|  |  |
| --- | --- |
| slice\_segment\_header( ) { | Descriptor |
| … |  |
| **slice\_bi\_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\_ bi \_sample\_mode\_enable\_flag** specifies whether bi-color mode can be used for intra mode coding. If slice\_bi\_sample\_mode\_enabled\_flag is equal to 0, the syntax elements of the current picture shall be constrained such that no bi-color mode is used in decoding of the current picture. Otherwise (slice\_bi\_sample\_mode\_enabled\_flag is equal to 1), bi-color mode may be used in decoding of the current picture. When not present, the value of slice\_bi\_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) |
| nCbS = ( 1  <<  log2CbSize ) |  |
| if( cu\_skip\_flag[ x0 ][ y0 ] ) |  |
| prediction\_unit( x0, y0, nCbS, nCbS ) |  |
| else { |  |
| **bi\_sample\_flag**[ x0 ][ y0 ] | ae(v) |
| if(**bi\_sample\_flag** [ x0 ][ y0 ] ) { |  |
| **bi\_sample\_idx**[ x0 ][ y0 ] | ae(v) |
| previous\_run\_type\_flag = INDEX\_MODE |  |
| scanPos = 0 |  |
| while( scanPos < nCbS \* nCbS ) { |  |
| xC = scanPos % nCbS |  |
| yC = scanPos / nCbS |  |
| if ( yC != 0 && previous\_run\_type\_flag != COPY\_ABOVE\_MODE ) |  |
| **palette\_run\_type\_flag**[ xC ][ yC ] | ae(v) |
| else |  |
| palette\_run\_type\_flag[ xC ][ yC ] = INDEX\_MODE |  |
| previous\_run\_type\_flag = palette\_run\_type\_flag[ xC ][ yC ] |  |
| if( scanPos == 0 ) |  |
| **palette\_index** | ae(v) |
| else if( palette\_run\_type\_flag[ xC ][ yC ] = = INDEX\_MODE ) |  |
| palette\_index = ( palette\_run\_type\_flag[ xC-1 ][ yC ] = = INDEX\_MODE )? |  |
| ( palette\_index\_Prev[xC-1][yC] + 1 ) % 2 : |  |
| ( palette\_index\_Prev[xC-nCbS][yC] + 1 ) % 2 |  |
| palette\_index\_Prev[xC][yC] = palette\_index |  |
| **palette\_run** | ae(v) |
| runPos = 0 |  |
| while ( runPos <= palette\_run ) { |  |
| if( palette\_run\_type\_flag[ xC ][ yC ] = = INDEX\_MODE ) |  |
| paletteMap[ xC ][ yC ] = palette\_index |  |
| else |  |
| paletteMap[ xC ][ yC ] = paletteMap[ xC ][ yC − 1 ] |  |
| runPos++ |  |
| scanPos++ |  |
| xC = scanPos % nCbS |  |
| yC = scanPos / 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( 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[ 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( palette\_mode\_flag[ x0 ][ y0 ] ) |  |
| }// **bi\_sample\_flag**[ x0 ][ y0 ] |  |
| } |  |
| } |  |
|  |  |
|  |  |
|  |  |
|  |  |

**bi\_sample\_flag** [ x0 ][ y0 ] equal to 1 specifies that for the current coding unit, no more syntax elements except the bi-color candidate index bi\_sample\_idx[ x0 ][ y0 ] are parsed after bi\_sample\_flag[ x0 ][ y0 ]. bi\_sample\_flag[ x0 ][ y0 ] equal to 0 specifies that the coding unit is not bi color mode. 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 bi\_sample\_flag[ x0 ][ y0 ] is not present, it is inferred to be equal to 0.

**bi\_sample\_idx**[ x0 ][ y0 ] specifies the bi-color candidate index of the bi-color 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 bi\_sample\_idx[ x0 ][ y0 ] is not present, it is inferred to be equal to 0.

**palette\_run\_type\_flag**[ xC ][ yC ] 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.

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

**palette\_index** is an color index to the selected bi-color candidate. It is smaller than 2.

**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 ], **bi\_sample\_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 **bi\_sample\_flag** [ xCb ][ yCb ] is equal to 1), ), The following applies for cIdx from 0 to 2:

The bi-color intra prediction process as specified in subclause 8.x.x.x.x is invoked with the luma location ( xCb, yCb ), the variable log2TrafoSize set equal to log2CbSize, and the variable cIdx as inputs, and the output is a modified reconstructed picture before deblocking filtering.

– Otherwise if (pcm\_flag[ xCb ][ yCb ] is equal to 0, **bi\_sample\_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, **bi\_sample\_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, **bi\_sample\_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 and **bi\_sample\_flag** [ xCb ][ yCb ] is 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.x.x.x.x. Specification of bi-color intra prediction process

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 availability derivation process for a block in z-scan order as specified in subclause 6.4.1 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, availableFlagB0 are set equal to TRUE.

predSample An is set equal to p[ −1 ][ nTbS −1  ].

predSample Bn is set equal to p[ nTbS −1  ][ −1  ].

predSample A1 is set equal to p[ −1 ][ 0 ].

predSample B1 is set equal to p[ 0 ][ −1 ].

predSample B0 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 ][ nTbS −1  ] 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[ nTbS −1  ][ −1 ] is marked as "not available for intra prediction".
* availableFlagAn is equal to TRUE and predSample Bn is equal to predSample An.

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

* p[ 0 ][ −1 ] is marked as "not available for intra prediction".
* availableFlagAn is equal to TRUE and predSample B1 is equal to predSample An.
* availableFlagBn is equal to TRUE and predSample B1 is equal to predSample Bn.

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

* p[ −1 ][ 0 ] is marked as "not available for intra prediction".
* availableFlagAn is equal to TRUE and predSample A1 is equal to predSample An.
* availableFlagBn is equal to TRUE and predSample A1 is equal to predSample Bn.
* availableFlagB1 is equal to TRUE and predSample A1 is equal to predSample B1.

When one or more of the following conditions are true, availableFlag B0is set equal to FALSE:

* p[ −1 ][ −1 ] is marked as "not available for intra prediction".
* availableFlagAn is equal to TRUE and predSample B0is equal to predSample An.
* availableFlagBn is equal to TRUE and predSample B0is equal to predSample Bn.
* availableFlagB1 is equal to TRUE and predSample B0is equal to predSample B1.
* availableFlagA1 is equal to TRUE and predSample B0 is equal to predSample A1.

The sample candidate list, sampleCandList[i][j], is constructed as follows, where j = 0, 1:

i = 0  
if( availableFlagAn or availableFlagBn)  
sampleCandList[ i++ ][0] = An

sampleCandList[ i++ ][1] = Bn  
if( availableFlagAn or availableFlagB1)  
sampleCandList[ i++ ][0] = An

sampleCandList[ i++ ][1] = B1  
if( availableFlagBn or availableFlagB1)  
sampleCandList[ i++ ][0] = Bn

sampleCandList[ i++ ][1] = B1if( availableFlagAn or availableFlagA1)  
sampleCandList[ i++ ][0] = An

sampleCandList[ i++ ][1] = A1

if( availableFlagBn or availableFlagA1)  
sampleCandList[ i++ ][0] = Bn

sampleCandList[ i++ ][1] = A1

if( availableFlagB1 or availableFlagA1)  
sampleCandList[ i++ ][0] = B1

sampleCandList[ i++ ][1] = A1

if( availableFlagAn or availableFlagB0)  
sampleCandList[ i++ ][0] = An

sampleCandList[ i++ ][1] = B0

if( availableFlagBn or availableFlagB0)  
sampleCandList[ i++ ][0] = Bn

sampleCandList[ i++ ][1] = B0

if( availableFlagB1 or availableFlagB0)  
sampleCandList[ i++ ][0] = B1

sampleCandList[ i++ ][1] = B0

if( availableFlagA1 or availableFlagB0)  
sampleCandList[ i++ ][0] = A1

sampleCandList[ i++ ][1] = B0

numSampleMergeCand is set equal to i.

When numSampleMergeCand is less than MaxNumSampleCand, the default values 1<<( bitDepth − 1 ), 0 and (1<<bitDepth)-1 are padded to empty sampleCandList[ i][j] until numSampleMergeCand is equal to MaxNumSampleCand.

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][palette\_index[x][y]]

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

predSamples[ x ][ y ] = 0

The picture reconstruction process prior to in-loop filtering for a colour component as specified in subclause 8.6.6 is invoked with the transform block location (xCurr, yCurr ), 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.