derived as specified below

$$tx = (16384 + Abs(td / 2)) / td$$

$$\text{(8-10410477)}$$

$$DistScaleFactor = Clip3(-1024, 1023, (tb * tx + 32) >> 6)$$

$$\text{(8-10510577)}$$

$$mvLXA = ClipMv((DistScaleFactor * mvLXA + 128) >> 8)$$

$$\text{(8-10610677)}$$

[Ed. (WJ): software has a clip function for scaled motion vector. Do we need this? AVC does not have it]

where td and tb are derived as

$$td = Clip3(-128, 127, PicOrderCnt(currPic) - RefPicOrderCnt(currPic, refIdxA, ListA))$$

$$\text{(8-10710777)}$$

$$tb = Clip3(-128, 127, PicOrderCnt(currPic) - RefPicOrderCnt(currPic, refIdxLX, LX))$$

$$\text{(8-10810877)}$$

The motion vector $mvLXB$ and the availability flag $availableFlagLXB$ are derived in the following ordered steps:

1. Let a set of three sample location (x_{B_k}, y_{B_k}) , with $k = 0, 1, 2$, specifies sample locations with $x_{B_0} = x_P + nPSW$, $x_{B_1} = x_{B_0} - MinPuSize$, $x_{B_2} = x_P - MinPuSize$ and $y_{B_k} = y_P - 1$. The set of sample locations (x_{B_k}, y_{B_k}) represent the sample locations immediately to the upper side of the above partition boundary and its extended line. [Ed.: (WJ) $MinPuSize$ should be defined somewhere]
2. Let the availability flag $availableFlagLXB$ be initially set equal to 0 and the both components of $mvLXB$ are set equal to 0.
3. For (x_{B_k}, y_{B_k}) from (x_{B_0}, y_{B_0}) to (x_{B_2}, y_{B_2}) where $x_{B_0} = x_P + nPSW$, $x_{B_1} = x_{B_0} - MinPuSize$, and $x_{B_2} = x_P - MinPuSize$, the following applies repeatedly until $availableFlagLXB$ is equal to 1:
 - When the prediction unit covering luma location (x_{B_k}, y_{B_k}) is available, $PredMode$ is not $MODE_INTRA$, $predFlagLX[x_{B_k}][y_{B_k}]$ is equal to 1, and the reference index $refIdxLX[x_{B_k}][y_{B_k}]$ is equal to the reference index of the current prediction unit $refIdxLX$ and $mvLX[x_{B_k}][y_{B_k}]$ is not identical to $mvLXA$, $availableFlagLXB$ is set equal to 1 and the motion vector $mvLXB$ is set equal to the motion vector $mvLX[x_{B_k}][y_{B_k}]$, $refIdxB$ is set equal to $refIdxLX[x_{B_k}][y_{B_k}]$ and $ListB$ is set equal to LX .

HEVC

4. When availableFlagLXB is equal to 0, for (xB_k, yB_k) from (xB₀, yB₀) to (xB₂, yB₂) where xB₀ = xP + nPSW, xB₁ = xB₀ - MinPuSize , and xB₂ = xP - MinPuSize, the following applies repeatedly until availableFlagLXB is equal to 1:

- If the prediction unit covering luma location (xB_k, yB_k) is available, PredMode is not MODE_INTRA, predFlagLY[xB_k][yB_k](with Y = !X) is equal to 1 and RefPicOrderCnt(currPic , refIdxLY[xB_k][yB_k], LY) is equal to RefPicOrderCnt(currPic, refIdxLX, LX), availableFlagLXB is set equal to 1, the motion vector mvLXB is set equal to the motion vector mvLY[xB_k][yB_k], refIdxB is set equal to refIdxLY[xB_k][yB_k] and ListB is set equal to LY.
- Otherwise if the prediction unit covering luma location (xB_k, yB_k) is available, PredMode is not MODE_INTRA, predFlagLX[xB_k][yB_k] is equal to 1, availableFlagLXB is set equal to 1, the motion vector mvLXB is set equal to the motion vector mvLX[xB_k][yB_k], refIdxB is set equal to refIdxLX[xB_k][yB_k], ListB is set equal to LX.
- Otherwise if the prediction unit covering luma location (xB_k, yB_k) is available, PredMode is not MODE_INTRA, predFlagLY[xB_k][yB_k](with Y = !X) is equal to 1, availableFlagLXB is set equal to 1, the motion vector mvLXB is set equal to the motion vector mvLY[xB_k][yB_k], refIdxB is set equal to refIdxLY[xB_k][yB_k], ListB is set equal to LY.
- If availableFlagLXB is equal to 1 and RefPicOrderCnt(currPic , refIdxB, ListB) is equal to RefPicOrderCnt(currPic, refIdxLX, LX), mvLXB is set equal to mvLXB.
- Otherwise if availableFlagLXB is equal to 1 and RefPicOrderCnt(currPic , refIdxB, ListB) is not equal to RefPicOrderCnt(currPic, refIdxLX, LX), mvLXB is derived as specified below.

$$tx = (16384 + Abs(td / 2)) / td$$

$$DistScaleFactor = Clip3(-1024, 1023, (tb * tx + 32) >> 6)$$

$$mvLXB = ClipMv((DistScaleFactor * mvLXB + 128) >> 8)$$

[Ed. (WJ): software has a clip function for scaled motion vector. Do we need this? AVC does not have it]

where td and tb are derived as

$$td = Clip3(-128, 127, PicOrderCnt(currPic) - RefPicOrderCnt(currPic , refIdxB, ListB))$$

$$tb = Clip3(-128, 127, PicOrderCnt(currPic) - RefPicOrderCnt(currPic, refIdxLX, LX))$$

- When availableFlagLXB is equal to 1 and mvLXB is identical to mvLXA, availableFlagLXB is set equal to 0.

8.4.2.1.58.4.2.1.6 Derivation process for temporal luma motion vector prediction

Inputs to this process are

- a luma location (xP, yP) specifying the top-left luma sample of the current prediction unit relative to the top-left sample of the current picture,
- variables specifying the width and the height of the prediction unit for luma, nPSW and nPSH,
- the reference index of the current prediction unit partition refIdxLX (with X being 0 or 1).

Outputs of this process are

- the motion vector prediction mvLXCol,
- the availability flag availableFlagLXCol.

The function RefPicOrderCnt(pic, refidx, LX) is specified by the value of PicOrderCnt of the picture that is the reference picture RefPicListX[refidx] of pic with X being 0 or 1. PicOrderCnt of the reference picture shall be maintained until the picture is marked as “non-existing.”

Depending on the values of slice_type and collocated_from_10_flag, the variable colPic, specifying the picture that contains the co-located partition, is derived as follows.

HEVC

- If slice_type is equal to B and collocated_from_l0_flag is equal to 0, the variable colPic specifies the picture that contains the co-located partition as specified by RefPicList1[0].
- Otherwise (slice_type is equal to B and collocated_from_l0_flag is equal to 1 or slice_type is equal to P) , the variable colPic specifies the picture that contains the co-located partition as specified by RefPicList0[0].

Variable colPu and its position (xPCol, yPCol) are derived in the following ordered steps:

1. Right-bottom luma position of the current prediction unit is defined by

$$xPRb = xP + nPSW$$

$$(8-114+1477)$$

$$yPRb = yP + nPSH$$

$$(8-115+1577)$$

2. The variable colPu is set as the prediction unit covering the modified position given by $((xPRb \gg 4) \ll 4, (yPRb \gg 4) \ll 4)$ inside the colPic.
3. If colPu is coded in an intra prediction mode or colPu is unavailable, the following applies.

- Central luma position of the current prediction unit is defined by

$$xPCtr = (xP + (nPSW \gg 1) - 1)$$

$$(8-116+1677)$$

$$yPCtr = (yP + (nPSH \gg 1) - 1)$$

$$(8-117+1778)$$

- The variable colPu is set as the prediction unit covering the modified position given by $((xPCtr \gg 4) \ll 4, (yPCtr \gg 4) \ll 4)$ inside the colPic.

4. (xPCol, yPCol) is set equal to the top-left luma sample of the colPu relative to the top-left luma sample of the colPic.

The variables mvLXCol and availableFlagLXCol are derived as follows.

- If colPu is coded in an intra prediction mode or colPu is unavailable, both components of mvLXCol are set equal to 0 and availableFlagLXCol is set equal to 0.
- Otherwise (colPu is not coded in an intra prediction mode and colPu is available), the variables mvCol and refldxCol are derived as follows,
- If PredFlagL0[xPCol][yPCol] is equal to 0, the motion vector mvCol and the reference index refldxCol are set equal to MvL1[xPCol][yPCol] and RefldxL1[xPCol][yPCol], respectively.
- Otherwise (PredFlagL0[xPCol][yPCol] is equal to 1), the following applies.
 - If PredFlagL1[xPCol][yPCol] is equal to 0, the motion vector mvCol and the reference index refldxCol are set equal to MvL0[xPCol][yPCol] and RefldxL0[xPCol][yPCol], respectively.
 - Otherwise (PredFlagL1[xPCol][yPCol] is equal to 1), the following applies.
 - a. The following assignments are made with X being 0 or 1.
 - RefldxColLX = RefldxLX[xPCol][yPCol]
 - If PicOrderCnt(colPic) is less than PicOrderCnt(currPic) and RefPicOrderCnt(colPic, RefldxColLX, LX) is greater than PicOrderCnt(currPic) or PicOrderCnt(colPic) is greater than PicOrderCnt(currPic) and RefPicOrderCnt(colPic, RefldxColLX, LX) is less than PicOrderCnt(currPic), the variable MvXCross is derived as follows.
 - MvXCross = 1
 - Otherwise (PicOrderCnt(colPic) is less than PicOrderCnt(currPic) and RefPicOrderCnt(colPic, RefldxColLX, LX) is less than or equal to PicOrderCnt(currPic) or PicOrderCnt(colPic) is greater than PicOrderCnt(currPic) and RefPicOrderCnt(colPic, RefldxColLX, LX) is greater than or equal to PicOrderCnt(currPic)), the variable MvXCross is derived as follows.
 - MvXCross = 0

HEVC

- b. If one of the following conditions is true, the motion vector mvCol, the reference index refIdxCol and ListCol are set equal to MvL1[xPCol][yPCol], RefIdxColL1 and L1, respectively.
 - Mv0Cross is equal to 0 and Mv1Cross is equal to 1
 - Mv0Cross is equal to Mv1Cross and reference index list is equal to L1
- c. Otherwise, the motion vector mvCol, the reference index refIdxCol and ListCol are set equal to MvL0[xPCol][yPCol], RefIdxColL0 and L0, respectively.

and the variable availableFlagLXCcol is set equal to 1 and the following applies.

- If PicOrderCnt(colPic) – RefPicOrderCnt(colPic, refIdxCol, ListCol) is equal to PicOrderCnt(currPic) – RefPicOrderCnt(currPic, refIdxLX, LX),

$$\text{mvLXCcol} = \text{mvCol} \quad (8-118+1879)$$

- Otherwise, mvLXCcol is derived as scaled version of the motion vector mvCol as specified below

$$\text{tx} = (16384 + \text{Abs}(\text{td} / 2)) / \text{td} \quad (8-119+1980)$$

$$\text{DistScaleFactor} = \text{Clip3}(-1024, 1023, (\text{tb} * \text{tx} + 32) \gg 6) \quad (8-120+2084)$$

$$\text{mvLXCcol} = \text{ClipMv}((\text{DistScaleFactor} * \text{mvCol} + 128) \gg 8) \quad (8-121+2182)$$

[Ed. (WJ): software has a clip function for scaled motion vector. Do we need this? AVC does not have it]

where td and tb are derived as

$$\text{td} = \text{Clip3}(-128, 127, \text{PicOrderCnt}(\text{colPic}) - \text{RefPicOrderCnt}(\text{colPic}, \text{refIdxCol}, \text{ListCol})) \quad (8-122+2283)$$

$$\text{tb} = \text{Clip3}(-128, 127, \text{PicOrderCnt}(\text{currPic}) - \text{RefPicOrderCnt}(\text{currPic}, \text{refIdxLX}, \text{LX})) \quad (8-123+2384)$$

8.4.2.1.68.4.2.1.7 Derivation process for chroma motion vectors

[Ed.: (WJ) 4:2:0 assumption yet]

Inputs to this process are a luma motion vector mvLX and a reference index refIdxLX.

Output of this process is a chroma motion vector mvCLX.

A chroma motion vector is derived from the corresponding luma motion vector.

For the derivation of the chroma motion vector mvCLX, the following applies.

$$\text{mvCLX}[0] = \text{mvLX}[0] \quad (8-124+2483)$$

$$\text{mvCLX}[1] = \text{mvLX}[1] \quad (8-125+2586)$$

8.4.2.2 Decoding process for inter prediction samples

Inputs to this process are:

- a luma location (xC, yC) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture,
- a luma location (xB, yB) specifying the top-left luma sample of the current prediction unit relative to the top-left luma sample of the current coding unit,
- a variable nCS specifying the size of the current coding unit,
- variables specifying the width and the height of the prediction unit, nPSW and nPSH,
- luma motion vectors mvL0 and mvL1, and chroma motion vectors mvCL0 and mvCL1,
- reference indices refIdxL0 and refIdxL1,
- prediction list utilization flags, predFlagL0 and predFlagL1.

HEVC

Outputs of this process are:

- a $(nCS_L) \times (nCS_L)$ array predSamples_L of luma prediction samples, where nCS_L is derived as specified below,
- a $(nCS_C) \times (nCS_C)$ array preSamples_{Cb} of chroma prediction samples for the component Cb, where nCS_C is derived as specified below,
- a $(nCS_C) \times (nCS_C)$ array predSamples_{Cr} of chroma residual samples for the component Cr, where nCS_C is derived as specified below.

The variable nCS_L is set equal to nCS and the variable nCS_C is set equal to $nCS \gg 1$. [Ed: (WJ) revisit for supporting other chroma formats]

Let predSamples_{L0L} and predSamples_{L1L} be $(nPSW) \times (nPSH)$ arrays of predicted luma sample values and predSample_{L0Cb} , predSample_{L1Cb} , predSample_{L0Cr} , and predSample_{L1Cr} be $(nPSW/2) \times (nPSH/2)$ arrays of predicted chroma sample values.

For LX being replaced by either L0 or L1 in the variables predFlag_{LX} , RefPicList_X , refIdx_{LX} , refPic_{LX} , and predPart_{LX} , the following is specified.

When predFlag_{LX} is equal to 1, the following applies.

- The reference picture consisting of an ordered two-dimensional array refPic_{LXL} of luma samples and two ordered two-dimensional arrays refPic_{LXCb} and refPic_{LXCr} of chroma samples is derived by invoking the process specified in subclause 8.4.2.2.1 with refIdx_{LX} and RefPicList_X given as input.
- The arrays predSamples_{LXL} , $\text{predSamples}_{LXCb}$, and $\text{predSamples}_{LXCr}$ are derived by invoking the fractional sample interpolation process specified in subclause 8.4.2.2.2 with the luma locations (x_C, y_C) , (x_B, y_B) , the width and the height of the current prediction unit $nPSW$, $nPSH$, the motion vectors mv_{LX} , mv_{CLX} , and the reference arrays with refPic_{LXL} , refPic_{LXCb} and refPic_{LXCr} given as input.

The array predSample_L of the prediction samples of luma component is derived by invoking the weighted sample prediction process specified in subclause 8.4.2.2.3 with the luma location (x_B, y_B) , the width and the height of the current prediction unit $nPSW$, $nPSH$, and the sample arrays predSamples_{L0L} and predSamples_{L1L} as well as predFlag_{L0} , predFlag_{L1} and BitDepth_Y given as input.

For C being replaced by Cb, or Cr, the array predSample_C of the prediction samples of component C is derived by invoking the weighted sample prediction process specified in subclause 8.4.2.2.3 with the chroma location $(x_B/2, y_B/2)$, the width and the height of the current prediction unit $nPSW_C$ set equal to $nPSW/2$, $nPSH_C$ set equal to $nPSH/2$, and the sample arrays predSamples_{L0C} and predSamples_{L1C} as well as predFlag_{L0} , predFlag_{L1} and BitDepth_C given as input.

8.4.2.2.1 Reference picture selection process

[Ed: (WJ) same as AVC]

8.4.2.2.2 Fractional sample interpolation process

Inputs to this process are:

- a luma location (x_C, y_C) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture,
- a luma location (x_B, y_B) specifying the top-left luma sample of the current prediction unit relative to the top left luma sample of the current coding unit,
- the width and height of this prediction unit, $nPSW$ and $nPSH$, in luma-sample units,
- a luma motion vector mv_{LX} given in quarter-luma-sample units,
- a chroma motion vector mv_{CLX} given in eighth-chroma-sample units,
- the selected reference picture sample arrays refPic_{LXL} , refPic_{LXCb} , and refPic_{LXCr} .

Outputs of this process are:

- a $(nPSW) \times (nPSH)$ array predSample_{LXL} of prediction luma sample values,
- two $(nPSW/2) \times (nPSH/2)$ arrays predSample_{LXCb} , and predSample_{LXCr} of prediction chroma sample values.

The location (x_P, y_P) given in full-sample units of the upper-left luma samples of the current prediction unit relative to the upper-left luma sample location of the given reference sample arrays is derived by

HEVC

$$xP = xC + xB \quad (8-12612687)$$

$$yP = yC + yB \quad (8-12712788)$$

Let (x_{IntL} , y_{IntL}) be a luma location given in full-sample units and (x_{FracL} , y_{FracL}) be an offset given in quarter-sample units. These variables are used only inside this subclause for specifying general fractional-sample locations inside the reference sample arrays $refPicLX_L$, $refPicLX_{Cb}$, and $refPicLX_{Cr}$.

For each luma sample location ($0 \leq x_L < nPSW$, $0 \leq y_L < nPSH$) inside the prediction luma sample array $predSampleLX_L$, the corresponding prediction luma sample value $predSampleLX_L[x_L, y_L]$ is derived as follows:

- The variables x_{IntL} , y_{IntL} , x_{FracL} , and y_{FracL} are derived by

$$x_{IntL} = xP + (mvLX[0] \gg 2) + x_L \quad (8-12812889)$$

$$y_{IntL} = yP + (mvLX[1] \gg 2) + y_L \quad (8-12912990)$$

$$x_{FracL} = mvLX[0] \& 3 \quad (8-13013091)$$

$$y_{FracL} = mvLX[1] \& 3 \quad (8-13113192)$$

- The prediction luma sample value $predSampleLX_L[x_L, y_L]$ is derived by invoking the process specified in subclause 8.4.2.2.2.1 with (x_{IntL} , y_{IntL}), (x_{FracL} , y_{FracL}) and $refPicLX_L$ given as input.

Let (x_{IntC} , y_{IntC}) be a chroma location given in full-sample units and (x_{FracC} , y_{FracC}) be an offset given in one-eighth sample units. These variables are used only inside this subclause for specifying general fractional-sample locations inside the reference sample arrays $refPicLX_{Cb}$ and $refPicLX_{Cr}$.

For each chroma sample location ($0 \leq x_C < nPSW/2$, $0 \leq y_C < nPSH/2$) inside the prediction chroma sample arrays $predSampleLX_{Cb}$ and $predSampleLX_{Cr}$, the corresponding prediction chroma sample values $predSampleLX_{Cb}[x_C, y_C]$ and $predSampleLX_{Cr}[x_C, y_C]$ are derived as follows:

- The variables x_{IntC} , y_{IntC} , x_{FracC} , and y_{FracC} are derived by

$$x_{IntC} = (xP / 2) + (mvCLX[0] \gg 3) + x_C \quad (8-13213293)$$

$$y_{IntC} = (yP / 2) + (mvCLX[1] \gg 3) + y_C \quad (8-13313394)$$

$$x_{FracC} = mvLX[0] \& 7 \quad (8-13413495)$$

$$y_{FracC} = mvLX[1] \& 7 \quad (8-13513596)$$

- The prediction sample value $predSampleLX_{Cb}[x_C, y_C]$ is derived by invoking the process specified in subclause 8.4.2.2.2 with (x_{IntC} , y_{IntC}), (x_{FracC} , y_{FracC}) and $refPicLX_{Cb}$ given as input.
- The prediction sample value $predSampleLX_{Cr}[x_C, y_C]$ is derived by invoking the process specified in subclause 8.4.2.2.2 with (x_{IntC} , y_{IntC}), (x_{FracC} , y_{FracC}) and $refPicLX_{Cr}$ given as input.

8.4.2.2.2.1 Luma sample interpolation process

Inputs to this process are:

- a luma location in full-sample units (x_{IntL} , y_{IntL}),
- a luma location in fractional-sample units (x_{FracL} , y_{FracL}),
- the luma reference sample array $refPicLX_L$.

Output of this process is a predicted luma sample value $predSampleLX_L[x_L, y_L]$

A _{-1,-1}				A _{0,-1}	a _{0,-1}	b _{0,-1}	c _{0,-1}	A _{1,-1}				A _{2,-1}
A _{-1,0}				A _{0,0}	a _{0,0}	b _{0,0}	c _{0,0}	A _{1,0}				A _{2,0}
d _{-1,0}				d _{0,0}	e _{0,0}	f _{0,0}	g _{0,0}	d _{1,0}				d _{2,0}
h _{-1,0}				h _{0,0}	i _{0,0}	j _{0,0}	k _{0,0}	h _{1,0}				h _{2,0}
n _{-1,0}				n _{0,0}	p _{0,0}	q _{0,0}	r _{0,0}	n _{1,0}				n _{2,0}
A _{-1,1}				A _{0,1}	a _{0,1}	b _{0,1}	c _{0,1}	A _{1,1}				A _{2,1}
A _{-1,2}				A _{0,2}	a _{0,2}	b _{0,2}	c _{0,2}	A _{1,2}				A _{2,2}

Figure 8-6 – Integer samples (shaded blocks with upper-case letters) and fractional sample positions (un-shaded blocks with lower-case letters) for quarter sample luma interpolation

In Figure 8-6, the positions labelled with upper-case letters $A_{i,j}$ within shaded blocks represent luma samples at full-sample locations inside the given two-dimensional array refPicXL of luma samples. These samples may be used for generating the predicted luma sample value $\text{predSampleXL}[x_L, y_L]$. The locations $(x_{A_{i,j}}, y_{A_{i,j}})$ for each of the corresponding luma samples $A_{i,j}$ inside the given array refPicXL of luma samples are derived as follows:

$$x_{A_{i,j}} = \text{Clip3}(0, \text{PicWidthInSamples}_L - 1, x_{\text{IntL}} + i) \quad (8-13613697)$$

$$y_{A_{i,j}} = \text{Clip3}(0, \text{PicHeightInSamples}_L - 1, y_{\text{IntL}} + j) \quad (8-13713798)$$

Variables shift1 , shift2 , shift3 , offset1 and offset2 are derived as follows.

- The variable shift1 is set equal to $\text{BitDepth}_Y - 8$, the variable shift2 is set equal to $\text{BitDepth}_Y - 2$, and the variable shift3 is set equal to $14 - \text{BitDepth}_Y$.
- If the variable shift1 is equal to 0, the variable offset1 is set equal to 0, otherwise, the variable offset1 is set equal to $1 \ll (\text{shift1} - 1)$.
- The variable offset2 is set equal to $1 \ll (\text{shift2} - 1)$.

Given the luma samples $A_{i,j}$ at full-sample locations $(x_{A_{i,j}}, y_{A_{i,j}})$, the luma samples 'a_{0,0}' to 'r_{0,0}' at fractional sample positions are derived by the following rules.

- The samples labelled $a_{0,0}$, $b_{0,0}$, $c_{0,0}$, $d_{0,0}$, $h_{0,0}$, and $n_{0,0}$ shall be derived by applying the 8-tap filter to the nearest integer position samples and clipping the filtered value:

$$a_{0,0} = (-A_{-3,0} + 4*A_{-2,0} - 10*A_{-1,0} + 57*A_{0,0} + 19*A_{1,0} - 7*A_{2,0} + 3*A_{3,0} - A_{4,0} + \text{offset1}) \gg \text{shift1} \quad (8-13813899)$$

$$b_{0,0} = (-A_{-3,0} + 4*A_{-2,0} - 11*A_{-1,0} + 40*A_{0,0} + 40*A_{1,0} - 11*A_{2,0} + 4*A_{3,0} - A_{4,0} + \text{offset1}) \gg \text{shift1} \quad (8-139139100)$$

$$c_{0,0} = (-A_{-3,0} + 3*A_{-2,0} - 7*A_{-1,0} + 19*A_{0,0} + 57*A_{1,0} - 10*A_{2,0} + 4*A_{3,0} - A_{4,0} + \text{offset1}) \gg \text{shift1} \quad (8-140140101)$$

HEVC

$$d_{0,0} = (-A_{0,-3} + 4*A_{0,-2} - 10*A_{0,-1} + 57*A_{0,0} + 19*A_{0,1} - 7*A_{0,2} + 3*A_{0,3} - A_{0,4} + \text{offset1}) \gg \text{shift1} \quad (8-141141102)$$

$$h_{0,0} = (-A_{0,-3} + 4*A_{0,-2} - 11*A_{0,-1} + 40*A_{0,0} + 40*A_{0,1} - 11*A_{0,2} + 4*A_{0,3} - A_{0,4} + \text{offset1}) \gg \text{shift1} \quad (8-142142103)$$

$$n_{0,0} = (-A_{0,-3} + 3*A_{0,-2} - 7*A_{0,-1} + 19*A_{0,0} + 57*A_{0,1} - 10*A_{0,2} + 4*A_{0,3} - A_{0,4} + \text{offset1}) \gg \text{shift1} \quad (8-143143104)$$

- The samples labelled $e_{0,0}$, $f_{0,0}$, $g_{0,0}$, $i_{0,0}$, $j_{0,0}$, $k_{0,0}$, $p_{0,0}$, $q_{0,0}$ and $r_{0,0}$ shall be derived by first calculating intermediate values denoted as $d_{1,i,0}$, $h_{1,i,0}$ and $n_{1,i,0}$ where $i = -3..4$ by applying the 8-tap filter to the nearest integer position samples in vertical direction: [Ed: (WJ) horizontal first yields the same result]

$$d_{1,i,0} = -A_{i,-3} + 4*A_{i,-2} - 10*A_{i,-1} + 57*A_{i,0} + 19*A_{i,1} - 7*A_{i,2} + 3*A_{i,3} - A_{i,4} \quad (8-144144105)$$

$$h_{1,i,0} = -A_{i,-3} + 4*A_{i,-2} - 11*A_{i,-1} + 40*A_{i,0} + 40*A_{i,1} - 11*A_{i,2} + 4*A_{i,3} - A_{i,4} \quad (8-145145106)$$

$$n_{1,i,0} = -A_{i,-3} + 3*A_{i,-2} - 7*A_{i,-1} + 19*A_{i,0} + 57*A_{i,1} - 10*A_{i,2} + 4*A_{i,3} - A_{i,4} \quad (8-146146107)$$

- The final prediction values $e_{0,0}$, $f_{0,0}$, $g_{0,0}$, $i_{0,0}$, $j_{0,0}$, $k_{0,0}$, $p_{0,0}$, $q_{0,0}$ and $r_{0,0}$ shall be derived by applying the 8-tap filter to the intermediate values $d_{1,i,0}$, $h_{1,i,0}$ and $n_{1,i,0}$ where $i = -3..4$ in horizontal direction:

$$e_{0,0} = (-d_{1,-3,0} + 4*d_{1,-2,0} - 10*d_{1,-1,0} + 57*d_{1,0,0} + 19*d_{1,1,0} - 7*d_{1,2,0} + 3*d_{1,3,0} - d_{1,4,0} + \text{offset2}) \gg \text{shift2} \quad (8-147147108)$$

$$f_{0,0} = (-d_{1,-3,0} + 4*d_{1,-2,0} - 11*d_{1,-1,0} + 40*d_{1,0,0} + 40*d_{1,1,0} - 11*d_{1,2,0} + 4*d_{1,3,0} - d_{1,4,0} + \text{offset2}) \gg \text{shift2} \quad (8-148148109)$$

$$g_{0,0} = (-d_{1,-3,0} + 3*d_{1,-2,0} - 7*d_{1,-1,0} + 19*d_{1,0,0} + 57*d_{1,1,0} - 10*d_{1,2,0} + 4*d_{1,3,0} - d_{1,4,0} + \text{offset2}) \gg \text{shift2} \quad (8-149149110)$$

$$i_{0,0} = (-h_{1,-3,0} + 4*h_{1,-2,0} - 10*h_{1,-1,0} + 57*h_{1,0,0} + 19*h_{1,1,0} - 7*h_{1,2,0} + 3*h_{1,3,0} - h_{1,4,0} + \text{offset2}) \gg \text{shift2} \quad (8-150150111)$$

$$j_{0,0} = (-h_{1,-3,0} + 4*h_{1,-2,0} - 11*h_{1,-1,0} + 40*h_{1,0,0} + 40*h_{1,1,0} - 11*h_{1,2,0} + 4*h_{1,3,0} - h_{1,4,0} + \text{offset2}) \gg \text{shift2} \quad (8-151151112)$$

$$k_{0,0} = (-h_{1,-3,0} + 3*h_{1,-2,0} - 7*h_{1,-1,0} + 19*h_{1,0,0} + 57*h_{1,1,0} - 10*h_{1,2,0} + 4*h_{1,3,0} - h_{1,4,0} + \text{offset2}) \gg \text{shift2} \quad (8-152152113)$$

$$p_{0,0} = (-n_{1,-3,0} + 4*n_{1,-2,0} - 10*n_{1,-1,0} + 57*n_{1,0,0} + 19*n_{1,1,0} - 7*n_{1,2,0} + 3*n_{1,3,0} - n_{1,4,0} + \text{offset2}) \gg \text{shift2} \quad (8-153153114)$$

$$q_{0,0} = (-n_{1,-3,0} + 4*n_{1,-2,0} - 11*n_{1,-1,0} + 40*n_{1,0,0} + 40*n_{1,1,0} - 11*n_{1,2,0} + 4*n_{1,3,0} - n_{1,4,0} + \text{offset2}) \gg \text{shift2} \quad (8-154154115)$$

$$r_{0,0} = (-n_{1,-3,0} + 3*n_{1,-2,0} - 7*n_{1,-1,0} + 19*n_{1,0,0} + 57*n_{1,1,0} - 10*n_{1,2,0} + 4*n_{1,3,0} - n_{1,4,0} + \text{offset2}) \gg \text{shift2} \quad (8-155155116)$$

The positions labelled with lower-case letters within un-shaded blocks represent luma samples at quarter-pel sample fractional locations. The luma location offset in fractional-sample units (x_{FracL} , y_{FracL}) specifies which of the generated luma samples at full-sample and fractional-sample locations is assigned to the predicted luma sample value $\text{predSampleLXL}[x_L, y_L]$. This assignment is done according to [Table 8-11](#)/[Table 8-9](#). The value of $\text{predSampleLXL}[x_L, y_L]$ shall be the output.

Table 8-11/**8-9** – Assignment of the luma prediction sample $\text{predSampleLXL}[x_L, y_L]$

xFracL	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3
yFracL	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
predSampleLXL[x_L, y_L]	A << shift3	d	h	n	a	e	i	p	b	f	j	q	c	g	k	r

8.4.2.2.2 Chroma sample interpolation process

Inputs to this process are:

HEVC

- a chroma location in full-sample units (x_{Intc} , y_{Intc}),
- a chroma location in fractional-sample units (x_{Fracc} , y_{Fracc}),
- the chroma reference sample array $refPicLXC$.

Output of this process is a predicted chroma sample value $predSampleLXC[x_c, y_c]$

	$ha_{0,-1}$	$hb_{0,-1}$	$hc_{0,-1}$	$hd_{0,-1}$	$he_{0,-1}$	$hf_{0,-1}$	$hg_{0,-1}$	$hh_{0,-1}$	
$ah_{-1,0}$	$B_{0,0}$	$ab_{0,0}$	$ac_{0,0}$	$ad_{0,0}$	$ae_{0,0}$	$af_{0,0}$	$ag_{0,0}$	$ah_{0,0}$	$B_{1,0}$
$bh_{-1,0}$	$ba_{0,0}$	$bb_{0,0}$	$bc_{0,0}$	$bd_{0,0}$	$be_{0,0}$	$bf_{0,0}$	$bg_{0,0}$	$bh_{0,0}$	$ba_{1,0}$
$ch_{-1,0}$	$ca_{0,0}$	$cb_{0,0}$	$cc_{0,0}$	$cd_{0,0}$	$ce_{0,0}$	$cf_{0,0}$	$cg_{0,0}$	$ch_{0,0}$	$ca_{1,0}$
$dh_{-1,0}$	$da_{0,0}$	$db_{0,0}$	$dc_{0,0}$	$dd_{0,0}$	$de_{0,0}$	$df_{0,0}$	$dg_{0,0}$	$dh_{0,0}$	$da_{1,0}$
$eh_{-1,0}$	$ea_{0,0}$	$eb_{0,0}$	$ec_{0,0}$	$ed_{0,0}$	$ee_{0,0}$	$ef_{0,0}$	$eg_{0,0}$	$eh_{0,0}$	$ea_{1,0}$
$fh_{-1,0}$	$fa_{0,0}$	$fb_{0,0}$	$fc_{0,0}$	$fd_{0,0}$	$fe_{0,0}$	$ff_{0,0}$	$fg_{0,0}$	$fh_{0,0}$	$fa_{1,0}$
$gh_{-1,0}$	$ga_{0,0}$	$gb_{0,0}$	$gc_{0,0}$	$gd_{0,0}$	$ge_{0,0}$	$gf_{0,0}$	$gg_{0,0}$	$gh_{0,0}$	$ga_{1,0}$
$hh_{-1,0}$	$ha_{0,0}$	$hb_{0,0}$	$hc_{0,0}$	$hd_{0,0}$	$he_{0,0}$	$hf_{0,0}$	$hg_{0,0}$	$hh_{0,0}$	$ha_{1,0}$
	$B_{0,1}$	$ab_{0,1}$	$ac_{0,1}$	$ad_{0,1}$	$ae_{0,1}$	$af_{0,1}$	$ag_{0,1}$	$ah_{0,1}$	$B_{1,1}$

Figure 8-7 – Integer samples (shaded blocks with upper-case letters) and fractional sample positions (un-shaded blocks with lower-case letters) for eighth sample chroma interpolation

In [Figure 8-6](#)[Figure 8-5](#), the positions labelled with upper-case letters $B_{i,j}$ within shaded blocks represent chroma samples at full-sample locations inside the given two-dimensional array $refPicLXC$ of chroma samples. These samples may be used for generating the predicted chroma sample value $predSampleLXC[x_c, y_c]$. The locations ($x_{B_{i,j}}$, $y_{B_{i,j}}$) for each of the corresponding chroma samples $B_{i,j}$ inside the given array $refPicLXC$ of chroma samples are derived as follows:

$$x_{B_{i,j}} = \text{Clip3}(0, \text{PicWidthInSamples}_c - 1, x_{Intc} + i) \quad (8-156156117)$$

$$y_{B_{i,j}} = \text{Clip3}(0, \text{PicHeightInSamples}_c - 1, y_{Intc} + j) \quad (8-157157118)$$

Variables $shift1$, $shift2$, $shift3$, $offset1$ and $offset2$ are derived as follows.

- The variable $shift1$ is set equal to $\text{BitDepth}_c - 8$, the variable $shift2$ is set equal to $\text{BitDepth}_c - 2$, and the variable $shift3$ is set equal to $14 - \text{BitDepth}_c$.
- If the variable $shift1$ is equal to 0, the variable $offset1$ is set equal to 0, otherwise, the variable $offset1$ is set equal to $1 \ll (shift1 - 1)$.
- The variable $offset2$ is set equal to $1 \ll (shift2 - 1)$.

Given the chroma samples $B_{i,j}$ at full-sample locations ($x_{B_{i,j}}$, $y_{B_{i,j}}$), the chroma samples ‘ $ab_{0,0}$ ’ to ‘ $hh_{0,0}$ ’ at fractional sample positions are derived by the following rules.

- The samples labelled $ab_{0,0}$, $ac_{0,0}$, $ad_{0,0}$, $ae_{0,0}$, $af_{0,0}$, $ag_{0,0}$, and $ah_{0,0}$ shall be derived by applying the 4-tap filter to the nearest integer position samples and clipping the filtered value:

HEVC

$$ab_{0,0} = (-3*B_{-1,0} + 60*B_{0,0} + 8*B_{1,0} - B_{2,0} + offset1) \gg shift1 \quad (8-158158119)$$

$$ac_{0,0} = (-4*B_{-1,0} + 54*B_{0,0} + 16*B_{1,0} - 2*B_{2,0} + offset1) \gg shift1 \quad (8-159159120)$$

$$ad_{0,0} = (-5*B_{-1,0} + 46*B_{0,0} + 27*B_{1,0} - 4*B_{2,0} + offset1) \gg shift1 \quad (8-160160121)$$

$$ae_{0,0} = (-4*B_{-1,0} + 36*B_{0,0} + 36*B_{1,0} - 4*B_{2,0} + offset1) \gg shift1 \quad (8-161161122)$$

$$af_{0,0} = (-4*B_{-1,0} + 27*B_{0,0} + 46*B_{1,0} - 5*B_{2,0} + offset1) \gg shift1 \quad (8-162162123)$$

$$ag_{0,0} = (-2*B_{-1,0} + 16*B_{0,0} + 54*B_{1,0} - 4*B_{2,0} + offset1) \gg shift1 \quad (8-163163124)$$

$$ah_{0,0} = (-B_{-1,0} + 8*B_{0,0} + 60*B_{1,0} - 3*B_{2,0} + offset1) \gg shift1 \quad (8-164164125)$$

- The samples labelled $ba_{0,0}$, $ca_{0,0}$, $da_{0,0}$, $ea_{0,0}$, $fa_{0,0}$, $ga_{0,0}$, and $ha_{0,0}$ shall be derived by applying the 4-tap filter to the nearest integer position samples and clipping the filtered value:

$$ba_{0,0} = (-3*B_{0,-1} + 60*B_{0,0} + 8*B_{0,1} - B_{0,2} + offset1) \gg shift1 \quad (8-165165126)$$

$$ca_{0,0} = (-4*B_{0,-1} + 54*B_{0,0} + 16*B_{0,1} - 2*B_{0,2} + offset1) \gg shift1 \quad (8-166166127)$$

$$da_{0,0} = (-5*B_{0,-1} + 46*B_{0,0} + 27*B_{0,1} - 4*B_{0,2} + offset1) \gg shift1 \quad (8-167167128)$$

$$ea_{0,0} = (-4*B_{0,-1} + 36*B_{0,0} + 36*B_{0,1} - 4*B_{0,2} + offset1) \gg shift1 \quad (8-168168129)$$

$$fa_{0,0} = (-4*B_{0,-1} + 27*B_{0,0} + 46*B_{0,1} - 5*B_{0,2} + offset1) \gg shift1 \quad (8-169169130)$$

$$ga_{0,0} = (-2*B_{0,-1} + 16*B_{0,0} + 54*B_{0,1} - 4*B_{0,2} + offset1) \gg shift1 \quad (8-170170131)$$

$$ha_{0,0} = (-B_{0,-1} + 8*B_{0,0} + 60*B_{0,1} - 3*B_{0,2} + offset1) \gg shift1 \quad (8-171171132)$$

- The samples labelled $Xb_{0,0}$, $Xc_{0,0}$, $Xd_{0,0}$, $Xe_{0,0}$, $Xf_{0,0}$, $Xg_{0,0}$ and $Xh_{0,0}$ for X being replaced by b, c, d, e, f, g and h, respectively, shall be derived by first calculating intermediate values denoted as $ba_{i,0}$, $ca_{i,0}$, $da_{i,0}$, $ea_{i,0}$, $fa_{i,0}$, $ga_{i,0}$ and $ha_{i,0}$ where $i = -1..2$ by applying the 4-tap filter to the nearest integer position samples in vertical direction: [Ed: (WJ) horizontal first yields the same result]

$$ba_{i,0} = -3*B_{0,-1} + 60*B_{0,0} + 8*B_{0,1} - B_{0,2} \quad (8-172172133)$$

$$ca_{i,0} = -4*B_{0,-1} + 54*B_{0,0} + 16*B_{0,1} - 2*B_{0,2} \quad (8-173173134)$$

$$da_{i,0} = -5*B_{0,-1} + 46*B_{0,0} + 27*B_{0,1} - 4*B_{0,2} \quad (8-174174135)$$

$$ea_{i,0} = -4*B_{0,-1} + 36*B_{0,0} + 36*B_{0,1} - 4*B_{0,2} \quad (8-175175136)$$

$$fa_{i,0} = -4*B_{0,-1} + 27*B_{0,0} + 46*B_{0,1} - 5*B_{0,2} \quad (8-176176137)$$

$$ga_{i,0} = -2*B_{0,-1} + 16*B_{0,0} + 54*B_{0,1} - 4*B_{0,2} \quad (8-177177138)$$

$$ha_{i,0} = -B_{0,-1} + 8*B_{0,0} + 60*B_{0,1} - 3*B_{0,2} \quad (8-178178139)$$

- The final prediction values $Xb_{0,0}$, $Xc_{0,0}$, $Xd_{0,0}$, $Xe_{0,0}$, $Xf_{0,0}$, $Xg_{0,0}$ and $Xh_{0,0}$ for X being replaced by b, c, d, e, f, g and h, respectively, shall be derived by applying the 4-tap filter to the intermediate values $Xa_{i,0}$ where $i = -1..2$ in horizontal direction:

$$Xb_{0,0} = (-3*Xa_{-1,0} + 60*Xa_{0,0} + 8*Xa_{1,0} - Xa_{2,0} + offset2) \gg shift2 \quad (8-179179140)$$

$$Xc_{0,0} = (-4*Xa_{-1,0} + 54*Xa_{0,0} + 16*Xa_{1,0} - 2*Xa_{2,0} + offset2) \gg shift2 \quad (8-180180141)$$

$$Xd_{0,0} = (-5*Xa_{-1,0} + 46*Xa_{0,0} + 27*Xa_{1,0} - 4*Xa_{2,0} + offset2) \gg shift2 \quad (8-181181142)$$

$$Xe_{0,0} = (-4*Xa_{-1,0} + 36*Xa_{0,0} + 36*Xa_{1,0} - 4*Xa_{2,0} + offset2) \gg shift2 \quad (8-182182143)$$

$$Xf_{0,0} = (-4*Xa_{-1,0} + 27*Xa_{0,0} + 46*Xa_{1,0} - 5*Xa_{2,0} + offset2) \gg shift2 \quad (8-183183144)$$

HEVC

$$X_{g_{0,0}} = (-2 * X_{a_{-1,0}} + 16 * X_{a_{0,0}} + 54 * X_{a_{1,0}} - 4 * X_{a_{2,0}} + \text{offset2}) \gg \text{shift2} \quad (8-184184145)$$

$$X_{h_{0,0}} = (-X_{a_{-1,0}} + 8 * X_{a_{0,0}} + 60 * X_{a_{1,0}} - 3 * X_{a_{2,0}} + \text{offset2}) \gg \text{shift2} \quad (8-185185146)$$

The positions labelled with lower-case letters within un-shaded blocks represent chroma samples at eighth-pel sample fractional locations. The chroma location offset in fractional-sample units (xFracC, yFracC) specifies which of the generated chroma samples at full-sample and fractional-sample locations is assigned to the predicted chroma sample value predSampleLXC[xc, yc]. This assignment is done according to [Table 8-12](#)/[Table 8-10](#). The value of predSampleLXC[xc, yc] shall be the output.

Table 8-128-108-10 – Assignment of the chroma prediction sample predSampleLXC[xc, yc] for (X, Y) being replaced by (1, b), (2, c), (3, d), (4, e), (5, f), (6, g), and (7, h), respectively

xFracC	0	0	0	0	0	0	0	0	0	X	X	X	X	X	X	X
yFracC	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
predSampleLXC[xc, yc]	B << shift3	ba	ca	da	ea	fa	ga	ha	aY	bY	cY	dY	eY	fY	gY	hY

8.4.2.2.3 Weighted sample prediction process

[Ed.: (KU) only the default weighted prediction part is added for bi-directional averaging]

Inputs to this process are:

- a location (xB, yB) specifying the top-left sample of the current prediction unit relative to the top left sample of the current coding unit,
- the width and height of this prediction unit, nPSW and nPSH,
- two (nPSW)x(nPSH) arrays predSamplesL0 and predSamplesL1,
- prediction list utilization flags, predFlagL0 and predFlagL1,
- the bit-depth of the chroma component, bitDepth.

Outputs of this process are:

- the (nPSW)x(nPSH) array predSamples of prediction sample values.

Variables shift1, shift2, offset1 and offset2 are derived as follows.

- The variable shift1 is set equal to 14 – bitDepth and the variable shift2 is set equal to 15 – bitDepth,
- The variable offset1 is set equal to 1 << (shift1 – 1) and the variable offset2 is set equal to 1 << (shift2 – 1).

Depending on the value of predFlagL0 and predFlagL1, the prediction samples predSamples[x, y] with x = 0..(nPSW)-1 and y = 0..(nPSH)-1 are derived as follows.

- If predFlagL0 is equal to 1 and predFlagL1 is equal to 0,

$$\text{predSamples}[x, y] = (\text{predSamplesL0}[x, y] + \text{offset1}) \gg \text{shift1} \quad (8-186186147)$$

- Otherwise, if predFlagL0 is equal to 0 and predFlagL1 is equal to 1,

$$\text{predSamples}[x, y] = (\text{predSamplesL1}[x, y] + \text{offset1}) \gg \text{shift1} \quad (8-187187148)$$

- Otherwise (both predFlagL0 and predFlagL1 are equal to 1),

$$\text{predSamples}[x, y] = \text{Clip3}(0, (1 \ll \text{bitDepth}) - 1, \text{predSamplesL0}[x, y] + \text{predSamplesL1}[x, y] + \text{offset2}) \gg \text{shift2} \quad (8-188188149)$$

8.4.3 Decoding process for the residual signal of coding units coded in inter prediction mode

Inputs to this process are:

- a luma location (xC, yC) specifying the top-left luma sample of the current coding unit relative to the top-left luma sample of the current picture,
- a variable log2CUSize specifying the size of the current coding unit.

HEVC

Outputs of this process are:

- a $(nCS_L) \times (nCS_L)$ array $resSamples_L$ of luma residual samples, where nCS_L is derived as specified below,
- a $(nCS_C) \times (nCS_C)$ array $resSamples_{Cb}$ of chroma residual samples for the component Cb, where nCS_C is derived as specified below,
- a $(nCS_C) \times (nCS_C)$ array $resSamples_{Cr}$ of chroma residual samples for the component Cr, where nCS_C is derived as specified below.

The variable nCS_L is set equal to $1 \ll \log_2 CUSize$ and the variable nCS_C is set equal to $(1 \ll \log_2 CUSize) \gg 1$.

Let $resSamples_L$ be a $(nCS_L) \times (nCS_L)$ array of luma residual samples and let $resSamples_{Cb}$ and $resSamples_{Cr}$ be two $(nCS_C) \times (nCS_C)$ arrays of chroma residual samples.

Depending on `no_residual_data_flag`, the following applies:

- If `no_residual_data_flag` is equal to 1, all samples of the $(nCS_L) \times (nCS_L)$ array $resSamples_L$ and all samples of the two $(nCS_C) \times (nCS_C)$ arrays $resSamples_{Cb}$ and $resSamples_{Cr}$ are set equal to 0.
- Otherwise (`no_residual_data_flag` is equal to 0), the following ordered steps apply:
 1. The decoding process for luma residual blocks as specified in subclause 8.4.3.1 below is invoked with the luma location (x_C, y_C) , the luma location (x_B, y_B) set equal to $(0, 0)$, the variable `log2TrafoSize` set equal to `log2CUSize`, the variable `trafoDepth` set equal to 0, the variable `nCS` set equal to nCS_L , and the $(nCS_L) \times (nCS_L)$ array $resSamples_L$ as the inputs and the output is a modified version of the $(nCS_L) \times (nCS_L)$ array $resSamples_L$.
 2. The decoding process for chroma residual blocks as specified in subclause 8.4.3.2 below is invoked with the luma location (x_C, y_C) , the luma location (x_B, y_B) set equal to $(0, 0)$, the variable `log2TrafoSize` set equal to `log2CUSize`, the variable `trafoDepth` set equal to 0, the variable `cldx` set equal to 1, the variable `nCS` set equal to nCS_C , and the $(nCS_C) \times (nCS_C)$ array $resSamples_{Cb}$ as the inputs and the output is a modified version of the $(nCS_C) \times (nCS_C)$ array $resSamples_{Cb}$.
 3. The decoding process for chroma residual blocks as specified in subclause 8.4.3.2 below is invoked with the luma location (x_C, y_C) , the luma location (x_B, y_B) set equal to $(0, 0)$, the variable `log2TrafoSize` set equal to `log2CUSize`, the variable `trafoDepth` set equal to 0, the variable `cldx` set equal to 2, the variable `nCS` set equal to nCS_C , and the $(nCS_C) \times (nCS_C)$ array $resSamples_{Cr}$ as the inputs and the output is a modified version of the $(nCS_C) \times (nCS_C)$ array $resSamples_{Cr}$.

8.4.3.1 Decoding process for luma residual blocks

Inputs to this process are:

- a luma location (x_C, y_C) specifying the top-left luma sample of the current coding unit relative to the top-left luma sample of the current picture,
- a luma location (x_B, y_B) specifying the top-left luma sample of the current block relative to the top-left luma sample of the current coding unit,
- a variable `log2TrafoSize` specifying the size of the current block,
- a variable `trafoDepth` specifying the hierarchy depth of the current block relative to the coding unit,
- a variable `nCS` specifying the size, in luma samples, of the current coding unit,
- a $(nCS) \times (nCS)$ array $resSamples$ of luma residual samples.

Output of this process is:

- a modified version of the $(nCS) \times (nCS)$ array of luma residual samples.

Depending `split_transform_flag[xB][yB][trafoDepth]`, the following applies:

- If `split_transform_flag[xB][yB][trafoDepth]` is equal to 1, the following ordered steps apply:
 1. The variable `xB1` is set equal to $x_B + ((1 \ll \log_2 TrafoSize) \gg 1)$.
 2. The variable `yB1` is set equal to $y_B + ((1 \ll \log_2 TrafoSize) \gg 1)$.
 3. The decoding process for luma residual blocks as specified in this subclause is invoked with the luma location (x_C, y_C) , the luma location (x_B, y_B) , the variable `log2TrafoSize` set equal to `log2TrafoSize - 1`, the variable `trafoDepth` set equal to `trafoDepth + 1`, the variable `nCS`, and the $(nCS) \times (nCS)$ array $resSamples$ as the inputs and the output is a modified version of the $(nCS) \times (nCS)$ array $resSamples$.

HEVC

4. The decoding process for luma residual blocks as specified in this subclause is invoked with the luma location (x_C, y_C), the luma location (x_{B1}, y_{B1}), the variable $\log_2\text{TrafoSize}$ set equal to $\log_2\text{TrafoSize} - 1$, the variable trafoDepth set equal to $\text{trafoDepth} + 1$, the variable nCS , and the (nCS) \times (nCS) array resSamples as the inputs and the output is a modified version of the (nCS) \times (nCS) array resSamples .
 5. The decoding process for luma residual blocks as specified in this subclause is invoked with the luma location (x_C, y_C), the luma location (x_B, y_{B1}), the variable $\log_2\text{TrafoSize}$ set equal to $\log_2\text{TrafoSize} - 1$, the variable trafoDepth set equal to $\text{trafoDepth} + 1$, the variable nCS , and the (nCS) \times (nCS) array resSamples as the inputs and the output is a modified version of the (nCS) \times (nCS) array resSamples .
 6. The decoding process for luma residual blocks as specified in this subclause is invoked with the luma location (x_C, y_C), the luma location (x_{B1}, y_{B1}), the variable $\log_2\text{TrafoSize}$ set equal to $\log_2\text{TrafoSize} - 1$, the variable trafoDepth set equal to $\text{trafoDepth} + 1$, the variable nCS , and the (nCS) \times (nCS) array resSamples as the inputs and the output is a modified version of the (nCS) \times (nCS) array resSamples .
- Otherwise ($\text{split_transform_flag}[x_B][y_B][\text{trafoDepth}]$ is equal to 0), the following ordered steps apply:
1. The variable nS is set equal to $1 \ll \log_2\text{TrafoSize}$.
 2. The scaling and transformation process as specified in subclause 8.5.18-5-4 is invoked with the luma location ($x_C + x_B, y_C + y_B$), the variable trafoDepth , the variable cIdx set equal to 0, and the transform size trafoSize set equal to nS as the inputs and the output is a (nS) \times (nS) array resSamplesBlock .
 3. The array construction process as specified in subclause 8.5.5 is invoked with the luma location (x_B, y_B), the variable cIdx set equal to 0, the variable inputArraySize set equal to nS , the variable outputArraySize set equal to nCS , the (nS) \times (nS) array resSamplesBlock , and the (nCS) \times (nCS) array resSamples as the inputs and the output is a modified version of the (nCS) \times (nCS) array resSamples .

8.4.3.2 Decoding process for chroma residual blocks

Inputs to this process are:

- a luma location (x_C, y_C) specifying the top-left luma sample of the current coding unit relative to the top-left luma sample of the current picture,
- a luma location (x_B, y_B) specifying the top-left luma sample of the current block relative to the top-left luma sample of the current coding unit,
- a variable $\log_2\text{TrafoSize}$ specifying the size of the current block,
- a variable trafoDepth specifying the hierarchy depth of the current block relative to the coding unit,
- a variable cIdx specifying the chroma component of the current block,
- a variable nCS specifying the size, in chroma samples, of the current coding unit,
- a (nCS) \times (nCS) array resSamples of chroma residual samples.

Output of this process is:

- a modified version of the (nCS) \times (nCS) array of chroma residual samples.

The variable splitChromaFlag is derived as follows:

- If $\text{split_transform_flag}[x_B][y_B][\text{trafoDepth}]$ is equal to 1 and $\log_2\text{TrafoSize}$ is greater than $\text{Log2MinTrafoSize} + 1$, splitChromaFlag is set equal to 1.
- Otherwise ($\text{split_transform_flag}[x_B][y_B][\text{trafoDepth}]$ is equal to 0 or $\log_2\text{TrafoSize}$ is equal to $\text{Log2MinTrafoSize} + 1$), splitChromaFlag is set equal to 0.

Depending splitChromaFlag , the following applies:

- If splitChromaFlag is equal to 1, the following ordered steps apply:
 1. The variable x_{B1} is set equal to $x_B + ((1 \ll \log_2\text{TrafoSize}) \gg 1)$.
 2. The variable y_{B1} is set equal to $y_B + ((1 \ll \log_2\text{TrafoSize}) \gg 1)$.
 3. The decoding process for residual chroma blocks as specified in this subclause is invoked with the luma location (x_C, y_C), the luma location (x_B, y_B), the variable $\log_2\text{TrafoSize}$ set equal to $\log_2\text{TrafoSize} - 1$, the variable trafoDepth set equal to $\text{trafoDepth} + 1$, the variable cIdx , the variable nCS , and the (nCS) \times (nCS) array resSamples as the inputs and the output is a modified version of the (nCS) \times (nCS) array resSamples .

HEVC

8.5.2 Inverse scanning process for transform coefficients

Inputs to this process are:

- a variable nS specifying the size of the current transform unit,
- a list of $(nS) \times (nS)$ values $\text{transCoeffLevel}[xT][yT][\text{trafoDepth}][cIdx]$.
- a variable $cIdx$ specifying the chroma component of the current block,

Output of this process is a variable c containing a two-dimensional array of $(nS) \times (nS)$ values.

- The variable $c[i][j]$ which is located in the (i, j) position in the array c is derived as follows.
- The current transform unit is divided into $(nS \gg 2) * (nS \gg 2)$ subsets of $4 * 4$ samples. The subsets are processed in zig-zag scan order. The position $(k0, l0)$ in the array c , corresponding to the top left position of a subset s , is derived as follows.

$$\begin{aligned} k0 &= \text{ZigZag}[\log_2(nS \gg 2)][s][0] \ll 2 \\ l0 &= \text{ZigZag}[\log_2(nS \gg 2)][s][1] \ll 2, \end{aligned}$$

$$\text{with } s = 0..((nS \gg 2) * (nS \gg 2)) - 1 \quad (8-191+91+52)$$

- For every subset s , the 16 transform coefficient level at scanning position n are mapped to the position (i, j) in the array c using the reverse zig-zag scan.

$$\begin{aligned} i &= k0 + \text{ZigZag}[2][n][0] \\ j &= l0 + \text{ZigZag}[2][n][1] \\ c[i][j] &= \text{transCoeffLevel}[xT][yT][\text{trafoDepth}][cIdx][n + (s \ll 4)], \end{aligned}$$

$$\text{with } n = 15..0 \quad (8-192+92+52)$$

8.5.3 Scaling process for transform coefficients

Inputs of this process are:

- a variable nS specifying the size of the current transform unit,
- a $(nS) \times (nS)$ array c of transform coefficients with elements c_{ij} ,
- a variable $cIdx$ specifying the chroma component of the current block,
- a variable qP specifying the quantization parameter.

Output of this process is scaled transform coefficients as a $(nS) \times (nS)$ array of d with elements d_{ij} .

The variable trafoPrecisionExt is derived as follows:

- If nS is greater than 4,
 - If $cIdx$ is equal to 0 and $\text{bit_depth_luma_minus8}$ is equal to 0, trafoPrecisionExt is set equal to 2.
 - Otherwise, if $cIdx$ is not equal to 0 and $\text{bit_depth_chroma_minus8}$ is equal to 0, trafoPrecisionExt is set equal to 2.
 - Otherwise, trafoPrecisionExt is set equal to 0.
- Otherwise, trafoPrecisionExt is set equal to 0.

[Ed.: (WJ) wait decision on transform precision extension]

The scaled transform coefficient array d_{ij} is derived as follows.

$$d_{ij} = (c_{ij} * \text{LevelScale}_{(nS) \times (nS)}[qP\%6][i][j]) \ll (qP/6 + \text{trafoPrecisionExt}), \text{ with } i, j = 0..nS-1 \quad (8-193+93+58)$$

[Ed.: (WJ) $\text{LevelScale}_{(nS) \times (nS)}$ should be very large table up to 32×32 . Recent contributions (JCTVC-C209 and JCTVC-C255) shown that this matrix could be replaced by one constants per qP without coding efficiency penalty. Maybe it is better to wait.]

[Ed.: (WJ) dependent on transformation process – should be modified again after transformation process is fixed]

8.5.4 Transformation process for scaled transform coefficients

Inputs of this process are:

HEVC

- a variable nS specifying the size of the current transform unit,
- a (nS)x(nS) array d of scaled transform coefficients with elements d_{ij}.
- a variable cIdx specifying the chroma component of the current block,

Output of this process is residual samples as a (nS)x(nS) array r with elements r_{ij}.

The variable trafoPrecisionExt is derived as follows:

- If nS is greater than 4,
 - If cIdx is equal to 0 and bit_depth_luma_minus8 is equal to 0, trafoPrecisionExt is set equal to 4.
 - Otherwise, if cIdx is not equal to 0 and bit_depth_chroma_minus8 is equal to 0, trafoPrecisionExt is set equal to 4.
 - Otherwise, trafoPrecisionExt is set equal to 0.
- Otherwise, trafoPrecisionExt is set equal to 0.

[Ed.: (WJ) wait decision on transform precision extension]

[Ed: (WJ) derivation of trafoPrecisionExt is duplicated to that of the scaling process. Better to remove one of two.]

~~Depending on cIdx, a variable planarFlag is derived as follows:~~

- ~~- If cIdx is equal to 0, planarFlag is set equal to planar_flag_luma~~
- ~~- Otherwise, planarFlag is set equal to planar_flag_chroma [Ed.: (WJ): it is not necessary now since DST is not applied to chroma anyway]~~

~~Depending on PredMode and IntraPredMode, the following applies:~~

- ~~- If PredMode is equal to MODE_INTRA, nS is equal to 4, and cIdx is equal to 0, the variables horizTrType and vertTrType are specified as Table 8-13Table 8-14 with IntraPredMode as input. [Ed. (WJ): DST is applied only for luma 4x4 block]~~
- ~~- Otherwise, the variables horizTrType and vertTrType are set equal to 0.~~

Table 8-138-11 – Specification of horizTrType and vertTrType

IntraPredMode	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
horizTrType	0	1	0	1	1	0	0	1	1	1	1	1	0	0	1	1	1	1
vertTrType	1	0	0	1	1	1	1	1	0	0	1	1	1	1	1	1	0	0

IntraPredMode	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
horizTrType	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1
vertTrType	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	1

The constructed residual samples are derived as specified in the following ordered steps.

1. Each (horizontal) row of scaled transform coefficients d_{ij} (i, j=0..nS-1) is transformed to e_{ij} (i, j=0..nS-1) by invoking the one-dimensional transformation process as specified in subclause 8.5.4.1 to 8.5.4.4 according to the size of the transform unit nS, with the (nS)x(nS) array d and the transform type variable horizTrType as the inputs and the output is the (nS)x(nS) array e.
2. Each (vertical) column of the resulting matrix e_{ij} (i, j=0..nS-1) is transformed to f_{ij} (i, j=0..nS-1) by invoking the one-dimensional transformation process as specified in subclause 8.5.4.1 to 8.5.4.4 according to the size of the transform unit nS, with the (nS)x(nS) array e and the transform type variable vertTrType as the inputs and the output is the (nS)x(nS) array f.
3. If nS is equal to 4 or 8, the residual sample value r_{ij} is derived by

$$r_{ij} = (f_{ij} + 2^{5+\text{trafoPrecisionExt}}) \gg (6 + \text{trafoPrecisionExt}), \text{ with } i, j = 0..(nS)x(nS)-1 \quad (8-194+194+59)$$

Formatted: Indent: Left: 0", Hanging: 0.3", Tab stops: 0.3", Left

Formatted: Caption

Formatted: Indent: Left: 0", Hanging: 0.3", Tab stops: 0.3", Left

HEVC

- a variable nS specifying the size of the current block,
- a variable nCS specifying the size of the current coding unit,
- a (nS)x(nS) array resSampleBlock specifying the current block samples,
- a (nCS)x(nCS) array resSample specifying the samples of the current coding unit.

Output of this process is a modified (nCS)x(nCS) array resSample specifying the samples of the current coding unit.

The modified array resSample is derived as follows.

$$\text{resSample}[xB+i, yB+j] += \text{resSampleBlock}[i, j], \text{ with } i, j = 0..nS-1 \quad (8-204204427)$$

8.6 In-loop filter process

[Ed: (WJ) overall process description need to be inserted]

8.6.1 Deblocking filter process

A conditional filtering process shall be performed on a treeblock basis after the completion of the picture construction process prior to deblocking filter process for the entire decoded picture (as specified in subclauses XXX and YYY) [Ed.: (WJ) those subclauses seem not defined yet], with all treeblocks in a picture processed in order of increasing treeblock addresses.

Each treeblock is processed on a coding unit basis with the same order as decoding process. For each coding unit, vertical edges are filtered first, starting with the edge on the left-hand side of the coding unit proceeding through the edges towards the right-hand side of the coding unit in their geometrical order. Then, the horizontal edges are filtered starting with the edge on the top of the coding unit proceeding through the edges towards the bottom of the coding unit in their geometrical order.

Sample values above of the current coding unit that may have already been modified by the filtering of horizontal edges of deblocking filter process operation on previous coding unit shall be used as inputs to the deblocking filter process on the current coding unit and may be further modified during the filtering of the current coding unit. Sample values to the left of the current coding unit shall be used as inputs to the deblocking filter process on the current coding unit and may be further modified during the filtering of the current coding unit. Sample values to the left of the current coding unit may be modified by the filtering of vertical edge and may be further modified by the filtering of horizontal edges.

Sample values modified during filtering of vertical edges are used as input for the filtering of the horizontal edges. For sample values modified by both filtering of horizontal edges and filtering of vertical edges, filtering of horizontal edges is applied after filtering of vertical edges.

The deblocking filter process shall be applied to all prediction unit edges and transform unit edges of a picture, except edges at the boundary of the picture—and, any edges for which the deblocking filter process is disabled by disable_deblocking_filter_idc and any edges coincide with slice boundaries when loop_filter_across_slice_flag is equal to 0. For the transform units and prediction units with edges smaller than 8 samples in either vertical or horizontal direction, only the edges lying on the 8x8 sample grid are filtered.

~~When disable_deblocking_filter_idc is not equal to 1, the deblocking filter process is invoked. If disable_deblocking_filter_idc is equal to 1, the deblocking filter process is simply omitted and if loop_filter_across_slice_flag is equal to 0, the deblocking filter process is omitted at slice boundaries, otherwise as the following ordered steps apply for each coding unit with the same order as decoding process.~~

1. The coding unit size nS is set equal to $1 \lll \log_2 \text{CuSize}$.
2. The variables FilterInternalEdgesFlag, FilterLeftCuEdgeFlag and FilterTopCuEdgeFlag are derived as follows.
 - The variable FilterInternalEdges is set equal to 1.
 - If the left boundary of current coding unit is the left boundary of the picture or if the left boundary of current coding unit is the left boundary of the slice and loop_filter_across_slice_flag is equal to 0, the variable FilterLeftCuEdgeFlag is set equal to 0, otherwise set equal to 1.
 - If the top boundary of current coding unit is the top boundary of the picture or if the top boundary of current coding unit is the top boundary of the slice and loop_filter_across_slice_flag is equal to 0, the variable FilterTopCuEdgeFlag is set equal to 0, otherwise set equal to 1.
3. All elements of two-dimensional array of size (nS)x(nS), horEdgeFlags and verEdgeFlags are initialized to zero.

HEVC

4. The derivation process of transform unit boundary specified in subclause 8.6.1.1 are invoked with the luma location (x_B, y_B) set equal to (0, 0), the transform unit size $\log_2\text{TrafoSize}$ set equal to $\log_2\text{CUSize}$ and the variable trafoDepth set equal to 0 as the inputs and the modified horEdgeFlags and verEdgeFlags as outputs.
5. The derivation process of prediction unit boundary specified in subclause 8.6.1.2 are invoked with the coding unit size $\log_2\text{CUSize}$ and the prediction partition mode PartMode as inputs, and the modified horEdgeFlags and verEdgeFlags as outputs.
6. The derivation process of the boundary filtering strength specified in subclause 8.6.1.3 is invoked with the luma location (x_C, y_C), the coding unit size $\log_2\text{CUSize}$, horEdgeFlags and verEdgeFlags as inputs and an array of size $(2 \times n_S) \times (n_S)$, b_S as output.
7. The filtering process for coding unit specified in subclause 8.6.1.4 are invoked with the luma location (x_C, y_C) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture, the coding unit size $\log_2\text{CUSize}$ and the array b_S as inputs and the modified reconstructed picture as output.

8.6.1.1 Derivation process of transform unit boundary

Inputs of this process are:

- a luma location (x_B, y_B) specifying the top-left luma sample of the current block relative to the top-left luma sample of the current coding unit,
- a transform unit size $\log_2\text{TrafoSize}$,
- a variable trafoDepth .

Outputs of this process are:

- two-dimensional arrays of $(n_S) \times (n_S)$, horEdgeFlags and verEdgeFlags .

Depending on $\text{split_transform_flag}[x_B][y_B][\text{trafoDepth}]$, the following applies:

- If $\text{split_transform_flag}[x_B][y_B][\text{trafoDepth}]$ is equal to 1, the following ordered steps apply:
 1. The variable x_{B1} is set equal to $x_B + ((1 \ll \log_2\text{TrafoSize}) \gg 1)$.
 2. The variable y_{B1} is set equal to $y_B + ((1 \ll \log_2\text{TrafoSize}) \gg 1)$.
 3. The derivation process of transform unit boundary as specified in this subclause is invoked with the luma location (x_B, y_B), the variable $\log_2\text{TrafoSize}1$ set equal to $\log_2\text{TrafoSize} - 1$ and the variable $\text{trafoDepth}1$ set equal to $\text{trafoDepth} + 1$ as inputs and the outputs are the modified versions of two arrays, horEdgeFlags and verEdgeFlags .
 4. The derivation process of transform unit boundary as specified in this subclause is invoked with the luma location (x_{B1}, y_{B1}), the variable $\log_2\text{TrafoSize}1$ set equal to $\log_2\text{TrafoSize} - 1$ and the variable $\text{trafoDepth}1$ set equal to $\text{trafoDepth} + 1$ as inputs and the outputs are the modified versions of two arrays, horEdgeFlags and verEdgeFlags .
 5. The derivation process of transform unit boundary as specified in this subclause is invoked with the luma location (x_B, y_{B1}), the variable $\log_2\text{TrafoSize}1$ set equal to $\log_2\text{TrafoSize} - 1$ and the variable $\text{trafoDepth}1$ set equal to $\text{trafoDepth} + 1$ as inputs and the outputs are the modified versions of two arrays, horEdgeFlags and verEdgeFlags .
 6. The derivation process of transform unit boundary as specified in this subclause is invoked with the luma location (x_{B1}, y_{B1}), the variable $\log_2\text{TrafoSize}1$ set equal to $\log_2\text{TrafoSize} - 1$ and the variable $\text{trafoDepth}1$ set equal to $\text{trafoDepth} + 1$ as inputs and the outputs are the modified versions of two arrays, horEdgeFlags and verEdgeFlags .
- Otherwise ($\text{split_transform_flag}[x_B][y_B][\text{trafoDepth}]$ is equal to 0), the following applies:
 - If y_B is equal to zero, $\text{horEdgeFlags}[x_B + k][y_B]$ is set equal to $\text{FilterTopCuEdgeFlag}$, otherwise $\text{horEdgeFlags}[x_B + k][y_B]$ is set equal to $\text{FilterInternalEdgesFlag}$ for $k = 0..(1 \ll \log_2\text{TrafoSize}) - 1$.
 - If x_B is equal to zero, $\text{verEdgeFlags}[x_B][y_B + k]$ is set equal to $\text{FilterLeftCuEdgeFlag}$, otherwise $\text{verEdgeFlags}[x_B][y_B + k]$ is set equal to $\text{FilterInternalEdgesFlag}$ for $k = 0..(1 \ll \log_2\text{TrafoSize}) - 1$.

8.6.1.2 Derivation process of prediction unit boundary

Inputs of this process are:

- a variable $\log_2\text{CUSize}$ specifying the coding unit size,

HEVC

- a prediction partition mode PartMode.

Outputs of this process are:

- two-dimensional arrays of (nS)x(nS), horEdgeFlags and verEdgeFlags.

Depending on PartMode, the following applies:

- If PartMode is equal to PART_2NxN or PART_NxN, horEdgeFlags[k][1 << (log2CUSize – 1)] is set equal to FilterInternalEdgesFlag for k = 0.. (1 << log2CUSize) – 1.
- If PartMode is equal to PART_Nx2N or PART_NxN, verEdgeFlags[1 << (log2CUSize – 1)][k] is set equal to FilterInternalEdgesFlag for k = 0.. (1 << log2CUSize) – 1.

8.6.1.3 Derivation process of boundary filtering strength

Inputs of this process are:

- a luma location (xC, yC) specifying the top-left luma sample of the current coding unit relative to the top-left luma sample of the current picture,
- a variable log2CUSize specifying the size of the current coding unit,
- a two-dimensional arrays of size (nS)x(nS), horEdgeFlags and verEdgeFlags.

Output of this process is an array of size (2)x(nS)x(nS), bS specifying the boundary filtering strength.

Let (xE_k, yE_j) with k = 0..nE-1 and j = 0..nE-1 specify a set of edge sample locations where nE is set equal to ((1 << log2CUSize) >> 2), xE₀ = 0, yE₀ = 0, xE_{k+1} = xE_k + 4 and yE_{j+1} = yE_j + 4.

For (xE_k, yE_j) with k = 0..nE-1 and j = 0..nE-1, the following applies.

- If horEdgeFlags[xE_k][yE_j] is equal to 1,
 - Set sample p₀ = recPicture[xC + xE_k][yC + yE_j – 1] and q₀ = recPicture[xC + xE_k][yC + yE_j].
 - The variable filterDir is set equal to 1.
- Otherwise, if verEdgeFlags[xE_k][yE_j] is equal to 1,
 - Set sample p₀ = recPicture[xC + xE_k – 1][yC + yE_j] and q₀ = recPicture[xC + xE_k][yC + yE_j].
 - The variable filterDir is set equal to 0.
- Depending on the value of filterDir, the variable bS[filterDir][xE_k][yE_j] is derived as follows.
 - If the block edge is also a coding unit edge and the following condition is true, the variable bS[0][xE_k][yE_j] is set equal to 4.
 - The sample p₀ or q₀ is in a coding unit coded with intra prediction mode
 - Otherwise, if the following condition is true, the variable bS[filterDir][xE_k][yE_j] is set equal to 3.
 - The sample p₀ or q₀ is in a coding unit coded with intra prediction mode
 - Otherwise, if the following condition is true, the variable bS[filterDir][xE_k][yE_j] is set equal to 2.
 - The sample p₀ or q₀ is in a transform unit which contains non-zero transform coefficient level.
 - Otherwise, if any of the following conditions are true, the variable bS[filterDir][xE_k][yE_j] is set equal to 1.
 - The prediction unit containing sample p₀ has different reference pictures or a different number of motion vectors with the prediction unit containing the sample q₀.
NOTE – The determination of whether the reference pictures used for the two prediction are the same or different is based on which pictures are referenced, without regard to whether a prediction is formed using an index into list 0 or an index into list 1, and also without regard to whether or not the index position within a reference picture list is different or not.
 - One motion vector is used to predict the prediction unit containing sample p₀, one motion vector is used to predict the prediction unit containing sample q₀, and the absolute difference between the horizontal or vertical component of the motion vector used is greater than or equal to 4 in units of quarter luma samples.
 - [Ed.: (WJ) needs to be checked again whether this condition covers all 2-motion cases] Two motion vectors are used to predict the prediction unit containing sample p₀, two motion vectors are used to predict

HEVC

the prediction unit containing sample q_0 , and at least one of the motion vector pairs corresponding the same reference pictures and the different boundary samples p_0 and q_0 satisfies the following condition:

1. The absolute difference between the horizontal or vertical component of a motion vector used in the prediction of the two prediction units is greater than or equal to 4 in units of quarter luma samples.
- Otherwise, the variable $bS[\text{filterDir}][xE_k][yE_j]$ is set equal to 0.

8.6.1.4 Filtering process for coding unit

Inputs of this process are:

- a luma location (xC, yC) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture,
- a variable $\log_2\text{CUSize}$ specifying the coding unit size,
- an array bS specifying the boundary filtering strength.

Output of this process is:

- modified reconstruction of the picture.

The filtering process for luma edges in the current coding unit consists of the following ordered steps:

1. The variable nD is set equal to $1 \ll (\log_2\text{CUSize} - 3)$.
2. All elements of the three-dimensional array of size $(2) \times (nD) \times (nD)$, $dEdge$ are initialized to zero.
3. All elements of the three-dimensional array of size $(2) \times (nD) \times (1 \ll \log_2\text{CUSize})$, $dSample$ are initialized to zero.
4. All elements of the three-dimensional array of size $(2) \times (nD) \times (nD)$, $bStrength$ are initialized to zero.
5. For xD_k set equal to $xC + (k \ll 3)$, $k=0..nD - 1$, the following applies:
 - For yD_m set equal to $yC + (m \ll 3)$, $m=0..nD - 1$, the following ordered steps apply:
 - a. Boundary filtering strength $bSVer$ is derived as follows:

$$bSVer = \text{Max}(bS[0][xD_k][yD_m + i]) \text{ for } i = 0..7 \quad (8-205205428)$$
 - b. $bStrength[1][k][m]$ is set equal to $bSVer$.
 - c. Decision process for luma block edge in subclause 8.6.1.4.1 is invoked with the luma location of the coding unit (xC, yC) , the luma location of the block (xD_k, yD_m) , a variable $verticalEdgeFlag$ set equal to 1, and the boundary filtering strength $bSVer$ as inputs and the decision $dEdge[1][k][m]$ and an array dS of size (8) as outputs.
 - d. $dSample[1][k][(m \ll 3) + i]$ is set equal to $dS[i]$ for $i=0..7$.
 - e. Boundary filtering strength $bSHor$ is derived as follows:

$$bSHor = \text{Max}(bS[1][xD_k + i][yD_m]) \text{ for } i = 0..7 \quad (8-206206428)$$
 - f. $bStrength[0][k][m]$ is set equal to $bSHor$.
 - g. Decision process for luma block edge in subclause 8.6.1.4.1 is invoked with the luma location of the coding unit (xC, yC) , the luma location of the block (xD_k, yD_m) , a variable $verticalEdgeFlag$ set equal to 0, the boundary filtering strength $bSHor$ as inputs, the decision $dEdge[0][k][m]$ and an array dS of size (8) as outputs.
 - h. $dSample[0][m][(k \ll 3) + i]$ is set equal to $dS[i]$ for $i=0..7$.
6. For xD_k set equal to $xC + (k \ll 3)$, $k=0..nD - 1$, the following applies:
 - For yD_m set equal to $yB + (m \ll 3)$, $m=0..nD - 1$, the following ordered steps apply:
 - a. $dS[i]$ is set equal to $dSample[1][k][(m \ll 3) + i]$ for $i=0..7$.
 - b. Filtering process for luma block edge in subclause 8.6.1.4.2 is invoked with the luma location of the coding unit (xC, yC) , the luma location of the block (xD_k, yD_m) , a variable $verticalEdgeFlag$ set equal to 1, the boundary filtering strength $bStrength[1][k][m]$, the decision $dEdge[1][k][m]$, and the array of size (8), dS as inputs and the modified luma picture buffer as outputs.

HEVC

7. For yD_m set equal to $yC+(m \ll 3)$, $m=0..nD-1$, the following applies:
 - For xD_k set equal to $xC+(k \ll 3)$, $k=0..nD-1$, the following ordered steps apply:
 - a. If xD_k is equal to 0, the parameter $xPOS$ is set equal to 1. If xD_k is equal to $xB+(nD-1) \ll 3$ $xPOS$ is set equal to 2. Otherwise $xPOS$ is set to 0.
 - b. $dS[i]$ is set equal to $dSample[0][m][(k \ll 3) + i]$ for $i = 0..7$.
 - c. Filtering process for luma block edge in subclause 8.6.1.4.2 is invoked with the luma location of the coding unit (xC, yC), the luma location of the block (xD_k, yD_m), a variable $verticalEdgeFlag$ set equal to 0, the boundary filtering strength $bStrength[0][k][m]$, the decision $dEdge[0][k][m]$, and the array of size (8), $dS, xPOS, dS_L[m][i], dE_L[m], bS_L[m]$, and $t_{cL}[m]$, as inputs and the modified luma picture buffer as output.
 - The elements of the two dimensional array of size $(3) \times nD$, dS_L are set as follows. $dS_L[m][0], dS_L[m][1]$, and $dS_L[m][2]$, are set equal to $dS[5], dS[6]$ and $dS[7]$.
 - The elements of the array of size (nD) , dE_L are set as follows. $dE_L[m]$ is set equal to $dEdge[0][k][m]$.
 - The elements of the array of size (nD) , bS_L are set as follows. $bS_L[m]$ is set equal to $bStrength[0][k][m]$.
 - The elements of the array of size (nD) , t_c are set as follows. $t_{cL}[m]$ is set equal to t_c .

The filtering process for chroma edges in the current coding unit consists of the following ordered steps:

1. The variable nD is set equal to $1 \ll (\text{Max}(\log_2 \text{CUsSize}, 4) - 4)$.
2. For xD_k set equal to $(xC/2)+(k \ll 3)$, $k=0..nD-1$, the following applies:
 - For yD_m set equal to $(yC/2)+(m \ll 2)$, $m=0..nD*2-1$, the following ordered steps apply:
 - a. Boundary filtering strength $bSVer$ is derived as follows:

$$bSVer = bS[0][xD_k*2][yD_m*2] \quad (8-207207430)$$
 - b. Filtering process for chroma block edge in subclause 8.6.1.4.3 is invoked with the chroma location of the coding unit ($xC/2, yC/2$), the chroma location of the block (xD_k, yD_m), a variable $verticalEdgeFlag$ set equal to 1, a chroma component index $cldx$ set equal to 1 and the boundary filtering strength $bSVer$ as inputs and the modified chroma picture buffer as output.
 - c. Filtering process for chroma block edge in subclause 8.6.1.4.3 is invoked with the chroma location of the coding unit ($xC/2, yC/2$), the chroma location of the block (xD_k, yD_m), a variable $verticalEdgeFlag$ set equal to 1, a chroma component index $cldx$ set equal to 2 and the boundary filtering strength $bSVer$ as inputs and the modified chroma picture buffer as output.
3. For yD_m set equal to $(yC/2)+(m \ll 3)$, $m=0..nD-1$, the following applies:
 - For xD_k set equal to $(xC/2)+(k \ll 2)$, $k=0..nD*2-1$, the following ordered steps apply:
 - a. If xD_k is equal to 0, the parameter $xPOS$ is set equal to 1. If xD_k is equal to $xB+(nD*2-1) \ll 2$ $xPOS$ is set equal to 2. Otherwise $xPOS$ is set to 0.
 - b. Boundary filtering strength $bSHor$ is derived as follows:

$$bSHor = bS[1][xD_k*2][yD_m*2] \quad (8-208208431)$$
 - c. Filtering process for chroma block edge in subclause 8.6.1.4.3 is invoked with the chroma location of the coding unit ($xC/2, yC/2$), the chroma location of the block (xD_k, yD_m), a variable $verticalEdgeFlag$ set equal to 0, a chroma component index $cldx$ set equal to 1 and the boundary filtering strength $bSHor, xPOS, bS_L[m]$ and $t_{cL}[m]$ as inputs and the modified chroma picture buffer as output.
 - d. Filtering process for chroma block edge in subclause 8.6.1.4.3 is invoked with the chroma location of the coding unit ($xC/2, yC/2$), the chroma location of the block (xD_k, yD_m), a variable $verticalEdgeFlag$ set equal to 0, a chroma component index $cldx$ set equal to 2 and the boundary filtering strength $bSHor, xPOS, bS_L[m]$ and $t_{cL}[m]$ as inputs and the modified chroma picture buffer as output.

8.6.1.4.1 Decision process for luma block edge

Inputs of this process are:

Formatted: Heading 5

HEVC

- a luma location (xC, yC) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture,
- a luma location (xB, yB) specifying the top-left luma sample of the current block relative to the top left luma sample of the current coding unit,
- a variable verticalEdgeFlag,
- a variable bS specifying the boundary filtering strength,

Output of this process is:

- a variable dE containing a decision,
- one-dimensional array of size (8), dS containing decisions.

Let s' represent the luma sample array recPicture_L of the current picture.

A variables β is specified as [Table 8-14Table 8-13](#) with luma quantization parameter qp_L as input.

A variable t_C is specified as follows:

- If bS is greater than 2, the variable t_C is specified as [Table 8-14Table 8-13](#) with luma quantization parameter (qp_L + 4) as input,
- Otherwise (bS is equal or less than 2), the variable t_C is specified as [Table 8-14Table 8-13](#) with luma quantization parameter qp_L as input.

Depending on verticalEdgeFlag, the following applies:

- If verticalEdgeFlag is equal to 1, the following ordered steps apply:

1. The sample values p_{i,k} and q_{i,k} with i = 0..3 and k = 2,5 are derived as follows:

$$q_{i,k} = s'[xC + xB + i, yC + yB + k] \quad (8-209209447)$$

$$p_{i,k} = s'[xC + xB - i - 1, yC + yB + k] \quad (8-210210448)$$

2. The variable d is derived as follows:

$$d = | p_{2,2} - 2 * p_{1,2} + p_{0,2} | + | q_{2,2} - 2 * q_{1,2} + q_{0,2} | + | p_{2,5} - 2 * p_{1,5} + p_{0,5} | + | q_{2,5} - 2 * q_{1,5} + q_{0,5} | \quad (8-211211449)$$

3. The variable dE is set equal to 0.

4. If bS is not equal to 0 and d is less than β, the following ordered steps apply:

- a. for each sample location (xC + xB, yC + yB + k), k = 0..7, the following ordered steps apply:

- a. The decision process for a luma sample specified in subclause 8.6.1.4.4 is invoked with sample values p_{i,k}, q_{i,k} with i = 0..3, the boundary filtering strength bS and the variables d, β and t_C as inputs and a decision dSam as output.

- b. The variable dS[k] is set equal to dSam

- b. The variable dE is set equal to 1.

- Otherwise (verticalEdgeFlag is equal to 0), the following ordered steps apply:

1. The sample values p_{i,k} and q_{i,k} with i = 0..3 and k = 2,5 are derived as follows:

$$q_{i,k} = s'[xC + xB + k, yC + yB + i] \quad (8-212212435)$$

$$p_{i,k} = s'[xC + xB + k, yC + yB - i - 1] \quad (8-213213436)$$

2. The variable d is derived as follows:

$$d = | p_{2,2} - 2 * p_{1,2} + p_{0,2} | + | q_{2,2} - 2 * q_{1,2} + q_{0,2} | + | p_{2,5} - 2 * p_{1,5} + p_{0,5} | + | q_{2,5} - 2 * q_{1,5} + q_{0,5} | \quad (8-214214437)$$

3. The variable dE is set equal to 0.

4. If bS is not equal to 0 and d is less than β, the following ordered steps apply:

- a. For each sample location (xC + xB + k, yC + yB), k = 0..7, the following ordered steps apply:

HEVC

- a. The decision process for a luma sample specified in subclause 8.6.1.4.4 is invoked with sample values $p_{i,k}$, $q_{i,k}$ with $i = 0..3$, the boundary filtering strength bS and the variables d , β and t_C as inputs and a decision $dSam$ as output.
- b. The variable $dS[k]$ is set equal to $dSam$.
- b. The variable dE is set equal to 1.

8.6.1.4.18.6.1.4.2 Filtering process for luma block edge

Inputs of this process are:

- a luma location (x_C , y_C) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture,
- a luma location (x_B , y_B) specifying the top-left luma sample of the current block relative to the top left luma sample of the current coding unit,
- a variable $verticalEdgeFlag$,
- a variable bS specifying the boundary filtering strength,
- a variable dE containing a decision,
- one-dimensional array of size (8), dS containing decisions,
- a variable bS_L ,
- a variable t_{CL} ,

Output of this process is:

- modified reconstruction of the picture.

Let s' represent the luma sample array $recPicture_L$ of the current picture.

Depending on pcm_flag , a variable β is specified as follows:

- If pcm_flag is equal to 1, β is specified as Table 8-14 with luma quantization parameter qP_L as input.
- Otherwise, β is specified as Table 8-14 with luma quantization parameter qP_L as input.

A variable t_C is specified as follows:

- If bS is greater than 2, the variable t_C is specified as Table 8-14 with luma quantization parameter ($qP_L + 4$) as input,
- Otherwise (bS is equal or less than 2), the variable t_C is specified as Table 8-14 with luma quantization parameter qP_L as input.

Depending on $verticalEdgeFlag$, the following applies:

- If $verticalEdgeFlag$ is equal to 1, the following ordered steps apply:

1. The sample values $p_{i,k}$ and $q_{i,k}$ with $i = 0..3$ and $k = 0..7$ are derived as follows:

$$q_{i,k} = s'[x_C + x_B + i, y_C + y_B + k] \quad (8-215215432)$$

$$p_{i,k} = s'[x_C + x_B - i - 1, y_C + y_B + k] \quad (8-216216433)$$

2. If dE is not equal to 0, for each sample location ($x_C + x_B$, $y_C + y_B + k$), $k = 0..7$, the following ordered steps apply:

- a. The filtering process for a luma sample specified in subclause 8.6.1.4.5 is invoked with sample values $p_{i,k}$, $q_{i,k}$ with $i = 0..3$, the decision $dS[k]$, the boundary filtering strength bS and the variable t_C as inputs and the number of filtered samples nD and the filtered sample values p_i' and q_i' as outputs.

- b. The filtered sample values p_i' and q_i' with $i = 0..nD - 1$ replace the corresponding samples inside the sample array s' as follows:

$$s'[x_C + x_B + i, y_C + y_B + k] = q_i' \quad (8-217217435)$$

$$s'[x_C + x_B - i - 1, y_C + y_B + k] = p_i' \quad (8-218218436)$$

Formatted: Indent: Left: 0", Hanging: 0.2", Tab stops: 0.2", Left

HEVC

– Otherwise (verticalEdgeFlag is equal to 0), the following ordered steps apply:

1. If xPOS is equal to 1, the parameters k_s and k_e are set to -3 and 4 respectively. If xD is equal to 2, the parameters k_s and k_e are set to 0 and 4 respectively. Otherwise k_s and k_e are set to 0 and 7 respectively.
2. The sample values $p_{i,k}$ and $q_{i,k}$ with $i = 0..3$ and $k = k_s..k_e$ are derived as follows:

$$q_{i,k} = s'[xC + xB + k, yC + yB + i] \quad (8-219219437)$$

$$p_{i,k} = s'[xC + xB + k, yC + yB - i - 1] \quad (8-220220438)$$

3. If xPOS is less than 0 and dE_L is not equal to 0, for each sample location $(xC + xB + k, yC + yB)$, $k = -3..-1$, the following ordered steps apply:

- a. The filtering process for a luma sample specified in subclause 8.6.1.4.5 is invoked with sample values $p_{i,k}$, $q_{i,k}$ with $i = 0..3$, decision $dS_L[k+3]$, the boundary filtering strength bS_L and the variable t_{CL} as inputs and the number of filtered samples nD and the filtered sample values p_i' and q_i' as outputs.
- b. The filtered sample values p_i' and q_i' with $i = 0..nD - 1$ replace the corresponding samples inside the sample array s' as follows:

$$s'[xC + xB + k, yC + yB + i] = q_i' \quad (8-221221459)$$

$$s'[xC + xB + k, yC + yB - i - 1] = p_i' \quad (8-222222460)$$

5. If dE is not equal to 0, for each sample location $(xC + xB + k, yC + yB)$, $k = 0..k_e$, the following ordered steps apply:

- a. The filtering process for a luma sample specified in subclause 8.6.1.4.5 is invoked with sample values $p_{i,k}$, $q_{i,k}$ with $i = 0..3$, decision $dS[k]$, the boundary filtering strength bS and the variable t_c as inputs and the number of filtered samples nD and the filtered sample values p_i' and q_i' as outputs.
- b. The filtered sample values p_i' and q_i' with $i = 0..nD - 1$ replace the corresponding samples inside the sample array s' as follows:

$$s'[xC + xB + k, yC + yB + i] = q_i' \quad (8-223223461)$$

$$s'[xC + xB + k, yC + yB - i - 1] = p_i' \quad (8-224224462)$$

8.6.1.4.3 Filtering process for chroma block edge

Formatted: Heading 5

[Ed.: (WJ) cIdx cannot be 0 here]

Inputs of this process are:

- a luma location (xC, yC) specifying the top-left chroma sample of the current coding unit relative to the top left chroma sample of the current picture,
- a luma location (xB, yB) specifying the top-left chroma sample of the current block relative to the top left chroma sample of the current coding unit,
- a variable verticalEdgeFlag,
- a variable bS specifying the boundary filtering strength,
- a variable cIdx specifying the chroma component index.
- a variable xPOS,
- a variable bS_L ,
- a variable t_{CL} .

Output of this process is:

- modified reconstruction of the picture.

Let s' be a variable specifying chroma sample array which is derived as follows.

- If cIdx is equal to 1, s' represents the chroma sample array $recPicture_{Cb}$ of the current picture.
- Otherwise (cIdx is equal to 2), s' represents the chroma sample array $recPicture_{Cr}$ of the current picture.

HEVC

A variable t_C is specified as follows:

- If b_S is greater than 2, the variable t_C is specified as [Table 8-14](#) with luma quantization parameter ($q_{P_L} + 4$) as input,
- Otherwise (b_S is equal or less than 2), the variable t_C is specified as [Table 8-14](#) with luma quantization parameter q_{P_L} as input.

Depending on `verticalEdgeFlag`, the following applies:

- If `verticalEdgeFlag` is equal to 1, for each sample location ($x_C + x_B, y_C + y_B + k$), $k = 0..3$, the following ordered steps apply:

1. The sample values p_i and q_i with $i = 0..1$ are derived as follows:

$$q_i = s'[x_C + x_B + i, y_C + y_B + k] \quad (8-225225442)$$

$$p_i = s'[x_C + x_B - i - 1, y_C + y_B + k] \quad (8-226226443)$$

2. If b_S is greater than 2, the following ordered steps apply:

- a. The filtering process for a sample specified in subclause 8.6.1.4.6 is invoked with sample values p_i, q_i with $i = 0..1$, the boundary filtering strength b_S and the variable t_C as inputs and the filtered sample values p_0' and q_0' as outputs.
- b. The filtered sample values p_0' and q_0' replace the corresponding samples inside the sample array s' as follows:

$$s'[x_C + x_B, y_C + y_B + k] = q_0' \quad (8-227227444)$$

$$s'[x_C + x_B - 1, y_C + y_B + k] = p_0' \quad (8-228228445)$$

- Otherwise (`verticalEdgeFlag` is equal to 0), the following ordered steps apply:

1. If x_{POS} is equal to 1, the parameters k_s and k_e are set to -1 and 2 respectively. If x_D is equal to 2, the parameters k_s and k_e are set to 0 and 2 respectively. Otherwise k_s and k_e are set to 0 and 3 respectively.
2. The sample values p_i and q_i with $i = 0..1$ and $k = k_s..k_e$ are derived as follows:

$$q_i = s'[x_C + x_B + k, y_C + y_B + i] \quad (8-229229446)$$

$$p_i = s'[x_C + x_B + k, y_C + y_B - i - 1] \quad (8-230230447)$$

3. If x_{POS} is less than 0, and if b_{S_L} is greater than 2, for each sample location ($x_C + x_B - 1, y_C + y_B$), the following ordered steps apply:

- a. The filtering process for a sample specified in subclause 8.6.1.4.6 is invoked with sample values p_i, q_i with $i = 0..1$, the boundary filtering strength b_{S_L} and the variable t_{C_L} as inputs and the filtered sample values p_0' and q_0' as outputs.
- b. The filtered sample values p_0' and q_0' replace the corresponding samples inside the sample array s' as follows:

$$s'[x_C + x_B + k, y_C + y_B] = q_0' \quad (8-231231452)$$

$$s'[x_C + x_B + k, y_C + y_B - i - 1] = p_i' \quad (8-232232445)$$

4. If b_S is greater than 2, for each sample location ($x_C + x_B + k, y_C + y_B$), $k = 0..k_e$, the following ordered steps apply:

- c. The filtering process for a sample specified in subclause 8.6.1.4.6 is invoked with sample values p_i, q_i with $i = 0..1$, the boundary filtering strength b_S and the variable t_C as inputs and the filtered sample values p_0' and q_0' as outputs.
- d. The filtered sample values p_0' and q_0' replace the corresponding samples inside the sample array s' as follows:

$$s'[x_C + x_B + k, y_C + y_B] = q_0' \quad (8-233233448)$$

HEVC

$$s'[xC + xB + k, yC + yB - 1] = p_0' \quad (8-234234449)$$

8.6.1.4.28.6.1.4.4 Decision process for a luma sample

[Ed: (WJ) no filtering when bS is equal to 0]

Inputs of this process are:

- sample values, p_i and q_i with $i = 0..2$,
- a variable bS specifying the boundary filtering strength,
- variables d, β and t_C .

Output of this process is:

- a variable dSam containing a decision

When the variable bS is not equal to 0, the following applies:

- If d is less than $(\beta \gg 2)$, $|p_3 - p_0| + |q_0 - q_3|$ is less than $(\beta \gg 3)$ and $|p_0 - q_0|$ is less than $(5 * t_C + 1) \gg 1$, dSam is set equal to 1.
- Otherwise, dSam is set equal to 0.

8.6.1.4.38.6.1.4.5 Filtering process for a luma sample

Inputs of this process are:

- sample values, p_i and q_i with $i = 0..3$,
- a variable dSam containing a decision,
- a variable t_C .

Output of this process is:

- number of filtered samples nD,
- filtered sample values, p_i' and q_i' with $i = 0..nD - 1$,

Depending on dSam, the following applies:

- When the variable dSam is equal to 1, the following strong filtering applies while nD is set equal to 3:

$$p_0' = \text{Clip1}_Y((p_2 + 2 * p_1 + 2 * p_0 + 2 * q_0 + q_1 + 4) \gg 3) \quad (8-235235450)$$

$$p_1' = \text{Clip1}_Y((p_2 + p_1 + p_0 + q_0 + 2) \gg 2) \quad (8-236236451)$$

$$p_2' = \text{Clip1}_Y((2 * p_3 + 3 * p_2 + p_1 + p_0 + q_0 + 4) \gg 3) \quad (8-237237452)$$

$$q_0' = \text{Clip1}_Y((p_1 + 2 * p_0 + 2 * q_0 + 2 * q_1 + q_2 + 4) \gg 3) \quad (8-238238453)$$

$$q_1' = \text{Clip1}_Y((p_0 + q_0 + q_1 + q_2 + 2) \gg 2) \quad (8-239239454)$$

$$q_2' = \text{Clip1}_Y((p_0 + q_0 + q_1 + 3 * q_2 + 2 * q_3 + 4) \gg 3) \quad (8-240240455)$$

- Otherwise, the following weak filtering applies while nD is set equal to 2:

$$\Delta = \text{Clip3}(-t_C, t_C, (13 * (q_0 - p_0) + 4 * (q_1 - p_1) - 5 * (q_2 - p_0) + 16) \gg 5) \quad (8-241241456)$$

$$p_0' = \text{Clip1}_Y(p_0 + \Delta) \quad (8-242242457)$$

$$q_0' = \text{Clip1}_Y(q_0 - \Delta) \quad (8-243243458)$$

$$p_1' = \text{Clip1}_Y(p_1 + \Delta/2) \quad (8-244244459)$$

$$q_1' = \text{Clip1}_Y(q_1 - \Delta/2) \quad (8-245245460)$$

Each of the filtered sample values, p_i' with $i = 0..nD-1$, is substituted by the corresponding input sample value p_i if all of the following conditions are true.

Formatted: Subscript

Formatted: Subscript

HEVC

- p_i is a sample of an I_PCM block.
- `pcm_loop_filter_disable_flag` value is equal to 1.

Formatted: Indent: Left: 0.2", Hanging: 0.2", Tab stops: 0.39", Left

Similarly, each of the filtered sample values, q_i , with $i = 0..nD-1$, is substituted by the corresponding input sample value q_i if all of the following conditions are true.

Formatted: Subscript

- q_i is a sample of an I_PCM block.
- `pcm_loop_filter_disable_flag` value is equal to 1.

Formatted: Subscript

[Ed. (WJ): for PCM case, deblocking filter applies first and the filtered pixels are restored. Rather than this, it's better to skip the filtering itself for PCM samples since first filtering is actually not needed.]

Table 8-148-128-11 – Derivation of threshold variables β and t_c from input Q

Q	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
β	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	7	8
t_c	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Q	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37
β	9	10	11	12	13	14	15	16	17	18	20	22	24	26	28	30	32	34	36
t_c	1	1	1	1	1	1	1	1	2	2	2	2	3	3	3	3	4	4	4
Q	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	
β	38	40	42	44	46	48	50	52	54	56	58	60	62	64	64	64	64	64	
t_c	5	5	6	6	7	8	9	9	10	10	11	11	12	12	13	13	14	14	

8.6.1.4.48.6.1.4.6 Filtering process for a chroma sample

[Ed. (WJ) no filtering when bS is equal or less than 2]

Inputs of this process are:

- sample values, p_i and q_i with $i = 0..1$,
- a variable bS specifying the boundary filtering strength.
- a variable t_c .

Output of this process is:

- The filtered sample values, p_0' and q_0' .

When the variable bS is greater than 2, the filtered sample values p_0' and q_0' are derived by

$$\Delta = \text{Clip3}(-t_c, t_c, (((q_0 - p_0) << 2) + p_1 - q_1 + 4) >> 3)) \quad (8-246246461)$$

$$p_0' = \text{Clip1}_c(p_0 + \Delta) \quad (8-247247462)$$

$$q_0' = \text{Clip1}_c(q_0 - \Delta) \quad (8-248248463)$$

The filtered sample value, p_0' is substituted by the corresponding input sample value p_0 if all of the following conditions are true.

- p_0 is a sample of an I_PCM block.
- `pcm_loop_filter_disable_flag` value is equal to 1.

Similarly, the filtered sample value, q_0' is substituted by the corresponding input sample value q_0 if all of the following conditions are true.

- q_0 is a sample of an I_PCM block.
- `pcm_loop_filter_disable_flag` value is equal to 1.

Formatted: Normal, Indent: Hanging: 0.2", Tab stops: 0.39", Left + Not at 0.59" + 0.79" + 0.98"

HEVC

[Ed. (WJ): for PCM case, deblocking filter applies first and the filtered pixels are restored. Rather than this, it's better to skip the filtering itself for PCM samples since first filtering is actually not needed.]

8.6.2 Sample adaptive offset process

A sample adaptive offset process shall be conditionally performed after the completion of the deblocking filter process for the decoded picture.

This process is invoked when both `sample_adaptive_offset_enabled_flag` and `sample_adaptive_offset_flag` are equal to 1.

This process is performed on a region basis which is aligned with the LCU boundaries after the completion of the slice construction process prior to sample adaptive offset process for the entire decoded process.

The sample adaptive offset process is invoked **with the region equal to entire slice here**.

[Ed.: (WJ) invoking is needed]

8.6.2.1 Modification process for sample adaptive offset

[Ed.: (WJ) recursive partitioning is invoked here]

8.6.2.1.1 Modification process for luma samples

[Ed.: (WJ) offset is added here]

8.6.3 Adaptive loop filter process

An adaptive loop filtering process shall be conditionally performed on a treeblock basis after the completion of the sample adaptive offset process for the entire decoded picture, with all treeblocks in a picture processed in order of increasing treeblock addresses.

Each treeblock is processed on a coding unit basis with the same order as decoding process.

This process is invoked when both `adaptive_loop_filter_enabled_flag` and `adaptive_loop_filter_flag` are equal to 1.

This process is performed on a coding unit basis after the completion of the slice construction process prior to adaptive loop filter process for the entire decoded slice, with all coding units in a slice processed in order of coding unit scan order.

Filter coefficients c_L for luma samples and c_C for chroma samples are derived by invoking the process specified in subclause [8.6.3.28-6.2.1](#).

Filter index array `fldx` for luma samples are derived by invoking the process specified in subclause [8.6.3.38-6.2.2](#) with the luma location (`xC`, `yC`) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture and the coding unit size `log2CUSize` as inputs.

When interpreting the luma samples of the coding unit as to be filtered, depending on `pcm_flag`, `pcm_loop_filter_disable_flag` and `alf_cu_control_flag`, the following applies.

- If `pcm_loop_filter_disable_flag` is equal to 1 and `pcm_flag` is equal to 1, the luma samples are not filtered.
- Otherwise, If if `alf_cu_control_flag` is equal to 0, the luma samples are filtered,
- Otherwise (if `alf_cu_control_flag` is equal to 1), if `AlfCuFlag[xC][yC]` is equal to 1 where `xC` and `yC` are the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture, the luma samples are filtered.
- Otherwise, the luma samples are not filtered.

When interpreting the chroma samples of the coding unit as to be filtered, depending on `pcm_flag`, `pcm_loop_filter_disable_flag` and `alf_chroma_idc`, the following applies.

- If `pcm_loop_filter_disable_flag` is equal to 1 and `pcm_flag` is equal to 1, both chroma samples are not filtered.
- Otherwise, If if `alf_chroma_idc` is equal to 1, the C_r samples are filtered.
- Otherwise, if `alf_chroma_idc` is equal to 2, the C_b samples are filtered.
- Otherwise, if `alf_chroma_idc` is equal to 3, both chroma samples are filtered.
- Otherwise (if `alf_chroma_idc` is equal to 0), both chroma samples are not filtered.

HEVC

For the luma samples of the coding unit interpreted as to be filtered, the filtering process for luma samples specified in subclause 8.6.3.48.6.2.3 is invoked with the luma location (x_C , y_C) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture, the coding unit size $\log_2\text{CUsSize}$, and the filter index array fldx as inputs and the output is the modified filtered picture, recFiltPicture_L .

For the chroma samples of the coding unit interpreted as to be filtered, the filtering process for chroma samples specified in subclause 8.6.3.58.6.2.4 is invoked with the chroma location ($x_C/2$, $y_C/2$) specifying the top-left chroma sample of the current coding unit relative to the top left chroma sample of the current picture, the coding unit size $\log_2\text{CUsSize}-1$ and the chroma component index cldx equal to 1 as inputs and the output is the modified filtered picture, $\text{recFiltPicture}_{Cb}$.

For the chroma samples of the coding unit interpreted as to be filtered, the filtering process for chroma samples specified in subclause 8.6.3.58.6.2.4 is invoked with the chroma location ($x_C/2$, $y_C/2$) specifying the top-left chroma sample of the current coding unit relative to the top left chroma sample of the current picture, the coding unit size $\log_2\text{CUsSize}-1$ and the chroma component index cldx equal to 2 as inputs and the output is the modified filtered picture, $\text{recFiltPicture}_{Cr}$.

[Ed.: (WJ) recPicture : deblocked/output picture and recFiltPicture : ALFed picture]

[Ed.: (WJ) depending adaptive_loop_filter_flag, recFiltPicture should be copied to recPicture in subclause 8.6]

8.6.3.1 Boundary padding process

Inputs of this process are:

- a luma location (x_C , y_C) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture,
- a variable $\log_2\text{CUsSize}$ specifying the size of the current coding unit,
- a variable cldx specifying the chroma component index.

Output of this process is the padded luma sample array s'' .

Depending on cldx , the following applies:

- If cldx is equal to 0, a luma sample array s' is set equal to recPicture_L , a variable alfTap is set equal to $(\text{alf_length_luma_minus_5_div2} \ll 1) + 5$, a variable nS is set equal to $(1 \ll \log_2\text{CUsSize})$ and a variable $n\text{ExtPixels}$ is set equal to $(\text{alfTap} \gg 1)$.
- Otherwise, if cldx is equal to 1, a chroma sample array s' is set equal to recPicture_{Cb} , a variable alfTap is set equal to $(\text{alf_length_chroma_minus_5_div2} \ll 1) + 5$, a variable nS is set equal to $(1 \ll (\log_2\text{CUsSize} - 1))$ and a variable $n\text{ExtPixels}$ is set equal to $(\text{alfTap} \gg 1)$.
- Otherwise (cldx is equal to 2), a chroma sample array s' is set equal to recPicture_{Cr} , a variable alfTap is set equal to $(\text{alf_length_chroma_minus_5_div2} \ll 1) + 5$, a variable nS is set equal to $(1 \ll (\log_2\text{CUsSize} - 1))$ and a variable $n\text{ExtPixels}$ is set equal to $(\text{alfTap} \gg 1)$.

Depending on loop_filter_across_slice_flag, a variable s'' specifying the padded luma sample array is derived as:

- If loop_filter_across_slice_flag is equal to 1, $s''[x_C+x][y_C+y]$ is set equal to $s'[x_C+x][y_C+y]$ for $x = -n\text{ExtPixels}..nS+n\text{ExtPixels}-1$ and $y = -n\text{ExtPixels}..nS+n\text{ExtPixels}-1$.

[Ed. (WJ): picture boundaries should be considered even for this case]

- Otherwise (loop_filter_across_slice_flag is equal to 0), $s''[x_C+x][y_C+y]$ is derived by the following ordered steps:

1. $s''[x_C+x][y_C+y]$ is set equal to $s'[x_C+x][y_C+y]$ for $x=0..nS-1$ and $y=0..nS-1$.

2. If the left boundary of current block coincides the slice,

$$s''[x_C+x][y_C+y] = s'[x_C][y_C+y] \text{ where } x = -n\text{ExtPixels}..-1 \text{ and } y = 0..nS-1 \quad (8-249249462)$$

3. If the right boundary of current block coincides the slice,

$$s''[x_C+nS+x][y_C+y] = s'[x_C+nS-1][y_C+y] \text{ where } x = 0..n\text{ExtPixels}-1 \text{ and } y = 0..nS-1 \quad (8-250250462)$$

4. If the top boundary of current block coincides the slice,

$$s''[x_C+x][y_C+y] = s'[x_C+x][y_C] \text{ where } x = 0..nS-1 \text{ and } y = -n\text{ExtPixels}..-1 \quad (8-251251462)$$

5. If the bottom boundary of current block coincide the slice,

Formatted: Heading 4,Heading 4 Char Char,Heading 4 Char1

Formatted: Subscript

Formatted: Bulleted + Level: 1 + Aligned at: 0" + Tab after: 0.28" + Indent at: 0.28", Tab stops: Not at 0.55"

Formatted: Subscript

HEVC

$$s'[xC+x][yC+nS+y] = s'[xC+x][yC+nS-1] \text{ where } x = 0..nS-1 \text{ and } y = 0..nExtPixels-1 \quad (8-252252462)$$

6. If both the left and the top boundaries of current block coincide the slice and the current coding unit and its upper-left coding unit are not at the same slice,

$$s'[xC+x][yC+y] = s'[xC][yC] \text{ where } x = -nExtPixels..-1 \text{ and } y = -nExtPixels..-1 \quad (8-253253462)$$

7. If both the right and the top boundaries of current block coincide the slice and the current coding unit and its upper-right coding unit are not at the same slice,

$$s'[xC+nS+x][yC+y] = s'[xC+nS-1][yC] \text{ where } x = 0..nExtPixels-1 \text{ and } y = -nExtPixels..-1 \quad (8-254254462)$$

8. If both the left and the bottom boundaries of current block coincide the slice and the current coding unit and its bottom-left coding unit are not at the same slice,

$$s'[xC+x][yC+nS+y] = s'[xC][yC+nS-1] \text{ where } x = -nExtPixels..-1 \text{ and } y = 0..nExtPixels-1 \quad (8-255255462)$$

9. If both the right and the bottom boundaries of current block coincide the slice and the current coding unit and its bottom-right coding unit are not at the same slice,

$$s'[xC+nS+x][yC+nS+y] = s'[xC+nS-1][yC+nS-1] \text{ where } x = 0..nExtPixels-1 \text{ and } y = 0..nExtPixels-1 \quad (8-256256462)$$

Formatted: Indent: Left: 0.21", Numbered + Level: 1 + Numbering Style: 1, 2, 3, ... + Start at: 1 + Alignment: Left + Aligned at: 0" + Tab after: 0.28" + Indent at: 0.28", Tab stops: 0.5", Left + 0.75", Left + 1", Left + 1.18", Left + Not at 0.2" + 0.55" + 0.83" + 1.1" + 1.38"

Formatted: (Asian) Korean, (Other) (none)

8.6.3.18.6.3.2 Derivation process for filter coefficients

Outputs of this process are filter coefficients c_L for the luma samples and filter coefficients c_C for the chroma samples.

The luma filter coefficients c_L with elements $c_L[i][j]$, $i = 0..AlfNumFilters - 1$, $j = 0..AlfCodedLengthLuma - 1$ is derived as follows:

- If `alf_pred_method` is equal to 0 or the value of i is equal to 0,

$$c_L[i][j] = \text{alf_coeff_luma}[i][j] \quad (8-257257464)$$

- Otherwise (`alf_pred_method` is equal to 1 and the value of i is greater than 1),

$$c_L[i][j] = \text{alf_coeff_luma}[i][j] + c_L[i-1][j] \quad (8-258258465)$$

Considering the symmetry of the filter, the luma filter coefficients c_L with elements $c_L[i][j]$, $i = 0..AlfNumFilters - 1$ is derived as follows:

$$c_L[i][AlfLengthLuma - 1] = c_L[i][AlfCodedLengthLuma - 1] \quad (8-259259466)$$

$$c_L[i][AlfLengthLuma - 2 - j] = c_L[i][j] \text{ for } j = 0..AlfCodedLengthLuma - 2 \quad (8-260260467)$$

The chroma filter coefficients c_C with elements $c_C[i]$, $i = 0..AlfCodedLengthChroma - 1$ is derived as follows:

- If i is equal to `AlfCodedLengthChroma - 1`, the coefficient $c_C[i]$ is derived as

$$c_C[i] = 255 - \text{sum} - \text{alf_coeff_chroma}[i] \quad (8-261261468)$$

where

$$\text{sum} = \text{alf_coeff_chroma}[AlfCodedLengthChroma - 2] + \sum_j (\text{alf_coeff_chroma}[j] \ll 1) \quad (8-262262469)$$

with $j = 0..AlfCodedLengthChroma - 3$

- Otherwise (i is less than `AlfCodedLengthChroma - 1`),

$$c_C[i] = \text{alf_coeff_chroma}[i] \quad (8-263263470)$$

Considering the symmetry of the filter, the chroma filter coefficients c_C with elements $c_C[i]$, $i = 0..AlfLengthChroma - 1$ is derived as follows:

HEVC

$cc[i][AlfLengthChroma - 1] = cc[i][AlfCodedLengthChroma - 1]$ (8-264264471)

$cc[i][AlfLengthChroma - 2 - j] = cc[i][j]$ for $j = 0..AlfCodedLengthChroma - 2$ (8-265265472)

Formatted: Heading 4,Heading 4 Char Char,Heading 4 Char1, Indent: Left: 0", Tab stops: Not at 0.59" + 0.79" + 0.98"

8.6.3.28.6.3.3 Derivation process for filter index array for luma samples

Inputs of this process are:

- a luma location (xC, yC) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture,
- a variable log2CUSize specifying the size of the current coding unit.

Output of this process is the two-dimensional filter index array of (nS)x(nS), fldx.

A variable nS is set equal to $(1 \ll \log_2 \text{CUSize})$ and s' represents the luma sample array recPictureL of the current picture.

The boundary padding process specified in subclause 8.6.3.1 is invoked with the luma location (xC, yC), the size of coding unit log2CUSize and the chroma component index cldc set equal to 0, and the output is assigned to the luma sample array s'. [Ed. (WJ): s' is now a picture-size array, but actually CU size + appropriate margin is enough]

The filter index array fldx is specified in the follows:

- When alf_region_adaptation_flag is equal to 1, the following ordered steps apply.

1. The variables xIdx and yIdx are derived as

$regionTab[16] = \{ 0, 1, 4, 5, 15, 2, 3, 6, 14, 11, 10, 7, 13, 12, 9, 8 \}$ (8-266266474)
 $offset = 1 \ll (\text{Log2MaxCUSize} - 1)$ (8-267267474)
 $xInterval = ((PicWidthInSamples_L + offset) \gg \text{Log2MaxCUSize} + 1) \gg 2$ (8-268268474)
 $yInterval = ((PicHeightInSamples_L + offset) \gg \text{Log2MaxCUSize} + 1) \gg 2$ (8-269269474)
 $xIdx = \text{Min}(3, \text{Floor}((xC + x) / (xInterval \ll \text{Log2MaxCUSize})))$ (8-270270474)
 $yIdx = \text{Min}(3, \text{Floor}((yC + y) / (yInterval \ll \text{Log2MaxCUSize})))$ (8-271271474)

2. The filter index $fldx[x, y]$ with $x, y = 0..(nS)-1$ is derived as

$fldx[x][y] = regionTab[(yIdx \ll 2) + xIdx]$ (8-272272474)

- Otherwise (alf_region_flag is equal to 0), the following ordered steps apply.

3. The variables varTempH[x][y], varTempV[x][y] and varTemp1[x][y] with $x, y = -1..(nS)+1$ is derived as

$varTempH[x][y] = |(s'[xC+x, yC+y] \ll 1) - s'[xC+x-1, yC+y] - s'[xC+x+1, yC+y]|$ (8-273273474)
 $varTempV[x][y] = |(s'[xC+x, yC+y] \ll 1) - s'[xC+x, yC+y-1] - s'[xC+x, yC+y+1]|$ (8-274274474)
 $varTemp1[x][y] = varTempH[x][y] + varTempV[x][y]$ (8-275275474)

4. The variable varTemp2[x, y] with $x, y = 0..(nS)-1$ is derived as

$varTemp2[x][y] = \sum_i \sum_j varTemp1[x+i][y+j]$ with $i, j = -1..1$ (8-276276474)

5. The variables varTemp3[x, y], varTempH1[x, y] and varTempV1[x, y] with $x, y = 0..(nS) - 1 \gg 2$ are derived as

$varTemp3[x][y] = (\sum_i \sum_j varTemp2[(x \ll 2) + i][(y \ll 2) + j]) \gg 4$ with $i, j = 0..3$ (8-277277474)
 $varTempH1[x][y] = \sum_i \sum_j varTempH[(x \ll 2) + i][(y \ll 2) + j]$ with $i, j = 0..3$ (8-278278474)
 $varTempV1[x][y] = \sum_i \sum_j varTempV[(x \ll 2) + i][(y \ll 2) + j]$ with $i, j = 0..3$ (8-279279474)

6. The variable direction is derived as

- If $varTempV1[x \gg 2][y \gg 2]$ is greater than $varTempH1[x \gg 2][y \gg 2] \ll 1$,
direction = 1
- Otherwise, if $varTempH1[x \gg 2][y \gg 2]$ is greater than $varTempV1[x \gg 2][y \gg 2] \ll 1$,

HEVC

direction = 2

– Otherwise,

direction = 0

7. The variable avgvar is derived as

$$\text{varTab}[16] = \{ 0, 1, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4 \} \quad (8-280280475)$$

$$\text{avgVar} = \text{Clip3}(0, 15, (\text{varTemp3}[x \gg 2][y \gg 2] * 114) \gg (3 + \text{BitDepthY})) \quad (8-281281475)$$

8. The filter index fIdx[x, y] with x, y = 0..(nS)-1 is derived as

$$\text{fIdx}[x][y] = \text{Clip3}(0, 4, \text{varTab}[\text{avgVar}]) + 5 * \text{direction} \quad (8-282282504)$$

8.6.3.38.6.3.4 Filtering process for luma samples

Inputs of this process are:

- a luma location (xC, yC) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture,
- a variable log2CUsSize specifying the size of the current coding unit,
- a filter index array of (nS)x(nS), fIdx.

Output of this process is the filtered reconstruction of luma picture.

~~Let s' be a variable specifying luma sample array recPictureL. The boundary padding process specified in subclause 8.6.3.1 is invoked with the luma location (xC, yC), the size of coding unit log2CUsSize and the chroma component index cIdx set equal to 0, and the output is assigned to the luma sample array s'. [Ed. (WJ): s' is now a picture-size array, but actually CU size + appropriate margin is enough]~~

A variable nS is set equal to (1 << log2CUsSize) and a variable alfTap is set equal to (alf_length_luma_minus_5_div2 << 1) + 5.

Each sample of luma picture recFiltPictureL[xC + x][yC + y] with x, y = 0..(nS)-1, is derived as follows.

$$\text{recFiltPicture}_L[xC + x][yC + y] = c_L[\text{fIdx}[x][y]][N] + \sum_{i=0}^{N-1} (s''[xC + x + \text{horPos}[i], yC + y + \text{verPos}[i]] * c_L[\text{fIdx}[x][y]][i])$$

(8-283283476)

where N is set equal to AlfNumCoeffLuma-1 and horPos[i] and verPos[i] are specified in [Table 8-15Table 8-14](#) and [Table 8-16Table 8-15](#), respectively.

Table 8-158-138-12 – Specification of horPos[i] according to alfTap for adaptive loop filter process of luma samples

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
alfTap = 5	0	-1	0	1	-2	-1	0	1	2	-1	0	1	0	-	-	-	-	-	-	-
alfTap = 7	0	-1	0	1	-2	-1	0	1	2	-3	-2	-1	0	1	2	3	-2	-1	0	1
alfTap = 9	-1	0	1	-2	-1	0	1	2	-3	-2	-1	0	1	2	3	-4	-3	-2	-1	0

HEVC

i	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
alfTap = 5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
alfTap = 7	2	-1	0	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-
alfTap = 9	1	2	3	4	-3	-2	-1	0	1	2	3	-2	-1	0	1	2	-1	0	1

Table 8-168-148-13 – Specification of verPos[i] according to alfTap for adaptive loop filter process of luma samples

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
alfTap = 5	-2	-1	-1	-1	0	0	0	0	0	1	1	1	2							
alfTap = 7	-3	-2	-2	-2	-1	-1	-1	-1	-1	0	0	0	0	0	0	1	1	1	1	1
alfTap = 9	-3	-3	-3	-2	-2	-2	-2	-2	-1	-1	-1	-1	-1	-1	0	0	0	0	0	0

i	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
alfTap = 5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
alfTap = 7	1	2	2	2	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-
alfTap = 9	0	0	0	0	1	1	1	1	1	1	1	2	2	2	2	2	3	3	3

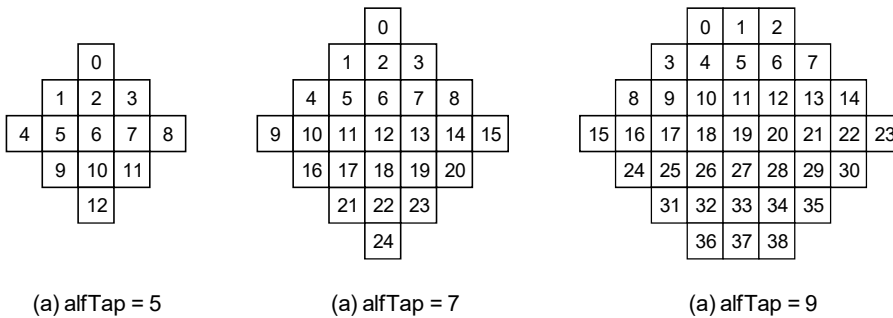


Figure 8-8 Mapping between geometric position and luma adaptive loop filter index according to alfTap (informative)

8.6.3.48.6.3.5 Filtering process for chroma samples

Inputs of this process are:

- a chroma location (xC, yC) specifying the top-left luma sample of the current coding unit relative to the top left chroma sample of the current picture,
- a variable log2CUSize specifying the size of the current coding unit.
- a variable cIdx specifying the chroma component index.

Output of this process is the filtered reconstruction of chroma picture.

The boundary padding process specified in subclause 8.6.3.1 is invoked with the chroma location (xC, yC), the size of coding unit log2CUSize and the chroma component index cIdx, and the output is assigned to the luma sample array s'. [Ed. (WJ): s' is now a picture-size array, but actually CU size + appropriate margin is enough]

Let s' be a variable specifying chroma sample array which is derived as follows.

— If cIdx is equal to 1, s' represents the chroma sample array recPictureCb of the current picture and recFiltPicture represents the filtered reconstruction of chroma picture recFiltPictureCb.

— Otherwise (eIdx is equal to 2), s' represents the chroma sample array $recPicture_{c_c}$ of the current picture and $recFiltPicture$ represents the filtered reconstruction of chroma picture $recFiltPicture_{c_c}$.

A variable nS is set equal to $(1 \ll \log_2 CUSize)$ and a variable $alfTapChroma$ is set equal to $(alf_length_chroma_minus_5_div2 \ll 1) + 5$.

Filtered samples of chroma picture $recFiltPicture[xC + x][yC + y]$ with $x, y = 0..(nS)-1$, are derived as follows.

$$recFiltPicture[xC + x][yC + y] = c_c[N] + \sum_{i=0}^{N-1} (s'[xC + x + horPos[i], yC + y + verPos[i]] * c_c[i])$$

(8-284284477)

where

$$N = AlfNumCoeffChroma - 1, \quad (8-285285478)$$

$$horPos[i] = (i \% alfTapChroma) - (alfTapChroma \gg 1), \text{ and} \quad (8-286286479)$$

$$verPos[i] = (i / alfTapChroma) - (alfTapChroma \gg 1) \quad (8-287287480)$$

Formatted: (Asian Korean, (Other) (none)

Formatted: Normal, Indent: Left: 0", Tab stops: Not at 0.59" + 0.79" + 0.98"

9 Parsing process

Inputs to this process are bits from the RBSP.

Outputs of this process are syntax element values.

This process is invoked when the descriptor of a syntax element in the syntax tables in subclause 7.3 is equal to $ue(v)$, $me(v)$, $se(v)$, $te(v)$ (see subclause 9.1), $ce(v)$ (see subclause 9.2), or $ae(v)$ (see subclause 9.3).

9.1 Parsing process for Exp-Golomb codes

This process is invoked when the descriptor of a syntax element in the syntax tables in subclause 7.3 is equal to $ue(v)$, $me(v)$, $se(v)$, or $te(v)$. For syntax elements in subclauses 7.3.4 and 7.3.5, this process is invoked only when $entropy_coding_mode_flag$ is equal to 0.

Inputs to this process are bits from the RBSP.

Outputs of this process are syntax element values.

Syntax elements coded as $ue(v)$, $me(v)$, or $se(v)$ are Exp-Golomb-coded. Syntax elements coded as $te(v)$ are truncated Exp-Golomb-coded. The parsing process for these syntax elements begins with reading the bits starting at the current location in the bitstream up to and including the first non-zero bit, and counting the number of leading bits that are equal to 0. This process is specified as follows:

$$\begin{aligned} leadingZeroBits &= -1 \\ \text{for}(b = 0; !b; leadingZeroBits++) & \\ b &= read_bits(1) \end{aligned} \quad (9-1)$$

The variable $codeNum$ is then assigned as follows:

$$codeNum = 2^{leadingZeroBits} - 1 + read_bits(leadingZeroBits) \quad (9-2)$$

where the value returned from $read_bits(leadingZeroBits)$ is interpreted as a binary representation of an unsigned integer with most significant bit written first.

Table 9-1 Table 9-4 illustrates the structure of the Exp-Golomb code by separating the bit string into "prefix" and "suffix" bits. The "prefix" bits are those bits that are parsed in the above pseudo-code for the computation of $leadingZeroBits$, and are shown as either 0 or 1 in the bit string column of Table 9-1 Table 9-4. The "suffix" bits are those bits that are parsed in the computation of $codeNum$ and are shown as x_i in Table 9-1 Table 9-4, with i being in the range 0 to $leadingZeroBits - 1$, inclusive. Each x_i can take on values 0 or 1.

HEVC

Table 9-19-19-1 – Bit strings with "prefix" and "suffix" bits and assignment to codeNum ranges (informative)

Bit string form	Range of codeNum
1	0
0 1 x_0	1..2
0 0 1 $x_1 x_0$	3..6
0 0 0 1 $x_2 x_1 x_0$	7..14
0 0 0 0 1 $x_3 x_2 x_1 x_0$	15..30
0 0 0 0 0 1 $x_4 x_3 x_2 x_1 x_0$	31..62
...	...

Table 9-2 illustrates explicitly the assignment of bit strings to codeNum values.

Table 9-29-29-2 – Exp-Golomb bit strings and codeNum in explicit form and used as ue(v) (informative)

Bit string	codeNum
1	0
0 1 0	1
0 1 1	2
0 0 1 0 0	3
0 0 1 0 1	4
0 0 1 1 0	5
0 0 1 1 1	6
0 0 0 1 0 0 0	7
0 0 0 1 0 0 1	8
0 0 0 1 0 1 0	9
...	...

Depending on the descriptor, the value of a syntax element is derived as follows.

- If the syntax element is coded as ue(v), the value of the syntax element is equal to codeNum.
- Otherwise, if the syntax element is coded as se(v), the value of the syntax element is derived by invoking the mapping process for signed Exp-Golomb codes as specified in subclause 9.1.1 with codeNum as the input.
- Otherwise, if the syntax element is coded as me(v) [Ed. (TW): insert text]
- Otherwise (the syntax element is coded as te(v)), the range of possible values for the syntax element is determined first. The range of this syntax element may be between 0 and x, with x being greater than or equal to 1 and the range is used in the derivation of the value of the syntax element value as follows.
 - If x is greater than 1, codeNum and the value of the syntax element is derived in the same way as for syntax elements coded as ue(v).
 - Otherwise (x is equal to 1), the parsing process for codeNum which is equal to the value of the syntax element is given by a process equivalent to:

$$\begin{aligned}
 & b = \text{read_bits}(1) \\
 & \text{codeNum} = !b
 \end{aligned}
 \tag{9-3}$$

HEVC

9.1.1 Mapping process for signed Exp-Golomb codes

Input to this process is codeNum as specified in subclause 9.1.

Output of this process is a value of a syntax element coded as se(v).

The syntax element is assigned to the codeNum by ordering the syntax element by its absolute value in increasing order and representing the positive value for a given absolute value with the lower codeNum. [Table 9-3](#) provides the assignment rule.

Table 9-3 – Assignment of syntax element to codeNum for signed Exp-Golomb coded syntax elements se(v)

codeNum	syntax element value
0	0
1	1
2	-1
3	2
4	-2
5	3
6	-3
k	$(-1)^{k+1} \text{Ceil}(k/2)$

9.2 CAVLC parsing process for slice data

This process is invoked when the descriptor of a syntax element in the syntax tables in subclause 7.3 is equal to ce(v).

Inputs to this process are bits from the RBSP.

Outputs of this process are syntax element values.

9.2.1 Parsing process for VLC codes

Inputs to this process are bits from the RBSP and the parameter vlcNum specifying the VLC code. If vlcNum is equal to 14, inputs to this process additionally include a parameter cMax.

Output of this process is the variable codeNum specifying an integer value to be used when deriving syntax elements.

If vlcNum is not equal to 8 or 11, and is less than 14, the parsing process for the VLC codes begins with reading the bits starting at the current location in the bitstream up to and including the first non-zero bit, and counting the number of leading bits that are equal to 0. This process is specified as follows.

```

leadingZeroBits = -1
for( b = 0; !b; leadingZeroBits++ )
    b = read_bits( 1 )
    
```

(9-4)

Depending on the value of vlcNum, the value of codeNum is assigned as follows.

- If vlcNum is equal to 0,
 - If leadingZeroBits is less than or equal to 6,

$$\text{codeNum} = \text{leadingZeroBits} \tag{9-5}$$

- Otherwise (leadingZeroBits is greater than 6),

$$\text{numBits} = \text{leadingZeroBits} - 6 \tag{9-6}$$

$$\text{codeNum} = 5 + (1 \ll \text{numBits}) + \text{read_bits}(\text{numBits})$$

- Otherwise, if vlcNum is less than 5,
 - If leadingZeroBits is less than or equal to 6,

$$\text{codeNum} = (\text{leadingZeroBits} \ll \text{vlcNum}) + \text{read_bits}(\text{vlcNum}) \tag{9-7}$$

HEVC

- Otherwise (leadingZeroBits is greater than 6),
 - numBits = leadingZeroBits - 6 + vlcNum (9-8)
 - codeNum = 5*(1<<vlcNum) + (1<< numBits) + read_bits(numBits)
 - Otherwise, if vlcNum is less than 8,
 - codeNum = (leadingZeroBits << (vlcNum - 4)) + read_bits(vlcNum - 4) (9-9)
 - Otherwise, if vlcNum is equal to 8,
 - b = read_bits(1) (9-10)
 - codeNum = b ? 2 - read_bits(1) : 0
 - Otherwise, if vlcNum is equal to 9,
 - If leadingZeroBits is equal to 0,
 - b = read_bits(1)
 - if (b) {
 - b = read_bits(1)
 - if (b) {
 - codeNum = 3 + read_bits(3) (9-11)
 - else {
 - b = read_bits(1)
 - codeNum = b ? 1 + read_bits(1) : 0
 - Otherwise (leadingZeroBits is greater than 0),
 - codeNum = (leadingZeroBits << 4) + read_bits(4) + 11 (9-12)
- Otherwise, if vlcNum is equal to 10,
 - codeNum = (1<< leadingZeroBits) - 1 + read_bits(leadingZeroBits) (9-13)
- Otherwise, if vlcNum is equal to 11,
 - codeNum = read_bits(3)
 - if(codeNum) {
 - b = read_bits(1)
 - codeNum = (codeNum << 1) + b -1 (9-14)
- Otherwise, if vlcNum is equal to 12,
 - codeNum = (leadingZeroBits << 6) + read_bits(6) (9-15)
- Otherwise, if vlcNum is equal to 13,
 - codeNum = (leadingZeroBits << 4) + read_bits(4) (9-16)
- Otherwise, if vlcNum is equal to 14,
 - leadingZeroBits = -1
 - for(b = 0; !b && leadingZeroBits < cMax; leadingZeroBits++) (9-17)
 - b = read_bits(1)
 - codeNum = leadingZeroBits
- Otherwise, if vlcNum is equal to 15, codeNum is determined according to Table 9-4.
- Otherwise, if vlcNum is equal to 16
 - codeNum = read_bits(3)
 - if(codeNum >= 6)
 - codeNum = 7 - codeNum
 - else if (codeNum > 0) {
 - b = read_bits(1)
 - codeNum = 13 - (codeNum << 1) - b (9-18)
 - }
 - else {
 - b = read_bits(2)

HEVC

```

        codeNum = 15 - b
    }
-   Otherwise, if vlcNum is equal to 17
    codeNum = read_bits(3)
    if( codeNum ) {
        b = read_bits(1)
        codeNum = 16 - (codeNum << 1) - b
    }

```

(9-19)

```

-   Otherwise, if vlcNum is equal to 18
    codeNum = read_bits(3)
    if( codeNum && codeNum < 4) {
        b = read_bits(1)
        codeNum = 8 - (codeNum << 1) - b
    } else if( codeNum < 6) {
        b = read_bits(2)
        codeNum = 30 - (codeNum << 2) - b
    } else {
        b = read_bits(3)
        codeNum = (codeNum << 3) + b
        if( codeNum < 62)
            codeNum = 76 - codeNum
        else {
            b = read_bits(1)
            codeNum = 156 - (codeNum << 1) - b
        }
    }

```

(9-20)

```

-   Otherwise, if vlcNum is equal to 19
    codeNum = read_bits(3)
    if( codeNum && codeNum < 3) {
        b = read_bits(1)
        codeNum = 6 - (codeNum << 1) - b
    } else {
        b = read_bits(2)
        codeNum = (codeNum << 2) + b
        if( codeNum < 25)
            codeNum = 29 - codeNum
        else {
            b = read_bits(1)
            codeNum = 81 - (codeNum << 1) - b
        }
    }

```

(9-21)

Table 9-49-4 – codeNum and codeword with vlcNum equal to 15

codeNum	Codeword
0	0
1	11
2	1001
3	1011
4	1000
5	10100
6	101010
7	101011

9.2.2 Initialisation process

Outputs of this process are initialised CAVLC internal variables.

HEVC

The processes in subclauses 9.2.2.2 through 9.2.2.6 are invoked when starting the parsing of the slice data of a slice in subclause 7.3.4.

9.2.2.1 Counter initialisation for codeword index adaptive mapping

Outputs of this process are initial values of counters that are used for parsing a syntax element.

If the number of counters used for a syntax element is N (N>0), values of the N counters are all initialized to 0. For each set of counters used for parsing a syntax element, there is an associated value, counterSum, to store the number of counter increments since its previous reset. The value of counterSum is also initialized to 0.

9.2.2.2 Initialisation process for lastPosVlcNumIndex

Outputs of this process are initial values of the variable array lastPosVlcNumIndex.

The variable array lastPosVlcIndex is initialized as follows.

$$\text{lastPosVlcNumIndex [blockType]} = 0, \text{ for blockType} = 0 \dots 7 \quad (9-22224)$$

9.2.2.3 Initialisation process for lastPosTable

Outputs of this process are initial values of the variable array lastPosTable.

The variable array lastPosTable is initialized as specified in Table 9-5.

Table 9-59-59-5 – Specification of initial values of lastPosTable[tableNum][codeNum]

	codeNum															
tableNum	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	2	3	5	4	7	6	11	8	14	9	16	15	10	13
1	0	1	2	5	7	4	6	3	9	12	11	8	10	14	13	15
2	0	1	2	5	7	4	6	3	9	12	11	8	10	14	13	15

	codeNum															
tableNum	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	12	17	18	19	22	31	23	21	27	20	25	30	24	26	28	29
1	16	17	21	22	25	18	28	23	30	26	20	19	27	24	29	31
2	16	17	21	22	25	18	28	23	30	26	20	19	27	24	29	31

9.2.2.4 Initialisation process for splitPredPartModeTable

Outputs of this process are initial values of the variable array splitPredPartModeTable and associated counters.

The variable array splitPredPartModeTable is initialized as follows:

- For each cuDepth with a value from 0 to (Log2MaxCUSize – Log2MinCUSize), set splitPredPartModeTable[cuDepth][codeNum] equal to codeNum, with codeNum = 0...6.

For each cuDepth with a value from 0 to (Log2MaxCUSize – Log2MinCUSize), there are four counters used for codeword index adaptive mapping. These counters are initialized according to subclause 9.2.2.1.

9.2.2.5 Initialisation process for intraModeTable

Outputs of this process are initial values of the variable array intraModeTable.

The variable array intraModeTable is initialized as specified in Table 9-6.

Table 9-69-6 – Specification of intraModeTable[k][codeNum]

k	codeNum																
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	15	11	10	13	7	9	4	14	2	3	6	8	5	12	1	
1	14	10	9	0	13	7	2	8	3	12	6	4	11	1	5		
2	2	0	29	30	20	1	21	28	15	7	16	8	11	31	22	19	32
3	2	1	28	0	29	20	27	19	51	21	7	14	10	11	30	31	18

k	codeNum																
	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
2	10	27	14	4	18	12	23	17	24	26	5	25	5	9	3	13	
3	16	26	8	17	4	22	9	23	6	25	13	24	12	3	5		

9.2.2.6 Initialisation process for cbpSplitTransTable

Outputs of this process are initial values of the variable array cbpSplitTransTable and associated counters.

The values of the variable array cbpSplitTransTable[k][n] are initialized as specified in Table 9-7 where the values of k = 0...5 and n = 0,1. The number of counters used in each case for codeword index adaptive mapping is defined in Table 9-16. These counters are initialized according to subclause 9.2.2.1.

Table 9-79-7 – Specification of initial values of cbpSplitTransTable[flagPattern-8][n][codeNum]

codeNum	k (n = 0)						k (n = 1)					
	0	1	2	3	4	5	0	1	2	3	4	5
0	14	2	3	6	4	4	0	2	0	0	0	0
1	13	3	1	4	7	7	1	3	2	5	4	4
2	12	1	0	5	0	0	3	0	1	4	5	5
3	10	0	2	7	5	5	7		3	7	6	6
4	6			0	6	6	2			6	1	1
5	11			1	1	1	4			2	7	7
6	9			2	2	2	5			3	2	2
7	8			3	3	3	8			1	3	3
8	5						9					
9	4						11					
10	2						6					
11	7						10					
12	3						12					
13	1						13					
14	0						14					

9.2.3 Codeword index mapping update process

Inputs to this process are a VLC codeword index mapping table indexMappingTable that provides the mapping between VLC codeword indexes and syntax element values, a codeword index value codeNum, and a value counterNum that specifies the number of counters used. If counterNum is not equal to 0, inputs additionally include an array counterArray indexed from 0 to (counterNum-1) that saves the value of counters, a value counterSum that saves the number of all counter increments since its previous reset.

Outputs of this process are indexMappingTable, and if counterNum is not equal to 0 counterArray and counterSum with updated values.

HEVC

The process is specified in the following ordered steps:

- Syntax element value `syntaxVal` is set equal to `indexMappingTable [codeNum]`.
- If `codeNum` is not less than 1, `codeNumPre` is set equal to `(codeNum-1)`. Otherwise, `codeNumPre` is set equal to `codeNum`. Syntax element value `syntaxValPre` is set equal to `indexMappingTable [codeNumPre]`.
- If `codeNum` is greater or equal to `counterNum`, the values of `indexMappingTable[codeNumPre]` and `indexMappingTable[codeNum]` are exchanged with each other. No further step is carried out. Otherwise
 - The value of `counterArray[codeNum]` is increased by 1.
 - If `counterArray[codeNum] >= counterArray[codeNumPre]`, the counter values of `counterArray[codeNum]` and `counterArray[codeNumPre]` are swapped, and the values of `indexMappingTable [codeNumPre]` and `indexMappingTable [codeNum]` are also swapped.
 - If `counterSum` is not less than 15, the value of `counterSum` is reset to 0. For $i = 0 \dots (\text{counterNum}-1)$, `counterArray[i]` is set equal to `(counterArray[i]>>1)`.
 - Otherwise, the value of `counterSum` is increased by 1.

9.2.4 Parsing process for transform coefficient syntax elements

This process is invoked when parsing syntax elements with descriptor equal to `ce(v)` in subclause 7.3.11 and 7.3.12 and when `entropy_coding_mode_flag` is equal to 0.

Inputs to this process are bits from RBSP, and the variables `log2TrafoSize` and `cIdx`, and the variable arrays `lastPosVlcNumIndex` and `lastPosTable`.

Outputs of this process are CAVLC syntax elements `last_pos_level_one`, `level_minus2_and_sign`, `run_level_one`, and `level`, and the variable arrays `lastPosTable` and `lastPosVlcNumIndex`.

9.2.4.1 Parsing process for last_pos_level_one

Inputs to this process are bits from RBSP and the variable arrays `lastPosVlcNumIndex` and `lastPosTable`.

Output of this process is the syntax element `last_pos_level_one`, the variables `trOne` and `vlcNumLevel`, and the variable arrays `lastPosVlcNumIndex` and `lastPosTable`.

The parsing process for `last_pos_level_one` is specified as follows.

- The variable `N` is set equal to $(1 \ll \log_2 \text{TrafoSize}) \gg (\text{cIdx} > 0 ? 2 : 0)$.
- If `N` is greater than 4, `last_pos_level_one` is derived in the following ordered steps:
 1. The variable `blockType` is derived as

$$\text{blockType} = (\text{cIdx} == 0 ? (\text{PredMode} == \text{MODE_INTRA} ? 0 : \text{slice_type} + 1) + (\text{N} > 8 ? 5 : 2) : \text{cIdx} - 1)$$
 (9-23)
 2. The variable `vlcNumIndex` is derived as

$$\text{vlcNumIdx} = \text{Min}(16, \text{lastPosVlcNumIndex}[\text{blockType}])$$
 (9-24)
 3. The variable `vlcNum` is derived as

$$\text{vlcNum} = \text{lastPosVlcNumTable}[\text{blockType}][\text{vlcNumIdx}]$$
 (9-25)

The values of the array `lastPosVlcNumTable` are shown in Table 9-8.
 4. The parsing process described in subclause 9.2.1 is invoked with `vlcNum` as input and the variable `codeNum` as output.
 5. The variable `lastPos` is derived as follows

$$\begin{aligned} &\text{if } (\text{codeNum} \leq \text{N} + \text{N} * \text{N}) \\ &\quad \text{if } (\text{codeNum} \neq 0 \ \&\& \ (\text{codeNum} \ \& \ (\text{N} - 1)) = 0) \\ &\quad \quad \text{lastPos} = (\text{codeNum} \gg \text{Log}_2(\text{N})) - 1 \\ &\quad \quad \text{else} \\ &\quad \quad \quad \text{lastPos} = \text{codeNum} - (\text{codeNum} \gg \text{Log}_2(\text{N})) \\ &\quad \text{else} \\ &\quad \quad \text{lastPos} = \text{codeNum} - \text{N} * \text{N} \end{aligned}$$
 (9-26)
 6. The variable `levelGreaterThanOneFlag` is derived as

HEVC

$$\text{levelGreaterThanOneFlag} = (\text{codeNum} > (N + N*N)) \mid \mid (\text{codeNum} \neq 0 \ \&\& \ (\text{codeNum} \ \& \ (N-1))) \quad (9-27)$$

7. The syntax element `last_pos_level_one` is derived as

$$\text{last_pos_level_one} = \text{levelGreaterThanOneFlag} ? \text{codeNum} + N : \text{codeNum} \quad (9-28)$$

8. The variable array `lastPosVlcNumIndex[blockType]` is updated as follows.

$$\begin{aligned} &\text{if } ((N = 8 ? \text{codeNum} : \text{codeNum} >> 2) < \text{lastPosVlcNumIndex}[\text{blockType}]) \\ &\quad \text{lastPosVlcNumIndex}[\text{blockType}] -= 1 \\ &\text{else if } ((N = 8 ? \text{codeNum} : \text{codeNum} >> 2) > \text{lastPosVlcNumIndex}[\text{blockType}]) \\ &\quad \text{lastPosVlcNumIndex}[\text{blockType}] += 1 \end{aligned} \quad (9-29)$$

– Otherwise, (N equal to 4), `last_pos_level_one` is derived in the following ordered steps:

1. The parsing process described in subclause 9.2.1 is invoked with 2 as input and the variable `codeNum` as output.

2. The variable `tableNum` is derived as

$$\text{tableNum} = (\text{PredMode} == \text{MODE_INTRA} \mid \mid \text{cIdx} > 0) ? 0 : \text{slice_type} + 1 \quad (9-30)$$

3. The syntax element `last_pos_level_one` is derived as

$$\text{last_pos_level_one} = \text{lastPosTable}[\text{tableNum}][\text{codeNum}] \quad (9-31)$$

4. The variable array `lastPosTable[tableNum]` is updated by invoking process 9.2.3 with `lastPosTable[tableNum]`, `codeNum` and 0 as inputs.

– The variable `trOne` is derived as

$$\text{trOne} = ! \text{levelGreaterThanOneFlag} \quad (9-32)$$

– The variable `vlcNumLevel` is set equal to 0

Table 9-89-8-8 – Specification of `lastPosVlcNumTable[blockType][vlcNumIdx]`

blockType	vlcNumIdx																
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	10	10	10	10	2	2	2	7	9	9	9	9	9	4	4	4	4
1	10	10	10	10	10	2	9	9	9	9	9	9	9	4	4	4	4
2	2	2	2	2	2	7	7	7	7	7	7	7	7	7	4	4	13
3	2	2	2	2	7	7	7	7	7	7	7	7	7	7	7	7	13
4	2	2	2	2	7	7	7	7	7	7	7	7	7	7	7	7	13
5	10	10	10	4	4	4	4	12	12	12	12	12	12	12	12	12	12
6	10	10	10	10	4	4	12	12	12	12	12	12	12	12	12	12	12
7	10	10	10	10	4	4	12	12	12	12	12	12	12	12	12	12	12

9.2.4.2 Parsing process for `level_minus2_and_sign`

Input to this process are bits from slice data.

Output of this process is `level_minus2_and_sign`.

The syntax element `level_minus2_and_sign` is derived by invoking the parsing process of subclause 9.2.1 with 0 as input and `level_minus2_and_sign` as output.

9.2.4.3 Parsing process for `run_level_one`

Inputs to this process are bits from slice data, the scan position n of the previous non-zero transform coefficient in inverse scan order, and the variable `TrOne`.

Outputs of this process are the syntax element `run_level_one` and the variable `TrOne`.

The value of `run_level_one` is derived as follows.

– The variable N is set equal to $(1 \ll \log_2 \text{TrafoSize}) \gg (\text{cIdx} > 0 ? 2 : 0)$.

– The variable `blockType` is derived as

HEVC

$$\text{blockType} = (\text{cIdx} == 0 ? (\text{PredMode} == \text{MODE_INTRA} ? 0 : \text{slice_type} + 1) + (\text{N} > 8 ? 5 : 2) : \text{cIdx} - 1) \quad (9-33)$$

- The variable tableIdx is derived as

$$\text{tableIdx} = (\text{blockType} == 2 \mid \mid \text{blockType} == 5) ? (\text{N} \leq 8 ? 0 : 1) : (\text{N} \leq 8 ? 2 : 3) \quad (9-34)$$

- The variable maxRun is set equal to n.
- The variable maxRunIdx is set equal to Min(28, maxRun).
- The variable vlcNum is derived from tableIdx and maxRunIdx as shown in Table 9-9.
- The parsing process described in subclause 9.2.1 is invoked with vlcNum as input and the variable codeNum as output. If blockType is equal to 2 or blockType is equal to 5, the value of run_level_one is derived as follows.
 - If N is equal to 4, the variable largeOnePos is derived from trOne and maxRunIdx as shown in Table 9-10.
 - Otherwise (N is greater than 4), the variable largeOnePos is derived from trOne and maxRunIdx as shown in Table 9-11.
 - The variables levelGreaterThanOneFlag and runOfZeros are derived as follows.

```

if ( largeOnePos > 0 ) {
    if( codeNum < min(largeOnePos, maxRunIdx + 2 ) ) {
        levelGreaterThanOneFlag = 0
        runOfZeros = codeNum
    } else if( codeNum < (maxRunIdx << 1 ) + 4 - largeOnePos ) {
        if( (codeNum + largeOnePos) & 1 ) {
            levelGreaterThanOneFlag = 0
            runOfZeros = (codeNum + largeOnePos - 1) >> 1
        } else {
            levelGreaterThanOneFlag = 1
            runOfZeros = (codeNum - largeOnePos) >> 1
        }
    } else {
        levelGreaterThanOneFlag = 1
        runOfZeros = codeNum - maxRunIdx - 2
    }
} else {
    if( codeNum & 1 ) {
        levelGreaterThanOneFlag = 0
        runOfZeros = (codeNum - 1) >> 1
    } else {
        runOfZeros = codeNum >> 1
        levelGreaterThanOneFlag = ((codeNum >> 1) <= maxRunIdx) ? 1 : 0
    }
}

```

- The syntax element run_level_one is derived as

$$\text{run_level_one} = \text{levelGreaterThanOneFlag} ? \text{runOfZeros} + \text{maxRun} : \text{runOfZeros} \quad (9-36)$$
- Otherwise (blockType is not equal to 2 and blockType is not equal to 5), the value of run_level_one is derived as follows.

- The variable levelGreaterThanOneFlag is derived as

$$\text{levelGreaterThanOneFlag} = \text{codeNum} \leq \text{maxRun} + 1 ? 0 : 1 \quad (9-37)$$

- If maxRun is less than 28, the variable runOfZeros is derived from maxRun and codeNum as shown in Table 9-12.

- Otherwise (maxRun is not less than 28), the variable runOfZeros is derived as

$$\text{runOfZeros} = \text{codeNum} \leq \text{maxRun} + 1 ? \text{codeNum} : \text{codeNum} - \text{maxRun} - 2 \quad (9-38)$$

- The syntax element run_level_one is derived as

$$\text{run_level_one} = \text{levelGreaterThanOneFlag} ? \text{runOfZeros} + \text{n} : \text{runOfZeros} \quad (9-39)$$

- The variable trOne is derived as

HEVC

$$\text{trOne} = (\text{trOne} == 0 \mid \mid \text{levelGreaterThanOneFlag}) ? 0 : \text{Max}(4, \text{trOne} + 1) \quad (9-40)$$

Table 9-99-99-9 – Derivation of vlcNum from tableIdx and maxRunIdx

	maxRunIdx														
tableIdx	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	8	0	0	5	5	5	5	5	5	5	6	6	6	6	6
1	8	0	0	5	5	5	5	5	5	5	6	6	6	6	6
2	8	0	0	0	0	5	5	5	5	5	5	5	5	5	5
3	8	0	1	1	1	1	2	2	2	2	2	2	2	6	6

	maxRunIdx														
tableIdx	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	6	6	6	6	6	6	6	6	6	6	6	6	6	3	6
1	6	6	6	6	6	6	6	6	6	6	6	6	6	3	6
2	5	5	5	1	1	1	1	1	1	1	1	1	1	2	5
3	6	6	6	6	3	3	3	3	3	3	3	3	3	3	6

Table 9-109-109-10 – Derivation of largeOnePos from tableIdx and maxRunIdx when N is equal to 4

	maxRunIdx														
trOne	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0
1	2	3	4	5	6	5	6	7	7	7	7	7	6	4	2
2	2	2	2	3	3	3	3	4	4	4	3	3	3	2	NA
3	2	1	1	2	2	2	2	2	2	2	2	2	1	NA	NA
4	1	1	1	1	1	1	1	1	1	1	1	1	NA	NA	NA

Table 9-119-119-11 – Derivation of largeOnePos from tableIdx and maxRunIdx when N is greater than 4

	maxRunIdx														
trOne	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0	1	2	2	2	2	2	4	5	5	6	6	6	6	6
1	2	4	4	6	6	8	8	10	11	13	15	13	14	15	16
2	2	3	4	5	4	5	6	6	7	8	8	9	9	10	10
3	2	1	2	2	3	3	4	4	5	5	5	6	6	7	7
4	2	1	2	2	2	2	3	3	3	3	4	4	4	4	4

	maxRunIdx													
trOne	15	16	17	18	19	20	21	22	23	24	25	26	27	28
0	6	7	8	7	8	8	9	10	10	10	12	10	9	8
1	18	18	21	20	21	22	23	25	25	26	27	28	29	27
2	11	11	12	13	14	15	16	16	17	18	19	20	19	19
3	7	8	9	9	10	10	11	11	12	12	13	13	13	13
4	5	5	5	5	5	6	6	6	6	6	6	7	7	8

HEVC

Table 9-129-129-12 – Derivation of runOfZeros from maxRun and codeNum

	codeNum%(maxRun+1)														
maxRun	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1	0	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
1	2	1	0	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
2	3	0	2	1	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
3	4	1	0	2	3	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
4	5	0	3	2	4	1	NA	NA	NA	NA	NA	NA	NA	NA	NA
5	6	0	1	4	5	3	2	NA	NA	NA	NA	NA	NA	NA	NA
6	7	1	0	2	4	3	6	5	NA	NA	NA	NA	NA	NA	NA
7	8	0	3	4	2	1	5	7	6	NA	NA	NA	NA	NA	NA
8	9	0	5	1	6	4	8	3	2	7	NA	NA	NA	NA	NA
9	10	0	1	6	7	2	9	5	4	3	8	NA	NA	NA	NA
10	11	1	0	2	3	6	7	4	5	8	10	9	NA	NA	NA
11	12	0	3	2	1	4	6	5	7	11	8	9	10	NA	NA
12	13	0	1	5	6	4	2	3	7	8	12	10	11	9	NA
13	14	0	1	7	2	8	6	5	3	4	13	12	9	11	10
14	15	0	1	2	8	3	9	7	6	4	5	14	10	13	12
15	16	0	1	2	3	4	8	7	5	9	6	10	15	11	13
16	17	0	3	1	2	4	5	8	9	7	6	10	11	12	16
17	18	0	1	5	2	6	4	3	7	8	9	10	17	13	14
18	19	0	1	7	8	2	6	3	9	4	5	10	18	15	16
19	20	0	9	1	10	2	8	3	7	4	16	17	6	19	5
20	21	0	1	10	2	11	3	9	5	4	17	20	18	8	7
21	22	1	0	2	3	4	10	11	5	6	9	7	12	8	13
22	23	0	3	4	2	1	5	7	6	10	11	8	9	12	13
23	24	0	5	6	1	4	3	7	2	8	10	11	9	12	15
24	25	0	7	1	8	6	5	2	9	10	4	11	3	12	17
25	26	0	1	9	10	2	8	11	7	3	6	5	12	4	19
26	27	0	1	11	2	10	12	3	9	7	8	4	6	20	13
27	28	0	1	2	12	3	4	11	13	9	5	8	10	7	6

	codeNum%(maxRun+1)													
maxRun	15	16	17	18	19	20	21	22	23	24	25	26	27	28
0	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
1	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
2	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
3	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
4	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
5	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
6	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
7	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
8	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
9	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
10	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
11	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA

HEVC

12	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
13	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
14	11	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
15	14	12	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
16	13	14	15	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
17	12	15	11	16	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
18	14	11	13	17	12	NA	NA	NA	NA	NA	NA	NA	NA	NA
19	11	15	12	14	13	18	NA	NA	NA	NA	NA	NA	NA	NA
20	6	12	16	13	15	19	14	NA	NA	NA	NA	NA	NA	NA
21	18	21	17	19	14	20	16	15	NA	NA	NA	NA	NA	NA
22	14	22	18	15	19	20	17	16	21	NA	NA	NA	NA	NA
23	14	13	23	16	19	17	18	21	20	22	NA	NA	NA	NA
24	16	13	24	18	19	15	20	23	14	21	22	NA	NA	NA
25	13	18	20	14	17	25	21	16	23	24	15	22	NA	NA
26	5	14	26	21	19	18	25	15	16	17	24	22	23	NA
27	21	14	27	15	22	19	16	20	17	26	18	25	23	24

9.2.4.4 Parsing process for level

Inputs to this process are bits from slice data and the variable vlcNumLevel.

Outputs of this process are the syntax element level and the variable vlcNumLevel.

- The syntax element level is determined by invoking the parsing process described in subclause 9.2.1 with vlcNumLevel as input and level as output.
- The variable vlcNumLevel derived in the following ordered steps updated as follows.
 1. The variable vlcIdx is set equal to $vlcIdx = \text{Min}(3, vlcNumLevel)$
 2. The variable levelThreshold is derived from vlcIdx as shown in Table 9-13.
 3. The variable vlcNumLevel is derived as

$$vlcNumLevel = vlcNumLevel + (\text{level} > \text{levelThreshold} \ \&\& \ vlcNumLevel < 4 ? 1 : 0) \quad (9-41)$$

Table 9-139-13 – Derivation of levelThreshold from vlcIdx

vlcIdx	levelThreshold
0	4
1	6
2	14
3	28

9.2.4.5 Parsing process for cu_split_pred_part_mode

This process is invoked when entropy_coding_mode_flag is equal to 0 for parsing syntax element cu_split_pred_part_mode in subclause 7.3.5.

Inputs to this process are bits from slice data, splitPredPartModeTable.

Output of this process is the syntax element cu_split_pred_part_mode.

The value of cu_split_pred_part_mode is derived as follows.

- If $\log_2 \text{CUsSize}$ is greater than $\log_2 \text{MinCUsSize}$, the parsing process described in subclause 9.2.1 is invoked with vlcNum of 14 and cMax of 6 as inputs and the variable codeNum as output.
- Otherwise, the parsing process described in subclause 9.2.1 is invoked with vlcNum of 14 and cMax of 5 as inputs and the variable codeNum as output.
- The syntax element cu_split_pred_part_mode is derived as

HEVC

$cu_split_pred_part_mode = splitPredPartModeTable[Log2MaxCUsSize - log2CUsSize][codeNum]$ (9-424244)

- If $log2CUsSize$ is equal to $Log2MinCUsSize$ and $cu_split_part_mode$ has a value of 5, $PredMode$ and $PartMode$ are determined as follows
 1. Read one bit and assign its value to a variable b
 2. If b is equal to 1, set $PredMode$ to $MODE_INTRA$ and $PartMode$ to $PART_2Nx2N$
 3. Otherwise, read the next one bit and assign its value to the variable b .
 - if b is equal to 1, set $PredMode$ to $MODE_INTRA$ and $PartMode$ to $PART_NxN$
 - Otherwise, set $PredMode$ to $MODE_INTER$ and $PartMode$ to $PART_NxN$
- The variable array $splitPredPartModeTable[Log2MaxCUsSize - log2CUsSize]$ is updated by invoking process 9.2.3 with $splitPredPartModeTable[Log2MaxCUsSize - log2CUsSize]$, $codeNum$ and a value of 4 as inputs.

9.2.4.6 Parsing process for $rem_intra_luma_pred_mode$

This process is invoked when $entropy_coding_mode_flag$ is equal to 0 for parsing syntax element $rem_intra_luma_pred_mode$ in subclause 7.3.7.

Inputs to this process are bits from slice data, a variable $puSize$ specifying the size of the current prediction unit, $NumMPCMand$ and a variable array $intraModeTable$.

Outputs of this process are the syntax element $rem_intra_luma_pred_mode$ and value-updated $intraModeTable$.

The value of $rem_intra_luma_pred_mode$ is derived as follows.

- Based on the input $puSize$, the value of a variable $intraPredModeNum$ indicating the number of intra prediction modes for the given size of prediction unit is obtained according to [Table 7-12Table 7-11](#).
- If $intraPredModeNum$ is equal to 3, read one bit and assign its value to $rem_intra_luma_pred_mode$. No further step is carried out. Otherwise,
- The values of variable k and $vlcNum$ are obtained as follows
 - If $intraPredModeNum$ is equal to 17
 - If $NumMPCMand$ is equal to 1, set $vlcNum$ equal to 16 and k equal to 0.
 - Otherwise, set $vlcNum$ equal to 17 and k equal to 1.
 - If $intraPredModeNum$ is equal to 34
 - If $NumMPCMand$ is equal to 1, set $vlcNum$ equal to 18 and k equal to 2.
 - Otherwise, set $vlcNum$ equal to 19 and k equal to 3.
- The parsing process described in subclause 9.2.1 is invoked with $vlcNum$ as input and the variable $codeNum$ as output.
- The value of $rem_intra_luma_pred_mode$ is set equal to $intraModeTable[k][codeNum]$.
- The value of a variable $counterNum$ is set to 0. The variable array $intraModeTable[k]$ is updated by invoking process in subclause 9.2.3 with $intraModeTable[k]$, $codeNum$ and $counterNum$ as inputs.

9.2.4.7 Parsing process for $cbp_and_split_transform$

This process is invoked when $entropy_coding_mode_flag$ is equal to 0 for parsing syntax element $cbp_and_split_transform$ in subclause 7.3.8.

Inputs to this process are bits from slice data, a variable $cbpVlcNumIdx$, $trafoDepth$ and a variable array $cbpSplitTransTable$.

Outputs of this process are the syntax element $cbp_and_split_transform$ and value-updated $cbpVlcNumIndex$ and $cbpSplitTransTable$.

The value of $cbp_and_split_transform$ is derived as follows.

- Obtain a 4-bit value $flagPattern$ according to subclause 7.4.8.
- Derive the value of a variable k according to Table 9-16.
- Set the value of a variable n as

$n = (PredMode == MODE_INTRA ? 0 : 1)$ (9-43)

HEVC

- Obtain a value of vlcNum as follows
 - If flagPattern is equal to 8,

$$\text{vlcNum} = (n == 0 ? \text{cbpVlcNumTable}[\text{cbpVlcNumIdx}]:11) \quad (9-44)$$
 - Otherwise, if flagPattern is equal to 11, 13 or 15 and n is equal to 0, set vlcNum to 15;
 - Otherwise, set vlcNum to 14.
- If vlcNum is equal to 14, obtain the value of a variable cMax according to Table 9-15.
- The parsing process described in subclause 9.2.1 is invoked with vlcNum and cMax as inputs and the variable codeNum as output.
- Set the value of cbp_and_split_transform equal to cbpSplitTransTable[k][n][codeNum]
- If flagPattern is not equal to 10 or 12, a variable counterNum is obtained according to Table 9-16 and the variable array cbpSplitTransTable[k][n] is updated by invoking process 9.2.3 with cbpSplitTransTable[k][n], codeNum and counterNum as inputs.
- If flagPattern is equal to 8 and n is equal to 0, the value of cbpVlcNumIdx is updated as follows

$$\begin{aligned} &\text{if}(\text{codeNum} < \text{cbpVlcNumIdx}) \\ &\quad \text{cbpVlcNumIdx} -= 1 \\ &\text{else if}(\text{codeNum} > \text{cbpVlcNumIdx}) \\ &\quad \text{cbpVlcNumIdx} += 1 \end{aligned} \quad (9-45)$$
- If flagPattern is equal to 15, the exact values of Cb and Cr in Table 7-14 are determined as follows.
 - The parsing process described in subclause 9.2.1 is invoked with vlcNum of 14 and cMax of 2 as inputs and the variable codeNum as output.
 - Set the value of variable b as

$$b = n ? (\text{codeNum} + 1) : (3 - \text{codeNum}) \quad (9-46)$$
 - The values of Cb and Cr are set equal to

$$\text{Cb} = (b \gg 1) \quad (9-47)$$

$$\text{Cr} = (b \& 0x01) \quad (9-48)$$

Table 9-149-149-14 – Specification of cbpVlcNumTable[cbpVlcNumIdx]

cbpVlcNumIdx	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
vlcNum	1	2	2	2	11	11	11	11	11	11	11	11	11	11	11

Table 9-159-159-15 – Derivation of cMax from flagPattern and n

n	flagPattern			
	9	10, 12	11,13,15	14
0	3	3	-	7
1	2	3	6	7

Table 9-169-169-16 – Specification of cbpSplitTransTable and counterNum associated

flagPattern	8	9	10, 12	11, 13, 15	14 (trafoDepth=0)	14 (trafoDepth>0)
k	0	1	2	3	4	5
counterNum	2	3	0	4	4	4

HEVC

9.3 CABAC parsing process for slice data

[Ed. (TW) needs to be extended]

This process is invoked when parsing syntax elements with descriptor $ae(v)$ in subclauses 7.3.4 and 7.3.5 when $entropy_coding_mode_flag$ is equal to 1.

Inputs to this process are a request for a value of a syntax element and values of prior parsed syntax elements.

Output of this process is the value of the syntax element.

When starting the parsing of the slice data of a slice in subclause 7.3.4, the initialisation process of the CABAC parsing process is invoked as specified in subclause 9.3.1.

The parsing of syntax elements proceeds as follows:

For each requested value of a syntax element a binarization is derived as described in subclause 9.3.2.

The binarization for the syntax element and the sequence of parsed bins determines the decoding process flow as described in subclause 9.3.2.109.3.3.

For each bin of the binarization of the syntax element, which is indexed by the variable $binIdx$, a context index $ctxIdx$ is derived as specified in subclause 9.3.3.1.

For each $ctxIdx$ the arithmetic decoding process is invoked as specified in subclause 9.3.3.2.

The resulting sequence ($b_0..b_{binIdx}$) of parsed bins is compared to the set of bin strings given by the binarization process after decoding of each bin. When the sequence matches a bin string in the given set, the corresponding value is assigned to the syntax element.

In case the request for a value of a syntax element is processed for the syntax element mb_type and the decoded value of mb_type is equal to I_PCM , the decoding engine is initialised after the decoding of any $pcm_alignment_zero_bit$ and all pcm_sample_luma and pcm_sample_chroma data as specified in subclause 9.3.1.2.

9.3.1 Initialisation process

Outputs of this process are initialised CABAC internal variables.

The processes in subclauses 9.3.1.1 and 9.3.1.2 are invoked when starting the parsing of the slice data of a slice in subclause 7.3.4.

9.3.1.1 Initialisation process for context variables

Outputs of this process are the initialised CABAC context variables indexed by $ctxIdx$.

Table 9-18 Tables 9-5 to Table 9-43 9-26 contain the values of the variables n and m used in the initialisation of context variables that are assigned to all syntax elements in subclauses 7.3.4 to 7.3.10 except for the end-of-slice flag and those related to CAVLC.

For each context variable, the two variables $pStateIdx$ and $valMPS$ are initialised.

NOTE 1 – The variable $pStateIdx$ corresponds to a probability state index and the variable $valMPS$ corresponds to the value of the most probable symbol as further described in subclause 9.3.3.2.

The two values assigned to $pStateIdx$ and $valMPS$ for the initialisation are derived from $SliceQP_Y$, which is derived in Equation $SliceQP_Y = 26 + pic_init_qp_minus26 + slice_qp_delta(7-127-9)$. Given the two table entries (m, n), the initialisation is specified by the following pseudo-code process:

```
preCtxState = Clip3( 1, 126, ( ( m * Clip3( 0, 51, SliceQP_Y ) ) >> 4 ) + n )
if( preCtxState <= 63 ) {
    pStateIdx = 63 - preCtxState
    valMPS = 0
} else {
    pStateIdx = preCtxState - 64
    valMPS = 1
}
```

(9-49494)

In Table 9-17 Table 9-4, the $ctxIdxTable$ and the $ctxIdx$ for which initialisation is needed for each of the slice types are listed. The referenced context index tables $ctxIdxTable$ include the values of m and n needed for the initialisation.

HEVC

Table 9-179-179-17 – Association of ctxIdx and syntax elements for each slice type in the initialisation process

	Syntax element	ctxIdxTable	Slice Type		
			I	P	B
slice_header()	alf_cu_flag	Table 9-18Table 9-5	0 2	3..5	6..8
coding_tree()	split_coding_unit_flag	Table 9-19Table 9-6	0..2	3..5	6..8
coding_unit()	skip_flag	Table 9-20Table 9-7		0..2	3..5
	cu_qp_delta	Table 9-21Table 9-8	0..3	4..7	8..11
	pred_type	Table 9-22Table 9-9	0	1..4	5..9
prediction_unit()	prev_intra_luma_pred_flag	Table 9-23Table 9-10	0	1	2
	rem_intra_luma_pred_mode	Table 9-24Table 9-11	0	1	2
	intra_chroma_pred_mode	Table 9-25Table 9-12	0..3	4..7	8..11
	merge_flag	Table 9-26Table 9-13		0..2	3..5
	merge_idx	Table 9-27Table 9-14		0..3	4..7
	inter_pred_flag	Table 9-28Table 9-15			0..2
	ref_idx_lc, ref_idx_l0, ref_idx_l1	Table 9-29Table 9-16		0..5	6..11
	mvd_l0[][][0]	Table 9-30Table 9-17		0..6	14..20
	mvd_l0[][][0], mvd_l1[][][0]	Table 9-30Table 9-17			14..20
	mvd_l0[][][1]	Table 9-30Table 9-17		7..13	21..27
mvd_l0[][][1]	Table			21..2	

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

HEVC

	}, mvd_11[][][]	Table 9-30F 9-17			7	
	mvp_idx_lc, mvp_idx_10, mvp_idx_11	Table 9-31F 9-18		0..1	2..3	
transfor m_tree()	no_residual_data_flag	Table 9-32F 9-19		0..3	4..7	
	split_transform_flag	Table 9-33F 9-20	0..3	4..7	8..11	
	cbf_luma	Table 9-34F 9-21	0..3	4..7	8..11	
	cbf_cb	Table 9-35F 9-22	0..3	4..7	8..11	
	cbf_cr	Table 9-36F 9-23	0..3	4..7	8..11	
residual coding()	last_significant_coeff_x	Table 9-37F 9-24	0..40	41..81	82..122	
	last_significant_coeff_y	Table 9-38F 9-25	0..40	41..81	82..122	
	needs update	Table 9-39F 9-26				
	significant_coeff_flag					

Initial isatio	significant coeff flag ctxIdx															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
m	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
n	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
	1	1	1	1	2	2	2	2	2	2	2	2	2	2	3	3
m	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
n	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
	3	3	3	3	3	3	3	3	4	4	4	4	4	4	4	4
m	-	-	-	-	5	-	-	-	-	-	0	-	2	0	-	0
n	1	9	9	1	3	6	1	7	6	1	2	4	5	6	9	6
	4	4	5	5	5	5	5	5	5	5	5	5	6	6	6	6
m	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
n	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
	6	6	6	6	6	6	7	7	7	7	7	7	7	7	7	7
m	-	0	-	-	1	2	-	1	2	-	6	5	2	0	-	0
n	8	6	7	7	7	5	8	3	5	8	1	4	5	6	8	6
	8	8	8	8	8	8	8	8	8	8	9	9	9	9	9	9
m	-	-	-	-	-	-	-	4	-	-	2	5	6	-	0	-
n	9	7	7	7	7	7	8	9	4	8	7	4	5	5	1	6
	9	9	9	9	1	1	1	1	1	1	1	1	1	1	1	1
m	-	-	-	-	5	3	-	-	0	-	8	4	-	-	0	-
n	9	7	7	7	3	5	9	5	5	8	3	3	5	6	8	6

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt, Highlight

Formatted: Font: Times New Roman, 8 pt, English (United Kingdom), Highlight

Formatted: Font: 8 pt, Highlight

Formatted Table

HEVC

		Table 9-40			
coeff_abs_level_greater1_flag	▲	0..79	80..159	160..239	
	Table 9-42F 9-29				
coeff_abs_level_greater2_flag	▲	0..79	80..159	160..239	
	Table 9-43F 9-30				

Formatted: Font: 8 pt
Formatted: Keep with next

Formatted: Font: 8 pt

NOTE 2 – ctxIdxTable equal to 0 and ctxIdx equal to 0 are associated with the end_of_slice_flag. The decoding process specified in subclause 9.3.3.2.4 applies to ctxIdxTable equal to 0 and ctxIdx equal to 0. This decoding process, however, may also be implemented by using the decoding process specified in subclause 9.3.3.2.1. In this case, the initial values associated with ctxIdxTable equal to 0 and ctxIdx equal to 0 are specified to be pStatIdx = 63 and valMPS = 0, where pStatIdx = 63 represents a non-adapting probability state.

Table 9-189-189-18 – Values of variables m and n for alf_cu_flag ctxIdx

Initialisation variables	alf_cu_flag ctxIdx								
	0	1	2	3	4	5	6	7	8
m	0	0	0	0	0	0	0	0	0
n	64	64	64	64	64	64	64	64	64

Table 9-199-199-19 – Values of variables m and n for split_coding_unit_flag ctxIdx

Initialisation variables	split_coding_unit_flag ctxIdx								
	0	1	2	3	4	5	6	7	8
m	-7	-10	-10	-14	-6	-6	-14	-7	-10
n	68	87	105	71	73	91	71	74	92

Table 9-209-209-20 – Values of variables m and n for skip_flag ctxIdx

Initialisation variables	skip_flag ctxIdx					
	0	1	2	3	4	5
m	0	0	0	0	0	0
n	64	64	64	64	64	64

Table 9-219-219-21 – Values of variables m and n for cu_qp_delta ctxIdx

Initialisation variables	cu_qp_delta flag ctxIdx											
	0	1	2	3	4	5	6	7	8	9	10	11
m	0	0	0	0	0	0	0	0	0	0	0	0
n	64	64	64	64	64	64	64	64	64	64	64	64

HEVC

Table 9-229-229-22 – Values of variables m and n for pred_type

Initialisation variables	pred_type ctxIdx									
	0	1	2	3	4	5	6	7	8	9
m	0	-25	-1	-3	6	6	-1	13	-11	-11
n	73	89	64	63	78	50	56	53	76	70

Table 9-239-239-23 – Values of variable m and n for prev_intra_luma_pred_flag ctxIdx

Initialisation variables	prev_intra_luma_pred_flag ctxIdx		
	0	1	2
m	2	0	0
n	54	50	51

Table 9-249-249-24 – Values of variable m and n for rem_intra_luma_pred_mode ctxIdx

Initialisation variables	rem_intra_luma_pred_mode ctxIdx		
	0	1	2
m	-3	-2	1
n	65	61	55

Table 9-259-259-25 – Values of variable m and n for intra_chroma_pred_mode ctxIdx

Initialisation variables	intra_chroma_pred_mode ctxIdx											
	0	1	2	3	4	5	6	7	8	9	10	11
m	0	0	0	0	0	0	0	0	0	0	0	0
n	64	64	64	64	64	64	64	64	64	64	64	64

Table 9-269-269-26 – Values of variable m and n for merge_flag ctxIdx

Initialisation variables	merge_flag ctxIdx					
	0	1	2	3	4	5
m	0	0	0	0	0	0
n	64	64	64	64	64	64

Table 9-279-279-27 – Values of variable m and n for merge_idx ctxIdx

Initialisation variables	merge_idx ctxIdx							
	0	1	2	3	4	5	6	7
m	0	0	0	0	1	6	-7	-4
n	64	64	64	64	65	42	75	72

HEVC

Table 9-289-289-28 – Values of variable m and n for inter_pred_flag ctxIdx

Initialisation variables	inter_pred_flag ctxIdx		
	0	1	2
m	-2	-5	-9
n	58	70	85

Table 9-299-299-29 – Values of variable m and n for ref_idx_lc, ref_idx_l0, ref_idx_l1 ctxIdx

Initialisation variables	ref_idx_lc, ref_idx_l0, ref_idx_l1 ctxIdx											
	0	1	2	3	4	5	6	7	8	9	10	11
m	-6	-10	-8	-17	1	0	-9	-9	-9	-12	-18	0
n	59	75	75	96	59	64	55	71	76	86	55	64

Table 9-309-309-30 – Values of variable m and n for mvd_lc, mvd_l0, mvd_l1 ctxIdx

Initialisation variables	mvd_lc, mvd_l0, mvd_l1 ctxIdx													
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
m	5	-2	-7	14	16	6	9	2	-3	-10	14	17	6	8
n	61	79	89	39	49	60	71	63	78	94	37	46	60	73
	14	15	16	17	18	19	20	21	22	23	24	25	26	27
m	1	-4	-10	11	14	4	7	1	-4	-10	11	15	5	8
n	67	83	97	49	56	69	77	65	81	96	48	52	67	76

Table 9-319-319-31 – Values of variable m and n for.mvp_idx_lc,.mvp_idx_l0,.mvp_idx_l1 ctxIdx

Initialisation variables	mvp_idx_lc, mvp_idx_l0, mvp_idx_l1 ctxIdx			
	0	1	2	3
m	0	0	0	0
n	64	64	64	64

Table 9-329-329-32 – Values of variable m and n for no_residual_data_flag ctxIdx

Initialisation variables	no_residual_data_flag ctxIdx							
	0	1	2	3	4	5	6	7
m	-14	-12	-8	-25	-26	-16	-14	-27
n	72	86	78	128	88	91	87	132

Table 9-339-339-33 – Values of variable m and n for split_transform_flag ctxIdx

Initialisation variables	split_transform_flag ctxIdx											
	0	1	2	3	4	5	6	7	8	9	10	11
m	0	12	22	-2	0	-28	-30	-34	-11	-31	-42	-47
n	64	12	4	49	13	89	99	106	38	88	118	130

HEVC

Table 9-349-349-34 – Values of variable m and n for cbf_luma ctxIdx

Initialisation variables	cbf_luma ctxIdx											
	0	1	2	3	4	5	6	7	8	9	10	11
m	-22	-5	-16	-16	-18	-41	-29	-23	-11	-32	-19	-16
n	116	75	112	111	98	120	117	108	80	83	89	85

Table 9-359-359-35 – Values of variable m and n for cbf_cb ctxIdx

Initialisation variables	cbf_cb ctxIdx											
	0	1	2	3	4	5	6	7	8	9	10	11
m	-35	-12	-9	-10	-46	-42	-11	-19	-22	-48	-7	-37
n	116	61	73	75	114	119	74	90	52	123	68	121

Table 9-369-369-36 – Values of variable m and n for cbf_cr ctxIdx

Initialisation variables	cbf_cr ctxIdx											
	0	1	2	3	4	5	6	7	8	9	10	11
m	-29	-12	-5	-6	-43	-41	-17	-25	-19	-48	-21	-9
n	104	59	65	67	107	118	86	101	45	123	94	73

HEVC

Table 9-379-379-37 – Values of variable m and n for last_significant_coefficient_x ctxIdx

Initialisation variables	last significant coeff v ctxIdx														
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
m	19	12	16	22	12	12	12	5	16	15	17	17	19	19	4
n	19	36	34	18	35	35	32	46	21	20	13	14	10	12	37
	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
m	13	22	27	26	18	6	12	34	38	24	14	41	45	56	30
n	22	-4	-19	-12	6	27	10	-33	-42	-15	7	7	1	-9	22
	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
m	14	23	19	25	29	29	19	10	12	-2	-37	15	14	15	25
n	40	24	32	26	10	4	20	37	38	60	114	29	36	41	8
	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
m	26	25	21	12	18	25	32	32	29	24	13	27	40	42	40
n	4	10	16	34	21	1	-12	-15	-11	-5	12	6	-32	-39	-39
	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74
m	45	43	45	38	15	11	11	26	28	21	15	27	29	23	-1
n	-51	-51	-56	-46	-6	-4	-4	29	29	46	45	17	12	15	57
	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89
m	6	22	31	37	43	43	48	15	14	15	25	26	25	21	12
n	65	17	-4	-25	-44	-46	-44	29	36	41	8	4	10	16	34
	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104
m	18	25	32	32	29	24	13	27	40	42	40	45	43	45	38
n	21	1	-12	-15	-11	-5	12	6	-32	-39	-39	-51	-51	-56	-46
	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119
m	15	11	11	26	28	21	15	27	29	23	-1	6	22	31	37
n	-6	-4	-4	29	29	46	45	17	12	15	57	65	17	-4	-25
	120	121	122												
m	43	43	48												
n	-44	-46	-44												

HEVC

Table 9-389-389-38 – Values of variable m and n for last_significant_coeff_y ctxIdx

Initialisation variables	last significant coeff y ctxIdx														
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
m	19	12	16	22	12	12	12	5	16	15	17	17	19	19	4
n	19	36	34	18	35	35	32	46	21	20	13	14	10	12	37
	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
m	13	22	27	26	18	6	12	34	38	24	14	41	45	56	30
n	22	-4	-19	-12	6	27	10	-33	-42	-15	7	7	1	-9	22
	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
m	14	23	19	25	29	29	19	10	12	-2	-37	15	14	15	25
n	40	24	32	26	10	4	20	37	38	60	114	29	36	41	8
	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
m	26	25	21	12	18	25	32	32	29	24	13	27	40	42	40
n	4	10	16	34	21	1	-12	-15	-11	-5	12	6	-32	-39	-39
	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74
m	45	43	45	38	15	11	11	26	28	21	15	27	29	23	-1
n	-51	-51	-56	-46	-6	-4	-4	29	29	46	45	17	12	15	57
	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89
m	6	22	31	37	43	43	48	15	14	15	25	26	25	21	12
n	65	17	-4	-25	-44	-46	-44	29	36	41	8	4	10	16	34
	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104
m	18	25	32	32	29	24	13	27	40	42	40	45	43	45	38
n	21	1	-12	-15	-11	-5	12	6	-32	-39	-39	-51	-51	-56	-46
	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119
m	15	11	11	26	28	21	15	27	29	23	-1	6	22	31	37
n	-6	-4	-4	29	29	46	45	17	12	15	57	65	17	-4	-25
	120	121	122												
m	43	43	48												
n	-44	-46	-44												

HEVC

Table 9-399-399-39 – Values of variable m and n for significant_coeff_flag ctxIdx (A)

Initialisation variables	significant_coeff_flag ctxIdx															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
m	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
n	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
m	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
n	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64
	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
m	-22	-14	-17	-26	5	-2	-32	-19	-7	-24	0	-1	2	0	-15	0
n	128	96	99	113	31	65	125	72	69	111	25	49	54	64	97	64
	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
m	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
n	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64
	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
m	-4	0	-4	-5	17	2	-8	1	2	-11	6	5	2	0	-8	0
n	88	68	75	74	7	56	80	36	52	85	12	41	54	65	82	64
	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
m	-8	-7	-7	-9	-18	-7	-11	-29	4	-14	-26	2	5	6	-22	0
n	95	77	73	74	77	72	81	92	47	83	79	46	50	52	104	64
	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
m	-7	-5	-6	-8	5	3	-18	-10	0	-11	-3	8	4	-1	-7	0
n	93	79	79	79	36	56	99	59	56	84	31	35	51	66	81	64

Table 9-409-409-40 – Values of variable m and n for significant_coeff_flag ctxIdx (B)

Initialisation variables	significant_coeff_flag ctxIdx															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
m	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
n	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
m	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
n	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64
	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
m	-22	-14	-17	-26	5	-2	-32	-19	-7	-24	0	-1	2	0	-15	0
n	128	96	99	113	31	65	125	72	69	111	25	49	54	64	97	64
	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
m	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
n	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64
	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
m	-4	0	-4	-5	17	2	-8	1	2	-11	6	5	2	0	-8	0
n	88	68	75	74	7	56	80	36	52	85	12	41	54	65	82	64
	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
m	-8	-7	-7	-9	-18	-7	-11	-29	4	-14	-26	2	5	6	-22	0
n	95	77	73	74	77	72	81	92	47	83	79	46	50	52	104	64
	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
m	-7	-5	-6	-8	5	3	-18	-10	0	-11	-3	8	4	-1	-7	0
n	93	79	79	79	36	56	99	59	56	84	31	35	51	66	81	64

HEVC

Table 9-419-419-41 – Values of variable m and n for significant_coeff_flag ctxIdx (C)

Initialisation variables	significant_coeff_flag ctxIdx															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
m	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
n	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
m	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
n	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64
	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
m	-34	-20	-23	38	3	-4	-17	-9	-4	-16	-1	0	-1	-12	-14	0
n	146	108	111	41	34	69	103	50	66	97	24	50	61	90	97	64
	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
m	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
n	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64
	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
m	-12	-4	-8	-7	15	2	-10	4	-10	-9	-9	6	3	-1	-14	0
n	99	75	81	80	13	57	85	27	75	81	46	41	56	69	96	64
	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
m	-7	-22	-12	-35	-5	-16	-9	-26	-28	-38	-3	-25	-9	6	-15	0
n	85	102	79	121	55	88	78	73	99	123	35	97	75	54	92	64
	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
m	-9	-14	-4	-14	7	3	-7	-8	-7	-8	-2	4	1	-5	-10	0
n	94	94	74	92	37	57	78	54	70	78	31	49	60	75	87	64

HEVC

Table 9-429-429-42 – Values of variable m and n for coeff_abs_level_greater1_flag ctxIdx

Initialisation variables	coeff_abs_level_greater1_flag ctxIdx														
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
m	-11	-20	-16	-13	-10	-5	-8	-8	-3	-9	0	-5	-12	-9	-1
n	87	64	68	71	73	67	26	37	36	56	63	39	56	57	52
	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
m	-4	-19	-28	-23	-3	-2	-27	-22	-14	-1	-10	-22	-13	-6	-5
n	72	73	88	85	59	72	97	89	77	58	86	74	63	57	63
	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
m	-4	16	9	5	-7	-6	-13	-12	-18	-14	-5	-8	-22	-35	-3
n	70	-2	23	41	67	63	-5	42	53	59	65	36	67	89	47
	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
m	-12	-20	-51	-44	-23	-1	-58	-71	-67	-91	-1	20	13	2	5
n	77	66	113	109	84	64	127	143	134	187	64	-3	21	46	48
	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74
m	-4	29	1	-6	-9	-6	23	3	-5	5	-12	-2	-15	-11	-3
n	71	-5	45	58	67	66	-7	26	42	31	79	45	67	62	54
	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89
m	-4	-13	-3	-26	-18	-9	-30	-24	-43	-6	4	34	21	14	5
n	70	68	100	91	84	83	103	90	122	66	59	-11	10	25	44
	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104
m	-4	40	19	-11	-6	-11	12	17	-6	11	-11	1	-26	-28	-2
n	69	-33	8	65	59	68	-2	-10	34	14	70	27	71	79	47
	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119
m	6	-23	-47	-55	-21	1	-38	-34	-45	10	9	41	32	14	17
n	47	67	107	117	83	59	82	77	95	25	46	-31	-17	19	18
	120	129	130	131	132	133	134	135	136	137	138	139	140	141	142
m	-2	18	10	-2	-7	-6	19	-5	-7	-4	-23	-3	2	-32	-16
n	65	22	33	55	64	67	11	50	53	54	99	51	41	102	79
	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149
m	-8	-21	-26	-33	-4	-31	-34	-25	-43	-6	3	23	12	11	8
n	77	84	91	104	61	122	110	96	124	70	60	12	30	33	40
	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164
m	-2	40	0	-5	7	-39	31	1	-35	-32	-15	16	-43	-75	-8
n	66	-20	46	54	37	116	-27	22	82	85	72	0	102	152	55
	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179
m	-9	-12	-84	-93	25	-104	-40	-51	110	3	17	55	28	-5	29
n	68	54	171	186	12	222	92	93	-169	52	33	-45	1	55	-4

HEVC

Table 9-439-439-43 – Values of variable m and n for coeff_abs_level_greater2_flag ctxIdx

Initialisation variables	coeff_abs_level_greater2_flag ctxIdx														
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
m	-12	-10	-11	-14	-35	-10	-1	-17	-5	-22	-13	-2	-13	-21	-3
n	72	79	87	94	136	58	54	86	70	105	70	59	81	96	73
	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
m	-24	-19	-1	1	-17	-7	-20	-11	-27	-12	-9	-8	-4	-3	-9
n	90	88	63	60	97	69	95	84	110	96	72	76	75	76	94
	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
m	-11	-4	-9	-24	-24	-18	-10	-19	-28	6	-12	-17	-8	-44	-17
n	65	66	82	107	111	58	61	82	97	47	50	73	66	115	86
	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
m	-26	-4	-43	-58	-51	-51	-85	-47	-95	-65	-2	7	-2	3	4
n	74	48	115	141	137	117	176	120	202	159	53	43	67	62	67
	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74
m	-2	-13	-11	-13	-24	-4	-17	-16	-19	-28	-12	-11	-18	-6	-19
n	49	81	82	88	112	44	77	82	89	110	64	70	88	67	97
	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89
m	-17	-27	-18	-13	-17	-11	-15	-10	-10	-12	-8	-7	-6	-8	-9
n	76	100	88	84	94	73	83	77	80	91	63	71	73	80	90
	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104
m	14	-7	-4	-8	-50	2	11	-4	-41	-60	-12	-6	1	-5	-12
n	20	68	66	76	148	22	15	49	117	149	49	55	44	58	73
	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119
m	13	5	-29	0	-23	-12	6	35	42	22	24	11	2	-5	4
n	9	26	101	46	92	57	30	-4	-24	24	0	34	61	75	62
	120	129	130	131	132	133	134	135	136	137	138	139	140	141	142
m	0	-6	-7	-31	-25	-20	-14	-20	-21	-6	-19	-34	-27	-7	-7
n	43	65	71	117	109	76	73	88	92	71	73	108	101	69	75
	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149
m	-18	-7	-20	-9	-7	-26	-13	-6	-12	-2	-3	-6	-7	-6	-11
n	77	64	91	75	78	98	81	69	83	70	50	66	73	75	91
	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164
m	19	2	4	-4	-32	13	-29	-64	-90	26	-24	8	-10	-16	-27
n	10	49	52	69	114	-1	85	143	186	4	68	30	61	78	96
	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179
m	31	-12	-15	-68	31	-8	-32	-36	-53	36	26	19	-22	5	8
n	-16	54	70	158	12	44	63	81	106	12	-5	17	101	54	53

9.3.1.2 Initialisation process for the arithmetic decoding engine

This process is invoked before decoding the first treeblock of a slice.

Outputs of this process are the initialised decoding engine registers codIRange and codIOffset both in 16 bit register precision.

The status of the arithmetic decoding engine is represented by the variables codIRange and codIOffset. In the initialisation procedure of the arithmetic decoding process, codIRange is set equal to 510 and codIOffset is set equal to the value returned from read_bits(9) interpreted as a 9 bit binary representation of an unsigned integer with most significant bit written first.

The bitstream shall not contain data that result in a value of codIOffset being equal to 510 or 511.

NOTE – The description of the arithmetic decoding engine in this Recommendation | International Standard utilizes 16 bit register precision. However, a minimum register precision of 9 bits is required for storing the values of the variables codIRange and

HEVC

codlOffset after invocation of the arithmetic decoding process (DecodeBin) as specified in subclause 9.3.3.2. The arithmetic decoding process for a binary decision (DecodeDecision) as specified in subclause 9.3.3.2.1 and the decoding process for a binary decision before termination (DecodeTerminate) as specified in subclause 9.3.3.2.4 require a minimum register precision of 9 bits for the variables codlRange and codlOffset. The bypass decoding process for binary decisions (DecodeBypass) as specified in subclause 9.3.3.2.3 requires a minimum register precision of 10 bits for the variable codlOffset and a minimum register precision of 9 bits for the variable codlRange.

9.3.2 Binarization process

Input to this process is a request for a syntax element.

Output of this process is the binarization of the syntax element, maxBinIdxCtx, ctxIdxOffset, and bypassFlag.

Table 9-31 specifies the type of binarization process, maxBinIdxCtx, ctxIdxTable, and ctxIdxOffset associated with each syntax element.

The specification of the unary (U) binarization process, the truncated unary (TU) binarization process, the Rice (R) binarization process, the truncated Rice (TR), the concatenated unary / k-th order Exp-Golomb (UEGk) binarization process, the concatenated truncated Rice / k-th order Exp-Golomb (TREGk) binarization process, and the fixed-length (FL) binarization process are given in subclauses 9.3.2.4 to 9.3.2.6, respectively. Other binarizations are specified in subclauses 9.3.2.7 to 9.3.2.10.

The binarizations for the syntax element coeff_abs_level_minus3 as specified in subclause 9.3.2.10 consist of bin strings given by a concatenation of prefix and suffix bit strings. The UEGk binarization as specified in subclause 9.3.2.3, which is used for the binarization of the syntax elements mvd_IX (X = 0, 1 C) also consist of a concatenation of prefix and suffix bit strings. For these binarization processes, the prefix and the suffix bit string are separately indexed using the binIdx variable as specified further in subclause 9.3.3. The two sets of prefix bit strings and suffix bit strings are referred to as the binarization prefix part and the binarization suffix part, respectively.

Associated with each binarization or binarization part of a syntax element is a specific value of the context index table (ctxIdxTable) variable and the related context index offset (ctxIdxOffset) variable and a specific value of the maxBinIdxCtx variable as given in Error! Reference source not found, Table 9-27. When two values for each of these variables are specified for one syntax element in Error! Reference source not found, Table 9-27, the value in the upper row is related to the prefix part while the value in the lower row is related to the suffix part of the binarization of the corresponding syntax element.

The use of the DecodeBypass process and the variable bypassFlag is derived as follows.

- If no value is assigned to ctxIdxOffset for the corresponding binarization or binarization part in Error! Reference source not found, Table 9-27 labelled as "na", all bins of the bit strings of the corresponding binarization or of the binarization prefix/suffix part are decoded by invoking the DecodeBypass process as specified in subclause 9.3.3.2.3. In such a case, bypassFlag is set equal to 1, where bypassFlag is used to indicate that for parsing the value of the bin from the bitstream the DecodeBypass process is applied.
- Otherwise, for each possible value of binIdx up to the specified value of maxBinIdxCtx given in Error! Reference source not found, Table 9-27, a specific value of the variable ctxIdx is further specified in subclause 9.3.3. bypassFlag is set equal to 0.

The possible values of the context index ctxIdx vary depending on the value of ctxIdxTable. The value assigned to ctxIdxOffset specifies the lower value of the range of ctxIdx assigned to the corresponding binarization or binarization part of a syntax element.

ctxIdxTable = 0 and ctxIdx = ctxIdxOffset = 0 are assigned to the syntax element end_of_slice_flag as further specified in subclause 9.3.3.1. For parsing the value of the corresponding bin from the bitstream, the arithmetic decoding process for decisions before termination (DecodeTerminate) as specified in subclause 9.3.3.2.4 is applied.

HEVC

Table 9-449-449-44 – Syntax elements and associated types of binarization, maxBinIdxCtx, ctxIdxTable, and ctxIdxOffset

Syntax element		Type of binarization	maxBinIdxCtx	ctxIdxTable	ctxIdxOffset
alf_cu_flag	I	FL, cMax = 1	0	Table 9-18Table 9-5	0
	P		0	Table 9-18Table 9-5	3
	B		0	Table 9-18Table 9-5	6
end_of_slice_flag	all	FL, cMax = 1	0	0	0
split_coding_unit_flag	I	FL, cMax = 1	0	Table 9-19Table 9-6	0
	P		0	Table 9-19Table 9-6	3
	B		0	Table 9-19Table 9-6	6
skip_flag	P	FL, cMax = 1	0	Table 9-20Table 9-7	0
	B		0	Table 9-20Table 9-7	3
cu_qp_delta	I	as specified in subclause 9.3.2.7	2	Table 9-21Table 9-8	0
	P		2	Table 9-21Table 9-8	4
	B		2	Table 9-21Table 9-8	8
pred_type	I	as specified in subclause 9.3.2.8	0	Table 9-22Table 9-9	0
	P		3	Table 9-22Table 9-9	1
	B		4	Table 9-22Table 9-9	5
pcm_flag	all	FL, cMax = 1	0	0	0
prev_intra_luma_pred_flag	I	FL, cMax = 1	0	Table 9-23Table 9-10	0
	P		0	Table 9-23Table 9-10	1
	B		0	Table 9-23Table 9-10	2
rem_intra_luma_pred_mode	I	as specified in subclause 9.3.2.9	0	Table 9-24Table 9-11	0
	P		0	Table 9-24Table 9-11	1
	B		0	Table 9-24Table 9-11	2
intra_chroma_pred_mode (IntraPredMode < 4)	I	TU, cMax = 3	3	Table 9-25Table 9-12	0
	P		3	Table 9-25Table 9-12	4
	B		3	Table 9-25Table 9-12	8
intra_chroma_pred_mode (4 <= IntraPredMode < 34)	I	TU, cMax = 4	4	Table 9-25Table 9-12	0
	P		4	Table 9-25Table 9-12	4

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

HEVC

Table 9-449-449-44 – Syntax elements and associated types of binarization, maxBinIdxCtx, ctxIdxTable, and ctxIdxOffset

Syntax element		Type of binarization	maxBinIdxCtx	ctxIdxTable	ctxIdxOffset
	B		4	Table 9-25Table 9-12	8
merge_flag	P	FL, cMax = 1	0	Table 9-26Table 9-13	0
	B		0	Table 9-26Table 9-13	3
merge_idx	P	TU, cMax = NumMergeCand	3	Table 9-27Table 9-14	0
	B		3	Table 9-27Table 9-14	4
inter_pred_flag	B	FL, cMax = 1	0	Table 9-28Table 9-15	0
ref_idx_lc	B	TU, cMax = num_ref_idx_IC_active_minus1	1	Table 9-29Table 9-16	6
ref_idx_l0	P	TU, cMax = num_ref_idx_l0_active_minus1	1	Table 9-29Table 9-16	0
	B		1	Table 9-29Table 9-16	6
ref_idx_l1	B	TU, cMax = num_ref_idx_l1_active_minus1	1	Table 9-29Table 9-16	6
mvd_l0[][][0]	P	prefix and suffix as given by UEG3 with signedValFlag=1, uCoff=9	prefix: 4 suffix: na	prefix: Table 9-30Table 9-17 suffix: na	prefix: 0 suffix: na, (uses Decode Bypass)
mvd_l0[][][1]	P		prefix: 4 suffix: na	prefix: Table 9-30Table 9-17 suffix: na	prefix: 7 suffix: na, (uses Decode Bypass)
mvd_l0[][][0], mvd_lc[][][0], mvd_l1[][][0]	B		prefix: 4 suffix: na	prefix: Table 9-30Table 9-17 suffix: na	prefix: 15 suffix: na, (uses Decode Bypass)
mvd_l0[][][1], mvd_lc[][][1], mvd_l1[][][1]	B		prefix: 4 suffix: na	prefix: Table 9-30Table 9-17 suffix: na	prefix: 21 suffix: na, (uses Decode Bypass)
mvp_idx_lc	B	TU, cMax = NumMVPCand(LcToLx)	1	Table 9-31Table 9-18	2
mvp_idx_l0	P	TU, cMax = NumMVPCand(L0)	1	Table 9-31Table 9-18	0
	B		1	Table 9-31Table 9-18	2
mvp_idx_l1	B	TU, cMax = NumMVPCand(L1)	1	Table 9-31Table 9-18	2
no_residual_data_flag	P	FL, cMax = 1	0	Table 9-32Table 9-19	0
	B		0	Table 9-32Table 9-19	4
split_transform_flag	I	FL, cMax = 1	0	Table 9-33Table 9-20	0

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

HEVC

Table 9-449-449-44 – Syntax elements and associated types of binarization, maxBinIdxCtx, ctxIdxTable, and ctxIdxOffset

Syntax element		Type of binarization	maxBinIdxCtx	ctxIdxTable	ctxIdxOffset
	P		0	Table 9-33Table 9-20	4
	B		0	Table 9-33Table 9-20	8
cbf_luma	I	FL, cMax = 1	0	Table 9-34Table 9-21	0
	P		0	Table 9-34Table 9-21	4
	B		0	Table 9-34Table 9-21	8
cbf_cb	I	FL, cMax = 1	0	Table 9-35Table 9-22	0
	P		0	Table 9-35Table 9-22	4
	B		0	Table 9-35Table 9-22	8
cbf_cr	I	FL, cMax = 1	0	Table 9-36Table 9-23	0
	P		0	Table 9-36Table 9-23	4
	B		0	Table 9-36Table 9-23	8
last_significant_coeff_x	I	TU, cMax = (1 << log2TrafoSize) - 1	30	Table 9-37Table 9-24	0
	P		30	Table 9-37Table 9-24	41
	B		30	Table 9-37Table 9-24	82

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

HEVC

Table 9-44 – Syntax elements and associated types of binarization, maxBinIdxCtx, ctxIdxTable, and ctxIdxOffset

Syntax element		Type of binarization	maxBinIdxCtx	ctxIdxTable	ctxIdxOffset																																																																		
last_significant_coeff_y	I	TU, cMax = (1 << log2TrafoSize) - 1	30	Table 9-38 9-25	0																																																																		
	P		30	Table 9-38 9-25	41																																																																		
	B		30	Table 9-38 9-25	82																																																																		
significant_coeff_flag	I	FL, cMax = 1	0	Table 9-39 9-26	0																																																																		
	P		0	<table border="1"> <thead> <tr> <th colspan="3">significant coeff</th> </tr> </thead> <tbody> <tr><td>6</td><td>7</td><td>8</td></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>64</td><td>64</td><td>64</td></tr> <tr><td>22</td><td>23</td><td>24</td></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>64</td><td>64</td><td>64</td></tr> <tr><td>38</td><td>39</td><td>40</td></tr> <tr><td>-32</td><td>-19</td><td>-18</td></tr> <tr><td>125</td><td>72</td><td>6</td></tr> <tr><td>54</td><td>55</td><td>5</td></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>64</td><td>64</td><td>64</td></tr> <tr><td>70</td><td>71</td><td>7</td></tr> <tr><td>-8</td><td>1</td><td>2</td></tr> <tr><td>80</td><td>36</td><td>5</td></tr> <tr><td>86</td><td>87</td><td>8</td></tr> <tr><td>-11</td><td>-29</td><td>-4</td></tr> <tr><td>81</td><td>92</td><td>4</td></tr> <tr><td>102</td><td>103</td><td>10</td></tr> <tr><td>-18</td><td>-10</td><td>0</td></tr> <tr><td>99</td><td>59</td><td>5</td></tr> </tbody> </table>	significant coeff			6	7	8	0	0	0	64	64	64	22	23	24	0	0	0	64	64	64	38	39	40	-32	-19	-18	125	72	6	54	55	5	0	0	0	64	64	64	70	71	7	-8	1	2	80	36	5	86	87	8	-11	-29	-4	81	92	4	102	103	10	-18	-10	0	99	59	5	0
	significant coeff																																																																						
6	7	8																																																																					
0	0	0																																																																					
64	64	64																																																																					
22	23	24																																																																					
0	0	0																																																																					
64	64	64																																																																					
38	39	40																																																																					
-32	-19	-18																																																																					
125	72	6																																																																					
54	55	5																																																																					
0	0	0																																																																					
64	64	64																																																																					
70	71	7																																																																					
-8	1	2																																																																					
80	36	5																																																																					
86	87	8																																																																					
-11	-29	-4																																																																					
81	92	4																																																																					
102	103	10																																																																					
-18	-10	0																																																																					
99	59	5																																																																					
B	0	Table 9-41 9-28	0																																																																				
coeff_abs_level_greater1_flag	I	FL, cMax = 1	0	Table 9-42 9-29	0																																																																		
	P		0	Table 9-42 9-29	60																																																																		

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: Times New Roman, 8 pt, English (United Kingdom)

Formatted: Table_Text, Left, Space Before: 0 pt, After: 0 pt, Line spacing: single, Don't keep with next, Don't keep lines together

Formatted: Font: 8 pt

Formatted: Table_Text, Centered, Keep with next

Formatted: Font: 8 pt

Formatted: Table_Text, Centered, Keep with next

HEVC

Table 9-449-449-44 – Syntax elements and associated types of binarization, maxBinIdxCtx, ctxIdxTable, and ctxIdxOffset

Syntax element		Type of binarization	maxBinIdxCtx	ctxIdxTable	ctxIdxOffset
	B		0	120	
coeff_abs_level_greater2_flag	I	FL, cMax = 1	0	0	
	P		0	60	
	B		0	120	
coeff_abs_level_minus3	all	prefix and suffix as specified in subclause 9.3.2.10	prefix: na suffix: na	prefix: na suffix: na	prefix: na, (uses Decode Bypass) suffix: na, (uses Decode Bypass)
coeff_sign_flag	all	FL, cMax = 1	na	na	na, (uses Decode Bypass)

Formatted: Font: 8 pt
 Formatted: Table_Text, Centered, Keep with next
 Formatted: Font: 8 pt
 Formatted: Font: 8 pt
 Formatted: Font: 8 pt

[Ed. (BB): The TU binarization for last_significant_coeff_x and last_significant_coeff_y is using 0s and a terminating 1 instead of using 1s and a terminating 0.]

9.3.2.1 Unary (U) binarization process

Input to this process is a request for a U binarization for a syntax element.

Output of this process is the U binarization of the syntax element.

The bin string of a syntax element having (unsigned integer) value synElVal is a bit string of length synElVal + 1 indexed by binIdx. The bins for binIdx less than synElVal are equal to 1. The bin with binIdx equal to synElVal is equal to 0.

Table 9-45Table 9-35 illustrates the bin strings of the unary binarization for a syntax element.

Table 9-459-459-45 – Bin string of the unary binarization (informative)

Value of syntax element	Bin string					
0 (L_NxN)	0					
1	1	0				
2	1	1	0			
3	1	1	1	0		
4	1	1	1	1	0	
5	1	1	1	1	1	0
...						
binIdx	0	1	2	3	4	5

9.3.2.2 Truncated unary (TU) binarization process

Input to this process is a request for a TU binarization for a syntax element and cMax.

Output of this process is the TU binarization of the syntax element.

HEVC

For syntax element (unsigned integer) values less than cMax, the U binarization process as specified in subclause 9.3.2.1 is invoked. For the syntax element value equal to cMax the bin string is a bit string of length cMax with all bins being equal to 1.

NOTE – TU binarization is always invoked with a cMax value equal to the largest possible value of the syntax element being decoded.

9.3.2.3 Truncated Rice (TR) binarization process

Input to this process is a request for a TR binarization for a syntax element, cRiceParam and cTRMax.

Output of this process is the TR binarization of the syntax element.

A TR bin string is a concatenation of a prefix bit string and a suffix bit string. The prefix of the binarization is specified by invoking the TU binarization process for the prefix part of the value specified by $\text{synElVal} \gg \text{cRiceParam}$ with $\text{cMax} = \text{cTRMax} \gg \text{cRiceParam}$. The suffix of the TR bin string is the binary representation of $\text{synElVal} - ((\text{synElVal} \gg \text{cRiceParam}) \ll \text{cRiceParam})$.

NOTE – For the input parameter $\text{cRiceParam} = 0$ the TR binarization is exactly the TU binarization.

9.3.2.4 k-th order Exp-Golomb (EGk) binarization process

Input to this process is a request for an EGk binarization for a syntax element and signedFlag.

Output of this process is the EGk binarization of the syntax element.

The bin string of the EGk binarization process of a syntax element synVal is specified by a process equivalent to the following pseudo-code:

```

absV = Abs( synVal )
stopLoop = 0
do {
    if( absV >= ( 1 << k ) ) {
        put( 1 )
        absV = absV - ( 1 << k )
        k++
    } else {
        put( 0 )
        while( k-- )
            put( ( absV >> k ) & 1 )
        stopLoop = 1
    }
} while( !stopLoop )
if( signedFlag && synVal != 0 ) {
    if( synVal > 0 )
        put( 0 )
    else
        put( 1 )
}

```

NOTE 1 – The specification for the k-th order Exp-Golomb (EGk) code uses 1's and 0's in reverse meaning for the unary part of the Exp-Golomb code of 0-th order as specified in subclause 9.1.

9.3.2.5 Concatenated unary/ k-th order Exp-Golomb (UEGk) binarization process

Input to this process is a request for a UEGk binarization for a syntax element, signedValFlag and uCoff.

Output of this process is the UEGk binarization of the syntax element.

A UEGk bin string is a concatenation of a prefix bit string and a suffix bit string. The prefix of the binarization is specified by invoking the TU binarization process for the prefix part $\text{Min}(\text{uCoff}, \text{Abs}(\text{synElVal}))$ of a syntax element value synElVal as specified in subclause 9.3.2.2 with $\text{cMax} = \text{uCoff}$, where $\text{uCoff} > 0$.

The variable k for a UEGk bin string is dependent on the syntax element for which a UEGk binarization is requested. [Error! Reference source not found. Table 9-34](#) specifies the associated types of binarization for syntax elements, including the value of k for syntax elements that use UEGk binarization.

NOTE 1 – For the syntax elements $\text{mvd_lc}[][]$, $\text{mvd_l0}[][]$ and $\text{mvd_l1}[][]$ a UEG3 binarization is used (k is equal to 3).

The UEGk bin string is derived as follows.

HEVC

- If one of the following is true, the bin string of a syntax element having value `synEIVal` consists only of a prefix bit string:
 - `signedValFlag` is equal to 0 and the prefix bit string is not equal to the bit string of length `uCoff` with all bits equal to 1,
 - `signedValFlag` is equal to 1 and the prefix bit string is equal to the bit string that consists of a single bit with value equal to 0.
- Otherwise, the bin string of the UEGk suffix part of a syntax element value `synEIVal` is specified by invoking the EGk binarization process as specified in subclause 9.3.2.4 with `k` being initialised to the value that is specified in [Table 9-4](#) for the requested UEGk binarization process.

Formatted: Highlight

9.3.2.6 Fixed-length (FL) binarization process

Input to this process is a request for a FL binarization for a syntax element and `cMax`.

Output of this process is the FL binarization of the syntax element.

FL binarization is constructed by using a `fixedLength`-bit unsigned integer bin string of the syntax element value, where $\text{fixedLength} = \text{Ceil}(\text{Log}_2(\text{cMax} + 1))$. The indexing of bins for the FL binarization is such that the `binIdx = 0` relates to the least significant bit with increasing values of `binIdx` towards the most significant bit.

9.3.2.7 Binarization process for `cu_qp_delta`

Input to this process is a request for a binarization for the syntax element `mb_qp_delta`.

Output of this process is the binarization of the syntax element.

For syntax elements `mb_qp_delta`, the U binarization process as specified in subclause [9.3.2.4](#) is invoked with the mapped value of the syntax element `mb_qp_delta` as input and the bin string as output. The assignment rule between the signed value of `mb_qp_delta` and its mapped value is given as specified in [Table 9-3](#).

9.3.2.8 Binarization process for `pred_type`

Input to this process is a request for a binarization for the syntax element `pred_type` and a variable `cLog2CUSize` specifying the current CU size.

Output of this process is the binarization of the syntax element.

The binarization for `pred_type` is given by [Table 9-4](#) depending on slice type and the size of the coding unit.

HEVC

Table 9-469-469-46 – Binarization for pred_type

Slice type	Value of pred_type	PredMode	PartMode	Bin string	
				cLog2CUSize > 3	cLog2CUSize = = 3e
I	0	MODE_INTRA	PART_2Nx2N	-	1
	1	MODE_INTRA	PART_NxN	-	0
P	0	MODE_INTER	PART_2Nx2N	01	01
	1	MODE_INTER	PART_2NxN	001	001
	2	MODE_INTER	PART_Nx2N	000	0001
	3	MODE_INTER	PART_NxN	-	0000
	4	MODE_INTRA	PART_2Nx2N	1	11
	5	MODE_INTRA	PART_NxN	-	10
B	0	MODE_INTER	PART_2Nx2N	1	1
	1	MODE_INTER	PART_2NxN	01	01
	2	MODE_INTER	PART_Nx2N	001	001
	3	MODE_INTER	PART_NxN	-	0001
	4	MODE_INTRA	PART_2Nx2N	000	0000 1
	5	MODE_INTRA	PART_NxN	-	0000 0

9.3.2.9 Binarization process for rem_intra_luma_pred_mode

Input to this process is a request for the syntax element rem_intra_luma_pred_mode and cNumBins.

Output of this process is the binarization of the syntax element.

The binarization for rem_intra_luma_pred_mode is given by [Table 9-47](#)/[Table 9-35](#).

Table 9-479-479-47 – Binarization for rem_intra_luma_pred_mode

Value of rem_intra_luma_pred_mode	Bin string
less than 32	FL, cMax = cNumBins
32	111110
33	111111

9.3.2.10 Binarization process for coeff_abs_level_minus3

Input to this process is a request for a binarization for the syntax element coeff_abs_level_minus3[n].

Output of this process is the binarization of the syntax element.

The variables cLastSE and cLastRiceParam are derived as follows.

- If n is equal to 0 or all previous syntax elements coeff_abs_level_minus3[pos] with pos less than n are derived to be equal to 0 instead of being explicitly parsed, cLastSE and cLastRiceParam are set equal to 0.
- Otherwise (n is not equal to 0), cLastSE is set equal to coeff_abs_level_minus3[n - 1] and cLastRiceParam is set equal to the value of cRiceParam that has been derived during the invocation of this subclause for the syntax element coeff_abs_level_minus2[n - 1] of the same transform block.

The variable cRiceParam is derived as follows.

- If cLastSE is equal to 0 or 1 and cLastRiceParam is equal to 0, cRiceParam is set equal to 0.

HEVC

- Otherwise if cLastSE is equal to 2 or 3 and cLastRiceParam is equal to 0 or 1, cRiceParam is set equal to 1.
- Otherwise if one of the following conditions is true, cRiceParam is set equal to 2.
 - cLastSE is in the range of 4 to 12 and cLastRiceParam is equal to 0.
 - cLastSE is in the range of 4 to 10 and cLastRiceParam is equal to 1.
 - cLastSE is in the range of 4 to 9 and cLastRiceParam is equal to 2.
- Otherwise, cRiceParam is set equal to 3.

The variable cTRMax is derived from cRiceParam as given in [Table 9-48](#) [Table 9-36](#).

Table 9-48 ~~489-489~~ ~~48~~ – Specification of cTRMax depending on cRiceParam

cRiceParam	cTRMax
0	7
1	20
2	42
3	70

The binarization of coeff_abs_level_minus3 consists of a prefix part and (when present) a suffix part. The prefix part of the binarization is derived by invoking the TR binarization process as specified in subclause 9.3.2.3 for the syntax element coeff_abs_level_minus3[n] with the variables cRiceParam and cTRMax as the inputs.

When the prefix bit string is not equal to the bin string that consists of (cTRMax + 1 >> cRiceParam) + cRiceParam ones, the bin string consists of a prefix bin string and a suffix bin string. The suffix bin string is derived using the EGO binarization as specified in subclause 9.3.2.4.

9.3.3 Decoding process flow

Input to this process is a binarization of the requested syntax element, maxBinIdxCtx, bypassFlag, ctxIdxTable and ctxIdxOffset as specified in subclause 9.3.2.

Output of this process is the value of the syntax element.

This process specifies how each bit of a bit string is parsed for each syntax element.

After parsing each bit, the resulting bit string is compared to all bin strings of the binarization of the syntax element and the following applies.

- If the bit string is equal to one of the bin strings, the corresponding value of the syntax element is the output.
- Otherwise (the bit string is not equal to one of the bin strings), the next bit is parsed.

While parsing each bin, the variable binIdx is incremented by 1 starting with binIdx being set equal to 0 for the first bin.

When the binarization of the corresponding syntax element consists of a prefix and a suffix binarization part, the variable binIdx is set equal to 0 for the first bin of each part of the bin string (prefix part or suffix part). In this case, after parsing the prefix bit string, the parsing process of the suffix bit string related to the binarizations specified in subclauses 9.3.2.5 and 9.3.2.10 is invoked depending on the resulting prefix bit string as specified in subclauses 9.3.2.5 and 9.3.2.10.

Depending on the variable bypassFlag, the following applies.

- If bypassFlag is equal to 1, the bypass decoding process as specified in subclause 9.3.3.2.3 is applied for parsing the value of the bins from the bitstream.
- Otherwise (bypassFlag is equal to 0), the parsing of each bin is specified by the following two ordered steps:
 1. Given binIdx, maxBinIdxCtx, ctxIdxTable, and ctxIdxOffset, ctxIdx is derived as specified in subclause 9.3.3.1.
 2. Given ctxIdxTable and ctxIdx, the value of the bin from the bitstream as specified in subclause 9.3.3.2 is decoded.

9.3.3.1 Derivation process for ctxIdx

Inputs to this process are binIdx, maxBinIdxCtx, ctxIdxTable, and ctxIdxOffset.

Output of this process is ctxIdx.

HEVC

[Table 9-49](#)[Table 9-37](#) shows the assignment of ctxIdx increments (ctxIdxInc) to binIdx for all ctxIdxTable and ctxIdxOffset.

The ctxIdx to be used with a specific binIdx is specified by first determining the ctxIdxTable and ctxIdxOffset associated with the given bin string or part thereof. The ctxIdxOffset is listed in [Table 9-49](#)[Table 9-37](#), the ctxIdx for a binIdx is the sum of ctxIdxOffset and ctxIdxInc, which is found in [Table 9-49](#)[Table 9-37](#). When more than one value is listed in [Table 9-49](#)[Table 9-37](#) for a binIdx, the assignment process for ctxIdxInc for that binIdx is further specified in the subclauses given in parenthesis of the corresponding table entry.

All bins with binIdx greater than maxBinIdxCtx are parsed using the value of ctxIdx being assigned to binIdx equal to maxBinIdxCtx.

All entries in [Table 9-49](#)[Table 9-37](#) labelled with "na" correspond to values of binIdx that do not occur for the corresponding ctxIdxOffset.

HEVC

Table 9-499-49 – Assignment of ctxIdxInc to binIdx for all ctxIdxTable and ctxIdxOffset values

Syntax element	ctxIdxTable, ctxIdxOffset	binIdx					
		0	1	2	3	>=4	
alf_cu_flag	Table 9-18 9-5	0	0,1,2 (subclause 9.3.3.1.1.1)	na	na	na	na
		3	0,1,2 (subclause 9.3.3.1.1.1)	na	na	na	na
		6	0,1,2 (subclause 9.3.3.1.1.1)	na	na	na	na
split_coding_unit_flag	Table 9-19 9-6	0	0,1,2 (subclause 9.3.3.1.1.1)	na	na	na	na
		3	0,1,2 (subclause 9.3.3.1.1.1)	na	na	na	na
		6	0,1,2 (subclause 9.3.3.1.1.1)	na	na	na	na
skip_flag	Table 9-20 9-7	0	0,1,2 (subclause 9.3.3.1.1.1)	na	na	na	na
		3	0,1,2 (subclause 9.3.3.1.1.1)	na	na	na	na
cu_qp_delta	Table 9-21 9-8	0	0	2	3	3	3
		4	0	2	3	3	3
		8	0	2	3	3	3
pred_type	Table 9-22 9-9	0	0	na	na	na	na
		1	0	1	2	3	na
		5	0	1	2	3	4
prev_intra_luma_pred_flag	Table 9-23 9-10	0	0	na	na	na	na
		1	0	na	na	na	na
		2	0	na	na	na	na
rem_intra_luma_pred_mode	Table 9-24 9-11	0	0	0	0	0	0
		1	0	0	0	0	0
		2	0	0	0	0	0
intra_chroma_pred_mode	Table 9-25 9-12	0	0,1,2 (subclause 9.3.3.1.1.1)	3	3	3	na
		4	0,1,2 (subclause 9.3.3.1.1.1)	3	3	3	na
		8	0,1,2 (subclause 9.3.3.1.1.1)	3	3	3	na
merge_flag	Table 9-26 9-13	0	0,1,2 (subclause 9.3.3.1.1.1)	na	na	na	na
		3	0,1,2 (subclause 9.3.3.1.1.1)	na	na	na	na
merge_idx	Table 9-27 9-14	0	0,1,2,3 (subclause 9.3.3.1.1.2)	0,1,2,3 (subclause 9.3.3.1.1.2)	0,2,3 (subclause 9.3.3.1.1.2)	3	na
		4	0,1,2,3 (subclause 9.3.3.1.1.2)	0,1,2,3 (subclause 9.3.3.1.1.2)	0,2,3 (subclause 9.3.3.1.1.2)	3	na
inter_pred_flag	Table 9-28 9-15	0	0,1,2 (subclause 9.3.3.1.1.1)	na	na	na	na

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

HEVC

Table 9-499-49 – Assignment of ctxIdxInc to binIdx for all ctxIdxTable and ctxIdxOffset values

Syntax element	ctxIdxTable, ctxIdxOffset	binIdx					
		0	1	2	3	>=4	
ref_idx_10	Table 9-29 Table 9-16	0	0,1,2,3 (subclause 9.3.3.1.1.1)	4	4	4	4
ref_idx_10, ref_idx_11, ref_idx_lc	Table 9-29 Table 9-16	6	0,1,2,3 (subclause 9.3.3.1.1.1)	4	4	4	4
mvd_10[][][0]	Table 9-30 Table 9-17	0	0,1,2 (subclause 9.3.3.1.1.1)	3	4	5	6
mvd_10[][][1]	Table 9-30 Table 9-17	7	0,1,2 (subclause 9.3.3.1.1.1)	3	4	5	6
mvd_10[][][0], mvd_lc[][][0], mvd_11[][][0]	Table 9-30 Table 9-17	15	0,1,2 (subclause 9.3.3.1.1.1)	3	4	5	6
mvd_10[][][1], mvd_lc[][][1], mvd_11[][][1]	Table 9-30 Table 9-17	21	0,1,2 (subclause 9.3.3.1.1.1)	3	4	5	6
mvp_idx_10	Table 9-31 Table 9-18	0	0	1	1	1	1
mvp_idx_10, mvp_idx_11, mvp_idx_lc	Table 9-31 Table 9-18	2	0	1	1	1	1
no_residual_data_flag	Table 9-32 Table 9-19	0	0,1,2,3 (subclause 9.3.3.1.1.1)	na	na	na	na
		4	0,1,2,3 (subclause 9.3.3.1.1.1)	na	na	na	na
split_transform_flag	Table 9-33 Table 9-20	0	cuDepth + trafoDepth	na	na	na	na
		4	cuDepth + trafoDepth	na	na	na	na
		8	cuDepth + trafoDepth	na	na	na	na
cbf_luma	Table 9-34 Table 9-21	0	0,1,2,3 (subclause 9.3.3.1.1.1)	na	na	na	na
		4	0,1,2,3 (subclause 9.3.3.1.1.1)	na	na	na	na
		8	0,1,2,3 (subclause 9.3.3.1.1.1)	na	na	na	na
cbf_cb (PredMode == MODE_INTRA)	Table 9-35 Table 9-22	0	0,1,2,3 (subclause 9.3.3.1.1.1)	na	na	na	na
		4	0,1,2,3 (subclause 9.3.3.1.1.1)	na	na	na	na
		8	0,1,2,3 (subclause 9.3.3.1.1.1)	na	na	na	na
cbf_cb (PredMode != MODE_INTRA)	Table 9-35 Table 9-22	0	trafoDepth	na	na	na	na
		4	trafoDepth	na	na	na	na
		8	trafoDepth	na	na	na	na
cbf_cr (PredMode == MODE_INTRA)	Table 9-36 Table 9-23	0	0,1,2,3 (subclause 9.3.3.1.1.1)	na	na	na	na
		4	0,1,2,3 (subclause 9.3.3.1.1.1)	na	na	na	na
		8	0,1,2,3 (subclause 9.3.3.1.1.1)	na	na	na	na

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

HEVC

Table 9-49 – Assignment of ctxIdxInc to binIdx for all ctxIdxTable and ctxIdxOffset values

Syntax element	ctxIdxTable, ctxIdxOffset	binIdx				
		0	1	2	3	>=4
cbf_cr (PredMode != MODE_INTRA)	0	trafoDepth	na	na	na	na
	4	trafoDepth	na	na	na	na
	8	trafoDepth	na	na	na	na

Formatted: Font: 8 pt

HEVC

Table 9-499-499-49 – Assignment of ctxIdxInc to binIdx for all ctxIdxTable and ctxIdxOffset values

Syntax element	ctxIdxTable, ctxIdxOffset	binIdx					
		0	1	2	3	>=4	
last_significant_coeff_x	Table 9-37 Table 9-24	0	0.40 (subclause 9.3.3.1.1.3)				
		41	0.40 (subclause 9.3.3.1.1.3)				
		82	0.40 (subclause 9.3.3.1.1.3)				
last_significant_coeff_y	Table 9-38 Table 9-25	0	0.40 (subclause 9.3.3.1.1.3)				
		41	0.40 (subclause 9.3.3.1.1.3)				
		82	0.40 (subclause 9.3.3.1.1.3)				
significant_coeff_flag	Table 9-39 Table 9-26	0	0.111 (subclause 9.3.3.1.1.4)	na	na	na	na
		0	0.111 (subclause 9.3.3.1.1.4)	na	na	na	na
		6	7				
		0	0				
		64	64				
		22	23				
		0	0				
		64	64				
		38	39				
		-32	-19				
		125	72				
		54	55				
		0	0				
		64	64				
		70	71				
		-8	1				
		80	36				
		86	87				
		-11	-29				
		81	92				
		102	103				
		-18	-10				
		99	59				
	Table 9-40 Table 9-27						
	Table 9-41 Table 9-28	0	0.111 (subclause 9.3.3.1.1.4)	na	na	na	na
coeff_abs_level_greater1_flag	Table 9-42 Table 9-29	0	0.59 (subclause 9.3.3.1.1.5)	na	na	na	na
		60	0.59 (subclause 9.3.3.1.1.5)	na	na	na	na

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: Times New Roman, 8 pt, English (United Kingdom)

Formatted: Font: 8 pt

Formatted: Font: 8 pt

HEVC

Table 9-499-49 – Assignment of ctxIdxInc to binIdx for all ctxIdxTable and ctxIdxOffset values

Syntax element	ctxIdxTable, ctxIdxOffset	binIdx				
		0	1	2	3	>=4
coeff_abs_level_greater2_flag	120 (subclause 9.3.3.1.1.5)	0.59	na	na	na	na
	0 (subclause 9.3.3.1.1.6)	0.59	na	na	na	na
	60 (subclause 9.3.3.1.1.6)	0.59	na	na	na	na
	120 (subclause 9.3.3.1.1.6)	0.59	na	na	na	na

Formatted: Font: 8 pt

Formatted: Heading 5, Indent: Left: 0", Tab stops: Not at 0.59" + 0.79" + 0.98"

9.3.3.1.1 Assignment process of ctxIdxInc using neighbouring syntax elements

Subclause 9.3.3.1.1.1 specifies the derivation process of ctxIdxInc for the syntax elements alf_cu_flag, split_coding_unit_flag, skip_flag, merge_flag, intra_chroma_pred_mode, inter_pred_flag, ref_idx_lc, ref_idx_l0, ref_idx_l1, mvd_l0, mvd_l1, mvd_lc, no_residual_data_flag, cbf_luma, cbf_cb and cbf_cr.

9.3.3.1.1.1 Derivation process of ctxIdxInc using left and above syntax elements

Input to this process is the luma location (xP, yP) specifying the top-left luma sample of the current prediction unit relative to the top-left sample of the current picture.

Output of this process is ctxIdxInc.

Let the luma location (xL, yL) specify a location covered by the prediction unit to the left of the top-left luma sample of the current prediction unit with xL = xP – MinPuSize and yL = yP, the variable availableL is derived as follows. [Ed.: (BB) MinPuSize should be defined somewhere]

- If the prediction unit covering location (xL, yL) is available, availableL is set equal to 1.
- Otherwise (the prediction unit covering location (xL, yL) is not available), availableL is set equal to 0.

Let the luma location (xA, yA) specify a location covered by the prediction unit above the top-left luma sample of the current prediction unit with xA = xP and yA = yP – MinPuSize, the variable availableA is derived as follows. [Ed.: (BB) MinPuSize should be defined somewhere]

- If the prediction unit covering location (xA, yA) is available, availableA is set equal to 1.
- Otherwise (the prediction unit covering location (xA, yA) is not available), availableA is set equal to 0

The assignment of ctxIdxInc for the syntax elements alf_cu_flag, split_coding_unit_flag, skip_flag, merge_flag, intra_chroma_pred_mode, inter_pred flag, ref_idx_lc, ref_idx_l0, ref_idx_l1, mvd_lc, mvd_l0, mvd_l1, no_residual_data_flag, cbf_luma, cbf_cb and cbf_cr is specified in [Table 9-50 – Specification of ctxIdxInc using left and above syntax elementsTable 9-38](#).

Table 9-509-509-50 – Specification of ctxIdxInc using left and above syntax elements

Syntax element	condL	condA	ctxIdxInc
alf_cu_flag	alf_cu_flag[xL][yL]	alf_cu_flag[xA][yA]	(condL && availableL) + (condA && availableA)
split_coding_unit_flag	cuDepth[xL][yL] > cuDepth[xP][yP]	cuDepth[xA][yA] > cuDepth[xP][yP]	(condL && availableL) + (condA && availableA)
skip_flag	skip_flag[xL][yL]	skip_flag[xA][yA]	(condL && availableL) + (condA && availableA)
merge_flag	merge_flag[xL][yL]	merge_flag[xA][yA]	(condL && availableL) + (condA && availableA)
intra_chroma_pred_mode	IntraPredMode[xL][yL] == 4	IntraPredMode [xA][yA] == 4	(condL && availableL) + (condA && availableA)
inter_pred_flag	inter_pred_flag[xL][yL]	inter_pred_flag[xA][yA]	(condL && availableL) + (condA && availableA)
ref_idx_lc	ref_idx_lc[xL][yL] > 0	ref_idx_lc[xA][yA] > 0	(condL && availableL) + (condA && availableA) << 1
ref_idx_l0	ref_idx_l0[xL][yL] > 0	ref_idx_l0[xA][yA] > 0	(condL && availableL) + (condA && availableA) << 1
ref_idx_l1	ref_idx_l1[xL][yL] > 0	ref_idx_l1[xA][yA] > 0	(condL && availableL) + (condA && availableA) << 1
mvd_lc	mvd_lc[xL][yL] > 16	mvd_lc[xA][yA] > 16	(condL && availableL) + (condA && availableA)
mvd_l0	mvd_l0[xL][yL] > 16	mvd_l0[xA][yA] > 16	(condL && availableL) + (condA && availableA)
mvd_l1	mvd_l1[xL][yL] > 16	mvd_l1[xA][yA] > 16	(condL && availableL) + (condA && availableA)
no_residual_data_flag	no_residual_data_flag[xL][yL]	no_residual_data_flag[xA][yA]	(condL && availableL) + (condA && availableA) << 1
cbf_luma	cbf_luma[xL][yL]	cbf_luma[xA][yA]	(condL && availableL) + (condA && availableA) << 1
cbf_cb	cbf_cb[xL][yL]	cbf_cb[xA][yA]	(condL && availableL) + (condA && availableA) << 1
cbf_cr	cbf_cr[xL][yL]	cbf_cr[xA][yA]	(condL && availableL) + (condA && availableA) << 1

9.3.3.1.1.2 Derivation process of ctxIdxInc for the syntax element merge_index

Input to this process is binIdx.

Output of this process is ctxIdxInc.

The variable ctxIdxInc is derived as follows.

- If binIdx is equal to 0, the following applies.
 - If availableLeft is equal to 1 and availableAbove is equal to 1, ctxIdxInc is set equal to 0.
 - Otherwise if availableLeft is equal to 1 or availableAbove is equal to 1, ctxIdxInc is set equal to 2.
 - Otherwise (availableLeft is equal to 0 and availableAbove is equal to 0), ctxIdxInc is set equal to 3.
- Otherwise if binIdx is equal to 1 and NumMergeCand is greater than 2, the following applies.
 - If availableAbove is equal to 1, ctxIdxInc is set equal to 2.
 - Otherwise (availableAbove is equal to 0), ctxIdxInc is set equal to 3.
- Otherwise if binIdx is equal to 2 and NumMergeCand is greater than 3, ctxIdxInc is set equal to 3.

HEVC

- Otherwise, $ctxIdxInc$ is set equal to 0.

When $ctxIdxInc$ is not equal to 0, $ctxIdxInc$ is derived as follows.

$$ctxIdxInc = ctxIdxInc - availableCollocated \quad (9-51516)$$

[Ed. (BB): The flags $availableLeft$, $availableAbove$, $availableCollocated$ need to be defined or a simpler context derivation should be used.]

9.3.3.1.1.3 Derivation process of $ctxIdxInc$ for the syntax elements $last_significant_coeff_x$ and $last_significant_coeff_y$

Inputs to this process are the $binIdx$, the color component index $cIdx$ and the transform block size $log2TrafoSize$.

Output of this process is $ctxIdxInc$.

Table 9-519-519-51 – Specification of $lastCtx$ depending on $binIdx$

$binIdx$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$lastCtx$	0	1	2	3	3	4	4	5	5	5	5	6	6	6	6	7
$binIdx$	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	
$lastCtx$	7	7	7	8	8	8	8	9	9	9	9	10	10	10	10	

The variable $lastCtx$ is derived from $binIdx$ as given by Table 9-51Table-9-39. For the derivation of $ctxIdxInc$, the following applies.

- If $log2TrafoSize$ is less than or equal to 2, $ctxIdxInc$ is derived as follows.

$$ctxIdxInc = lastCtx \quad (9-52527)$$

- Otherwise if $log2TrafoSize$ is equal to 3, $ctxIdxInc$ is derived as follows.

$$ctxIdxInc = lastCtx + 3 \quad (9-53538)$$

- Otherwise if $log2TrafoSize$ is equal to 4, $ctxIdxInc$ is derived as follows.

$$ctxIdxInc = lastCtx + 8 \quad (9-54549)$$

- Otherwise ($log2TrafoSize$ is greater than 4), $ctxIdxInc$ is derived as follows.

$$ctxIdxInc = lastCtx + 15 \quad (9-555510)$$

When $cIdx$ is greater than 0, $ctxIdxInc$ is modified as follows.

$$ctxIdxInc = ctxIdxInc + 26 \quad (9-565611)$$

[Ed. (BB): The context derivation assumes maximum transform sizes less than or equal to 32x32 for luma and 16x16 for chroma and minimum transform sizes greater than or equal to 4x4.]

9.3.3.1.1.4 Derivation process of $ctxIdxInc$ for the syntax element $significant_coeff_flag$

Inputs to this process are the color component index $cIdx$, the current coefficient scan position (xC , yC) and the transform block size $log2TrafoSize$.

Output of this process is $ctxIdxInc$.

The variable $sigCtx$ depends on the current position (xC , yC), the transform block size and previously decoded bins of the syntax element $significant_coeff_flag$. For the derivation of $sigCtx$, the following applies.

- If $log2TrafoSize$ is less than or equal to 3, $sigCtx$ is derived as follows.

$$\begin{aligned} shift &= log2TrafoSize == 3 ? 1 : 0 \\ sigCtx &= ((yC \gg shift) \ll 2) + (xC \gg shift) \end{aligned} \quad (9-575712)$$

- Otherwise if xC is less than 2 and yC is less than 2, $sigCtx$ is derived as follows.

$$sigCtx = (yC \ll 1) + xC \quad (9-585813)$$

HEVC

- Otherwise if yC is equal to 0, $sigCtx$ is derived as follows.

$$sigCtx = 4 + significant_coeff_flag[xC-1][yC] + significant_coeff_flag[xC-2][yC] \quad (9-595914)$$

- Otherwise if xC is equal to 0, $sigCtx$ is derived as follows.

$$sigCtx = 4 + significant_coeff_flag[xC][yC-1] + significant_coeff_flag[xC][yC-2] \quad (9-606015)$$

- Otherwise (xC is greater than 1 and yC is greater than 1), the following applies.

- The variable $sigCtx$ is initialized using previously decoded bins of the syntax element $significant_coeff_flag$ as follows.

$$sigCtx = significant_coeff_flag[xC-1][yC] + significant_coeff_flag[xC][yC-1] + significant_coeff_flag[xC-1][yC-1] \quad (9-616116)$$

- When xC is greater than 1, the following applies.

$$sigCtx = sigCtx + significant_coeff_flag[xC-2][yC] \quad (9-626217)$$

- When yC is greater than 1, the following applies.

$$sigCtx = sigCtx + significant_coeff_flag[xC][yC-2] \quad (9-636318)$$

- The variable $sigCtx$ is derived as follows.

$$sigCtx = 10 + \min(4, sigCtx) \quad (9-646419)$$

The context index increment $ctxIdxInc$ is derived using the color component index $cIdx$, the transform block size $\log2TrafoSize$ and $sigCtx$ as follows.

- If $cIdx$ is equal to 0, $ctxIdxInc$ is derived as follows.

$$ctxIdxInc = ((5 - \log2TrafoSize) * 16) + sigCtx \quad (9-656520)$$

- Otherwise ($cIdx$ is greater than 0), $ctxIdxInc$ is derived as follows.

$$ctxIdxInc = 64 + ((4 - \log2TrafoSize) * 16) + sigCtx \quad (9-666621)$$

[Ed. (BB): The context derivation assumes maximum transform sizes less than or equal to 32x32 for luma and 16x16 for chroma and minimum transform sizes greater than or equal to 4x4.]

9.3.3.1.1.5 Derivation process of $ctxIdxInc$ for the syntax element $coeff_abs_level_greater1_flag$

Inputs to this process are the color component index $cIdx$, the transform block size $\log2TrafoSize$, the 4x4 subset index i and the current coefficient scan index n within the current subset.

Output of this process is $ctxIdxInc$.

The variable $ctxSet$ specifies the current context set and for its derivation the following applies.

- If n is equal to 0 or all previous syntax elements $coeff_abs_level_greater1_flag[pos]$ with pos less than n are derived to be equal to 0 instead of being explicitly parsed, the following applies.

- If the current transform block size $\log2TrafoSize$ is equal to 2, $ctxSet$ is derived as follows.

$$ctxSet = 0 \quad (9-676722)$$

- Otherwise if the current 4x4 subset is the first one in scanning order (i is equal to 0), $ctxSet$ is derived as follows.

$$ctxSet = 5 \quad (9-686823)$$

- Otherwise (i is not equal to 0), the variable $lastGreater2Ctx$ is set equal to the variable $greater2Ctx$ that has been derived during the last invocation of subclause 9.3.3.1.1.6 for the syntax element $coeff_abs_level_greater2_flag$ for the subset $i - 1$ and $ctxSet$ is derived as follows.

$$ctxSet = ((lastGreater2Ctx + 1) >> 2) + 1 \quad (9-696924)$$

HEVC

- The variable `greater1Ctx` is set equal to 1.
- Otherwise (`coeff_abs_level_greater1_flag[n]` is not the first to be parsed within the current subset `i`), for the derivation of `ctxSet` and `greater1Ctx` the following applies.
 - The variable `ctxSet` is set equal to the variable `ctxSet` that has been derived during the last invocation of this subclause.
 - The variable `greater1Ctx` is set equal to the variable `greater1Ctx` that has been derived during the last invocation of this subclause.
 - When `greater1Ctx` is equal to 1, the variable `lastGreater1Flag` is set equal to the syntax element `coeff_abs_level_greater1_flag` that has been used during the last invocation of this subclause as follows.
 - If `lastGreater1Flag` is equal to 1, `greater1Ctx` is set equal to 0.
 - Otherwise (`lastGreater1Flag` is equal to 0), `greater1Ctx` is incremented by 1.

The context index increment `ctxIdxInc` is derived using the current context set `ctxSet` and the current context `greater1Ctx` as follows.

$$\text{ctxIdxInc} = (\text{ctxSet} * 5) + \min(4, \text{greater1Ctx}) \quad (9-707025)$$

When `cIdx` is greater than 0, `ctxIdxInc` is modified as follows.

$$\text{ctxIdxInc} = \text{ctxIdxInc} + 30 \quad (9-717126)$$

9.3.3.1.1.6 Derivation process of `ctxIdxInc` for the syntax element `coeff_abs_level_greater2_flag`

Inputs to this process are the color component index `cIdx`, the transform block size `log2TrafoSize`, the 4x4 subset index `i` and the current coefficient scan index `n` within the current subset.

Output of this process is `ctxIdxInc`.

The variable `ctxSet` specifies the current context set and for its derivation the following applies.

- If `n` is equal to 0 or all previous syntax elements `coeff_abs_level_greater1_flag[pos]` with `pos` less than `n` are derived to be equal to 0 instead of being explicitly parsed, the following applies.
 - If the current transform block size `log2TrafoSize` is equal to 2, `ctxSet` is derived as follows.

$$\text{ctxSet} = 0 \quad (9-727227)$$

- Otherwise if the current 4x4 subset is the first one in scanning order (`i` is equal to 0), `ctxSet` is derived as follows.

$$\text{ctxSet} = 5 \quad (9-737328)$$

- Otherwise (`i` is not equal to 0), the variable `lastGreater2Ctx` is set equal to the variable `greater2Ctx` that has been derived during the last invocation of this subclause for the subset `i - 1` and `ctxSet` is derived as follows.

$$\text{ctxSet} = (\text{lastGreater2Ctx} \gg 2) + 1 \quad (9-747429)$$

- The variable `greater2Ctx` is set equal to 1.
- Otherwise (`coeff_abs_level_greater1_flag[n]` is not the first to be parsed within the current subset `i`), for the derivation of `ctxSet` and `greater2Ctx` the following applies.
 - The variable `ctxSet` is set equal to the variable `ctxSet` that has been derived during the last invocation of this subclause.
 - The variable `greater2Ctx` is set equal to the variable `greater2Ctx` that has been derived during the last invocation of this subclause incremented by 1.

The context index increment `ctxIdxInc` is derived using the current context set `ctxSet` and the current context `greater2Ctx` as follows.

$$\text{ctxIdxInc} = (\text{ctxSet} * 5) + \min(4, \text{greater2Ctx}) \quad (9-757530)$$

When `cIdx` is greater than 0, `ctxIdxInc` is modified as follows.

HEVC

ctxIdxInc = ctxIdxInc + 30

(9-767634)

9.3.3.2 Arithmetic decoding process

Inputs to this process are the bypassFlag, ctxIdxTable, and ctxIdx as derived in subclause 9.3.3.1, and the state variables codlRange and codlOffset of the arithmetic decoding engine.

Output of this process is the value of the bin.

Figure 9-1 illustrates the whole arithmetic decoding process for a single bin. For decoding the value of a bin, the context index table ctxIdxTable and the ctxIdx is passed to the arithmetic decoding process DecodeBin(ctxIdxTable, ctxIdx), which is specified as follows.

- If bypassFlag is equal to 1, DecodeBypass() as specified in subclause 9.3.3.2.3 is invoked.
- Otherwise, if bypassFlag is equal to 0, ctxIdxTable is equal 0 and ctxIdx is equal to 0, DecodeTerminate() as specified in subclause 9.3.3.2.4 is invoked.
- Otherwise (bypassFlag is equal to 0, ctxIdxTable is not equal to 0 and ctxIdx is not equal to 0), DecodeDecision() as specified in subclause 9.3.3.2.1 is applied.

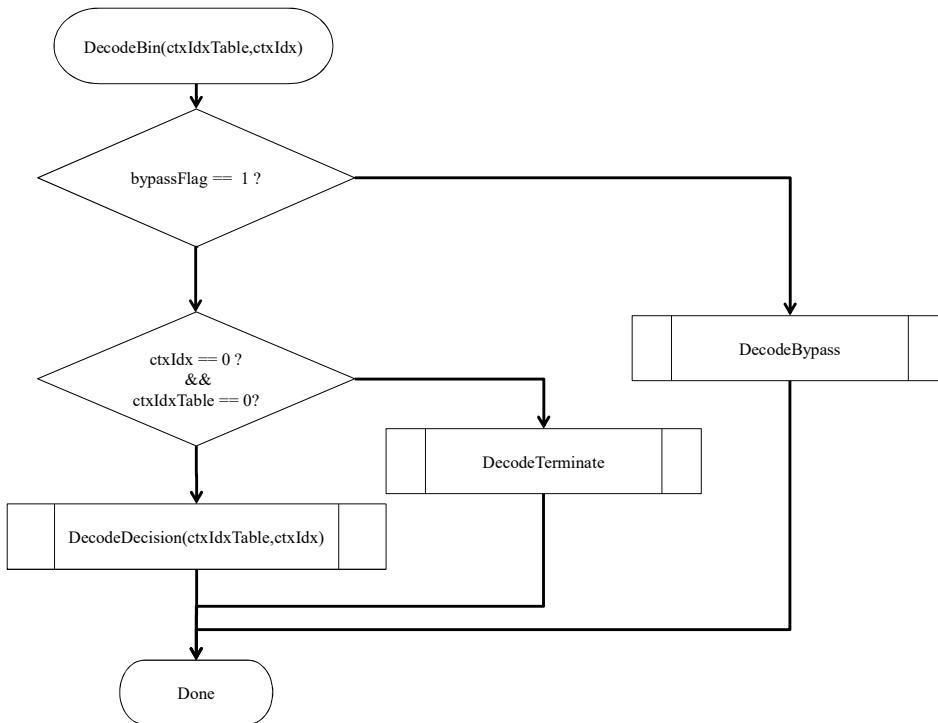


Figure 9-1 – Overview of the arithmetic decoding process for a single bin (informative)

NOTE – Arithmetic coding is based on the principle of recursive interval subdivision. Given a probability estimation $p(0)$ and $p(1) = 1 - p(0)$ of a binary decision (0, 1), an initially given code sub-interval with the range codlRange will be subdivided into two sub-intervals having range $p(0) * \text{codlRange}$ and $\text{codlRange} - p(0) * \text{codlRange}$, respectively. Depending on the decision, which has been observed, the corresponding sub-interval will be chosen as the new code interval, and a binary code string pointing into that interval will represent the sequence of observed binary decisions. It is useful to distinguish between the most probable symbol (MPS) and the least probable symbol (LPS), so that binary decisions have to be identified as either MPS or LPS, rather than 0 or 1. Given this terminology, each context is specified by the probability p_{LPS} of the LPS and the value of MPS (valMPS), which is either 0 or 1.

The arithmetic core engine in this Recommendation | International Standard has three distinct properties:

HEVC

- The probability estimation is performed by means of a finite-state machine with a table-based transition process between 64 different representative probability states $\{p_{LPS}(pStateIdx) | 0 \leq pStateIdx < 64\}$ for the LPS probability p_{LPS} . The numbering of the states is arranged in such a way that the probability state with index $pStateIdx = 0$ corresponds to an LPS probability value of 0.5, with decreasing LPS probability towards higher state indices.
- The range $codIRange$ representing the state of the coding engine is quantised to a small set $\{Q_1, \dots, Q_4\}$ of pre-set quantisation values prior to the calculation of the new interval range. Storing a table containing all 64×4 pre-computed product values of $Q_i * p_{LPS}(pStateIdx)$ allows a multiplication-free approximation of the product $codIRange * p_{LPS}(pStateIdx)$.
- For syntax elements or parts thereof for which an approximately uniform probability distribution is assumed to be given a separate simplified encoding and decoding bypass process is used.

9.3.3.2.1 Arithmetic decoding process for a binary decision

Inputs to this process are $ctxIdxTable$, $ctxIdx$, $codIRange$, and $codIOffset$.

Outputs of this process are the decoded value $binVal$, and the updated variables $codIRange$ and $codIOffset$.

Figure 9-2 [Figure 9-2](#) shows the flowchart for decoding a single decision (DecodeDecision):

1. The value of the variable $codIRangeLPS$ is derived as follows.
 - Given the current value of $codIRange$, the variable $qCodIRangeIdx$ is derived by

$$qCodIRangeIdx = (codIRange \gg 6) \& 3 \quad (9-77729)$$
 - Given $qCodIRangeIdx$ and $pStateIdx$ associated with $ctxIdxTable$ and $ctxIdx$, the value of the variable $rangeTabLPS$ as specified in [Table 9-52](#) [Table 9-40](#) is assigned to $codIRangeLPS$:

$$codIRangeLPS = rangeTabLPS[pStateIdx][qCodIRangeIdx] \quad (9-78730)$$
2. The variable $codIRange$ is set equal to $codIRange - codIRangeLPS$ and the following applies.
 - If $codIOffset$ is greater than or equal to $codIRange$, the variable $binVal$ is set equal to $1 - valMPS$, $codIOffset$ is decremented by $codIRange$, and $codIRange$ is set equal to $codIRangeLPS$.
 - Otherwise, the variable $binVal$ is set equal to $valMPS$.

Given the value of $binVal$, the state transition is performed as specified in subclause 9.3.3.2.1.1. Depending on the current value of $codIRange$, renormalization is performed as specified in subclause 9.3.3.2.2.

9.3.3.2.1.1 State transition process

Inputs to this process are the current $pStateIdx$, the decoded value $binVal$ and $valMPS$ values of the context variable associated with $ctxIdxTable$ and $ctxIdx$.

Outputs of this process are the updated $pStateIdx$ and $valMPS$ of the context variable associated with $ctxIdx$.

Depending on the decoded value $binVal$, the update of the two variables $pStateIdx$ and $valMPS$ associated with $ctxIdx$ is derived as specified by the following pseudo-code:

```

if( binVal == valMPS )
    pStateIdx = transIdxMPS( pStateIdx )
else {
    if( pStateIdx == 0 )
        valMPS = 1 - valMPS
    pStateIdx = transIdxLPS( pStateIdx )
}

```

(9-79731)

[Table 9-53](#) [Table 9-41](#) specifies the transition rules $transIdxMPS()$ and $transIdxLPS()$ after decoding the value of $valMPS$ and $1 - valMPS$, respectively.

HEVC

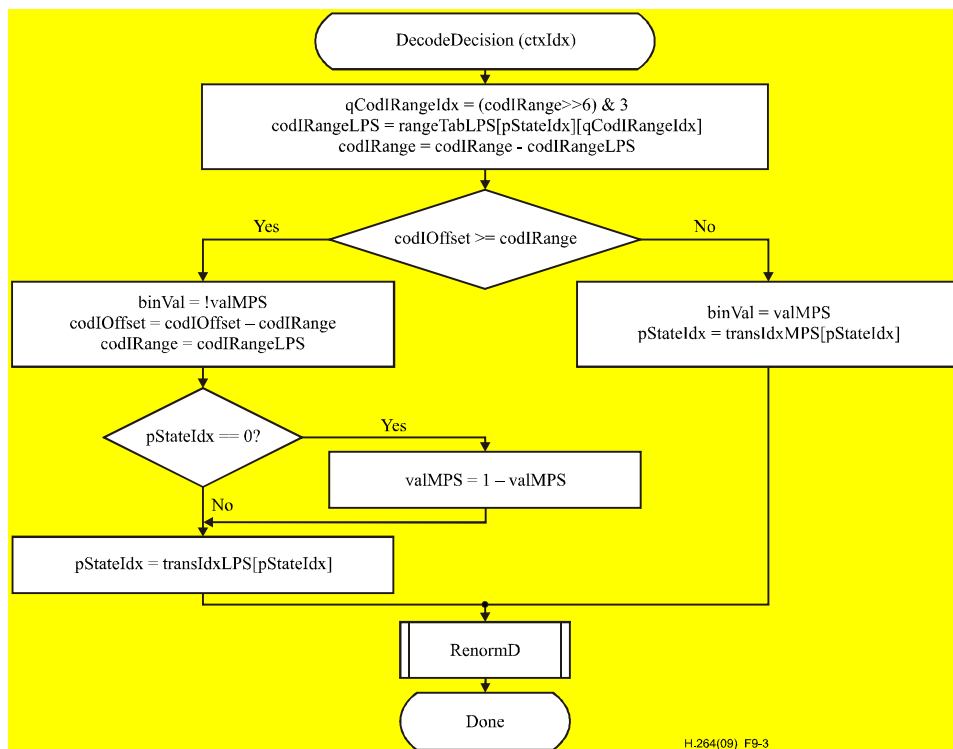


Figure 9-2 – Flowchart for decoding a decision

Table 9-529-529-52 – Specification of rangeTabLPS depending on pStateIdx and qCodIRangeIdx

pStateIdx	qCodIRangeIdx				pStateIdx	qCodIRangeIdx			
	0	1	2	3		0	1	2	3
0	128	176	208	240	32	27	33	39	45
1	128	167	197	227	33	26	31	37	43
2	128	158	187	216	34	24	30	35	41
3	123	150	178	205	35	23	28	33	39
4	116	142	169	195	36	22	27	32	37
5	111	135	160	185	37	21	26	30	35
6	105	128	152	175	38	20	24	29	33
7	100	122	144	166	39	19	23	27	31
8	95	116	137	158	40	18	22	26	30
9	90	110	130	150	41	17	21	25	28
10	85	104	123	142	42	16	20	23	27
11	81	99	117	135	43	15	19	22	25
12	77	94	111	128	44	14	18	21	24

HEVC

13	73	89	105	122	45	14	17	20	23
14	69	85	100	116	46	13	16	19	22
15	66	80	95	110	47	12	15	18	21
16	62	76	90	104	48	12	14	17	20
17	59	72	86	99	49	11	14	16	19
18	56	69	81	94	50	11	13	15	18
19	53	65	77	89	51	10	12	15	17
20	51	62	73	85	52	10	12	14	16
21	48	59	69	80	53	9	11	13	15
22	46	56	66	76	54	9	11	12	14
23	43	53	63	72	55	8	10	12	14
24	41	50	59	69	56	8	9	11	13
25	39	48	56	65	57	7	9	11	12
26	37	45	54	62	58	7	9	10	12
27	35	43	51	59	59	7	8	10	11
28	33	41	48	56	60	6	8	9	11
29	32	39	46	53	61	6	7	9	10
30	30	37	43	50	62	6	7	8	9
31	29	35	41	48	63	2	2	2	2

Table 9-539-539-53 – State transition table

pStateIdx	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
transIdxLPS	0	0	1	2	2	4	4	5	6	7	8	9	9	11	11	12
transIdxMPS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
pStateIdx	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
transIdxLPS	13	13	15	15	16	16	18	18	19	19	21	21	22	22	23	24
transIdxMPS	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
pStateIdx	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
transIdxLPS	24	25	26	26	27	27	28	29	29	30	30	30	31	32	32	33
transIdxMPS	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
pStateIdx	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
transIdxLPS	33	33	34	34	35	35	35	36	36	36	37	37	37	38	38	63
transIdxMPS	49	50	51	52	53	54	55	56	57	58	59	60	61	62	62	63

9.3.3.2.2 Renormalization process in the arithmetic decoding engine

Inputs to this process are bits from slice data and the variables codIRange and codIOffset.

HEVC

Outputs of this process are the updated variables `codIRange` and `codIOffset`.

A flowchart of the renormalization is shown in [Figure 9-3](#). The current value of `codIRange` is first compared to 256 and further steps are specified as follows.

- If `codIRange` is greater than or equal to 256, no renormalization is needed and the RenormD process is finished;
- Otherwise (`codIRange` is less than 256), the renormalization loop is entered. Within this loop, the value of `codIRange` is doubled, i.e., left-shifted by 1 and a single bit is shifted into `codIOffset` by using `read_bits(1)`.

The bitstream shall not contain data that result in a value of `codIOffset` being greater than or equal to `codIRange` upon completion of this process.

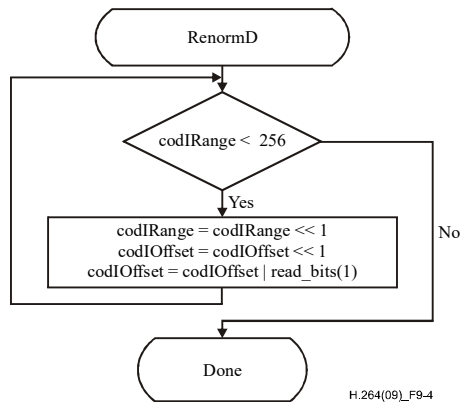


Figure 9-3 – Flowchart of renormalization

9.3.3.2.3 Bypass decoding process for binary decisions

Inputs to this process are bits from slice data and the variables `codIRange` and `codIOffset`.

Outputs of this process are the updated variable `codIOffset` and the decoded value `binVal`.

The bypass decoding process is invoked when `bypassFlag` is equal to 1. [Figure 9-4](#) shows a flowchart of the corresponding process.

First, the value of `codIOffset` is doubled, i.e., left-shifted by 1 and a single bit is shifted into `codIOffset` by using `read_bits(1)`. Then, the value of `codIOffset` is compared to the value of `codIRange` and further steps are specified as follows.

- If `codIOffset` is greater than or equal to `codIRange`, the variable `binVal` is set equal to 1 and `codIOffset` is decremented by `codIRange`.
- Otherwise (`codIOffset` is less than `codIRange`), the variable `binVal` is set equal to 0.

The bitstream shall not contain data that result in a value of `codIOffset` being greater than or equal to `codIRange` upon completion of this process.

HEVC

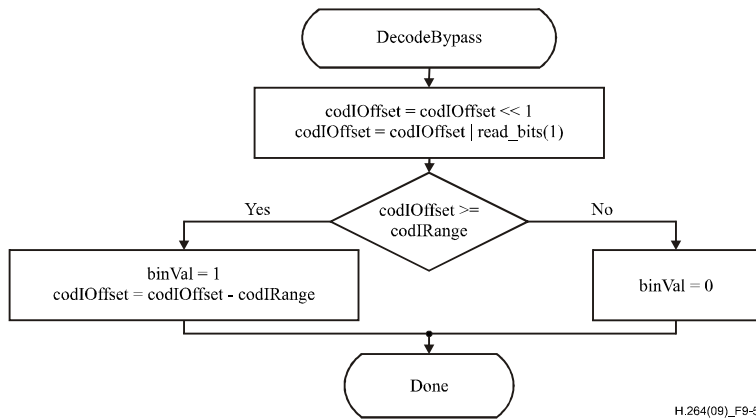


Figure 9-4 – Flowchart of bypass decoding process

9.3.3.2.4 Decoding process for binary decisions before termination

Inputs to this process are bits from slice data and the variables `codIRange` and `codIOffset`.

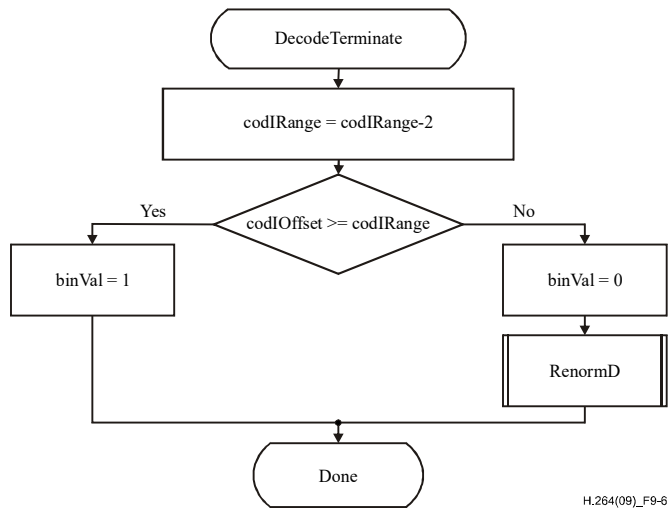
Outputs of this process are the updated variables `codIRange` and `codIOffset`, and the decoded value `binVal`.

This special decoding routine applies to decoding of `end_of_slice_flag` and `pcm_flag` corresponding to `ctxIdxTable` equal to 0 and `ctxIdx` equal to 0. Figure 9-5 shows the flowchart of the corresponding decoding process, which is specified as follows.

First, the value of `codIRange` is decremented by 2. Then, the value of `codIOffset` is compared to the value of `codIRange` and further steps are specified as follows.

- If `codIOffset` is greater than or equal to `codIRange`, the variable `binVal` is set equal to 1, no renormalization is carried out, and CABAC decoding is terminated. The last bit inserted in register `codIOffset` is equal to 1. When decoding `end_of_slice_flag`, this last bit inserted in register `codIOffset` is interpreted as `rsbsp_stop_one_bit`.
- Otherwise (`codIOffset` is less than `codIRange`), the variable `binVal` is set equal to 0 and renormalization is performed as specified in subclause 9.3.3.2.2.

NOTE – This procedure may also be implemented using `DecodeDecision(ctxIdxTable, ctxIdx)` with `ctxIdxTable = 0` and `ctxIdx = 0`. In the case where the decoded value is equal to 1, seven more bits would be read by `DecodeDecision(ctxIdxTable, ctxIdx)` and a decoding process would have to adjust its bitstream pointer accordingly to properly decode following syntax elements.



H.264(09)_F9-6

Figure 9-5 – Flowchart of decoding a decision before termination

9.3.4 Arithmetic encoding process (informative)

This subclause does not form an integral part of this Recommendation | International Standard.

Inputs to this process are decisions that are to be encoded and written.

Outputs of this process are bits that are written to the RBSP.

This informative subclause describes an arithmetic encoding engine that matches the arithmetic decoding engine described in subclause 9.3.3.2. The encoding engine is essentially symmetric with the decoding engine, i.e., procedures are called in the same order. The following procedures are described in this section: InitEncoder, EncodeDecision, EncodeBypass, EncodeTerminate, which correspond to InitDecoder, DecodeDecision, DecodeBypass, and DecodeTerminate, respectively. The state of the arithmetic encoding engine is represented by a value of the variable codILow pointing to the lower end of a sub-interval and a value of the variable codIRange specifying the corresponding range of that sub-interval.

9.3.4.1 Initialisation process for the arithmetic encoding engine (informative)

This subclause does not form an integral part of this Recommendation | International Standard.

This process is invoked before encoding the first macroblock of a slice, and after encoding any pcm_alignment_zero_bit and all pcm_sample_luma and pcm_sample_chroma data for a [macroblock-coding unit](#) of type I_PCM.

Outputs of this process are the values codILow, codIRange, firstBitFlag, bitsOutstanding, and BinCountsInNALunits of the arithmetic encoding engine.

In the initialisation procedure of the encoder, codILow is set equal to 0, and codIRange is set equal to 510. Furthermore, firstBitFlag is set equal to 1 and the counter bitsOutstanding is set equal to 0.

Depending on whether the current slice is the first slice of a coded picture, the following applies.

- If the current slice is the first slice of a coded picture, the counter BinCountsInNALunits is set equal to 0.
- Otherwise (the current slice is not the first slice of a coded picture), the counter BinCountsInNALunits is not modified. The value of BinCountsInNALunits is the result of encoding all the slices of a coded picture that precede the current slice in decoding order. After initialising for the first slice of a coded picture as specified in this subclause, BinCountsInNALunits is incremented as specified in subclauses 9.3.4.2, 9.3.4.4, and 9.3.4.5.

NOTE – The minimum register precision required for storing the values of the variables codILow and codIRange after invocation of any of the arithmetic encoding processes specified in subclauses 9.3.4.2, 9.3.4.4, and 9.3.4.5 is 10 bits and 9 bits, respectively. The encoding process for a binary decision (EncodeDecision) as specified in subclause 9.3.4.2 and the encoding process for a binary decision before termination (EncodeTerminate) as specified in subclause 9.3.4.5 require a minimum register precision of 10 bits for the variable codILow and a minimum register precision of 9 bits for the variable codIRange. The bypass encoding process for binary decisions (EncodeBypass) as specified in subclause 9.3.4.4 requires a minimum register precision of 11 bits for the variable codILow and a minimum register precision of 9 bits for the variable codIRange. The precision required for the

HEVC

counters bitsOutstanding and BinCountsInNALunits should be sufficiently large to prevent overflow of the related registers. When maxBinCountInSlice denotes the maximum total number of binary decisions to encode in one slice and maxBinCountInPic denotes the maximum total number of binary decisions to encode a picture, the minimum register precision required for the variables bitsOutstanding and BinCountsInNALunits is given by $\text{Ceil}(\text{Log}_2(\text{maxBinCountInSlice} + 1))$ and $\text{Ceil}(\text{Log}_2(\text{maxBinCountInPic} + 1))$, respectively.

9.3.4.2 Encoding process for a binary decision (informative)

This subclause does not form an integral part of this Recommendation | International Standard.

Inputs to this process are the context index ctxIdx, the value of binVal to be encoded, and the variables codIRange, codILow and BinCountsInNALunits.

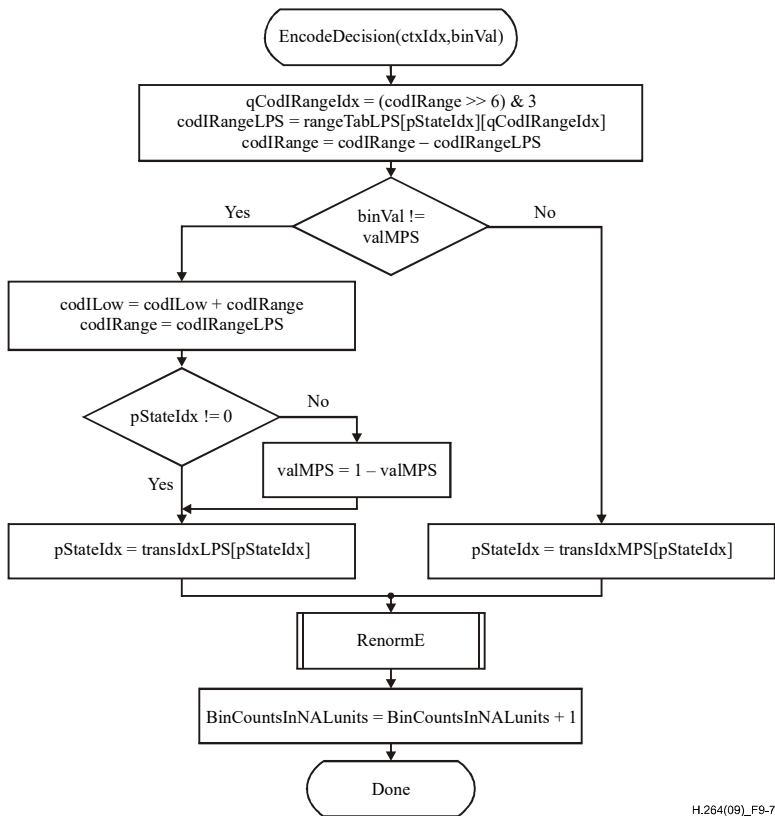
Outputs of this process are the variables codIRange, codILow, and BinCountsInNALunits.

Figure 9-6 shows the flowchart for encoding a single decision. In a first step, the variable codIRangeLPS is derived as follows.

Given the current value of codIRange, codIRange is mapped to the index qCodIRangeIdx of a quantised value of codIRange by using Equation 9-779-6. The value of qCodIRangeIdx and the value of pStateIdx associated with ctxIdx are used to determine the value of the variable rangeTabLPS as specified in Table 9-52 Table 9-8, which is assigned to codIRangeLPS. The value of $\text{codIRange} - \text{codIRangeLPS}$ is assigned to codIRange.

In a second step, the value of binVal is compared to valMPS associated with ctxIdx. When binVal is different from valMPS, codIRange is added to codILow and codIRange is set equal to the value codIRangeLPS. Given the encoded decision, the state transition is performed as specified in subclause 9.3.3.2.1.1. Depending on the current value of codIRange, renormalization is performed as specified in subclause 9.3.4.3. Finally, the variable BinCountsInNALunits is incremented by 1.

HEVC



H.264(09)_F9-7

Figure 9-6 – Flowchart for encoding a decision

9.3.4.3 Renormalization process in the arithmetic encoding engine (informative)

This subclause does not form an integral part of this Recommendation | International Standard.

Inputs to this process are the variables codIRange, codILow, firstBitFlag, and bitsOutstanding.

Outputs of this process are zero or more bits written to the RBSP and the updated variables codIRange, codILow, firstBitFlag, and bitsOutstanding.

Renormalization is illustrated in [Figure 9-7](#).

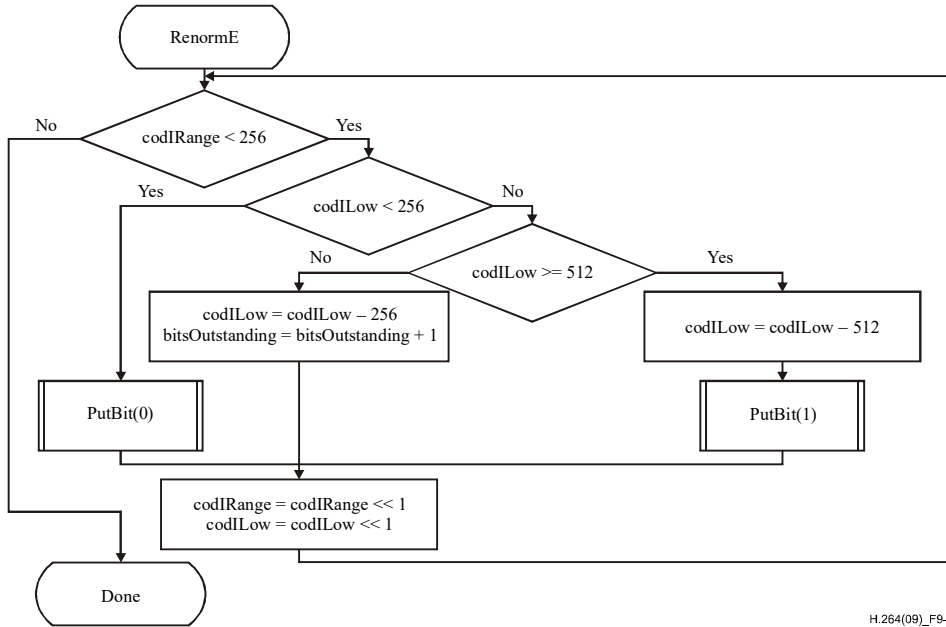


Figure 9-7 – Flowchart of renormalization in the encoder

The PutBit() procedure described in [Figure 9-8](#) provides carry over control. It uses the function WriteBits(B, N) that writes N bits with value B to the bitstream and advances the bitstream pointer by N bit positions. This function assumes the existence of a bitstream pointer with an indication of the position of the next bit to be written to the bitstream by the encoding process.

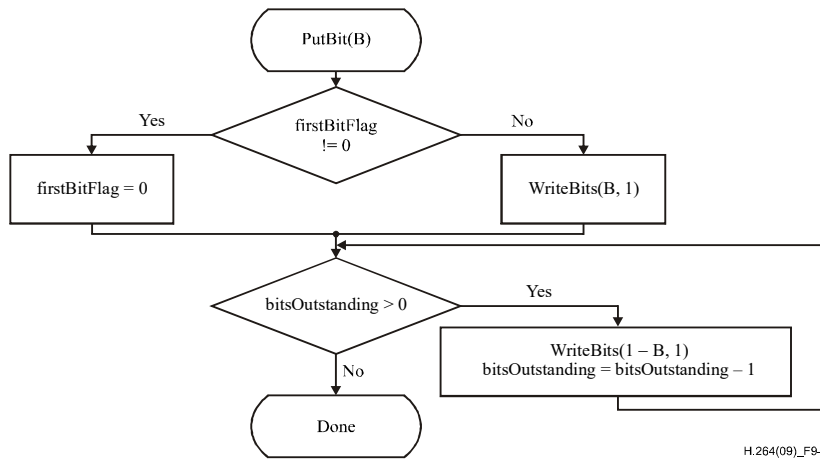


Figure 9-8 – Flowchart of PutBit(B)

9.3.4.4 Bypass encoding process for binary decisions (informative)

This subclause does not form an integral part of this Recommendation | International Standard.

Inputs to this process are the variables binVal, codiLow, codiRange, bitsOutstanding, and BinCountsInNALunits.

Output of this process is a bit written to the RBSP and the updated variables codiLow, bitsOutstanding, and BinCountsInNALunits.

This encoding process applies to all binary decisions with bypassFlag equal to 1. Renormalization is included in the specification of this process as given in [Figure 9-9](#).

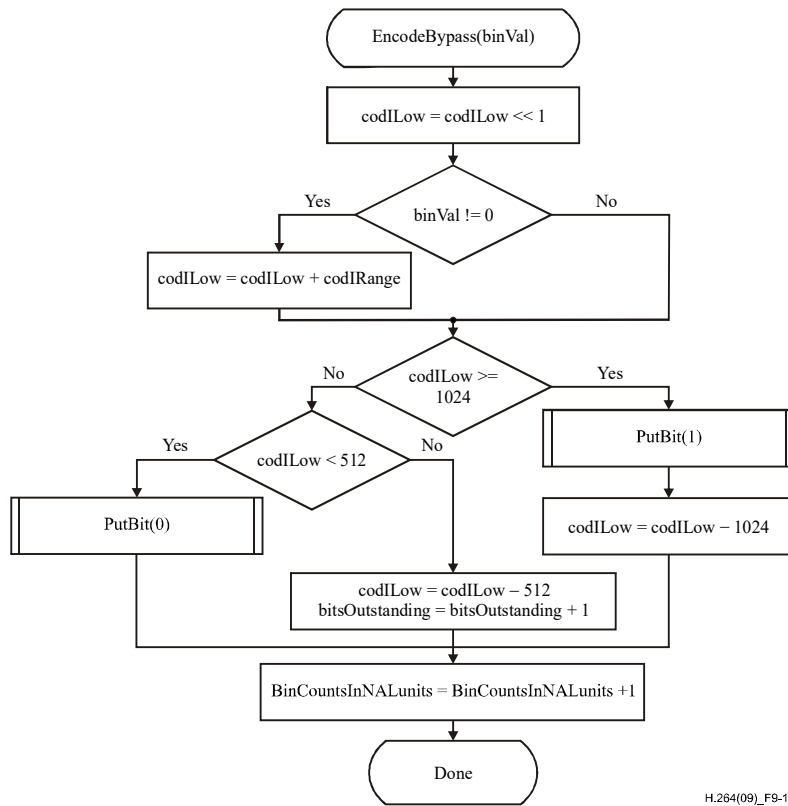


Figure 9-9 – Flowchart of encoding bypass

9.3.4.5 Encoding process for a binary decision before termination (informative)

This subclause does not form an integral part of this Recommendation | International Standard.

Inputs to this process are the variables binVal, codIRange, codILow, bitsOutstanding, and BinCountsInNALunits.

Outputs of this process are zero or more bits written to the Rbsp and the updated variables codILow, codIRange, bitsOutstanding, and BinCountsInNALunits.

This encoding routine shown in [Figure 9-10](#) applies to encoding of the end_of_slice_flag and pcm_flag of the bin indicating the I_PCM mb_type mode both associated with ctxIdx equal to 276.

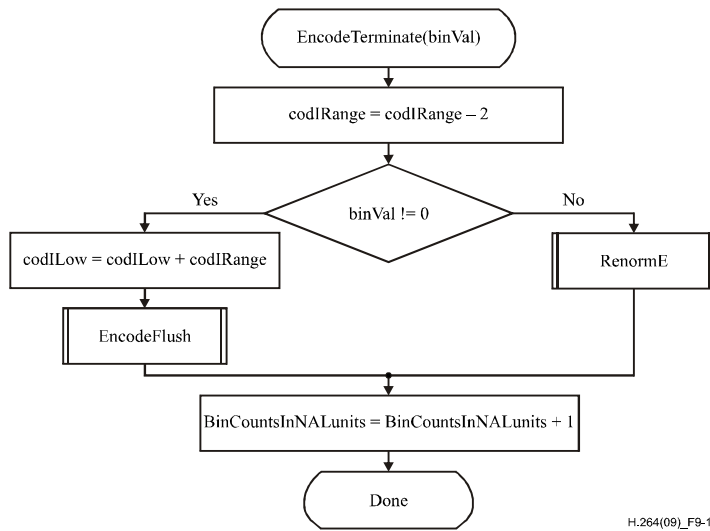


Figure 9-10 – Flowchart of encoding a decision before termination

When the value of binVal to encode is equal to 1, CABAC encoding is terminated and the flushing procedure shown in [Figure 9-11](#) is applied. In this flushing procedure, the last bit written by WriteBits(B, N) is equal to 1. When encoding end_of_slice_flag, this last bit is interpreted as the rbsp_stop_one_bit.

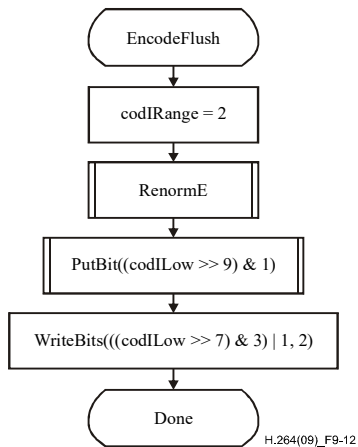


Figure 9-11 – Flowchart of flushing at termination

9.3.4.6 Byte stuffing process (informative)

This subclause does not form an integral part of this Recommendation | International Standard.

This process is invoked after encoding the last macroblock of the last slice of a picture and after encapsulation.

Inputs to this process are the number of bytes NumBytesInVclNALunits of all VCL NAL units of a picture, the number of macroblocks PicSizeInMbs in the picture, and the number of binary symbols BinCountsInNALunits resulting from encoding the contents of all VCL NAL units of the picture.

NOTE – The value of BinCountsInNALunits is the result of encoding all slices of a coded picture. After initialising for the first slice of a coded picture as specified in subclause 9.3.4.1, BinCountsInNALunits is incremented as specified in subclauses 9.3.4.2, 9.3.4.4, and 9.3.4.5.

Outputs of this process are zero or more bytes appended to the NAL unit.

Let the variable k be set equal to $\text{Ceil}(\text{Ceil}(3 * (32 * \text{BinCountsInNALunits} - \text{RawMbBits} * \text{PicSizeInMbs}) \div 1024) - \text{NumBytesInVclNALunits}) \div 3$. Depending on the variable k the following applies.

- If k is less than or equal to 0, no cabac_zero_word is appended to the NAL unit.
- Otherwise (k is greater than 0), the 3-byte sequence 0x000003 is appended k times to the NAL unit after encapsulation, where the first two bytes 0x0000 represent a cabac_zero_word and the third byte 0x03 represents an emulation_prevention_three_byte.

Annex A

Profiles and levels

(This annex forms an integral part of this Recommendation | International Standard)

Profiles and levels specify restrictions on bitstreams and hence limits on the capabilities needed to decode the bitstreams. Profiles and levels may also be used to indicate interoperability points between individual decoder implementations.

NOTE 1 – This Recommendation | International Standard does not include individually selectable "options" at the decoder, as this would increase interoperability difficulties.

Each profile specifies a subset of algorithmic features and limits that shall be supported by all decoders conforming to that profile.

NOTE 2 – Encoders are not required to make use of any particular subset of features supported in a profile.

Each level specifies a set of limits on the values that may be taken by the syntax elements of this Recommendation | International Standard. The same set of level definitions is used with all profiles, but individual implementations may support a different level for each supported profile. For any given profile, levels generally correspond to decoder processing load and memory capability.

The profiles that are specified in subclause A.2 are also referred to as the profiles specified in Annex A.

A.1 Requirements on video decoder capability

Capabilities of video decoders conforming to this Recommendation | International Standard are specified in terms of the ability to decode video streams conforming to the constraints of profiles and levels specified in this annex. For each such profile, the level supported for that profile shall also be expressed.

Specific values are specified in this annex for the syntax elements `profile_idc` and `level_idc`. All other values of `profile_idc` and `level_idc` are reserved for future use by ITU-T | ISO/IEC.

NOTE – Decoders should not infer that when a reserved value of `profile_idc` or `level_idc` falls between the values specified in this Recommendation | International Standard that this indicates intermediate capabilities between the specified profiles or levels, as there are no restrictions on the method to be chosen by ITU-T | ISO/IEC for the use of such future reserved values.

A.2 Profiles

All constraints for picture parameter sets that are specified are constraints for picture parameter sets that are activated in the bitstream. All constraints for sequence parameter sets that are specified in subclause A.2.1 are constraints for sequence parameter sets that are activated in the bitstream.

A.2.1 ABC profile

A.3 Levels

HEVC

Table A-9-549-54 – Level limits [Ed. (TW) Kept for convenience]

Level number	Max macroblock processing rate MaxMBPS (MB/s)	Max picture size MaxFS (MBs)	Max decoded picture buffer size MaxDpbMbs (MBs)	Max video bit rate MaxBR (1000 bits/s, 1200 bits/s, cpbBrVclFactor bits/s, or cpbBrNalFactor bits/s)	Max CPB size MaxCPB (1000 bits, 1200 bits, cpbBrVclFactor bits, or cpbBrNalFactor bits)	Vertical MV component range MaxVmvR (luma picture samples)	Min compression ratio MinCR	Max number of motion vectors per two consecutive MBs MaxMvsPer2Mb
1	1 485	99	396	64	175	[-64,+63.75]	2	-
1b	1 485	99	396	128	350	[-64,+63.75]	2	-
1.1	3 000	396	900	192	500	[-128,+127.75]	2	-
1.2	6 000	396	2 376	384	1 000	[-128,+127.75]	2	-
1.3	11 880	396	2 376	768	2 000	[-128,+127.75]	2	-
2	11 880	396	2 376	2 000	2 000	[-128,+127.75]	2	-
2.1	19 800	792	4 752	4 000	4 000	[-256,+255.75]	2	-
2.2	20 250	1 620	8 100	4 000	4 000	[-256,+255.75]	2	-
3	40 500	1 620	8 100	10 000	10 000	[-256,+255.75]	2	32
3.1	108 000	3 600	18 000	14 000	14 000	[-512,+511.75]	4	16
3.2	216 000	5 120	20 480	20 000	20 000	[-512,+511.75]	4	16
4	245 760	8 192	32 768	20 000	25 000	[-512,+511.75]	4	16
4.1	245 760	8 192	32 768	50 000	62 500	[-512,+511.75]	2	16
4.2	522 240	8 704	34 816	50 000	62 500	[-512,+511.75]	2	16
5	589 824	22 080	110 400	135 000	135 000	[-512,+511.75]	2	16
5.1	983 040	36 864	184 320	240 000	240 000	[-512,+511.75]	2	16

Levels with non-integer level numbers in Table A-9-54Table A-9-8 are referred to as "intermediate levels".

NOTE 4 – All levels have the same status, but some applications may choose to use only the integer-numbered levels.

HEVC

Annex B

Byte stream format

(This annex forms an integral part of this Recommendation | International Standard)

This annex specifies syntax and semantics of a byte stream format specified for use by applications that deliver some or all of the NAL unit stream as an ordered stream of bytes or bits within which the locations of NAL unit boundaries need to be identifiable from patterns in the data, such as ITU-T Rec. H.222.0 | ISO/IEC 13818-1 systems or ITU-T Rec. H.320 systems. For bit-oriented delivery, the bit order for the byte stream format is specified to start with the MSB of the first byte, proceed to the LSB of the first byte, followed by the MSB of the second byte, etc.

The byte stream format consists of a sequence of byte stream NAL unit syntax structures. Each byte stream NAL unit syntax structure contains one start code prefix followed by one nal_unit(NumBytesInNALunit) syntax structure. It may (and under some circumstances, it shall) also contain an additional zero_byte syntax element. It may also contain one or more additional trailing_zero_8bits syntax elements. When it is the first byte stream NAL unit in the bitstream, it may also contain one or more additional leading_zero_8bits syntax elements.

B.1 Byte stream NAL unit syntax and semantics

B.1.1 Byte stream NAL unit syntax

	C	Descriptor
byte_stream_nal_unit(NumBytesInNALunit) {		
while(next_bits(24) != 0x000001 && next_bits(32) != 0x00000001)		
leading_zero_8bits /* equal to 0x00 */		f(8)
if(next_bits(24) != 0x000001)		
zero_byte /* equal to 0x00 */		f(8)
start_code_prefix_one_3bytes /* equal to 0x000001 */		f(24)
nal_unit(NumBytesInNALunit)		
while(more_data_in_byte_stream() && next_bits(24) != 0x000001 && next_bits(32) != 0x00000001)		
trailing_zero_8bits /* equal to 0x00 */		f(8)
}		

B.1.2 Byte stream NAL unit semantics

The order of byte stream NAL units in the byte stream shall follow the decoding order of the NAL units contained in the byte stream NAL units (see subclause 7.4.1.2). The content of each byte stream NAL unit is associated with the same access unit as the NAL unit contained in the byte stream NAL unit (see subclause 7.4.1.2.3.7.4.1.2.3).

leading_zero_8bits is a byte equal to 0x00.

NOTE – The leading_zero_8bits syntax element can only be present in the first byte stream NAL unit of the bitstream, because (as shown in the syntax diagram of subclause B.1.1) any bytes equal to 0x00 that follow a NAL unit syntax structure and precede the four-byte sequence 0x00000001 (which is to be interpreted as a zero_byte followed by a start_code_prefix_one_3bytes) will be considered to be trailing_zero_8bits syntax elements that are part of the preceding byte stream NAL unit.

zero_byte is a single byte equal to 0x00.

When any of the following conditions are fulfilled, the zero_byte syntax element shall be present:

- the nal_unit_type within the nal_unit() is equal to 7 (sequence parameter set) or 8 (picture parameter set),
- the byte stream NAL unit syntax structure contains the first NAL unit of an access unit in decoding order, as specified by subclause 7.4.1.2.3.

start_code_prefix_one_3bytes is a fixed-value sequence of 3 bytes equal to 0x000001. This syntax element is called a start code prefix.

trailing_zero_8bits is a byte equal to 0x00.

HEVC

B.2 Byte stream NAL unit decoding process

Input to this process consists of an ordered stream of bytes consisting of a sequence of byte stream NAL unit syntax structures.

Output of this process consists of a sequence of NAL unit syntax structures.

At the beginning of the decoding process, the decoder initialises its current position in the byte stream to the beginning of the byte stream. It then extracts and discards each leading `zero_8bits` syntax element (if present), moving the current position in the byte stream forward one byte at a time, until the current position in the byte stream is such that the next four bytes in the bitstream form the four-byte sequence `0x00000001`.

The decoder then performs the following step-wise process repeatedly to extract and decode each NAL unit syntax structure in the byte stream until the end of the byte stream has been encountered (as determined by unspecified means) and the last NAL unit in the byte stream has been decoded:

1. When the next four bytes in the bitstream form the four-byte sequence `0x00000001`, the next byte in the byte stream (which is a `zero_byte` syntax element) is extracted and discarded and the current position in the byte stream is set equal to the position of the byte following this discarded byte.
2. The next three-byte sequence in the byte stream (which is a `start_code_prefix_one_3bytes`) is extracted and discarded and the current position in the byte stream is set equal to the position of the byte following this three-byte sequence.
3. `NumBytesInNALunit` is set equal to the number of bytes starting with the byte at the current position in the byte stream up to and including the last byte that precedes the location of any of the following conditions:
 - A subsequent byte-aligned three-byte sequence equal to `0x00000000`,
 - A subsequent byte-aligned three-byte sequence equal to `0x00000001`,
 - The end of the byte stream, as determined by unspecified means.
4. `NumBytesInNALunit` bytes are removed from the bitstream and the current position in the byte stream is advanced by `NumBytesInNALunit` bytes. This sequence of bytes is `nal_unit(NumBytesInNALunit)` and is decoded using the NAL unit decoding process.
5. When the current position in the byte stream is not at the end of the byte stream (as determined by unspecified means) and the next bytes in the byte stream do not start with a three-byte sequence equal to `0x00000001` and the next bytes in the byte stream do not start with a four byte sequence equal to `0x00000001`, the decoder extracts and discards each `trailing_zero_8bits` syntax element, moving the current position in the byte stream forward one byte at a time, until the current position in the byte stream is such that the next bytes in the byte stream form the four-byte sequence `0x00000001` or the end of the byte stream has been encountered (as determined by unspecified means).

B.3 Decoder byte-alignment recovery (informative)

This subclause does not form an integral part of this Recommendation | International Standard.

Many applications provide data to a decoder in a manner that is inherently byte aligned, and thus have no need for the bit-oriented byte alignment detection procedure described in this subclause.

A decoder is said to have byte-alignment with a bitstream when the decoder is able to determine whether or not the positions of data in the bitstream are byte-aligned. When a decoder does not have byte alignment with the encoder's byte stream, the decoder may examine the incoming bitstream for the binary pattern '00000000 00000000 00000000 00000001' (31 consecutive bits equal to 0 followed by a bit equal to 1). The bit immediately following this pattern is the first bit of an aligned byte following a start code prefix. Upon detecting this pattern, the decoder will be byte aligned with the encoder and positioned at the start of a NAL unit in the byte stream.

Once byte aligned with the encoder, the decoder can examine the incoming byte stream for subsequent three-byte sequences `0x000001` and `0x000003`.

When the three-byte sequence `0x000001` is detected, this is a start code prefix.

When the three-byte sequence `0x000003` is detected, the third byte (`0x03`) is an `emulation_prevention_three_byte` to be discarded as specified in subclause 7.4.1.

When an error in the bitstream syntax is detected (e.g., a non-zero value of the `forbidden_zero_bit` or one of the three-byte or four-byte sequences that are prohibited in subclause 7.4.1), the decoder may consider the detected condition as an indication that byte alignment may have been lost and may discard all bitstream data until the detection of byte alignment at a later position in the bitstream as described in this subclause.

HEVC

Annex C

Hypothetical reference decoder

(This annex forms an integral part of this Recommendation | International Standard)

This annex specifies the hypothetical reference decoder (HRD) and its use to check bitstream and decoder conformance.

Two types of bitstreams are subject to HRD conformance checking for this Recommendation | International Standard. The first such type of bitstream, called Type I bitstream, is a NAL unit stream containing only the VCL NAL units and filler data NAL units for all access units in the bitstream. The second type of bitstream, called a Type II bitstream, contains, in addition to the VCL NAL units and filler data NAL units for all access units in the bitstream, at least one of the following:

- additional non-VCL NAL units other than filler data NAL units,
- all `leading_zero_8bits`, `zero_byte`, `start_code_prefix_one_3bytes`, and `trailing_zero_8bits` syntax elements that form a byte stream from the NAL unit stream (as specified in Annex B).

Figure C-9-12 shows the types of bitstream conformance points checked by the HRD.

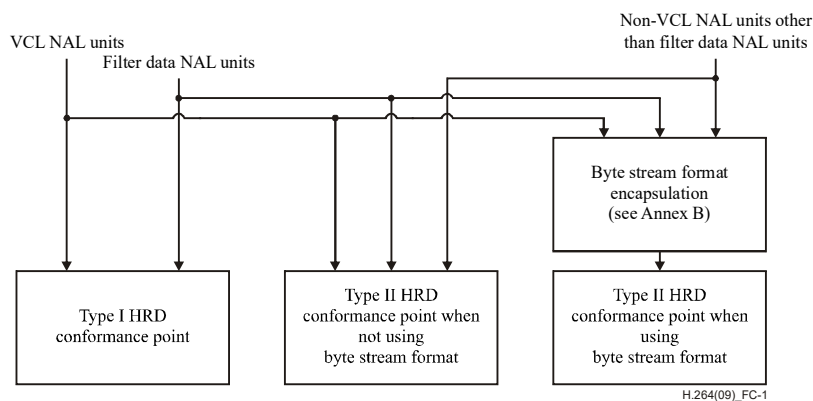


Figure C-9-12 – Structure of byte streams and NAL unit streams for HRD conformance checks

The syntax elements of non-VCL NAL units (or their default values for some of the syntax elements), required for the HRD, are specified in the semantic subclauses of clause 6.67, Annexes D and E.

The HRD contains a coded picture buffer (CPB), an instantaneous decoding process, a decoded picture buffer (DPB), and output cropping as shown in Figure C-9-13.

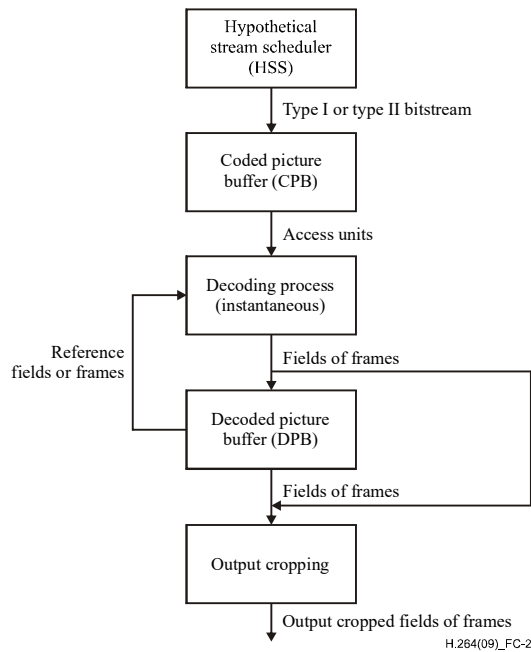


Figure C-9-13 – HRD buffer model

C.1 Operation of coded picture buffer (CPB)

The specifications in this subclause apply independently to each set of CPB parameters that is present and to both the Type I and Type II conformance points shown in [Figure C-9-12](#) [Figure C-9-12](#).

C.1.1 Timing of bitstream arrival

C.1.2 Timing of coded picture removal

C.2 Operation of the decoded picture buffer (DPB)

C.3 Bitstream conformance

C.4 Decoder conformance

HEVC

Annex D

Supplemental enhancement information

(This annex forms an integral part of this Recommendation | International Standard)

This annex specifies syntax and semantics for SEI message payloads.

SEI messages assist in processes related to decoding, display or other purposes. However, SEI messages are not required for constructing the luma or chroma samples by the decoding process. Conforming decoders are not required to process this information for output order conformance to this Recommendation | International Standard (see Annex 0 for the specification of conformance). Some SEI message information is required to check bitstream conformance and for output timing decoder conformance.

In Annex D, specification for presence of SEI messages are also satisfied when those messages (or some subset of them) are conveyed to decoders (or to the HRD) by other means not specified by this Recommendation | International Standard. When present in the bitstream, SEI messages shall obey the syntax and semantics specified in subclauses 7.3.2.3 and 0 and this annex. When the content of an SEI message is conveyed for the application by some means other than presence within the bitstream, the representation of the content of the SEI message is not required to use the same syntax specified in this annex. For the purpose of counting bits, only the appropriate bits that are actually present in the bitstream are counted.

D.1 SEI payload syntax

[Ed. (TW): insert table]

D.2 SEI payload semantics

HEVC

Annex E

Video usability information

(This annex forms an integral part of this Recommendation | International Standard)

This annex specifies syntax and semantics of the VUI parameters of the sequence parameter sets.

VUI parameters are not required for constructing the luma or chroma samples by the decoding process. Conforming decoders are not required to process this information for output order conformance to this Recommendation | International Standard (see Annex 0 for the specification of conformance). Some VUI parameters are required to check bitstream conformance and for output timing decoder conformance.

In Annex E, specification for presence of VUI parameters is also satisfied when those parameters (or some subset of them) are conveyed to decoders (or to the HRD) by other means not specified by this Recommendation | International Standard. When present in the bitstream, VUI parameters shall follow the syntax and semantics specified in subclauses 7.3.2.1 and 7.4.2.1 and this annex. When the content of VUI parameters is conveyed for the application by some means other than presence within the bitstream, the representation of the content of the VUI parameters is not required to use the same syntax specified in this annex. For the purpose of counting bits, only the appropriate bits that are actually present in the bitstream are counted.

E.1 VUI syntax

E.1.1 VUI parameters syntax

[Ed. (TW): insert table]

E.2 VUI semantics

HEVC

English (United Kingdom)

▲ Page 104: [1] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [1] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [1] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [1] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [1] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [1] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [1] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [1] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [1] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [1] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [1] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [1] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [1] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [1] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [1] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [1] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [1] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

English (United Kingdom)

▲ Page 104: [1] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [1] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [1] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [1] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [2] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [2] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [3] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [3] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [3] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [3] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [3] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [3] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [3] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [3] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [3] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [3] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [3] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

English (United Kingdom)

▲ Page 104: [3] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [3] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [3] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [3] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [3] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [3] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [3] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [3] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [3] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [3] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [3] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [3] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [3] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [3] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [3] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [3] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ Page 104: [3] Formatted JCTVC-E069 6/27/2011 1:32:00 PM

English (United Kingdom)

▲ **Page 104: [3] Formatted** JCTVC-E069 6/27/2011 1:32:00 PM
English (United Kingdom)

▲ **Page 104: [3] Formatted** JCTVC-E069 6/27/2011 1:32:00 PM
English (United Kingdom)

▲ **Page 104: [3] Formatted** JCTVC-E069 6/27/2011 1:32:00 PM
English (United Kingdom)

▲ **Page 104: [3] Formatted** JCTVC-E069 6/27/2011 1:32:00 PM
English (United Kingdom)

▲ **Page 104: [3] Formatted** JCTVC-E069 6/27/2011 1:32:00 PM
English (United Kingdom)

▲ **Page 104: [3] Formatted** JCTVC-E069 6/27/2011 1:32:00 PM
English (United Kingdom)

▲ **Page 104: [3] Formatted** JCTVC-E069 6/27/2011 1:32:00 PM
English (United Kingdom)

▲ **Page 104: [3] Formatted** JCTVC-E069 6/27/2011 1:32:00 PM
English (United Kingdom)

▲ **Page 104: [3] Formatted** JCTVC-E069 6/27/2011 1:32:00 PM
English (United Kingdom)

▲ **Page 104: [3] Formatted** JCTVC-E069 6/27/2011 1:32:00 PM
English (United Kingdom)

▲ **Page 104: [3] Formatted** JCTVC-E069 6/27/2011 1:32:00 PM
English (United Kingdom)