**Syntax and semantics description for SCCE3.D2 (Independent Uniform Prediction mode)**

This section provides the changes to HEVC Screen Content Coding Draft Text 1 (JCTVC-R1005-v3) syntax and semantics for the independent uniform prediction mode.

**7.3.6 Slice segment header syntax**

**7.3.6.1 General slice segment header syntaxasd**

|  |  |
| --- | --- |
| slice\_segment\_header( ) { | Descriptor |
| … |  |
| if( sample\_adaptive\_offset\_enabled\_flag ) { |  |
| **slice\_sao\_luma\_flag** | u(1) |
| if( ChromaArrayType != 0 ) |  |
| **slice\_sao\_chroma\_flag** | u(1) |
| } |  |
| if( independent\_uniform\_pred\_enabled\_flag ) { | u(v) |
| **num\_uniform\_colors** |  |
| for( i = 0; i < num\_uniform\_colors; i++) |  |
| for( cIdx = 0; cIdx < ( ChromaArrayType != 0 ? 3 : 1 ); cIdx++ ) |  |
| **uniform\_colors**[ cIdx ][ i ] | ae(v) |
| } |  |
| if( slice\_type = = P | | slice\_type = = B ) { |  |
| **num\_ref\_idx\_active\_override\_flag** | u(1) |
| if( num\_ref\_idx\_active\_override\_flag ) { |  |
| **num\_ref\_idx\_l0\_active\_minus1** | ue(v) |
| if( slice\_type = = B ) |  |
| **num\_ref\_idx\_l1\_active\_minus1** | ue(v) |
| } |  |
| … |  |

**7.3.8.5 Coding unit syntax**

|  |  |
| --- | --- |
| coding\_unit( x0, y0, log2CbSize ) { | Descriptor |
| if( transquant\_bypass\_enabled\_flag ) |  |
| **cu\_transquant\_bypass\_flag** | ae(v) |
| if( slice\_type != I ) |  |
| **cu\_skip\_flag**[ x0 ][ y0 ] | ae(v) |
| nCbS = ( 1  <<  log2CbSize ) |  |
| if( cu\_skip\_flag[ x0 ][ y0 ] ) |  |
| prediction\_unit( x0, y0, nCbS, nCbS ) |  |
| else { |  |
| if( independent\_uniform\_pred\_enabled\_flag && num\_uniform\_colors > 0 ) |  |
| **independent\_uniform\_pred\_flag**[ x0 ][ y0 ] | ae(v) |
| if( independent\_uniform\_pred\_ flag && num\_uniform\_colors > 1 ) |  |
| **uniform\_color\_index**[ x0 ][ y0 ] | ae(v) |
| 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\_mode\_enabled\_flag && ChromaArrayType = = 3 &&   CuPredMode[ x0 ][ y0 ] = = MODE\_INTRA && !intra\_bc\_flag[ x0 ][ y0 ] ) |  |
| **palette\_mode\_flag**[ x0 ][ y0 ] | ae(v) |
| if( palette\_mode\_flag[ x0 ][ y0 ] ) |  |
| palette\_coding( x0, y0, 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 && !intra\_bc\_flag[ x0 ][ y0 ] ) { |  |
| if( PartMode = = PART\_2Nx2N && pcm\_enabled\_flag &&   log2CbSize >= Log2MinIpcmCbSizeY &&  log2CbSize <= Log2MaxIpcmCbSizeY ) |  |
| **pcm\_flag**[ x0 ][ y0 ] | ae(v) |
| if( pcm\_flag[ x0 ][ y0 ] ) { |  |
| while( !byte\_aligned( ) ) |  |
| **pcm\_alignment\_zero\_bit** | f(1) |
| pcm\_sample( x0, y0, log2CbSize ) |  |
| } else { |  |
| pbOffset = ( PartMode = = PART\_NxN ) ? ( nCbS / 2 ) : nCbS |  |
| for( j = 0; j < nCbS; j = j + pbOffset ) |  |
| for( i = 0; i < nCbS; i = i + pbOffset ) |  |
| **prev\_intra\_luma\_pred\_flag**[ x0 + i ][ y0 + j ] | ae(v) |
| for( j = 0; j < nCbS; j = j + pbOffset ) |  |
| for( i = 0; i < nCbS; i = i + pbOffset ) |  |
| if( prev\_intra\_luma\_pred\_flag[ x0 + i ][ y0 + j ] ) |  |
| **mpm\_idx**[ x0 + i ][ y0 + j ] | ae(v) |
| else |  |
| **rem\_intra\_luma\_pred\_mode**[ x0 + i ][ y0 + j ] | ae(v) |
| if( ChromaArrayType = = 3 ) |  |
| for( j = 0; j < nCbS; j = j + pbOffset ) |  |
| for( i = 0; i < nCbS; i = i + pbOffset ) |  |
| **intra\_chroma\_pred\_mode**[ x0 + i ][ y0 + j ] | ae(v) |
| else if( ChromaArrayType != 0 ) |  |
| **intra\_chroma\_pred\_mode**[ x0 ][ y0 ] | ae(v) |
| } |  |
| } else { |  |
| if( PartMode = = PART\_2Nx2N ) |  |
| prediction\_unit( x0, y0, nCbS, nCbS ) |  |
| else if( PartMode = = PART\_2NxN ) { |  |
| prediction\_unit( x0, y0, nCbS, nCbS / 2 ) |  |
| prediction\_unit( x0, y0 + ( nCbS / 2 ), nCbS, nCbS / 2 ) |  |
| } else if( PartMode = = PART\_Nx2N ) { |  |
| prediction\_unit( x0, y0, nCbS / 2, nCbS ) |  |
| prediction\_unit( x0 + ( nCbS / 2 ), y0, nCbS / 2, nCbS ) |  |
| } else if( PartMode = = PART\_2NxnU ) { |  |
| prediction\_unit( x0, y0, nCbS, nCbS / 4 ) |  |
| prediction\_unit( x0, y0 + ( nCbS / 4 ), nCbS, nCbS \* 3 / 4 ) |  |
| } else if( PartMode = = PART\_2NxnD ) { |  |
| prediction\_unit( x0, y0, nCbS, nCbS \* 3 / 4 ) |  |
| prediction\_unit( x0, y0 + ( nCbS \* 3 / 4 ), nCbS, nCbS / 4 ) |  |
| } else if( PartMode = = PART\_nLx2N ) { |  |
| prediction\_unit( x0, y0, nCbS / 4, nCbS ) |  |
| prediction\_unit( x0 + ( nCbS / 4 ), y0, nCbS \* 3 / 4, nCbS ) |  |
| } else if( PartMode = = PART\_nRx2N ) { |  |
| prediction\_unit( x0, y0, nCbS \* 3 / 4, nCbS ) |  |
| prediction\_unit( x0 + ( nCbS \* 3 / 4 ), y0, nCbS / 4, nCbS ) |  |
| } else { /\* PART\_NxN \*/ |  |
| prediction\_unit( x0, y0, nCbS / 2, nCbS / 2 ) |  |
| prediction\_unit( x0 + ( nCbS / 2 ), y0, nCbS / 2, nCbS / 2 ) |  |
| prediction\_unit( x0, y0 + ( nCbS / 2 ), nCbS / 2, nCbS / 2 ) |  |
| prediction\_unit( x0 + ( nCbS / 2 ), y0 + ( nCbS / 2 ), nCbS / 2, nCbS / 2 ) |  |
| } |  |
| } |  |
| } |  |
| 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 ] ) | | independent\_uniform\_pred\_flag[ x0 ][ y0 ]) |  |
| **rqt\_root\_cbf** | ae(v) |
| if( rqt\_root\_cbf ) { |  |
| if( residual\_adaptive\_colour\_transform\_enabled\_flag &&   ( CuPredMode[ x0 ][ y0 ] = = MODE\_INTER | | intra\_bc\_flag[ x0 ][ y0 ] | |   intra\_chroma\_pred\_mode[ x0 ][ y0 ] = = 4 ) ) |  |
| **cu\_residual\_act\_flag** |  |
| MaxTrafoDepth = ( CuPredMode[ x0 ][ y0 ] = = MODE\_INTRA ?   ( max\_transform\_hierarchy\_depth\_intra + IntraSplitFlag ) :   max\_transform\_hierarchy\_depth\_inter ) |  |
| transform\_tree( x0, y0, x0, y0, log2CbSize, 0, 0 ) |  |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |

**7.3.8.12 Transform unit syntax**

|  |  |
| --- | --- |
| transform\_unit( x0, y0, xBase, yBase, log2TrafoSize, trafoDepth, blkIdx ) { | Descriptor |
| log2TrafoSizeC = Max( 2, log2TrafoSize − ( ChromaArrayType = = 3 ? 0 : 1 ) ) |  |
| cbfDepthC = trafoDepth − ( ChromaArrayType != 3 && log2TrafoSize = = 2 ? 1 : 0 ) |  |
| xC = ( ChromaArrayType != 3 && log2TrafoSize = = 2 ) ? xBase : x0 |  |
| yC = ( ChromaArrayType != 3 && log2TrafoSize = = 2 ) ? yBase : y0 |  |
| cbfLuma = cbf\_luma[ x0 ][ y0 ][ trafoDepth ] |  |
| cbfChroma =  cbf\_cb[ xC ][ yC ][ cbfDepthC ] | |  cbf\_cr[ xC ][ yC ][ cbfDepthC ] | |  ( ChromaArrayType = = 2 &&  ( cbf\_cb[ xC ][ yC + ( 1 << log2TrafoSizeC ) ][ cbfDepthC ] | |  cbf\_cr[ xC ][ yC + ( 1 << log2TrafoSizeC ) ][ cbfDepthC ] ) ) |  |
| if( cbfLuma | | cbfChroma ) { |  |
| if( cu\_qp\_delta\_enabled\_flag && !IsCuQpDeltaCoded ) { |  |
| **cu\_qp\_delta\_abs** | ae(v) |
| if( cu\_qp\_delta\_abs ) |  |
| **cu\_qp\_delta\_sign\_flag** | ae(v) |
| } |  |
| if( cu\_chroma\_qp\_offset\_enabled\_flag && cbfChroma &&  !cu\_transquant\_bypass\_flag && !IsCuChromaQpOffsetCoded ) { |  |
| **cu\_chroma\_qp\_offset\_flag** | ae(v) |
| if( cu\_chroma\_qp\_offset\_flag && chroma\_qp\_offset\_list\_len\_minus1 > 0 ) |  |
| **cu\_chroma\_qp\_offset\_idx** | ae(v) |
| } |  |
| if( cbfLuma ) |  |
| residual\_coding( x0, y0, log2TrafoSize, 0 ) |  |
| if( log2TrafoSize > 2 | | ChromaArrayType = = 3 ) { |  |
| if( cross\_component\_prediction\_enabled\_flag && cbfLuma &&   ( CuPredMode[ x0 ][ y0 ] = = MODE\_INTER | | intra\_bc\_flag[ x0 ][ y0 ] | | independent\_uniform\_pred\_ flag [ x0 ][ y0 ] | |   intra\_chroma\_pred\_mode[ x0 ][ y0 ] = = 4 ) ) |  |
| cross\_comp\_pred( x0, y0, 0 ) |  |
| for( tIdx = 0; tIdx < ( ChromaArrayType = = 2 ? 2 : 1 ); tIdx++ ) |  |
| if( cbf\_cb[ x0 ][ y0 + ( tIdx << log2TrafoSizeC ) ][ trafoDepth ] ) |  |
| residual\_coding( x0, y0 + ( tIdx << log2TrafoSizeC ), log2TrafoSizeC, 1 ) |  |
| if( cross\_component\_prediction\_enabled\_flag && cbfLuma &&  ( CuPredMode[ x0 ][ y0 ] = = MODE\_INTER | | intra\_bc\_flag[ x0 ][ y0 ] | | independent\_uniform\_pred\_ flag [ x0 ][ y0 ] | |   intra\_chroma\_pred\_mode[ x0 ][ y0 ] = = 4 ) ) |  |
| cross\_comp\_pred( x0, y0, 1 ) |  |
| for( tIdx = 0; tIdx < ( ChromaArrayType = = 2 ? 2 : 1 ); tIdx++ ) |  |
| if( cbf\_cr[ x0 ][ y0 + ( tIdx << log2TrafoSizeC ) ][ trafoDepth ] ) |  |
| residual\_coding( x0, y0 + ( tIdx << log2TrafoSizeC ), log2TrafoSizeC, 2 ) |  |
| } else if( blkIdx = = 3 ) { |  |
| for( tIdx = 0; tIdx < ( ChromaArrayType = = 2 ? 2 : 1 ); tIdx++ ) |  |
| if( cbf\_cb[ xBase ][ yBase + ( tIdx << log2TrafoSizeC ) ][ trafoDepth − 1 ] ) |  |
| residual\_coding( xBase, yBase + ( tIdx << log2TrafoSizeC ), log2TrafoSize, 1 ) |  |
| for( tIdx = 0; tIdx < ( ChromaArrayType = = 2 ? 2 : 1 ); tIdx++ ) |  |
| if( cbf\_cr[ xBase ][ yBase + ( tIdx << log2TrafoSizeC ) ][ trafoDepth − 1 ] ) |  |
| residual\_coding( xBase, yBase + ( tIdx << log2TrafoSizeC ), log2TrafoSize, 2 ) |  |
| } |  |
| } |  |
| } |  |

**7.3.8.13 Residual coding syntax**

|  |  |
| --- | --- |
| residual\_coding( x0, y0, log2TrafoSize, cIdx ) { | Descriptor |
| if( transform\_skip\_enabled\_flag && !cu\_transquant\_bypass\_flag &&   ( log2TrafoSize <= Log2MaxTransformSkipSize ) ) |  |
| **transform\_skip\_flag**[ x0 ][ y0 ][ cIdx ] | ae(v) |
| if( ( CuPredMode[ x0 ][ y0 ] = = MODE\_INTER | | intra\_bc\_flag[ x0 ][ y0 ] | | independent\_uniform\_pred\_ flag [ x0 ][ y0 ]) &&   explicit\_rdpcm\_enabled\_flag && ( transform\_skip\_flag[ x0 ][ y0 ][ cIdx ] | |   cu\_transquant\_bypass\_flag ) ) { |  |
| **explicit\_rdpcm\_flag**[ x0 ][ y0 ][ cIdx ] | ae(v) |
| if( explicit\_rdpcm\_flag[ x0 ][ y0 ][ cIdx ] ) |  |
| **explicit\_rdpcm\_dir\_flag**[ x0 ][ y0 ][ cIdx ] | ae(v) |
| } |  |
| … |  |
| } |  |

**7.4.3.3.1 General picture parameter set RBSP semantics**

**constrained\_intra\_pred\_flag** equal to 0 specifies that intra prediction allows usage of residual data and decoded samples of neighbouring coding blocks coded using either intra or inter prediction modes. constrained\_intra\_pred\_flag equal to 1 specifies constrained intra prediction, in which case the general intra prediction process only uses residual data and decoded samples from neighbouring coding blocks coded using intra prediction modes. [Ed. : review changes in this para, introduced by intra\_bc]

NOTE 5 – Samples reconstructed from available intra blocks with intra\_bc\_flag equal to 1 or independent\_uniform\_pred\_ flag equal to 1 are considered as available for the purposes of constrained intra prediction. However, no restriction is placed on the origin of reference samples used by intra blocks with intra\_bc\_flag equal to 1. Encoders that employ constrained intra prediction for the purpose of error propagation control should, in regions where error propagation is to be controlled through the use of constrained intra prediction, ensure that intra blocks with intra\_bc\_flag equal to 1 only use samples from neighbouring coding blocks that are coded using intra prediction modes.

**7.4.7.1 General slice segment header semantics**

**num\_uniform\_colors** specifies the total number of colors that may be used by the independent uniform prediction mode in in the current slice. When num\_uniform\_colors is not present, it is inferred to be equal to 0.

**uniform\_colors**[ cIdx ][ i ] specifies the i-th uniform color value for component cIdx. When uniform\_colors[ cIdx ][ i ] is not present, it is inferred to be equal to 0.

**7.4.9.5 Coding unit semantics**

**independent\_uniform\_pred\_flag**[ x0 ][ y0 ] equal to 1 specifies that the current coding unit is coded using the independent uniform prediction mode. independent\_uniform\_pred\_flag[ x0 ][ y0 ] equal to 0 specifies that the current coding unit is coded according to intra\_bc\_flag. When not present, the value of independent\_uniform\_pred\_flag[ x0 ][ y0 ] is inferred to be 0. 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.

**uniform\_color\_index**[ x0 ][ y0 ] is an index to the color values that will be used in the independent uniform prediction mode for the current coding unit. When uniform\_color\_index[ x0 ][ y0 ] is not present, it is inferred to be equal to 0. 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.

**pred\_mode\_flag** equal to 0 specifies that the current coding unit is coded in inter prediction mode. pred\_mode\_flag equal to 1 specifies that the current coding unit is coded in intra prediction mode. The variable CuPredMode[ x ][ y ] is derived as follows for x = x0..x0 + nCbS − 1 and y = y0..y0 + nCbS − 1:

* If independent\_uniform\_pred\_flag[ x0 ][ y0 ] is equal to 1, CuPredMode[ x ][ y ] is inferred to be equal to MODE\_INTRA.
* Otherwise, if intra\_bc\_flag[ x0 ][ y0 ] is equal to 1, CuPredMode[ x ][ y ] is inferred to be equal to MODE\_INTRA.
* Otherwise, if intra\_bc\_flag[ x0 ][ y0 ] is equal to 0 and pred\_mode\_flag is equal to 0, CuPredMode[ x ][ y ] is set equal to MODE\_INTER.
* Otherwise (intra\_bc\_flag[ x0 ][ y0 ] is equal to 0 and pred\_mode\_flag is equal to 1), CuPredMode[ x ][ y ] is set equal to MODE\_INTRA.

[Ed. (GS): The string "pred\_mode\_flag" is a substring of some other syntax element names.]

[Ed. (GS): The indentation above is not aligned.]

When pred\_mode\_flag is not present, the variable CuPredMode[ x ][ y ] is derived as follows for x = x0..x0 + nCbS − 1 and y = y0..y0 + nCbS − 1:

* If slice\_type is equal to I, CuPredMode[ x ][ y ] is inferred to be equal to MODE\_INTRA.
* Otherwise (slice\_type is equal to P or B), when cu\_skip\_flag[ x0 ][ y0 ] is equal to 1, CuPredMode[ x ][ y ] is inferred to be equal to MODE\_SKIP.

**part\_mode** specifies partitioning mode of the current coding unit. The semantics of part\_mode depend on CuPredMode[ x0 ][ y0 ]. The variables PartMode and IntraSplitFlag are derived from the value of part\_mode as defined in Table 7-10.

The value of part\_mode is restricted as follows:

* If CuPredMode[ x0 ][ y0 ] is equal to MODE\_INTRA, the following applies:
  + If independent\_uniform\_pred\_flag[ x0 ][ y0 ] is equal to 1, part\_mode shall be equal to 0.
  + Otherwise, if intra\_bc\_flag[ x0 ][ y0 ] is equal to 1, part\_mode shall be in the range of 0 to 3, inclusive.
* Otherwise (intra\_bc\_flag[ x0 ][ y0] is equal to 0 and independent\_uniform\_pred\_flag[ x0 ][ y0 ] is equal to 0), part\_mode shall be equal to 0 or 1.
* Otherwise (CuPredMode[ x0 ][ y0 ] is equal to MODE\_INTER), the following applies:
* If log2CbSize is greater than MinCbLog2SizeY and amp\_enabled\_flag is equal to 1, part\_mode shall be in the range of 0 to 2, inclusive, or in the range of 4 to 7, inclusive.
* Otherwise, if log2CbSize is greater than MinCbLog2SizeY and amp\_enabled\_flag is equal to 0, or log2CbSize is equal to 3, part\_mode shall be in the range of 0 to 2, inclusive.
* Otherwise (log2CbSize is greater than 3 and less than or equal to MinCbLog2SizeY), the value of part\_mode shall be in the range of 0 to 3, inclusive.

When part\_mode is not present, the variables PartMode and IntraSplitFlag are derived as follows:

* PartMode is set equal to PART\_2Nx2N.
* IntraSplitFlag is set equal to 0.

**8.4 Decoding process for coding units coded in intra prediction mode**

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

Inputs to this process are:

– a luma location ( xCb, yCb ) specifying the top-left sample of the current luma coding block relative to the top‑left luma sample of the current picture,

– a variable log2CbSize specifying the size of the current luma coding block.

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

The derivation process for quantization parameters as specified in subclause 8.6.1 is invoked with the luma location ( xCb, yCb ) as input.

A variable nCbS is set equal to 1  <<  log2CbSize.

Depending on the values of pcm\_flag[ xCb ][ yCb ], palette\_mode\_flag[ xCb ][ yCb ], independent\_uniform\_pred\_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 (pcm\_flag[ xCb ][ yCb ] is equal to 0), if palette\_mode\_flag[ xCb ][ yCb ] is equal to 1, the following order steps apply:

1. The decoding process for palette intra blocks as specified in subclause 8.4.5.2.8 is invoked with the luma location ( xCb, yCb ), nCbS, the variable cIdx set equal to 0, the palette modes array palette\_mode, the palette indices array paletteMap, and the array of quantized escape values palette\_escape\_val as inputs, and the output is an nCbS x nCbS array of reconstructed -palette sample values, recSamples[ x ][ y ], x, y = 0..nCbS − 1
2. The reconstructed picture is modified as follows:

– If palette\_transpose\_flag is true,

SL[ yCb + y ][ xCb + x ] = palette\_sample\_values[ x ][ y ]

– Otherwise (palette\_transpose\_flag is false)

SL[ xCb + x ][ yCb + y ] = palette\_sample\_values[ x ][ y ].

– 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. 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. When intra\_bc\_flag[ xCb ][ yCb ] is equal to 1, the derivation process for block vector components in intra block copying prediction mode as specified in subclause 8.4.4 is invoked with the luma location ( xCb, yCb ) and variable log2CbSize as inputs, and the output being bvIntra.
3. If cu\_residual\_act\_flag[ xCb ][ yCb ] is equal to 1, the following applies:

– For the variable cIdx proceeding over the values 0..2, the following ordered steps apply:

* + The variable comp is set equal to (!cIdx ? L : (cIdx = =1 ? Cb : Cr). [Ed. (GJS): Suggest reformulating to avoid this usage formulation and this variable name.]
  + The general decoding process for intra blocks as specified in subclause 8.4.4.1 is invoked with the 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 predModeIntraIUP set equal to independent\_uniform\_pred\_flag[ xCb ][ yCb ], the variable bvIntra, the variable cIdx, and variable controlParaACT equal to 1 as inputs, and the output is the residual sample array resSamplescomp.

…

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 values of pcm\_flag[ xCb ][ yCb ], independent\_uniform\_pred\_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..nCbS / 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..nCbS / SubHeightC − 1 (8-14)

– Otherwise (pcm\_flag[ xCb ][ yCb ] is equal to 0), if palette\_mode\_flag[ xCb ][ yCb ] is equal to 1 the following orderd steps apply:

1. The decoding process for palette intra blocks as specified in subclause 8.4.5.2.8 is invoked with the chroma location ( xCb, yCb ), nCbS, the variable cIdx set equal to 1, the palette modes array palette\_mode, the palette indices array paletteMap, and the array of quantized escape values palette\_escape\_val as inputs, and the output is an nCbS x nCbS array of reconstructed palette sample values, recSamples[ x ][ y ], x, y = 0..nCbS − 1.
2. The reconstructed picture is modified as follows:

– If palette\_transpose\_flag is true, SCb[ yCb + y ][ xCb + x ] is set equal to palette\_sample\_values[ x ][ y ],

– Otherwise (palette\_transpose\_flag is false), SCb[ xCb + x ][ yCb + y ] is set equal to palette\_sample\_values[ x ][ y ].

1. The decoding process for palette intra blocks as specified in subclause 8.4.5.2.8 is invoked with the chroma location ( xCb, yCb ), nCbS, the variable cIdx set equal to 2, the palette modes array palette\_mode, the palette indices array paletteMap, and the array of quantized escape values palette\_escape\_val as inputs, and the output is an nCbS x nCbS array of reconstructed palette sample values, recSamples[ x ][ y ], x, y = 0..nCbS − 1.
2. The reconstructed picture is modified as follows:

– If palette\_transpose\_flag is true, SCr[ yCb + y ][ xCb + x ] is set equal to palette\_sample\_values[ x ][ y ]

– Otherwise (palette\_transpose\_flag is false), SCr[ xCb + x ][ yCb + y ] is set equal to palette\_sample\_values[ x ][ y ]

– 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 independent\_uniform\_pred\_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 predModeIntraIUP set equal to independent\_uniform\_pred\_flag[ xCb ][ yCb ], the variable cIdx set equal to 1, and variable controlParaACT set to ( cu\_residual\_act\_flag[ xCb ][ yCb ] ? 2 : 3 ) 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 predModeIntraIUP set equal to independent\_uniform\_pred\_flag[ xCb ][ yCb ], the variable cIdx set equal to 2, and variable controlParaACT set to ( cu\_residual\_act\_flag[ xCb ][ yCb ] ? 2 : 3 ) as inputs, and the output is a modified reconstructed picture before deblocking filtering.

…

**8.4.5 Decoding process for intra blocks**

**8.4.5.1 General decoding process for intra blocks**

Inputs to this process are:

– a sample location ( xTb0, yTb0 ) specifying the top-left sample of the current transform block relative to the top‑left sample of the current picture,

– a variable log2TrafoSize specifying the size of the current transform block,

– a variable trafoDepth specifying the hierarchy depth of the current block relative to the coding unit,

– a variable predModeIntra specifying the intra prediction mode,

– a variable predModeIntraBc specifying the intra block copying mode,

– a variable predModeIntraIUP specifying the independent uniform prediction mode,

– a variable bvIntra specifying the intra block copying vector,

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

– a variable controlParaACT specifying the applicable processes. [Ed. (GJS): Check/fix variable name convention usage regarding strings of capital letters.]

Output of this process is a modified reconstructed picture before deblocking filtering when controlParaACT is not equal to 1, or residual sample array when controlParaACT is equal to 1.

…

1. When controlParaACT is not equal to 1, depending upon the value of predModeIntraBc and predModeIntraIUP, the following applies:

– When predModeIntraBc is equal to 0 and predModeIntraIUP is equal to 0, the general intra sample prediction process as specified in subclause 8.4.4.2.1 is invoked with the transform block location ( xTb0, yTb0 + yTbOffset ), the intra prediction mode predModeIntra, the transform block size nTbS, and the variable cIdx as inputs, and the output is an (nTbS)x(nTbS) array predSamples.

– Otherwise if predModeIntraIUP is equal to 0 and~~(~~predModeIntraBc is equal to 1~~)~~, then the intra block copying process as specified in subclause 8.4.4.2.7 is invoked with the transform block location ( xTb0, yTb0 + yTbOffset ), the transform block size nTbS, the variable trafoDepth, the variable bvIntra, and the variable cIdx as inputs, and the output is an (nTbS)x(nTbS) array predSamples.

– Otherwise (predModeIntraIUP is equal to 1 and predModeIntraBc is equal to 0), the independent uniform prediction process as specified in subclause 8.4.5.2.8 is invoked with the transform block location ( xTb0, yTb0 + yTbOffset ), the transform block size nTbS, the variable trafoDepth, the variable bvIntra, and the variable cIdx as inputs, and the output is an (nTbS)x(nTbS) array predSamples.

1. When controlParaACT is not equal to 2, the scaling and transformation process as specified in subclause 8.6.2 is invoked with the luma location ( xTbY, yTbY + yTbOffsetY ), the variable trafoDepth, the variable cIdx, and the transform size trafoSize set equal to nTbS as inputs, and the output is an (nTbS)x(nTbS) array resSamples.
2. When controlParaACT is not equal to 2 and residualDpcm is equal to 1, the directional residual modification process for blocks using a transform bypass as specified in subclause 8.6.5 is invoked with the variable mDir set equal to predModeIntra / 26, the variable nTbS, and the (nTbS)x(nTbS) array r set equal to the array resSamples as inputs, and the output is a modified (nTbS)x(nTbS) array resSamples.
3. When controlParaACT is not equal to 2 and cross\_component\_prediction\_enabled\_flag is equal to 1, ChromaArrayType is equal to 3, and cIdx is not equal to 0, the residual modification process for transform blocks using cross-component prediction as specified in subclause 8.6.6 is invoked with the current luma transform block location ( xTbY, yTbY ), the variable nTbS, the variable cIdx, the (nTbS)x(nTbS) array rY set equal to the corresponding luma residual sample array resSamples of the current transform block, and the (nTbS)x(nTbS) array r set equal to the array resSamples as inputs, and the output is a modified (nTbS)x(nTbS) array resSamples.
4. When controlParaACT is not equal to 1, the picture construction process prior to in-loop filtering for a colour component as specified in subclause 8.6.7 is invoked with the transform block location ( xTb0, yTb0 + yTbOffset ), the variables nCurrSw and nCurrSh both set equal to nTbS, the variable cIdx, the (nTbS)x(nTbS) array predSamples, and the (nTbS)x(nTbS) array resSamples as inputs.

**8.4.5.2.8 Specification of independent uniform prediction mode**

Inputs to this process are:

– a sample location ( xTb0, yTb0 ) specifying the top-left sample of the current transform block relative to the top left sample of the current picture,

– a variable nTbS specifying the transform block size,

– a variable trafoDepth specifying the hierarchy depth of the current block relative to the coding unit,

– a 2-D variable uniform\_colors specifying the uniform colors,

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

– a variable uniform\_color\_index specifying the index of selected uniform color for the current block.

Output of this process is the predicted samples predSamples[ x ][ y ], with x, y = 0..nTbS − 1.

predSamples[ x ][ y ]= uniform\_colors[cIdx] [uniform\_color\_index[xTb0, yTb0]]

**8.6.2 Scaling and transformation process**

…

The variable rotateCoeffs is derived as follows:

– If all of the following conditions are true, rotateCoeffs is set equal to 1:

– transform\_skip\_rotation\_enabled\_flag is equal to 1

– nTbS is equal to 4

– CuPredMode[ xTbY ][ yTbY ] is equal to MODE\_INTRA

– independent\_uniform\_pred\_flag [ xTbY ][ yTbY ] is equal to 0

– Otherwise, rotateCoeffs is set equal to 0

…

**9.3.2.2 Initialization process for context variables**

**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** |
| sao( ) | sao\_merge\_left\_flag sao\_merge\_up\_flag | Table 9-5 | 0 | 1 | 2 |
| sao\_type\_idx\_luma sao\_type\_idx\_chroma | Table 9-6 | 0 | 1 | 2 |
| coding\_quadtree( ) | split\_cu\_flag[ ][ ] | Table 9-7 | 0..2 | 3..5 | 6..8 |
| coding\_unit( ) | cu\_transquant\_bypass\_flag | Table 9-8 | 0 | 1 | 2 |
| cu\_skip\_flag | Table 9-9 |  | 0..2 | 3..5 |
| independent\_uniform\_pred\_flag | Table 9-X1 | 0 | 1 | 2 |
| uniform\_color\_index | Table 9-X2 | 0 | 1 | 2 |
| intra\_bc\_flag[ ][ ] | Table 9-38 | 0 | 1 | 2 |
| palette\_mode\_flag[ ][ ] | Table 9-39 | 0 | 1 | 2 |
| pred\_mode\_flag | Table 9-10 |  | 0 | 1 |
| part\_mode | Table 9-11 | 0 | 1..4 | 5..8 |
| prev\_intra\_luma\_pred\_flag[ ][ ] | Table 9-12 | 0 | 1 | 2 |
| intra\_chroma\_pred\_mode[ ][ ] | Table 9-13 | 0 | 1 | 2 |
| rqt\_root\_cbf | Table 9-14 |  | 0 | 1 |
| cu\_residual\_act\_flag | Table 9-40 | 0 | 1 | 2 |
| … |  |  |  |  |  |

**Table 9‑X1 – Values of initValue for ctxIdx of independent\_uniform\_pred\_flag**

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of independent\_uniform\_pred\_flag** | | |
| **0** | **1** | **2** |
| **initValue** | 185 | 197 | 197 |

**Table 9‑X2 – Values of initValue for ctxIdx of uniform\_color\_index**

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of uniform\_color\_index** | | |
| **0** | **1** | **2** |
| **initValue** | 185 | 197 | 197 |

**9.3.3 Binarization process**

**9.3.3.1 General**

**Table 9-38 – Syntax elements and associated binarizations**

| **Syntax structure** | **Syntax element** | **Binarization** | |
| --- | --- | --- | --- |
| **Process** | **Input parameters** |
| coding\_unit( ) | … | … | … |
| independent\_uniform\_pred\_flag | FL | cMax = 1 |
| uniform\_color\_index | TR | cMax = num\_uniform\_colors, cRiceParam=0 |
| … | … | … |

**Table 9-40 – Binarization for part\_mode**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **CuPredMode** [ xCb ][ yCb ] | **independent\_uniform\_pred\_flag**  **[ xCb ][ yCb ]** | **intra\_bc\_flag** [ xCb ][ yCb ] | **part\_mode** | **PartMode** | **Bin string** | | | |
| log2CbSize >  MinCbLog2SizeY | | log2CbSize  = =  MinCbLog2SizeY | |
| !amp\_enabled\_flag | amp\_enabled\_flag | log2CbSize  = =  3 | log2CbSize > 3 |
| MODE\_INTRA | 0 | 0 | 0 | PART\_2Nx2N | - | - | 1 | 1 |
| 1 | PART\_NxN | - | - | 0 | 0 |
| 1 | 0 | PART\_2Nx2N | 1 | 1 | 1 | 1 |
| 1 | PART\_2NxN | 01 | 01 | 01 | 01 |
| 2 | PART\_Nx2N | 00 | 00 | 001 | 001 |
| 3 | PART\_NxN | - | - | 000 | 000 |
| 1 | - | 0 | PART\_2Nx2N | - | - | - | - |
| MODE\_INTER |  |  | 0 | PART\_2Nx2N | 1 | 1 | 1 | 1 |
|  |  | 1 | PART\_2NxN | 01 | 011 | 01 | 01 |
|  |  | 2 | PART\_Nx2N | 00 | 001 | 00 | 001 |
|  |  | 3 | PART\_NxN | - | - | - | 000 |
|  |  | 4 | PART\_2NxnU | - | 0100 | - | - |
|  |  | 5 | PART\_2NxnD | - | 0101 | - | - |
|  |  | 6 | PART\_nLx2N | - | 0000 | - | - |
|  |  | 7 | PART\_nRx2N | - | 0001 | - | - |

**9.3.4.2 Derivation process for ctxTable, ctxIdx and bypassFlag**

**9.3.4.2.1 General**

**Table 9-43 – Assignment of ctxInc to syntax elements with context coded bins**

| **Syntax element** | **binIdx** | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **>= 5** |
| … |  |  |  |  |  |  |
| Independent\_uniform\_pred\_flag | 0 | na | na | na | na | na |
| uniform\_color\_index | 0 | bypass | bypass | bypass | bypass | bypass |
| … |  |  |  |  |  |  |