|  |  |
| --- | --- |
| slice\_segment\_header( ) { | Descriptor |
| … |  |
| **slice\_single\_sample\_mode\_enable\_flag** | ae(v) |
| if( slice\_segment\_header\_extension\_present\_flag ) { |  |
| **slice\_segment\_header\_extension\_length** | ue(v) |
| for( i = 0; i < slice\_segment\_header\_extension\_length; i++) |  |
| **slice\_segment\_header\_extension\_data\_byte**[ i ] | u(8) |
| } |  |
| byte\_alignment( ) |  |
| } |  |

7.4.7.1 General slice segment header semantics

**slice\_ single\_sample\_mode\_enable\_flag** specifies whether single sample mode can be used for intra mode coding. If slice\_single\_sample\_mode\_enabled\_flag is equal to 0, the syntax elements of the current picture shall be constrained such that no single sample mode is used in decoding of the current picture. Otherwise (slice\_single\_sample\_mode\_enabled\_flag is equal to 1), single sample mode may be used in decoding of the current picture. When not present, the value of slice\_single\_sample\_mode\_enabled\_flag is inferred to be equal to 0.

|  |  |
| --- | --- |
| 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) |
| if( !cu\_skip\_flag[ x0 ][ y0 ] ) |  |
| **single\_sample\_flag**[ x0 ][ y0 ] | ae(v) |
| If( !cu\_skip\_flag[ x0 ][ y0 ] ) { |  |
| **single\_sample\_idx**[ x0 ][ y0 ] | ae(v) |
| } else { |  |
| nCbS = ( 1  <<  log2CbSize ) |  |
| if( cu\_skip\_flag[ x0 ][ y0 ] ) |  |
| prediction\_unit( x0, y0, nCbS, nCbS ) |  |
| 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( 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[ x0 ][ y0 ] &&   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 if( intra\_bc\_flag[ x0 ][ y0 ] ) { |  |
| mvd\_coding( x0, y0, 2) |  |
| if( PartMode = = PART\_2NxN ) |  |
| mvd\_coding( x0, y0 + ( nCbS / 2 ), 2) |  |
| else if( PartMode = = PART\_Nx2N ) |  |
| mvd\_coding( x0 + ( nCbS / 2 ), y0, 2) |  |
| else if( PartMode = = PART\_NxN ) { |  |
| mvd\_coding( x0 + ( nCbS / 2 ), y0, 2) |  |
| mvd\_coding( x0, y0 + ( nCbS / 2 ), 2) |  |
| mvd\_coding( x0 + ( nCbS / 2 ), y0 + ( nCbS / 2 ), 2) |  |
| } |  |
| } 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 ] ) ) |  |
| **rqt\_root\_cbf** | ae(v) |
| if( rqt\_root\_cbf ) { |  |
| 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 ) |  |
| } |  |
| } |  |
| } |  |
| }// If( !cu\_skip\_flag[ x0 ][ y0 ] |  |
| } |  |

**single\_sample\_flag** [ x0 ][ y0 ] equal to 1 specifies that for the current coding unit, no more syntax elements except the single sample candidate index single\_sample\_idx[ x0 ][ y0 ] are parsed after single\_sample\_flag[ x0 ][ y0 ]. single\_sample\_flag[ x0 ][ y0 ] equal to 0 specifies that the coding unit is not single sample coded. 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.

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

**single\_sample\_idx**[ x0 ][ y0 ] specifies the single sample candidate index of the single sample candidate list where 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.

When single\_sample\_idx[ x0 ][ y0 ] is not present, it is inferred to be equal to 0.

#### I.8.4.4.1 General decoding process for intra blocks

1. 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 cIdx specifying the colour component of the current block.

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

The luma sample location ( xTbY, yTbY ) specifying the top-left sample of the current luma transform block relative to the top-left luma sample of the current picture is derived as follows:

* 1. ( xTbY, yTbY ) = ( cIdx  = =  0 ) ? ( xTb0, yTb0 ) : ( xTb0  <<  1, yTb0  <<  1 ) (I‑57)

The variable splitFlag is derived as follows:

* If cIdx is equal to 0, splitFlag is set equal to split\_transform\_flag[ xTbY ][ yTbY ][ trafoDepth ].
* Otherwise, if all of the following conditions are true, splitFlag is set equal to 1.
  + cIdx is greater than 0
  + split\_transform\_flag[ xTbY ][ yTbY ][ trafoDepth ] is equal to 1
  + log2TrafoSize is greater than 2
* Otherwise, splitFlag is set equal to 0.

Depending on the value of splitFlag, the following applies:

* If splitFlag is equal to 1, the following ordered steps apply:
  1. The variables xTb1 and yTb1 are derived as follows:
     + The variable xTb1 is set equal to xTb0 + ( 1  <<  ( log2TrafoSize − 1 ) ).
     + The variable yTb1 is set equal to yTb0 + ( 1  <<  ( log2TrafoSize − 1 ) ).
  2. The general decoding process for intra blocks as specified in this subclause is invoked with the location ( xTb0, yTb0 ), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the intra prediction mode predModeIntra, and the variable cIdx as inputs, and the output is a modified reconstructed picture before deblocking filtering.
  3. The general decoding process for intra blocks as specified in this subclause is invoked with the location ( xTb1, yTb0 ), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the intra prediction mode predModeIntra, and the variable cIdx as inputs, and the output is a modified reconstructed picture before deblocking filtering.
  4. The general decoding process for intra blocks as specified in this subclause is invoked with the location ( xTb0, yTb1 ), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the intra prediction mode predModeIntra, and the variable cIdx as inputs, and the output is a modified reconstructed picture before deblocking filtering.
  5. The general decoding process for intra blocks as specified in this subclause is invoked with the location ( xTb1, yTb1 ), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the intra prediction mode predModeIntra, and the variable cIdx as inputs, and the output is a modified reconstructed picture before deblocking filtering.
* Otherwise (splitFlag is equal to 0), the following ordered steps apply:
  1. The variable nTbS is set equal to 1  <<  log2TrafoSize.
  2. Depending on sdc\_flag[  xTb0 ][ xTb0 ] and **single\_sample\_flag** [ x0 ][ y0 ], the following applies:
     + If sdc\_flag[ xTb0 ][ yTb0 ] is equal to 0 and **single\_sample\_flag** [ x0 ][ y0 ] is equal to 0, following applies:
       - The general intra sample prediction process as specified in subclause 8.4.4.2.1 is invoked with the transform block location ( xTb0, yTb0 ), 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.
       - The scaling and transformation process as specified in subclause 8.6.2 is invoked with the luma location ( xTbY, yTbY ), 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.

[Ed. (GT): Meeting notes say: "Implement an enabling flag at (position t.b.d.)". However there seems to be no decision on the position yet.]

* + - * If DltFlag[ nuh\_layer\_id ] is equal to 1 and predModeIntra is equal to INTRA\_DC, INTRA\_ANGULAR10, INTRA\_ANGULAR26, INTRA\_DMM\_WFULL, INTRA\_DMM\_CPREDTEX, the following applies, for i in the range of 0 to nTbS − 1, inclusive, and j in the range of 0 to nTbS − 1, inclusive:

idx = DepthValue2Idx[ predSamples[ i ][ j ] ] + resSamples[ i ][ j ] (I‑58)

SL[ xTb0 + i ][ yTb0 + j ] =   
 Idx2DepthValue[ clip3( 0, NumDepthValuesInDlt[ nuh\_layer\_id ] − 1, idx ) ] (I‑59)

* + - * Otherwise, the following applies:
        + The picture reconstruction process prior to in-loop filtering for a colour component as specified in subclause 8.6.5 is invoked with the transform block location ( xTb0, yTb0 ), the transform block size nTbS, the variable cIdx, the (nTbS)x(nTbS) array predSamples, and the (nTbS)x(nTbS) array resSamples as inputs.
    - Otherwise if (sdc\_flag[ xTb0 ][ yTb0 ] is equal to 1) the following ordered steps apply:
    - The segmental depth intra coding process as specified in subclause I.8.4.4.3 is invoked with the location ( xTb0, yTb0 ), the transform size trafoSize set equal to nTbS, and the intra prediction mode predModeIntra, as inputs.
    - Otherwise if (**single\_sample\_flag** [ x0 ][ y0 ] is equal to 1) the following ordered steps apply:
      * the single sample prediction process as specified in subclause I.8.x.x.x is invoked with the location ( xTb0, yTb0 ), the transform size trafoSize set equal to nTbS, and the intra prediction mode predModeIntra, as inputs.

##### I.8.x.x.x. Specification of single color prediction mode

Inputs to this process are:

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

– the neighbouring samples p[ x ][ y ], with x = −1, y = −1..nTbS \* 2 − 1 and x = 0..nTbS \* 2 − 1, y = −1,

– a variable nTbS specifying the transform block size,

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

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

The nTbS \* 2 + 1 neighbouring samples p[ x ][ y ] that are constructed samples prior to the deblocking filter process, with x = −1, y = −1..nTbS − 1 and x = 0..nTbS  − 1, y = −1, are derived as follows:

– The neighbouring location (xNb, yNb ) is specified by:

( xNb, yNb ) = ( xCurr + x, yCurr + y ) (8‑27)

* The current sample location ( xTmp, yTmp ) and the neighbouring luma location (xNb, yNb ) are derived as follows:

( xCurr, yCurr ) = ( cIdx  = =  0 ) ? ( xCurr, yCurr ) : ( xCurr \* SubWidthC, yCurr  \* SubHeightC ) (8‑28)

( xNb, yNb) = ( cIdx  = =  0 ) ? ( xNb, yNb) : ( xNb \* SubWidthC, yNb \* SubHeightC ) (8‑29)

The availability derivation process for a block in z-scan order as specified in subclause  is invoked with the current luma location ( xCurr, yCurr ) and the neighbouring luma location ( xNb, yNb ) as inputs, and the output is assigned to availableN.

Each sample p[ x ][ y ] is derived as follows:

* If the variable availableN is equal to FALSE, the sample p[ x ][ y ] is marked as "not available for intra prediction".
* Otherwise, the sample p[ x ][ y ] is derived as the sample at the location ( xNb, yNb).

availableFlagAn, availableFlagBn, availableFlagA1, availableFlagB1, availableFlagBn+1 are set equal to TRUE.

predSampleAn is set equal to p[ −1 ][ 0 ].

predSampleBn is set equal to p[ 0 ][ −1  ].

predSampleA1 is set equal to p[ −1 ][ nTbS −1 ].

predSampleB1 is set equal to p[ nTbS −1 ][ −1 ].

predSampleBn+1 is set equal to p[ −1 ][ −1 ].

When one or more of the following conditions are true, availableFlagAn is set equal to FALSE:

* p[ −1 ][ 0 ] is marked as "not available for intra prediction".

When one or more of the following conditions are true, availableFlagBn is set equal to FALSE:

* p[ 0 ][ −1 ] is marked as "not available for intra prediction".
* availableFlagAn is equal to TRUE and predSampleBn is equal to predSampleAn.

When one or more of the following conditions are true, availableFlagB1 is set equal to FALSE:

* p[ nTbS −1 ][ −1 ] is marked as "not available for intra prediction".
* availableFlagAn is equal to TRUE and predSampleB1 is equal to predSampleAn.
* availableFlagBn is equal to TRUE and predSampleB1 is equal to predSampleBn.

When one or more of the following conditions are true, availableFlagA1 is set equal to FALSE:

* p[ −1 ][ nTbS −1 ] is marked as "not available for intra prediction".
* availableFlagAn is equal to TRUE and predSampleA1 is equal to predSampleAn.
* availableFlagBn is equal to TRUE and predSampleA1 is equal to predSampleBn.
* availableFlagB1 is equal to TRUE and predSampleA1 is equal to predSampleB1.

When one or more of the following conditions are true, availableFlagBn+1 is set equal to FALSE:

* p[ −1 ][ −1 ] is marked as "not available for intra prediction".
* availableFlagAn is equal to TRUE and predSample Bn+1 is equal to predSampleAn.
* availableFlagBn is equal to TRUE and predSample Bn+1 is equal to predSampleBn.
* availableFlagB1 is equal to TRUE and predSample Bn+1 is equal to predSampleB1.
* availableFlagA1 is equal to TRUE and predSample Bn+1 is equal to predSampleA1.

The sample candidate list, sampleCandList, is constructed as follows:

i = 0  
if( availableFlagAn )  
sampleCandList[ i++ ] = A1  
if( availableFlagBn )  
sampleCandList[ i++ ] = B1if( availableFlagB1 )  
sampleCandList[ i++ ] = B0 (8‑91)if( availableFlagA1 )  
sampleCandList[ i++ ] = A0if( availableFlagBn+1 )  
sampleCandList[ i++ ] = B2

numSampleMergeCand is set equal to i.

When numSampleMergeCand is equal to 0, sampleCandList[ numSampleMergeCand ++ ] = 1  <<  ( bitDepth − 1 )

When numSampleMergeCand is less than MaxNumSampleCand, the following steps are repeated until numSampleMergeCand is equal to MaxNumSampleCand:

i=1;

While (numSampleMergeCand < MaxNumSampleCand)

{

sampleCandList[ numSampleMergeCand ++ ] = sampleCandList[ 0 ] + Offset \* i

i++

}

The values of the prediction samples predSamples[ x ][ y ], with x, y = 0..nTbS − 1 are derived as follows:

predSamples[ x ][ y ] = sampleCandList[ single\_sample\_idx]