The modifications are based on JCTVC-P1005\_v1.

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( intra\_block\_copy\_enabled\_flag ) { |  |
| **intra\_bc\_skip\_flag**[ x0 ][ y0 ] | ae(v) |
| if( !intra\_bc\_skip\_flag[ x0 ][ y0 ] ) |  |
| **intra\_bc\_flag**[ x0 ][ y0 ] | ae(v) |
| } |  |
| if( !intra\_bc\_flag[ x0 ][ y0 ] && palette\_coding\_mode\_enabled\_flag ) |  |
| **palette\_mode\_flag**[ x0 ][ y0 ] | ae(v) |
| if( !intra\_bc\_flag[ x0 ][ y0 ] && !palette\_mode\_flag[ x0 ][ y0 ] ) |  |
| **dictionary\_mode\_flag**[ x0 ][ y0 ] | ae(v) |
| if( slice\_type != I && !intra\_bc\_flag[ x0 ][ y0 ] &&   !palette\_mode\_flag[ x0 ][ y0 ] && !dictionary\_mode\_flag[ x0 ][ y0 ] ) |  |
| **pred\_mode\_flag** | ae(v) |
| if( CuPredMode[ x0 ][ y0 ] != MODE\_INTRA | | ( intra\_bc\_flag[ x0 ][ y0 ] && !intra\_bc\_skip\_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 ] &&   !palette\_mode\_flag[ x0 ][ y0 ] &&  !dictionary\_mode\_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 ] ) { |  |
| if( PartMode == PART\_2Nx2N ) { |  |
| **intra\_bc\_merge\_flag**[ x0 ][ y0 ] | ae(v) |
| if(intra\_bc\_merge\_flag[ x0 ][ y0 ]) |  |
| **intra\_bc\_merge\_index**[ x0 ][ y0 ] | ae(v) |
| } |  |
| if( !intra\_bc\_merge\_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 if (palette\_mode\_flag[ x0 ][ y0 ] ) { |  |
| palette\_mode( x0, y0, nCbs, nCbs) |  |
| } else if (dictionary\_mode\_flag[ x0 ][ y0 ] ) { |  |
| dictionary\_mode( x0, y0, log2CbSize) |  |
| } else { |  |
| if( adaptive\_color\_space\_enable\_flag) |  |
| **cu\_color\_space**[ x0 ][ y0 ] | ae(v) |
| 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 ] && !palette\_mode\_flag[ x0 ][ y0 ] &&   !dictionary\_mode\_flag[ x0 ][ y0 ] ) { |  |
| if( CuPredMode[ x0 ][ y0 ] != MODE\_INTRA &&   !( PartMode = = PART\_2Nx2N && merge\_flag[ x0 ][ y0 ] ) | |   !( PartMode = = PART\_2Nx2N && intra\_bc\_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 ) |  |
| } |  |
| } |  |
| } |  |
| } |  |

7.3.8.9 Motion vector difference syntax

|  |  |
| --- | --- |
| mvd\_coding( x0, y0, refList ) { | **Descriptor** |
| if( refList == 2 ) |  |
| **intra\_bc\_flip\_flag**[ x0 ][ y0 ] | ae(v) |
| **abs\_mvd\_greater0\_flag**[ 0 ] | ae(v) |
| **abs\_mvd\_greater0\_flag**[ 1 ] | ae(v) |
| if( abs\_mvd\_greater0\_flag[ 0 ] ) |  |
| **abs\_mvd\_greater1\_flag**[ 0 ] | ae(v) |
| if( abs\_mvd\_greater0\_flag[ 1 ] ) |  |
| **abs\_mvd\_greater1\_flag**[ 1 ] | ae(v) |
| if( abs\_mvd\_greater0\_flag[ 0 ] ) { |  |
| if( abs\_mvd\_greater1\_flag[ 0 ] ) |  |
| **abs\_mvd\_minus2**[ 0 ] | ae(v) |
| **mvd\_sign\_flag**[ 0 ] | ae(v) |
| } |  |
| if( abs\_mvd\_greater0\_flag[ 1 ] ) { |  |
| if( abs\_mvd\_greater1\_flag[ 1 ] ) |  |
| **abs\_mvd\_minus2**[ 1 ] | ae(v) |
| **mvd\_sign\_flag**[ 1 ] | ae(v) |
| } |  |
| } |  |

7.3.8.X Palette mode syntax

|  |  |
| --- | --- |
| palette\_mode( x0, y0, CbWidth, CbHeight  ) { | **Descriptor** |
| for( cIdx = 0; cIdx < ( ChromaArrayType == 0 ) ? 1 : 3; cIdx++ ) { |  |
| **color\_table\_share\_flag**[ x0 ][ y0 ][ cIdx ] | ae(v) |
| if( !color\_table\_share\_flag[ x0 ][ y0 ][ cIdx ] ) { |  |
| **major\_color\_num**[ x0 ][ y0 ][ cIdx ] | ae(v) |
| for( n = 0; n < major\_color\_num[ x0 ][ y0 ][ cIdx ]; n++ ) { |  |
| **major\_color\_pred\_flag**[ x0 ][ y0 ][ cIdx ][ n**]** | ae(v) |
| if( major\_color\_pred\_flag[ x0 ][ y0 ][ cIdx ][ n ] ) { |  |
| **major\_color\_pred\_dir**[ x0 ][ y0 ][ cIdx ][ n ] | ae(v) |
| } else { |  |
| **major\_color\_value**[ x0 ][ y0 ][ cIdx ][ n ] | ae(v) |
| } |  |
| } |  |
| } |  |
| } |  |
| for( cIdx = 0; cIdx < ( ChromaArrayType == 3 ) ? 1 : 3; cIdx++ ) { |  |
| for( i = 0; i < CbHeight; i ++) { |  |
| **line\_mode**[ x0 ][ y0 ][ cIdx ][ i ] | ae(v) |
| if( line\_mode[ x0 ][ y0 ][ cIdx ][ i ] == HOR\_MODE ) { |  |
| ( cSubIdx = 0; cSubIdx < ( ChromaArrayType == 3 ) ? 3 : 1; cSubIdx++ ) { |  |
| **color\_index**[ x0 ][ y0 ][ cIdx + cSubIdx ][ i ][ 0 ] | ae(v) |
| } |  |
| } else if( line\_mode[ x0 ][ y0 ][ cIdx ][ i ] == VER\_MODE ) { |  |
| for( j = 0; j < CbWidth; j++ ) { |  |
| for(cSubIdx = 0; cSubIdx < ( ChromaArrayType == 3 ) ? 3 : 1; cSubIdx++ ) { |  |
| ColorIndex[ x0 ][ y0 ][ cIdx + cSubIdx ][ i ][ j ] =   ColorIndex[ x0 ][ y0 ][ cIdx + cSubIdx ][ i-1 ][ j ] |  |
| } |  |
| } |  |
| } else { //NOR\_MODE |  |
| for( j = 0; j < CbWidth; j++ ) { |  |
| **index\_pred\_symbol**[ x0 ][ y0 ][ cIdx ][ i ][ j ] | ae(v) |
| if( indx\_pred\_symbol[ x0 ][ y0 ][ cIdx ][ i ][ j ] != TOP\_PRED &&   indx\_pred\_symbol[ x0 ][ y0 ][ cIdx ][ i ][ j ] != LEFT\_PRED &&  indx\_pred\_symbol[ x0 ][ y0 ][ cIdx ][ i ][ j ] != TOP\_LEFT\_PRED &&  indx\_pred\_symbol[ x0 ][ y0 ][ cIdx ][ i ][ j ] != TOP\_RIGHT\_PRED &&) { |  |
| **index\_pred\_transition**[ x0 ][ y0 ][ cIdx ][ i ][ j ] | ae(v) |
| if( !index\_pred\_transition[ x0 ][ y0 ][ cIdx ][ i ][ j ] ) { |  |
| for(cSubIdx=0; cSubIdx < ( ChromaArrayType == 3 ) ? 3 : 1; cSubIdx++ ) { |  |
| **color\_index**[ x0 ][ y0 ][ cIdx ][ i ][ j ] | ae(v) |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |
| for( j = 0; j < CbWidth; j++ ) { |  |
| for( cSubIdx = 0; cSubIdx < ( ChromaArrayType == 3 ) ? 3 : 1; cSubIdx++ ) { |  |
| if( color\_index[ x0 ][ y0 ][ cIdx + cSubIdx ][ i ][ j ] ==  major\_color\_num[ x0 ][ y0 ][ cIdx + cSubIdx ] ) { |  |
| **quant\_escape\_value**[ cIdx + cSubIdx ][ i ][ j ] | ae(v) |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |

7.3.8.X Dictionary mode syntax

|  |  |
| --- | --- |
| dictionary\_mode( x0, y0, log2CbSize ) { | **Descriptor** |
| nCbS = ( 1  <<  log2CbSize ) |  |
| totalPixel = nCbS\*nCbS |  |
| currPixel = 0 |  |
| currIndex = 0 |  |
| **dictionary\_type\_flag**[ x0 ][ y0 ] | ae(v) |
| **dictionary\_scan\_flag**[ x0 ][ y0 ] | ae(v) |
| while (currPixel < totalPixel ) { |  |
| **dictionay\_pred\_mode\_flag**[ x0 ][ y0 ][ currIndex ] | ae(v) |
| if( dictionay\_pred\_mode\_flag[ x0 ][ y0 ][ currIndex ] ) { |  |
| if( dictionary\_type\_flag[ x0 ][ y0 ] == 0 ) { |  |
| **dictionay\_pred\_offset\_minus1**[ x0 ][ y0 ][ currIndex ] | ae(v) |
| } else { |  |
| **dictionay\_pred\_offsetX**[ x0 ][ y0 ][ currIndex ] | ae(v) |
| **dictionay\_pred\_offsetY**[ x0 ][ y0 ][ currIndex ] | ae(v) |
| } |  |
| **dictionary\_pred\_length\_minus1**[ x0 ][ y0 ][ currIndex ] | ae(v) |
| currPixel += (dictionary\_pred\_length\_minus1 [ x0 ][ y0 ][ currIndex ] + 1) |  |
| } else { |  |
| **dictionary\_direct\_component**[ x0 ][ y0 ][ currIndex ][ 0 ] | ae(v) |
| **dictionary\_direct\_component**[ x0 ][ y0 ][ currIndex ][ 1 ] | ae(v) |
| **dictionary\_direct\_component**[ x0 ][ y0 ][ currIndex ][ 2 ] | ae(v) |
| currPixel += 1 |  |
| } |  |
| currIndex += 1 |  |
| } |  |
| } |  |

7.4.9.5 Coding unit semantics

**intra\_bc\_skip\_flag**[ x0 ][ y0 ] equal to 1 specifies that the current coding unit is coded in skipped intra block copying mode (intra block copying with PartMode equaling to PART\_2Nx2N and without any residue). intra\_bc\_skip\_flag[ x0 ][ y0 ] equal to 0 specifies that the current coding unit is not coded in skipped intra block copying mode. When not present, the value of intra\_bc\_skip\_flag 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.

**intra\_bc\_flag**[ x0 ][ y0 ] equal to 1 specifies that the current coding unit is coded in intra block copying mode. intra\_bc\_flag[ x0 ][ y0 ] equal to 0 specifies that the current coding unit is coded according to pred\_mode\_flag. When not present, when intra\_bc\_skip\_flag[ x0 ][ y0 ] is equal to 1, the value of intra\_bc\_flag is inferred to be equal to 1. Otherwise, the value of intra\_bc\_flag 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.

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

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

**intra\_bc\_merge\_flag**[ x0 ][ y0 ] equal to 1 specifies that the block vector current prediction unit is coded in merge mode (merge with the left block vector or the above block vector). intra\_bc\_merge\_flag[ x0 ][ y0 ] equal to 0 specifies that the block vector of current prediction unit is not coded in merge mode. When not present, the value of intra\_bc\_merge\_flag 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.

**intra\_bc\_merge\_index**[ x0 ][ y0 ] equal to 1 specifies that the block vector of current prediction unit is the same as the above block vector. intra\_bc\_merge\_index[ x0 ][ y0 ] equal to 0 specifies that the block vector of current prediction unit is the same as the left block vector. 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.

**cu\_color\_space**[ x0 ][ y0 ] specifies the color space used for the current coding unit. 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.

7.4.9.9 Motion vector difference semantics

**intra\_bc\_flip\_flag** [ x0 ][ y0 ] equal to 1 specifies that the current prediction unit is predicted in vertically flipped mode. intra\_bc\_flip flag[ x0 ][ y0 ] equal to 0 specifies that the current prediction unit is predicted in normal 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.

7.4.9.X Palette mode semantics

**color\_table\_share\_flag**[ x0 ][ y0 ][ cIdx ] equal to 1 specifies that the current coding unit share the same color table with the last decoded coding unit for color component cIdx. color\_table\_share\_flag[ x0 ][ y0 ][ cIdx ] equal to 0 specifies that the color table should be transmitted for the current coding unit. 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.

**major\_color\_num**[ x0 ][ y0 ][ cIdx ] specifies the number of major colors for color component cIdx. 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. The array index cIdx specify the color components.

**major\_color\_pred\_flag**[ x0 ][ y0 ][ cIdx ][ n ] equal to 1 specifies the n-th major color value for color component cIdx could be predicted from the above or left CU. major\_color\_pred\_flag[ x0 ][ y0 ][ cIdx ][ n ] equal to 0 specifies current major color couldn’t be predicted. 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.

**major\_color\_pred\_dir**[ x0 ][ y0 ][ cIdx ][ n ] specifies the prediction direction of the n-th major color for color component cIdx. major\_color\_pred\_dir[ x0 ][ y0 ][ cIdx ][ n ] equal to 1 specifies that the current major color is the same as the corresponding major color in the above CU. major\_color\_pred\_dir[ x0 ][ y0 ][ cIdx ][ n ] equal to 0 specifies that the current major color is the same as the corresponding major color in the left CU. 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.

**major\_color\_value**[ x0 ][ y0 ][ cIdx ][ n ] specifies the n-th major color value for color component cIdx. 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.

**line\_mode**[ x0 ][ y0 ][ cIdx ][ i ] equal to HOR\_MODE specifies that the i-th pixel line should be decoded in horizontal line mode. line\_mode[ x0 ][ y0 ][ cIdx ][ i ] equal to VER\_MODE specifies that the i-th pixel line should be decoded in vertical line mode. line\_mode[ x0 ][ y0 ][ cIdx ][ i ] equal to NOR\_MODE specifies that the i-th pixel line should be decoded in normal line 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.

**index\_pred\_symbol**[ x0 ][ y0 ][ i ][ j ] specifies the prediction symbol of the index for the current pixel in position ( i, j ). index\_pred\_symbol[ x0 ][ y0 ][ i ][ j ] equal to TOP\_PRED specifies that index value of the current pixel is the same as the above one. index\_pred\_symbol[ x0 ][ y0 ][ i ][ j ] equal to LEFT\_PRED specifies that index value of the current pixel is the same as the left one. index\_pred\_symbol[ x0 ][ y0 ][ i ][ j ] equal to TOP\_LEFT\_PRED specifies that index value of the current pixel is the same as the top-left one. index\_pred\_symbol[ x0 ][ y0 ][ i ][ j ] equal to TOP\_RIGHT\_PRED specifies that index value of the current pixel is the same as the top-right one.

**index\_pred\_transition**[ x0 ][ y0 ][ cIdx ][ i ][ j ] equal to 1 specifies that transition copy is used for the prediction of component cIdx at the location ( i, j ). index\_pred\_transition[ x0 ][ y0 ][ cIdx ][ i ][ j ] equal to 0 specifies that transition copy is not used.

**quant\_escape\_value**[ x0 ][ y0 ][ cIdx ][ i ][ j ] specifies the quantized escape pixel value for color component cIdx at the location ( i, j ).

**color\_index**[ x0 ][ y0 ][ cIdx ][ i ][ j ] specifies the index value of the pixel at the location( i, j ) for color component cIdx.

7.4.9.X Dictionary mode semantics

**dictionary\_type\_flag**[ x0 ][ y0 ] equal to 0 specifies that the current coding unit is coded in dictionary mode and the prediction comes from the dictionary maintained by the decoder. dictionary\_type\_flag[ x0 ][ y0 ] equal to 1 specifies that the current coding unit is coded in dictionary mode and the prediction comes from the previously reconstructed pixels belonging to the same picture. 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.

**dictionary\_scan\_flag**[ x0 ][ y0 ] equal to 0 specifies that the horizontal scanning is applied to current coding unit. dictionary\_scan\_flag[ x0 ][ y0 ] equal to 1 specifies that the vertical scanning is applied to current coding unit. 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.

**dictionay\_pred\_mode\_flag**[ x0 ][ y0 ][ i ] equal to 1 specifies the prediction mode is applied to the i-th decoding sub-string in the current CU. dictionay\_pred\_mode\_flag[ x0 ][ y0 ][ i ] equal to 0 specifies the direct mode is applied to the i-th decoding sub-string in the current CU. 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.

**dictionay\_pred\_offset\_minus1**[ x0 ][ y0 ][ i ] plus 1 specifies the offset of the i-th decoding sub-string in the current CU. 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.

**dictionay\_pred\_offsetX**[ x0 ][ y0 ][ i ] specifies the horizontal offset of the i-th decoding sub-string in the current CU. 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.

**dictionay\_pred\_offsetY**[ x0 ][ y0 ][ i ] specifies the vertical offset of the i-th decoding sub-string in the current CU. 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.

**dictionary\_pred\_length\_minus1**[ x0 ][ y0 ][ i ] plus 1 specifies the length of the i-th decoding sub-string in the current CU. 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.

**dictionary\_direct\_component**[ x0 ][ y0 ][ i ][ comp ] specifies the pixel value of the comp-th color component in the i-th decoding sub-string in the current CU. 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.

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 cu\_color\_space[ x0 ][ y0 ], the following applies

If cu\_color\_space[ x0 ][ y0 ] is not equal to 0 (adaptive color space coding is used, the current chroma format should be 444), following storage process is performed. The surrounding will be stored in an array savedPixels[] with a size of 2\*nCbS+1.

savedPixels[x] will be set as the pixel value at (xCb-1+x, yCb-1 ), where x is in range of 0...nCbS, if the pixel is inside the picture.

savedPixels[nCbS+1+y] will be set as the pixel value at (xCb-1, yCb+y), where y is in range of 0…(nCbs-1), if the pixel is inside picture.

* If cu\_color\_space[ x0 ][ y0 ] is equal to 1 (GBR to YCoCg), the pixels in the above range are modified as follows
  + offset=1<<(BitDepthY-1)
  + maxValue=(1<<BitDepthY)-1
  + R = SCr[i][j]
  + G =SL[i][j]
  + B = SCb[i][j]
  + SL[i][j] = Clip3( 0, maxValue, (2\*R+4\*G+2\*B)/4 )
  + SCb[i][j] = Clip3( 0, maxValue, (R-B)/2+offset )
  + SCr[i][j] = Clip3( 0, maxValue, (2\*G-R-B)/2+offset )
* Otherwise, if cu\_color\_space[ x0 ][ y0 ] is equal to 2 (GBR to RGB), the pixels in the above range are modified as follows
  + R = SCr[i][j]
  + G =SL[i][j]
  + B = SCb[i][j]
  + SL[i][j] = R
  + SCb[i][j] = G
  + SCr[i][j] = B
* Otherwise, if cu\_color\_space[ x0 ][ y0 ] is equal to 3 (GBR to BGR), the pixels in the above range are modified as follows
* R = SCr[i][j]
* G =SL[i][j]
* B = SCb[i][j]
* SL[i][j] = B
* SCb[i][j] = G
* SCr[i][j] = R

……

Depending on the values of pcm\_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 IntraSplitFlag is equal to 0, the following ordered steps apply:

1. When intra\_bc\_flag[ xCb ][ yCb ] is equal to 0 and palette\_mode\_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. 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. When palette\_mode\_flag[ xCb ][ yCb ] is equal to 1, the decoding process is specified as follows:

For each i, j = 0 .. nCbS – 1

* If ColorIndexLuma[ i ][ j ] < MajorColorNumLuma[ xCb ][ yCb ],

RecL[ xCb + i ][ yCb + j ] = MajorColorValueLuma[ xCb ][ yCb ][ ColorIndexLuma[ i ][ j ] ]

* Otherwise

RecL[ xCb + i ][ yCb + j ] = QuanEscapeValueLuma[ i ][ j ] << EscapeShift + Offset,

where EscapeShift = EscapeQuantTable[ QP ], Offset = EscapeShift > 1? 1 << ( EscapeShift - 1 ): 0

1. 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, 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 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 0, 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), 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 and palette\_mode\_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. When palette\_mode\_flag[ xCb ][ yCb ] is equal to 1, the decoding process is specified as follows

For each i, j = 0 .. nCbS – 1

* If ColorIndexCb[ i ][ j ] < MajorColorNumCb[ xCb ][ yCb ],

RecCb[ xCb + i ][ yCb + j ] = MajorColorValueCb[ xCb ][ yCb ][ ColorIndexCb[ i ][ j ] ]

* Otherwise

RecCb[ xCb + i ][ yCb + j ] = QuanEscapeValueCb[ i ][ j ] << EscapeShift + Offset,

where EscapeShift = EscapeQuantTable[ QP ], Offset = EscapeShift > 1? 1 << ( EscapeShift - 1 ): 0

For each i, j = 0 .. nCbS – 1

* If ColorIndexCr[ i ][ j ] < MajorColorNumCr[ xCb ][ yCb ],

RecCr[ xCb + i ][ yCb + j ] = MajorColorValueCr[ xCb ][ yCb ][ ColorIndexCr[ i ][ j ] ]

* Otherwise

RecCr[ xCb + i ][ yCb + j ] = QuanEscapeValueCr[ i ][ j ] << EscapeShift + Offset,

where EscapeShift = EscapeQuantTable[ QP ], Offset = EscapeShift > 1? 1 << ( EscapeShift - 1 ): 0

1. 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, and the variable cIdx set equal to 1 as inputs, and the output is a modified reconstructed picture before deblocking filtering.
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, and the variable cIdx set equal to 2 as inputs, and the output is a modified reconstructed picture before deblocking filtering.

– Otherwise (pcm\_flag[ xCb ][ yCb ] is equal to 0, 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:

1. The variable xBS is set equal to xCb + ( nS  >>  1 ) \* ( blkIdx % 2 ).
2. The variable yBS is set equal to yCb + ( nS  >>  1 ) \* ( blkIdx / 2 ).
3. The derivation process for the chroma intra prediction mode as specified in 8.4.3 is invoked with the luma location ( xBS, yBS ) as input, and the output is the variable IntraPredModeC.
4. The general decoding process for intra blocks as specified in subclause 8.4.4.1 is invoked with the chroma location ( xBS, yBS), the variable log2TrafoSize set equal to log2CbSizeC − 1, the variable trafoDepth set equal to 1, the variable predModeIntra set equal to IntraPredModeC, the variable predModeIntraBc set equal to 0, and the variable cIdx set equal to 1 as inputs, and the output is a modified reconstructed picture before deblocking filtering.
5. The general decoding process for intra blocks as specified in subclause 8.4.4.1 is invoked with the chroma location ( xBS, yBS), the variable log2TrafoSize set equal to log2CbSizeC − 1, the variable trafoDepth set equal to 1, the variable predModeIntra set equal to IntraPredModeC, the variable predModeIntraBc set equal to 0, and the variable cIdx set equal to 2 as inputs, and the output is a modified reconstructed picture before deblocking filtering.

……

Depending cu\_color\_space[ x0 ][ y0 ], the following applies

If cu\_color\_space[ x0 ][ y0 ] is not equal to 0 (adaptive color space coding is used, the current chroma format should be 444), following recover process is performed. The surrounding will be recovered from the array savedPixels[] with a size of 2\*nCbS+1.

savedPixels[x] will be set to the pixel at (xCb-1+x, yCb-1 ), where x is in range of 0...nCbS, if the pixel is inside the picture.

savedPixels[nCbS+1+y] will be set to the pixel at (xCb-1, yCb+y), where y is in range of 0…(nCbs-1), if the pixel is inside picture.

* If cu\_color\_space[ x0 ][ y0 ] is equal to 1 (GBR to YCoCg), the pixels in the range of i=yCb…(yCb+nCbS-1) and i=xCb…(xCb+nCbS-1) are modified as follows
  + offset=1<<(bitDepthY-1)
  + maxValue=(1<<BitDepthY)-1
  + Y = SL[i][j]-offset
  + Co = SCr[i][j]
  + Cg = SCb[i][j]-offset
  + SCr[i][j] = Clip3( 0, maxValue, Y+Co-Cg )
  + SL[i][j] = Clip3( 0, maxValue, Y+Cg )
  + SCb[i][j] = Clip3( 0, maxValue, Y-Co-Cg )
* Otherwise, if cu\_color\_space[ x0 ][ y0 ] is equal to 2 (GBR to RGB), the pixels in the range of i=yCb…(yCb+nCbS-1) and i=xCb…(xCb+nCbS-1) are modified as follows
  + R = SL[i][j]
  + G = SCr[i][j]
  + B = SCb[i][j]
  + SL[i][j] = G
  + SCb[i][j] = B
  + SCr[i][j] = R
* Otherwise, if cu\_color\_space[ x0 ][ y0 ] is equal to 3 (GBR to BGR), the pixels in the range of i=yCb…(yCb+nCbS-1) and i=xCb…(xCb+nCbS-1) are modified as follows
  + R = SCr[i][j]
  + G = SCb[i][j]
  + B = SL[i][j]
  + SL[i][j] = G
  + SCb[i][j] = B
  + SCr[i][j] = R

8.4.4 Derivation process for block vector components in intra block copying prediction mode

The array of block vectors bvIntra is derived by the following ordered steps, for the variable blkIdx proceeding over the values 0..( numPartitions − 1 ):

1. The variable blkInc is set equal to ( PartMode = = PART\_2NxN ? 2 : 1 ).
2. The variable xPb is set equal to xCb + nPbSw \* ( blkIdx \* blkInc % 2 ).
3. The variable yPb is set equal to yCb + nPbSh \* ( blkIdx / 2 )
4. The following ordered steps apply, for the variable compIdx proceeding over the values 0..1:
5. If intra\_bc\_merge\_flag[ xPb ][ yPb ] is equal to 1, the following applies.

intra\_bc\_merge\_candidate[0][ compIdx ] is set equal bvIntra[ xPb - 1 ][ yPb ][ compIdx ]

intra\_bc\_merge\_candidate[1][ compIdx ] is set equal bvIntra[ xPb ][ yPb - 1 ][ compIdx ]

idx is set equal to intra\_bc\_merge\_flag[ xPb ][ yPb ]

bvIntra[ xPb ][ yPb ][ compIdx ] is set equal to intra\_bc\_merge\_candidate[idx][ compIdx ]

Otherwise (intra\_bc\_merge\_flag[ xPb ][ yPb ] is equal to 0), ~~D~~depending upon the number of times this process has been invoked for the current coding tree unit, the following applies:

* If this process is invoked for the first time for the current coding tree unit, bvIntra[ xPb ][ yPb ][ compIdx ] is derived as follows:

bvIntra[ xPb ][ yPb ][ 0 ] = BvdIntra[ xPb ][ yPb ][ 0 ] − nCbS (8‑25)

bvIntra[ xPb ][ yPb ][ 1 ] = BvdIntra[ xPb ][ yPb ][ 1 ] (8‑25)

* Otherwise, bvIntra[ xPb ][ yPb ][ compIdx ] is derived as follows:

bvIntra[ xPb ][ yPb ][ 0 ] = BvdIntra[ xPb ][ yPb ][ 0 ] + BvpIntra[ 0 ] (8‑25)

bvIntra[ xPb ][ yPb ][ 1 ] = BvdIntra[ xPb ][ yPb ][ 1 ] + BvpIntra[ 1 ] (8‑25)

The value of BvpIntra[ compIdx ] is updated to be equal to bvIntra[ xPb ][ yPb ][ compIdx ].

8.4.5.2.7 Specification of intra block copying prediction mode

……

– The (nTbS)x(nTbS) array of predicted samples samples, with x, y = 0..nTbS − 1, is derived as follows:

– The reference sample location (xRefCmp, yRefCmp ) is specified by:

Depending on intra\_bc\_flip\_flag[ xTb0][ yTb0 ], the following applies. If intra\_bc\_flip\_flag[ xTb0][ yTb0 ] is euqal to 1

( xRefCmp, yRefCmp ) = ( xTbCmp + x + bv[ 0 ], yTbCmp + nTbS-1- y + bv[ 1 ] )

Otherwise, (intra\_bc\_flip\_flag[ xTb0][ yTb0 ] is euqal to 0)

( xRefCmp, yRefCmp ) = ( xTbCmp + x + bv[ 0 ], yTbCmp + y + bv[ 1 ] ) (8‑65)

* Each sample at the location ( xRefCmp, yRefCmp ) is assigned to predSamples[ x ][ y ].

8.X Decoding process for dictionary mode

An array DictHor[] is used to maintain the virtual dictionary for normal dictionary mode in horizontal scanning. An array DictVer[] is used to maintain the virtual dictionary for normal dictionary mode in vertical scanning. Each element in the dictionary is a pixel including three components.

At the beginning of decoding one slice DictHor and DictVer are set to empty.

After decoding one CTU, if the size of DictHor is greater than (1<<18)\*3/2, the oldest (1<<18)/2 elements are removed from the dictionary. After decoding one CTU, if the size of DictVer is greater than (1<<18)\*3/2, the oldest (1<<18)/2 elements are removed from the dictionary.

If the current CU uses normal dictionary mode with horizontal scanning, 8.X.1 will be invoked.

If the current CU uses normal dictionary mode with vertical scanning, 8.X.2 will be invoked.

If the current CU uses reconstruction based dictionary mode with horizontal scanning, 8.X.3 will be invoked.

If the current CU uses reconstruction based dictionary mode with vertical scanning, 8.X.4 will be invoked.

After decoding one CU, if the current CU uses dictionary mode, the following applies.

If the current CU uses normal dictionary mode with horizontal scanning, the pixels in the current CU will also be added to DictVer using vertical scanning order.

Otherwise, if the current CU uses normal dictionary mode with vertical scanning, the pixels in the current CU will also be added to DictHor using horizontal scanning order.

Otherwise, (the current CU uses reconstruction based dictionary mode), the pixels in the current CU will added to both DictVer using vertical scanning order and DictHor using horizontal scanning order.

8.X.1 Decoding of CU in normal dictionary mode with horizontal scanning

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.

– arrays of dictInfoMatchFlag, dictInfoMatchOffset, dictInfoMatchOffset, dictInfoPixel containing the dictionary information.

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

decodedCount is set to 0.

currIdx is set to 0.

nCbS is set to 1<<log2CbSize.

totalPixel is set to nCbS\*nCbS.

currentDictSize is set to the current size of DictHor.

While decodedCount is smaller than totalPixel, the following steps applies.

* If dictInfoMatchFlag[currIdx] is equal to 1,
  + for(i=0;i<dictInfoMatchLength[currIdx];i++)  
    DictHor[currentDictSize+decodedCount+i] = DictHor[currentDictSize+decodedCount+i-dictInfoMatchOffset[currIdx]]
  + decodedCount += dictInfoMatchLength[currIdx]
* Otherwise (dictInfoMatchFlag[currIdx] is equal to 0)
  + DictHor[currentDictSize+decodedCount]= dictInfoPixel[currIdx]
  + decodedCount += 1
* currIdx++

The pixels at (x, y), where x is in range of  xCb… xCb+nCbS, and y is in range of yCb … yCb+nCbS  will be set to DictHor[currentDictSize+(y-yCb)\*nCbS+(x-xCb)]

8.X.2 Decoding of CU in normal dictionary mode with vertical scanning

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.

– arrays of dictInfoMatchFlag, dictInfoMatchOffset, dictInfoMatchOffset, dictInfoPixel containing the dictionary information.

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

decodedCount is set to 0.

currIdx is set to 0.

nCbS is set to 1<<log2CbSize.

totalPixel is set to nCbS\*nCbS.

currentDictSize is set to the current size of DictVer.

While decodedCount is smaller than totalPixel, the following steps applies.

* If dictInfoMatchFlag[currIdx] is equal to 1,
  + for(i=0;i<dictInfoMatchLength[currIdx];i++)  
    DictVer[currentDictSize+decodedCount+i] = DictVer[currentDictSize+decodedCount+i-dictInfoMatchOffset[currIdx]]
  + decodedCount += dictInfoMatchLength[currIdx]
* Otherwise (dictInfoMatchFlag[currIdx] is equal to 0)
  + DictVer[currentDictSize+decodedCount]= dictInfoPixel[currIdx]
  + decodedCount += 1
* currIdx++

The pixels at (x, y), where x is in range of  xCb… xCb+nCbS, and y is in range of yCb … yCb+nCbS  will be set to DictVer[currentDictSize+(x-xCb)\*nCbS+(y-yCb)]

8.X.3 Decoding of CU in reconstruction based dictionary mode with horizontal scanning

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.

– arrays of dictInfoMatchFlag, dictInfoMatchOffset, dictInfoMatchOffsetX, dictInfoMatchOffsetY, dictInfoPixel containing the dictionary information.

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

decodedCount is set to 0.

currIdx is set to 0.

nCbS is set to 1<<log2CbSize.

totalPixel is set to nCbS\*nCbS.

While decodedCount is smaller than totalPixel, the following steps applies.

* If dictInfoMatchFlag[currIdx] is equal to 1,
  + offsetX = dictInfoMatchOffsetX[currIdx]
  + offsetY = dictInfoMatchOffsetY[currIdx]
  + for(i=0;i<dictInfoMatchLength[currIdx];i++, decodedCount++)  
    The pixel at (xCb+decodedCount%nCbS,yCb+decocedCount/nCbS) is set to equal to the pixel at (xCb-offsetX+decodedCount%nCbS, yCb-offsetY+decocedCount/nCbS)
* Otherwise (dictInfoMatchFlag[currIdx] is equal to 0)
  + The pixel at (xCb+decodedCount%nCbS,yCb+decocedCount/nCbS) is set to equal to the pixel of dictInfoPixel[currIdx]
  + decodedCount += 1
* currIdx++

8.X.4 Decoding of CU in reconstruction based dictionary mode with vertical scanning

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.

– arrays of dictInfoMatchFlag, dictInfoMatchOffset, dictInfoMatchOffsetX, dictInfoMatchOffsetY, dictInfoPixel containing the dictionary information.

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

decodedCount is set to 0.

currIdx is set to 0.

nCbS is set to 1<<log2CbSize.

totalPixel is set to nCbS\*nCbS.

While decodedCount is smaller than totalPixel, the following steps applies.

* If dictInfoMatchFlag[currIdx] is equal to 1,
  + offsetX = dictInfoMatchOffsetX[currIdx]
  + offsetY = dictInfoMatchOffsetY[currIdx]
  + for(i=0;i<dictInfoMatchLength[currIdx];i++, decodedCount++)  
    The pixel at (xCb+decodedCount/nCbS,yCb+decocedCount%nCbS) is set to equal to the pixel at (xCb-offsetX+decodedCount/nCbS, yCb-offsetY+decocedCount%nCbS)
* Otherwise (dictInfoMatchFlag[currIdx] is equal to 0)
  + The pixel at (xCb+decodedCount/nCbS,yCb+decocedCount%nCbS) is set to equal to the pixel of dictInfoPixel[currIdx]
  + decodedCount += 1
* currIdx++