**Text specification for palette mode used as a basis in screen content coding core experiment 3 (SCCE3)**

This text specification is on top of JCTVC-P1005\_v4.

## Known issues in the text:

* max\_palette\_size is never signalled.
* Palette mode for 4:2:2 and 4:2:0 formats may not work correctly.
* Problems in the reconstruction of escape values

## Syntax

**7.3.2.2 Sequence parameter set RBSP synt****ax**

|  |  |
| --- | --- |
| seq\_parameter\_set\_rbsp( ) { | Descriptor |
| … |  |
| **sps\_extension\_present\_flag** | u(1) |
| if( sps\_extension\_present\_flag ) { |  |
| for( i = 0; i < 1; i++ ) |  |
| **sps\_extension\_flag**[ i ] | u(1) |
| **sps\_extension\_7bits** | u(7) |
| if( sps\_extension\_flag[ 0 ] ) { |  |
| **transform\_skip\_rotation\_enabled\_flag** | u(1) |
| **transform\_skip\_context\_enabled\_flag** | u(1) |
| **intra\_block\_copy\_enabled\_flag** | u(1) |
| **implicit\_rdpcm\_enabled\_flag** | u(1) |
| **explicit\_rdpcm\_enabled\_flag** | u(1) |
| **extended\_precision\_processing\_flag** | u(1) |
| **intra\_smoothing\_disabled\_flag** | u(1) |
| **high\_precision\_offsets\_enabled\_flag** | u(1) |
| **fast\_rice\_adaptation\_enabled\_flag** | u(1) |
| **cabac\_bypass\_alignment\_enabled\_flag** | u(1) |
| **palette\_enabled\_flag** | u(1) |
| } |  |
| if( sps\_extension\_7bits ) |  |
| while( more\_rbsp\_data( ) ) |  |
| **sps\_extension\_data\_flag** | u(1) |
| } |  |
| rbsp\_trailing\_bits( ) |  |
| } |  |

**7.3.8.5 Coding unit syntax**

|  |  |
| --- | --- |
| coding\_unit( x0, y0, log2CbSize ) { | Descriptor |
| … |  |
| else { |  |
| if( intra\_block\_copy\_enabled\_flag ) |  |
| **intra\_bc\_flag**[ x0 ][ y0 ] | ae(v) |
| if( slice\_type != I && !intra\_bc\_flag[ x0 ][ y0 ] ) |  |
| **pred\_mode\_flag** | ae(v) |
| if( palette\_enabled\_flag && CuPredMode[ x0 ][ y0 ] = = MODE\_INTRA ) |  |
| **palette\_mode\_flag**[ x0 ][ y0 ] | ae(v) |
| if( palette\_mode\_flag[ x0 ][ y0 ] ) |  |
| palette\_coding( x0, y0, nCbS, nCbS ) |  |
| else { |  |
| if( CuPredMode[ x0 ][ y0 ] != MODE\_INTRA | | intra\_bc\_flag[ x0 ][ y0 ] | |   log2CbSize = = MinCbLog2SizeY ) |  |
| **part\_mode** | ae(v) |
| if( CuPredMode[ x0 ][ y0 ] = = MODE\_INTRA ) { |  |
| if( PartMode = = PART\_2Nx2N && pcm\_enabled\_flag && !intra\_bc\_flag  log2CbSize >= Log2MinIpcmCbSizeY &&  log2CbSize <= Log2MaxIpcmCbSizeY ) |  |
| **pcm\_flag**[ x0 ][ y0 ] | ae(v) |
| … |  |
| if( !pcm\_flag[ x0 ][ y0 ] ) { |  |
| if( CuPredMode[ x0 ][ y0 ] != MODE\_INTRA &&   !( PartMode = = PART\_2Nx2N && merge\_flag[ x0 ][ y0 ] ) | |   CuPredMode[ x0 ][ y0 ] = = MODE\_INTRA && intra\_bc\_flag[ x0 ][ y0 ] ) |  |
| **rqt\_root\_cbf** | ae(v) |
| if( rqt\_root\_cbf ) { |  |
| … |  |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |

|  |  |
| --- | --- |
| palette\_coding(x0, y0, CbWidth, CbHeight) { | Descriptor |
| if( ChromaArrayType = = 3 ) { |  |
| palette\_coding\_component( x0, y0, CbWidth, CbHeight, 3 ) |  |
| } else if( ChromaArrayType = = 2 ) { |  |
| palette\_coding\_component( x0, y0, CbWidth, CbHeight, 1 ) |  |
| palette\_coding\_component( x0, y0, CbWidth >> 1, CbHeight, 2 ) |  |
| } else if( ChromaArrayType = = 1 ) { |  |
| palette\_coding\_component( x0, y0, CbWidth, CbHeight, 1 ) |  |
| palette\_coding\_component( x0, y0, CbWidth >> 1, CbHeight >> 1, 2 ) |  |
| } else if( ChromaArrayType = = 0 ) { |  |
| palette\_coding\_component( x0, y0, CbWidth, CbHeight, 1 ) |  |
| } |  |

|  |  |
| --- | --- |
| palette\_coding\_component( x0, y0, CbWidth, CbHeight, NumComp ) { | Descriptor |
| compOffset = ( NumComp = = 3 ) ? 0 : ( NumComp – 1 ) |  |
| nCbS = ( 1 << log2CbSize ) |  |
| numPredPreviousPalette = 0 |  |
| pred\_end\_flag = 0 |  |
| for( i = 0; i < previousPaletteSize && pred\_end\_flag != 0 && |  |
| numPredPreviousPalette < max\_palette\_size; i++ ) { |  |
| **previous\_palette\_entry\_flag**[ i ] | ae(v) |
| if ( previous\_palette\_entry\_flag[ i ] ) { |  |
| for ( cIdx = compOffset; cIdx < NumComp + compOffset; cIdx++ ) |  |
| palette\_entries[ cIdx ][ numPredPreviousPalette ] =   previousPaletteEntries[ cIdx ][ i ] |  |
| numPredPreviousPalette++ |  |
| if( i == 4 || ( i > 8 && i % 8 == 0 ) ) { |  |
| pred\_end\_flag | u(1) |
| if(pred\_end\_flag ) |  |
| for(; i < max\_pred\_size; i++) previous\_palette\_entry\_flag[ i ] = 0 |  |
| } |  |
| } |  |
| } |  |
| if( numPredPreviousPalette < max\_palette\_size) |  |
| **palette\_num\_signalled\_entries** | ae(v) |
| for ( cIdx = compOffset; cIdx < NumComp + compOffset; cIdx++ ) |  |
| for( i = 0; i < palette\_num\_signalled\_entries; i++ ) |  |
| **palette\_entries**[ cIdx ][ numPredPreviousPalette + i ] | ae(v) |
| palette\_size = numPredPreviousPalette + palette\_num\_signalled\_entries |  |
| previous\_run\_type\_flag = INDEX\_MODE |  |
| **palette\_limit\_run** | ae(v) |
| scanPos = 0 |  |
| last\_index = 1 |  |
| last\_transition = { 1, 0, } |  |
| while( scanPos < nCbS \* nCbS ) { |  |
| xC = scanPos % nCbS |  |
| yC = scanPos / nCbS |  |
|  |  |
|  |  |
|  |  |
| palette\_run\_type\_flag[ xC ][ yC ] |  |
| previous\_run\_type\_flag = palette\_run\_type\_flag[ xC ][ yC ] |  |
| if( palette\_run\_type\_flag[ xC ][ yC ] = = INDEX\_MODE ) { |  |
| coding\_size **=** palette\_size |  |
| if( coding\_size = = MAX\_PLT\_SIZE ) coding\_size++ |  |
| incVal = coding\_size |  |
| if( scanPos != 0 && paletteMap[ xC-1 ][ yC ] ! = MAX\_PLT\_SIZE ) { |  |
| if( palette\_run\_type\_flag[ xC-1 ][ yC ] ! = COPY\_ABOVE\_MODE &&  paletteMap[ xC-1 ][ yC ] <= palette\_limit\_run) { |  |
| incVal = paletteMap[ xC-1 ][ yC ] |  |
| coding\_size-- |  |
| } else if( yC > 0 && paletteMap[ xC ][ yC-1 ] ! = MAX\_PLT\_SIZE ) { |  |
| incVal = paletteMap[ xC-1 ][ yC ] |  |
| coding\_size-- |  |
| } |  |
| } |  |
| **truncated\_index** | ae(v) |
| if( truncated\_index > incVal ) truncated\_index++ |  |
| palette\_index = **truncated\_index** |  |
| if( palette\_index = = MAX\_PLT\_SIZE ) { /\* ESCAPE\_PIXEL \*/ |  |
|  |  |
|  |  |
| scanPos++ |  |
| for( cIdx = compOffset; cIdx < NumComp + compOffset; cIdx++ ) { |  |
| **palette\_escape\_val** | ae(v) |
| samples\_array[ cIdx ][ xC ][ yC ] = palette\_escape\_val |  |
| } |  |
| } |  |
| if( palette\_index != last\_index ) last\_transition[ last\_index ] == palette\_index |  |
| } |  |
| if( palette\_run\_type\_flag[ xC ][ yC ] = = COPY\_ABOVE\_MODE | |   palette\_run\_type\_flag[ xC ][ yC ] = = TRANSITION\_MODE | |   ( palette\_run\_type\_flag[ xC ][ yC ] = = INDEX\_MODE &&   palette\_index < palette\_size ) ) { |  |
| if( palette\_run\_type\_flag[ xC ][ yC ]  ! = COPY\_ABOVE\_MODE &&  palette\_index > palette\_limit\_run){ |  |
| palette\_run = 0 |  |
| else |  |
| **palette\_run** | ae(v) |
| runPos = 0 |  |
| while ( runPos <= palette\_run ) { |  |
| xC = scanPos % nCbS |  |
| yC = scanPos / nCbS |  |
| if( palette\_run\_type\_flag[ xC ][ yC ] = = INDEX\_MODE ) { |  |
| paletteMap[ xC ][ yC ] = palette\_index |  |
| } else if(palette\_run\_type\_flag[ xC ][ yC ] = = TRANSITION\_MODE ) { |  |
| paletteMap[ xC ][ yC ] = last\_transition[ last\_index ] |  |
| } else |  |
| paletteMap[ xC ][ yC ] = paletteMap[ xC ][ yC − 1 ] |  |
| for( cIdx = compOffset; cIdx < NumComp + compOffset; cIdx++ ) |  |
| samples\_array[ cIdx ][ xC ][ yC ] =   palette\_entries[ cIdx ][ paletteMap[ xC ][ yC ]] |  |
| runPos++ |  |
| scanPos++ |  |
| } |  |
| last\_index = paletteMap[ (scanPos-1) % nCbS ][ (scanPos-1) / nCbS ] |  |
| } |  |
| } |  |
| current\_size = palette\_size |  |
| for( i = 0; i < palette\_size; i++ ) { |  |
| for ( cIdx = compOffset; cIdx < NumComp + compOffset; cIdx++ ) |  |
| tempPaletteEntries[ cIdx ][ i ] = palette\_entries[ cIdx ][ i ] |  |
| } |  |
| } |  |
| for( i = 0; i < previousPaletteSize && current\_size < max\_pred\_size; i++ ) { |  |
| if(previous\_palette\_entry\_flag[ i ] == 0 ) { |  |
| for ( cIdx = compOffset; cIdx < NumComp + compOffset; cIdx++ ) |  |
| tempPaletteEntries[ cIdx ][ current\_size ] = previousPaletteEntries [ cIdx ][ i ] |  |
| current\_size++ |  |
| } |  |
| } |  |
| previousPaletteSize = current\_size |  |
| previousPaletteEntries = tempPaletteEntries |  |

7.4.3.2 Sequence parameter set RBSP semantics

**sps\_extension\_flag**[ 0 ] equal to 1 specifies that transform\_skip\_rotation\_enabled\_flag, transform\_skip\_context\_enabled\_flag, intra\_block\_copy\_enabled\_flag, palette\_enabled\_flag, implicit\_rdpcm\_enabled\_flag, explicit\_rdpcm\_enabled\_flag, extended\_precision\_processing\_flag, intra\_smoothing\_disabled\_flag, high\_precision\_offsets\_enabled\_flag, fast\_rice\_adaptation\_enabled\_flag, and cabac\_bypass\_alignment\_enabled\_flag are present in the SPS RBSP syntax structure. sps\_extension\_flag[ 0 ] equal to 0 specifies that these syntax elements are not present.

**palette\_enabled\_flag** equal to 1 specifies that the palette mode may be used for intra blocks. palette\_enabled\_flag equal to 0 specifies that the palette mode is not applied. When not present, the value of palette\_enabled\_flag is inferred to be equal to 0.

## Semantics

**palette\_mode\_flag**[ x0 ][ y0 ] equal to 1 specifies that the current coding unit is coded using the palette mode. palette\_mode\_flag[ x0 ][ y0 ] equal to 0 specifies that the current coding unit is not coded using the palette mode.

When palette\_mode\_flag[ x0 ][ y0 ] is not present, it is inferred to be equal to 0.

In the following sematics, the array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

**max\_pred\_size** is the maximum size of the predictor, 64.

**previous\_palette\_entry\_flag**[ i ]equal to 1 specifies that the palette entry from the palette predictor is copied. previous\_palette\_entry\_flag [ i ] equals to 0 specifies that the palette entry from the palette predictor is not copied.

previousPaletteSize is the size of the palette predictor made from the palette and palette predictor of the previous CU decoded in palette mode. The variable previousPaletteSize is set equal to 0 for the first decoded palette CU of each CTB line as well as slice.

**palette\_num\_signalled\_entries** specifies the number of entries in the palette that are explicitly signalled for the current coding unit.

When palette\_num\_signalled\_entries is not present, it is inferred to be equal to 0.

**palette\_entries**[ cIdx ][ i ]specifiesthe i-th element in the palette for the color component cIdx.

tempPaletteEntries is a temporary array to hold palette entries from both palette\_entries and previousPaletteEntries of the current CU. The array previousPaletteEntries is equal to the final content of the array tempPaletteEntries of the last CU decoded in palette mode.

**palette\_limit\_run:** specifies the minimum palette index that has no palette run coded for copy index mode.

**palette\_escape\_val** specifies the escape pixel value.

last\_index is the value of the last index in raster scan order, 1 initially when x0 and y0 are zero.

last\_transition is a table containing max\_palette\_size entries, initially containing all zeroes except for first value which is 1.

**palette\_run\_type\_flag** equal to 1 specifies thatthe decoding process is COPY\_ABOVE\_MODE where the decoded pixel value is equal to the pixel at the same location in the above row. palette\_run\_type\_flag equal to 0 specifies that the decoding process mode is INDEX\_MODE where the pixel’s palette index is coded in the bitstream. palette\_run\_type\_flag equal to 2 specifies that the decoding process mode is TRANSITION\_MODE where the pixel’s palette index is provided implicitly using last\_index and the transition table last\_transition.

When palette\_run\_type\_flag is not present, it is inferred to be equal to INDEX\_MODE.

**palette\_index** is an index to the palette entries. If palette\_index is equal to MAX\_PLT\_SIZE, the pixel is coded as ESCAPE\_PIXEL.

**truncated\_index** is a truncated binary syntax element taking into account the fact some values of **palette\_index** are not possible.

**palette\_run** the number of consecutive locations minus 1 with the same palette index as the position in the above row when palette\_run\_type\_flag is equal to COPY\_ABOVE\_MODE or represents the number of consecutive locations minus 1 with the same palette index when palette\_run\_type\_flag is equal to INDEX\_MODE.

**8.4.1** **General decoding process for coding units coded in intra prediction mode**

Depending on the values of pcm\_flag[ xCb ][ yCb ], palette\_mode\_flag[ xCb ][ yCb ] and IntraSplitFlag, the decoding process for luma samples is specified as follows:

– If pcm\_flag[ xCb ][ yCb ] is equal to 1, the reconstructed picture is modified as follows:

SL[ xCb + i ][ yCb + j ] =   
 pcm\_sample\_luma[ ( nCbS \* j ) + i ]  <<  ( BitDepthY − PcmBitDepthY ), with i, j = 0..nCbS − 1 (8‑12)

– Otherwise if (pcm\_flag[ xCb ][ yCb ] is equal to 0 and palette\_mode\_flag[ xCb ][ yCb ] is equal to 1), the decoding process is specified as follows:

For each i, j = 0..nCbS – 1,

SL[ xCb + i ][ yCb + j ] = sample\_array[ 0 ][ i ][ j ]

Otherwise (pcm\_flag[ xCb ][ yCb ] is equal to 0, palette\_mode\_flag[ xCb ][ yCb ] is equal to 0), if IntraSplitFlag is equal to 0, the following ordered steps apply:

1. When intra\_bc\_flag[ xCb ][ yCb ] is equal to 0, the derivation process for the intra prediction mode as specified in subclause 8.4.2 is invoked with the luma location ( xCb, yCb ) as input.
2. The general decoding process for intra blocks as specified in subclause 8.4.4.1 is invoked with the luma location ( xCb, yCb ), the variable log2TrafoSize set equal to log2CbSize, the variable trafoDepth set equal to 0, the variable predModeIntra set equal to IntraPredModeY[ xCb ][ yCb ], the variable predModeIntraBc set equal to intra\_bc\_flag[ xCb ][ yCb ], the variable bvIntra set equal to BvIntra[ xCb ][ yCb ], and the variable cIdx set equal to 0 as inputs, and the output is a modified reconstructed picture before deblocking filtering.

– Otherwise (pcm\_flag[ xCb ][ yCb ] is equal to 0, palette\_mode\_flag[ xCb ][ yCb ] is equal to 0 and IntraSplitFlag is equal to 1), for the variable blkIdx proceeding over the values 0..3, the following ordered steps apply:

1. The variable xPb is set equal to xCb + ( nCbS  >>  1 ) \* ( blkIdx % 2 ).
2. The variable yPb is set equal to yCb + ( nCbS  >>  1 ) \* ( blkIdx / 2 ).
3. The derivation process for the intra prediction mode as specified in subclause 8.4.2 is invoked with the luma location ( xPb, yPb ) as input.
4. The general decoding process for intra blocks as specified in subclause 8.4.4.1 is invoked with the luma location ( xPb, yPb ), the variable log2TrafoSize set equal to log2CbSize − 1, the variable trafoDepth set equal to 1, the variable predModeIntra set equal to IntraPredModeY[ xPb ][ yPb ], the variable predModeIntraBc set equal to intra\_bc\_flag[ xCb ][ yCb ], the variable bvIntra set equal to BvIntra[ xCb ][ yCb ], and the variable cIdx set equal to 0 as inputs, and the output is a modified reconstructed picture before deblocking filtering.

When ChromaArrayType is not equal to 0, the following applies.

The variable log2CbSizeC is set equal to log2CbSize − ( ChromaArrayType  = =  3 ? 0 : 1 ).

Depending on the value of pcm\_flag[ xCb ][ yCb ] and IntraSplitFlag, the decoding process for chroma samples is specified as follows:

– If pcm\_flag[ xCb ][ yCb ] is equal to 1, the reconstructed picture is modified as follows:

SCb[ xCb / SubWidthC + i ][ yCb / SubHeightC + j ] =  pcm\_sample\_chroma[ ( nCbS / SubWidthC \* j ) + i ]  <<  
 ( BitDepthC − PcmBitDepthC ), with i = 0..nCbS / SubWidthC − 1, and j = 0..nS / SubHeightC − 1 (8‑13)

SCr[ xCb / SubWidthC + i ][ yCb / SubHeightC + j ] = pcm\_sample\_chroma[ ( nCbS / SubWidthC \* ( j + nCbS / SubHeightC ) ) + i ]  <<  
 ( BitDepthC − PcmBitDepthC ), with i = 0..nCbS / SubWidthC − 1, and j = 0..nS / SubHeightC − 1 (8‑14)

– Otherwise (pcm\_flag[ xCb ][ yCb ] is equal to 0, palette\_mode\_flag[ xCb ][ yCb ] is equal to 1), the decoding process is specified as follows

For each i = 0..nCbS/ SubWidthC  – 1, j = 0..nCbS / SubHeightC – 1,

SCb[ xCb + i ][ yCb + j ] = sample\_array[ 1 ][ i ][ j ]

For each i = 0..nCbS/ SubWidthC  – 1, j = 0..nCbS / SubHeightC – 1,

SCr[ xCb + i ][ yCb + j ] = sample\_array[ 2 ][ i ][ j ]

Otherwise (pcm\_flag[ xCb ][ yCb ] is equal to 0, palette\_mode\_flag[ xCb ][ yCb ] is equal to 0), if IntraSplitFlag is equal to 0 or ChromaArrayType is not equal to 3, the following ordered steps apply:

1. When intra\_bc\_flag[ xCb ][ yCb ] is equal to 0, the derivation process for the chroma intra prediction mode as specified in 8.4.3 is invoked with the luma location ( xCb, yCb ) as input, and the output is the variable IntraPredModeC.
2. The general decoding process for intra blocks as specified in subclause 8.4.4.1 is invoked with the chroma location ( xCb / SubWidthC, yCb / SubHeightC ), the variable log2TrafoSize set equal to log2CbSizeC, the variable trafoDepth set equal to 0, the variable predModeIntra set equal to IntraPredModeC, the variable predModeIntraBc set equal to intra\_bc\_flag[ xCb ][ yCb ], the variable bvIntra set equal to BvIntra[ xCb ][ yCb ], and the variable cIdx set equal to 1 as inputs, and the output is a modified reconstructed picture before deblocking filtering.
3. The general decoding process for intra blocks as specified in subclause 8.4.4.1 is invoked with the chroma location ( xCb / SubWidthC, yCb / SubHeightC ), the variable log2TrafoSize set equal to log2CbSizeC, the variable trafoDepth set equal to 0, the variable predModeIntra set equal to IntraPredModeC, the variable predModeIntraBc set equal to intra\_bc\_flag[ xCb ][ yCb ], the variable bvIntra set equal to BvIntra[ xCb ][ yCb ], and the variable cIdx set equal to 2 as inputs, and the output is a modified reconstructed picture before deblocking filtering.

– Otherwise (pcm\_flag[ xC ][ yC ] is equal to 0 palette\_mode\_flag[ xCb ][ yCb ] is equal to 0 and IntraSplitFlag is equal to 1 and ChromaArrayType is equal to 3), for the variable blkIdx proceeding over the values 0..3, the following ordered steps apply:

…

**8.4.4.3 Scaling process for escape pixels**

Inputs to this process are:

– a value escValue specifying the sample value of the escape pixel within the current block,

– a variable cIdx specifying the colour component of the current block,

– a variable qP specifying the quantization parameter.

Output of this process is scaled escape pixel sample value escReco.

The variable escReco is derived as follows:

– If cIdx is equal to 0,

bdShift = BitDepthY + Log2( nTbS ) − 5

– Otherwise,

bdShift = BitDepthC + Log2( nTbS ) − 5

The list levelScale[ ] is specified as levelScale[ k ] = { 40, 45, 51, 57, 64, 72 } with k = 0..5.

For the derivation of the scaled escape pixel sample value escReco, the following applies:

– The scaling factor m[ x ][ y ] is derived as follows:

m[ x ][ y ] = 16

– The scaled escape pixel sample value escReco is derived as follows:

escReco = Clip3( −32768, 32767, ( ( escValue \* m[ x ][ y ] \*   
 levelScale[ qP%6 ]  <<  (qP / 6 ) ) + ( 1  <<  ( bdShift − 1 ) ) )  >>  bdShift )

Table 9-4 Association of ctxIdx and syntax elements for each initializationType in the initialization process

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Syntax structure** | **Syntax element** | **ctxTable** | **initType** | | |
| **0** | **1** | **2** |
| coding\_unit( ) | cu\_transquant\_bypass\_flag | Table 9‑8 | 0 | 1 | 2 |
| cu\_skip\_flag | Table 9‑9 |  | 0..2 | 3..5 |
| intra\_bc\_flag[ ][ ] | Table 9‑33 | 0..2 | 3..5 | 6..8 |
| palette\_mode\_flag[ ][ ] | Table 9-XX | 0..2 | 3..5 | 6..8 |
| pred\_mode\_flag | Table 9‑10 |  | 0 | 1 |
| part\_mode | Table 9‑11 | 0 | 1..4 | 5..8 |

| Table 9‑34 – Syntax elements and associated binarizations | | | |
| --- | --- | --- | --- |
| **Syntax structure** | **Syntax element** | **Binarization** | |
| **Process** | **Input parameters** |
| coding\_unit( ) | cu\_transquant\_bypass\_flag | FL | cMax = 1 |
| cu\_skip\_flag | FL | cMax = 1 |
| intra\_bc\_flag | FL | cMax = 1 |
| palette\_mode\_flag[ ][ ] | FL | cMax = 1 |
| pred\_mode\_flag | FL | cMax = 1 |
| part\_mode | 9.3.3.5 | ( xCb, yCb ) = ( x0, y0), log2CbSize |
| pcm\_flag[ ][ ] | FL | cMax = 1 |
| prev\_intra\_luma\_pred\_flag[ ][ ] | FL | cMax = 1 |
| mpm\_idx[ ][ ] | TR | cMax = 2, cRiceParam = 0 |
| rem\_intra\_luma\_pred\_mode[ ][ ] | FL | cMax = 31 |
| intra\_chroma\_pred\_mode[ ][ ] | 9.3.3.6 | - |
| rqt\_root\_cbf | FL | cMax = 1 |
| palette\_coding( ) | previous\_palette\_entry\_flag[] | FL | cMax = 1 |
| palette\_num\_signalled\_entries | TR | cMax = 31, cRiceParam = 0 |
| palette\_entries | FL | cMax = cIdx == 0 ? (1<<BitDepthY) – 1 : (1<<BitDepthC) – 1 |
| palette\_escape\_val | 9.3.3.10 | cIdx |
| palette\_run\_type\_flag | TU | cMax = ( yC != 0 && previous\_run\_type\_flag != COPY\_ABOVE\_MODE ) ? 2 : 1 |
| truncated\_index | TR | cMax = coding\_size, cRiceParam = (coding\_size>1)? log2(coding\_size):0 |
| palette\_limit\_run | TR | cMax = palette\_size-1, cRiceParam = 0 |
| palette\_run | 9.3.3.11 | - |

**Table 9-39 Assignment of ctxInc to syntax elements with context coded bins**

| **Syntax element** | **binIdx** | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **>= 5** |
| cu\_skip\_flag | 0,1,2 (subclause 9.3.4.2.2) | na | na | na | Na | na |
| pred\_mode\_flag | 0 | na | na | na | Na | na |
| palette\_mode\_flag | 0,1,2 (subclause 9.3.4.2.2) | na | na | na | Na | na |
| previous\_palette\_entry\_flag | bypass | na | na | na | na | na |
| palette\_num\_signalled\_entries | bypass | bypass | bypass | bypass | Bypass | bypass |
| palette\_entries | bypass | bypass | bypass | bypass | Bypass | bypass |
| palette\_escape\_val | cIdx | bypass | bypass | bypass | Bypass | bypass |
| palette\_run\_type\_flag | 0, 1, 2  (subclause 9.3.4.2.2) | 0, 1, 2 | na | na | Na | na |
| truncated\_index | bypass | bypass | bypass | bypass | Bypass | bypass |
| palette\_limit\_run | bypass | bypass | bypass | bypass | Bypass | bypass |
| palette\_run | 0 | 1 | 2 | bypass | Bypass | bypass |
| part\_mode log2CbSize = = MinCbLog2SizeY | 0 | 1 | 2 | bypass | Na | na |

Table 9‑40 – Specification of ctxInc using left and above syntax elements

|  |  |  |  |
| --- | --- | --- | --- |
| **Syntax element** | **condL** | **condA** | **ctxInc** |
| split\_cu\_flag[ x0 ][ y0 ] | CtDepth[ xNbL ][ yNbL ] > cqtDepth | CtDepth[ xNbA ][ yNbA ] > cqtDepth | ( condL  &&  availableL ) + ( condA  &&  availableA ) |
| cu\_skip\_flag[ x0 ][ y0 ] | cu\_skip\_flag[ xNbL ][ yNbL ] | cu\_skip\_flag[ xNbA ][ yNbA ] | ( condL  &&  availableL ) + ( condA  &&  availableA ) |
| intra\_bc\_flag[ x0 ][ y0 ] | intra\_bc\_flag[ xNbL ][ yNbL ] | intra\_bc\_flag[ xNbA ][ yNbA ] | ( condL  &&  availableL ) + ( condA  &&  availableA ) |
| palette\_mode\_flag[ x0 ][ y0 ] | palette\_mode\_flag [ xNbL ][ yNbL ] | palette\_mode\_flag[ xNbA ][ yNbA ] | ( condL  &&  availableL ) + ( condA  &&  availableA ) |
| palette\_run\_type\_flag [ x0 ][ y0 ] | palette\_run\_type\_flag [ xNbL ][ yNbL ] |  | ( condL  &&  availableL ) |

9.3.3.10 Binarization process for palette\_escape\_val

Input to this process is a request for a binarization for the syntax element palette\_escape\_val and color component index cIdx.

Output of this process is the binarization of palette\_escape\_val with FL and maximum parameter cMax.

The variable bitDepth is derived as follows:

bitDepth = ( cIdx = = 0 ) ? BitDepthY : BitDepthC

qp = ( cIdx = = 0 ) ? Qp′Y : ( ( cIdx = = 1 ) Qp′Cb ? Qp′Cr )

The variable numBins is derived as follows:

* A quantization step size parameter qStep is derived as follows:

qStep = ( qP= =0 ) ? Round( 2(qP - 4) / 6 ) : 1

* A maximum possible quantized value maxValue is derived as follows:

maxValue = Floor( ( 1<<bitDepth ) – 1 ) / Qstep )

* The number of bins numBins of the fixed length binarization codeword is derived as follows

while( maxValue ) {

maxValue = maxValue >> 1

numBins++

}

* The maximum parameter cMax for the fixed length binarization is derived as follows

cMax = ( 1 << numBins ) – 1

9.3.3.11 Binarization process for palette\_run

Input to this process is a request for a binarization for the syntax element palette\_run and color component index cIdx.

Output of this process is the binarization of palette\_run according to the following table.

|  |  |
| --- | --- |
| **palette\_run** | Codeword |
| 0 | ‘0’ |
| 1 | ‘10’ |
| 2 | ‘110’ |
| >2 | Prefix=‘111’, Suffix=TR code , cRiceParam = 3 |