#### 7.3.6.1 General slice segment header syntax

|  |  |
| --- | --- |
| slice\_segment\_header( ) { | Descriptor |
| …… |  |
| if( ( weighted\_pred\_flag && slice\_type = = P ) | |  ( weighted\_bipred\_flag && slice\_type = = B ) ) |  |
| pred\_weight\_table( ) |  |
| } |  |
| if( slice\_type = = P | | slice\_type = = B | | intra\_block\_copy\_enabled\_flag ) { |  |
| **five\_minus\_max\_num\_merge\_cand** | ue(v) |
| } |  |
| **slice\_qp\_delta** | se(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 | | intra\_block\_copy\_enabled\_flag) |  |
| **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\_flag~~**~~[ x0 ][ y0 ]~~ | ~~ae(v)~~ |
| if( slice\_type != I | | intra\_block\_copy\_enabled\_flag) |  |
| **pred\_mode\_flag** | ae(v) |
| if( palette\_mode\_enabled\_flag && ChromaArrayType = = 3 &&   CuPredMode[ x0 ][ y0 ] = = MODE\_INTRA ) |  |
| **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 ] )~~ ) |  |
| **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 ) |  |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |

#### 7.3.8.6 Prediction unit syntax

#### 7.3.8.6 Prediction unit syntax

|  |  |
| --- | --- |
| prediction\_unit( x0, y0, nPbW, nPbH ) { | Descriptor |
| if( cu\_skip\_flag[ x0 ][ y0 ] ) { |  |
| if( MaxNumMergeCand > 1 ) |  |
| **merge\_idx**[ x0 ][ y0 ] | ae(v) |
| ~~} else if( intra\_bc\_flag[ x0 ][ y0 ] ) { /\* Intra BC\*/~~ |  |
| ~~bvd\_coding( x0, y0, 2 )~~ |  |
| } else { /\* MODE\_INTER \*/ |  |
| **merge\_flag**[ x0 ][ y0 ] | ae(v) |
| if( merge\_flag[ x0 ][ y0 ] ) { |  |
| if( MaxNumMergeCand > 1 ) |  |
| **merge\_idx**[ x0 ][ y0 ] | ae(v) |
| } else { |  |
| if( intra\_block\_copy\_enabled\_flag && slice\_type != I &&  !( MinCbLog2SizeY = = 3 && PartMode = = PART\_NxN )) |  |
| **intra\_bc\_flag**[ x0 ][ y0 ] | ae(v) |
| if( slice\_type = = B && !intra\_bc\_flag[ x0 ][ y0 ]) |  |
| **inter\_pred\_idc**[ x0 ][ y0 ] | ae(v) |
| if( inter\_pred\_idc[ x0 ][ y0 ] != PRED\_L1 ) { |  |
| if( num\_ref\_idx\_l0\_active\_minus1 > 0 && !intra\_bc\_flag[ x0 ][ y0 ]) |  |
| **ref\_idx\_l0**[ x0 ][ y0 ] | ae(v) |
| mvd\_coding( x0, y0, 0 ) |  |
| **mvp\_l0\_flag**[ x0 ][ y0 ] | ae(v) |
| } |  |
| if( inter\_pred\_idc[ x0 ][ y0 ] != PRED\_L0 ) { |  |
| if( num\_ref\_idx\_l1\_active\_minus1 > 0 ) |  |
| **ref\_idx\_l1**[ x0 ][ y0 ] | ae(v) |
| if( mvd\_l1\_zero\_flag &&   inter\_pred\_idc[ x0 ][ y0 ] = = PRED\_BI ) { |  |
| MvdL1[ x0 ][ y0 ][ 0 ] = 0 |  |
| MvdL1[ x0 ][ y0 ][ 1 ] = 0 |  |
| } else |  |
| mvd\_coding( x0, y0, 1 ) |  |
| **mvp\_l1\_flag**[ x0 ][ y0 ] | ae(v) |
| } |  |
| } |  |
| } |  |
| } |  |

#### 7.3.8.9 Transform tree syntax

|  |  |
| --- | --- |
| transform\_tree( x0, y0, xBase, yBase, log2TrafoSize, trafoDepth, blkIdx ) { | Descriptor |
| if( log2TrafoSize <= MaxTbLog2SizeY &&  log2TrafoSize > MinTbLog2SizeY &&  trafoDepth < MaxTrafoDepth && !( IntraSplitFlag && ( trafoDepth = = 0 ) ) ) |  |
| **split\_transform\_flag**[ x0 ][ y0 ][ trafoDepth ] | ae(v) |
| if( ( log2TrafoSize > 2 && ChromaArrayType != 0 ) | | ChromaArrayType = = 3 ) { |  |
| if( trafoDepth = = 0 | | cbf\_cb[ xBase ][ yBase ][ trafoDepth − 1 ] ) { |  |
| **cbf\_cb**[ x0 ][ y0 ][ trafoDepth ] | ae(v) |
| if( ChromaArrayType = = 2 &&   ( !split\_transform\_flag[ x0 ][ y0 ][ trafoDepth ] | | log2TrafoSize = = 3 ) ) |  |
| **cbf\_cb**[ x0 ][ y0 + ( 1 << ( log2TrafoSize − 1 ) ) ][ trafoDepth ] | ae(v) |
| } |  |
| if( trafoDepth = = 0 | | cbf\_cr[ xBase ][ yBase ][ trafoDepth − 1 ] ) { |  |
| **cbf\_cr**[ x0 ][ y0 ][ trafoDepth ] | ae(v) |
| if( ChromaArrayType = = 2 &&   ( !split\_transform\_flag[ x0 ][ y0 ][ trafoDepth ] | | log2TrafoSize = = 3 ) ) |  |
| **cbf\_cr**[ x0 ][ y0 + ( 1 << ( log2TrafoSize − 1 ) ) ][ trafoDepth ] | ae(v) |
| } |  |
| } |  |
| if( split\_transform\_flag[ x0 ][ y0 ][ trafoDepth ] ) { |  |
| x1 = x0 + ( 1  <<  ( log2TrafoSize − 1 ) ) |  |
| y1 = y0 + ( 1  <<  ( log2TrafoSize − 1 ) ) |  |
| transform\_tree( x0, y0, x0, y0, log2TrafoSize − 1, trafoDepth + 1, 0 ) |  |
| transform\_tree( x1, y0, x0, y0, log2TrafoSize − 1, trafoDepth + 1, 1 ) |  |
| transform\_tree( x0, y1, x0, y0, log2TrafoSize − 1, trafoDepth + 1, 2 ) |  |
| transform\_tree( x1, y1, x0, y0, log2TrafoSize − 1, trafoDepth + 1, 3 ) |  |
| } else { |  |
| if( ~~(~~ CuPredMode[ x0 ][ y0 ] = = MODE\_INTRA ~~&& intra\_bc\_flag[ x0 ][ y0 ] != 1 )~~    | | trafoDepth != 0 | | cbf\_cb[ x0 ][ y0 ][ trafoDepth ] | |   cbf\_cr[ x0 ][ y0 ][ trafoDepth ] | | ( ChromaArrayType = = 2 &&   ( cbf\_cb[ x0 ][ y0 + ( 1 << ( log2TrafoSize − 1 ) ) ][ trafoDepth ] | |   cbf\_cr[ x0 ][ y0 + ( 1 << ( log2TrafoSize − 1 ) ) ][ trafoDepth ] ) ) ) |  |
| **cbf\_luma**[ x0 ][ y0 ][ trafoDepth ] | ae(v) |
| transform\_unit( x0, y0, xBase, yBase, log2TrafoSize, trafoDepth, blkIdx ) |  |
| } |  |
| } |  |

#### ~~7.3.8.10 Block vector difference syntax~~

|  |  |
| --- | --- |
| ~~bvd\_coding ( x0, y0, refList ) {~~ | **~~Descriptor~~** |
| **~~abs\_bvd\_greater0\_flag~~**~~[ 0 ]~~ | ~~ae(v)~~ |
| **~~abs\_bvd\_greater0\_flag~~**~~[ 1 ]~~ | ~~ae(v)~~ |
| ~~if( abs\_bvd\_greater0\_flag[ 0 ] ) {~~ |  |
| **~~abs\_bvd\_minus1~~**~~[ 0 ]~~ | ~~ae(v)~~ |
| **~~bvd\_sign\_flag~~**~~[ 0 ]~~ | ~~ae(v)~~ |
| ~~}~~ |  |
| ~~if( abs\_bvd\_greater0\_flag[ 1 ] ) {~~ |  |
| **~~abs\_bvd\_minus1~~**~~[ 1 ]~~ | ~~ae(v)~~ |
| **~~bvd\_sign\_flag~~**~~[ 1 ]~~ | ~~ae(v)~~ |
| ~~}~~ |  |
| ~~}~~ |  |

#### 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 ]~~  | |   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 ]~~ | |   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 ] )~~ &&   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) |
| } |  |
| …… |  |

##### 7.4.3.2.3 Sequence parameter set scc extension

**intra\_block\_copy\_enabled\_flag** equal to 1 specifies that ~~intra block copying~~ the process of adding the current decoded picture into reference picture list 0 may be invoked in the decoding process ~~for intra prediction~~. intra\_block\_copy\_enabled\_flag equal to 0 specifies that ~~intra block copying~~ the process of adding the current decoded picture into reference picture list 0 is not applied. When not present, the value of intra\_block\_copy\_enabled\_flag is inferred to be equal to 0.

#### 7.4.7.1 General slice segment header semantics

**num\_ref\_idx\_l0\_active\_minus1** specifies the maximum reference index for reference picture list 0 that may be used to decode the slice. num\_ref\_idx\_l0\_active\_minus1 shall be in the range of 0 to 14, inclusive. When the current slice is a P or B slice and num\_ref\_idx\_l0\_active\_minus1 is not present, num\_ref\_idx\_l0\_active\_minus1 is inferred to be equal to num\_ref\_idx\_l0\_default\_active\_minus1 for P and B slice, and is interred to be equal to −1 for I slice.

**num\_ref\_idx\_l1\_active\_minus1** specifies the maximum reference index for reference picture list 1 that may be used to decode the slice. num\_ref\_idx\_l1\_active\_minus1 shall be in the range of 0 to 14, inclusive. When num\_ref\_idx\_l1\_active\_minus1 is not present, num\_ref\_idx\_l1\_active\_minus1 is inferred to be equal to num\_ref\_idx\_l1\_default\_active\_minus1 for B slice, and is interred to be equal to −1 for I and P slice.

#### 7.4.9.5 Coding unit semantics

**~~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, the value of intra\_bc\_flag is inferred to be equal to 0. The array indices x0 and 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 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.

#### Prediction unit semantics

**intra\_bc\_flag**[ x0 ][ y0 ] equal to 1 specifies that the current decoded picture is used as the reference picture for current prediction unit. intra\_bc\_flag[ x0 ][ y0 ] equal to 0 specifies that the current prediction unit is coded according to pred\_mode\_flag. The array indices x0 and y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered prediction block relative to the top-left luma sample of the picture. When not present, the value of intra\_bc\_flag is inferred as follows:

* + if pred\_mode\_flag is equal to 0 and intra\_block\_copy\_enabled\_flag is equal to 1
    - if slice\_type is equal to I, intra\_bc\_flag is set equal to 1;
    - if PartMode is equal to PART\_NxN and MinCbLog2SizeY is equal to 3, intra\_bc\_flag is set equal to 1;
  + otherwise, intra\_bc\_flag is set equal to 0

**ref\_idx\_l0**[ x0 ][ y0 ] specifies the list 0 reference picture index for the current prediction unit. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered prediction block relative to the top-left luma sample of the picture.

When ref\_idx\_l0[ x0 ][ y0 ] is not present it is inferred as follows: ~~to be equal to 0.~~

* + When intra\_bc\_flag[ x0 ][ y0 ] is equal to 1, ref\_idx\_l0[ x0 ][ y0 ] is set equal to num\_ref\_idx\_l0\_active\_minus1 + 1.
  + Otherwise, ref\_idx\_l0[ x0 ][ y0 ] is set equal to 0.

### 8.3.6 Decoding process for reference picture list modification when intra\_block\_copy\_enabled\_flag is equal to 1

This process is invoked after 8.3.4 is invoked for each P or B slide when intra\_block\_copy\_enabled\_flag is equal to 1. This process is invoked at the beginning of decoding process for each I slice when intra\_block\_copy\_enabled\_flag is equal to 1.

num\_ref\_idx\_l0\_active\_minus1 += 1

RefPicList0[ num\_ref\_idx\_l0\_active\_minus1 ] = current decoded picture

The current decoded picture is marked as “used for long-term reference”.

### 8.3.7 Marking process for ending the decoding of a coded picture with intra\_block\_copy\_enabled\_flag equal to 1

This process is invoked at the ending of decoding of a coded picture with intra\_block\_copy\_enabled\_flag equal to 1.

The current decoded picture is marked as “used for short-term reference”.

### 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 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 ], 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 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 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 bvIntra,~~ the variable cIdx, and variable controlParaACT equal to 1 as inputs, and the output is the residual sample array resSamplescomp.

– The residual modification process for residual blocks using adaptive colour transform as specified in subclause 8.6.8 is invoked with the variable blkSize set equal to nCbS, the (nCbS)x(nCbS) array rY set equal to resSamplesL, the (nCbS)x(nCbS) array rCb set equal to resSamplesCb, and the (nCbS)x(nCbS) array rCr set equal to resSamplesCr as inputs, and the output are modified versions of the (nCbS)x(nCbS) arrays resSamplesL, resSamplesCb and resSamplesCr. [Ed. (GJS): Paragraph hanging indentation alignment problem.]

……

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

~~Inputs to this process are:~~

* ~~a luma location ( xCb, yCb ) of 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 the (nCbS)x(nCbX) array [Ed. some callees still call this a single vector] of block vectors bvIntra.~~

~~The variables nCbS, nPbSw, and nPbSh are derived as follows:~~

~~nCbS = 1  <<  log2CbSize (8‑25)~~

~~nPbSw = nCbS / ( PartMode = = PART\_2Nx2N | | PartMode = = PART\_2NxN ? 1 : 2 ) (8‑25)~~

~~nPbSh = nCbS / ( PartMode = = PART\_2Nx2N | | PartMode = = PART\_Nx2N ? 1 : 2 ) (8‑25)~~

~~The variable BvpIntra[ compIdx ] specifies a block vector predictor. The horizontal block vector component is assigned compIdx = 0 and the vertical block vector component is assigned compIdx = 1.~~

~~Depending upon PartMode, the variable numPartitions is derived as follows:~~

~~– If PartMode is equal to PART\_2Nx2N, numPartitions is set equal to 1.~~

~~– Otherwise, if PartMode is equal to either PART\_2NxN or PART\_Nx2N, numPartitions is set equal to 2.~~

~~– Otherwise (PartMode is equal to PART\_NxN), numPartitions is set equal to 4.~~

~~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. ~~The variable LastBvIntra[ 0 ][ compIdx ] and LastBvIntra[ 1 ][ compIdx ]specifies the last two block vector predictor. If this process is invoked for the first time for the current coding tree unit, LastBvIntra[ compIdx ] is derived as follows:~~

~~LastBvIntra[ 0 ][ 0 ] = −2\* nCbS; LastBvIntra[ 0 ][ 1 ] = 0~~

~~LastBvIntra[ 1 ][ 0 ] = −nCbS; LastBvIntra[ 1 ][ 1 ] = 0~~

~~Depending upon the number of times this process has been invoked for the current coding tree unit, subclause  is invoked with the luma coding block location ( xCb, yCb ), the coding block size nCbS, the luma prediction block location ( xPb, yPb ), the luma prediction block width nPbSw, the luma prediction block height nPbSh, the last block vectors LastBvIntra, and the partition index blkIdx as inputs, and the block vector predictor BvpIntra[ xPb ][ yPb ] as the output, and bvIntra[ xPb ][ yPb ][ compIdx ] is set equal to BvdIntra[ xPb ][ yPb ][ compIdx ] + BvpIntra[ xPb ][ yPb ][ compIdx ] [Ed. (GJS): Needs further formatting cleanup.]~~

~~When bvIntra[ xPb ][ yPb ][ 0 ] is not equal to LastBvIntra[ 0 ][ 0 ] or bvIntra[ xPb ][ yPb ][ 1 ] is not equal to LastBvIntra[ 0 ][ 1 ], the value of LastBvIntra[ 1 ][ compIdx ] is updated to be LastBvIntra[ 0 ][ compIdx ], and the value of LastBvIntra[ 0 ][ compIdx ] is updated to be bvIntra[ xPb ][ yPb ][ compIdx ].~~

1. ~~For use in derivation processes of variables invoked later in the decoding process, the following assignment is made for x = 0..nPbSw − 1 and y = 0..nPbSh − 1:~~

~~bvIntra[ xPb + x ][ yPb + y ][ compIdx ] = bvIntra[ xPb ][ yPb ][ compIdx ] (8‑25)~~

~~– When the derivation process for z-scan order block availability as specified in subclause  is invoked with ( xCurr, yCurr ) set equal to ( xCb, yCb ) and the neighbouring luma location ( xNbY, yNbY ) set equal to ( xPb + bvIntra[ xPb ][ yPb ][ 0 ], yPb + bvIntra[ xPb ][ yPb ][ 1 ] ) as inputs, the output is set equal to TRUE.~~

~~– When the derivation process for z-scan order block availability as specified in subclause  is invoked with ( xCurr, yCurr ) set equal to ( xCb, yCb ) and the neighbouring luma location ( xNbY, yNbY ) set equal to ( xPb + bvIntra[ xPb ][ yPb ][ 0 ] + nPbSw − 1, yPb + bvIntra[ xPb ][ yPb ][ 1 ] + nPbSh – 1 ) as inputs, the output is set equal to TRUE.~~

~~– One or both of the following conditions shall be true: [Ed. (GJS): Clarify that this is a bitstream constraint (if that is the correct interpretation).]~~

~~– bvIntra[ xPb ][ yPb ][ 0 ] + nPbSw is less than or equal to 0~~

~~– bvIntra[ xPb ][ yPb ][ 1 ] + nPbSh is less than or equal to 0~~

#### ~~8.4.4.1 Derivation process for intra block copy block vector prediction~~

~~Inputs to this process are:~~

* ~~a luma location ( xCb, yCb ) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,~~
* ~~a variable nCbS specifying the size of the current luma coding block,~~
* ~~a luma location ( xPb, yPb ) specifying the top-left sample of the current luma prediction block relative to the top-left luma sample of the current picture,~~
* ~~two variables nPbW and nPbH specifying the width and the height of the luma prediction block,~~
* ~~the reference index of the current prediction unit partition refIdxL2~~
* ~~the last block vector LastBvIntra~~
* ~~a variable partIdx specifying the index of the current prediction unit within the current coding unit.~~

~~Output of this process is the block vector prediction BvpIntra[ xPb ][ yPb ].~~

~~The block vector predictor BvpIntra is derived in the following ordered steps:~~

1. ~~The derivation process for block vector predictor candidates from neighbouring prediction unit partitions in subclause is invoked with the luma coding block location ( xCb, yCb ), the coding block size nCbS, the luma prediction block location ( xPb, yPb ), the luma prediction block width nPbW, the luma prediction block height nPbH, and the partition index partIdx as inputs, and the availability flags availableFlagN and the block vectors bvIntraN, with N being replaced by A or B, as output.~~
2. ~~The variables bvpIntraVirtual[ i ][ j ] (with i and j being equal to 0 or 1) specify two virtual block vector predictors, and are derived as follows:~~

~~bvpIntraVirtual[ 0 ][ 0 ] = LastBvIntra[ 0 ][ 0 ], bvpIntraVirtual[ 0 ][ 1 ] = LastBvIntra[ 0 ][ 1 ];~~

~~bvpIntraVirtual[ 1 ][ 0 ] = LastBvIntra[ 1 ][ 0 ], bvpIntraVirtual[ 1 ][ 1 ] = LastBvIntra[ 1 ][ 1 ];~~

~~[Ed. (GJS): Correct equation formatting (e.g., adding equation numbers).]~~

1. ~~The block vector predictor candidate list, bvpIntraList, is constructed as follows:~~

~~i = 0 [Ed. (GJS): Correct equation formatting (e.g., adding equation numbers).]  
if(availableFlagA )  
 bvpIntraList[ i++ ] = bvIntraA  
if( availableFlagB && bvIntraA != bvIntraB )  
 bvpIntraList[ i++ ] = bvIntraB  
for( j=0; j < 2 && i < 2; j++)  
 if( ( j = =1 | | bvpIntraList[ 0 ] != bvpIntraVirtual[ j ] )  
 bvpIntraList[ i++ ] = bvpIntraVirtual[ j ]~~

~~[Ed. (GJS): Check/fix usage of bvpIntraVirtual with two indices versus with one index and the assumption of being a two-dimensional vector. What do we do elsewhere for similar uses?]~~

1. ~~The block vector bvIntra[ xPb ][ yPb ] is derived as follows.~~

~~for (i =0; i< 2; i++)  
 BvpIntra[ xPb ][ yPb ][ i ]= bvpIntraList[ i ]~~

~~[Ed. (GJS): Correct equation formatting (e.g., adding equation numbers). Is this supposed to say bvIntra or BvpIntra? Check/fix substring problem.]~~

#### ~~8.4.4.2 Derivation process for intra block copy block vector prediction candidates~~

~~Inputs to this process are:~~

* ~~a luma location ( xCb, yCb ) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,~~
* ~~a variable nCbS specifying the size of the current luma coding block,~~
* ~~a luma location ( xPb, yPb ) specifying the top-left sample of the current luma prediction block relative to the top-left luma sample of the current picture,~~
* ~~two variables nPbW and nPbH specifying the width and the height of the luma prediction block,~~
* ~~a variable partIdx specifying the index of the current prediction unit within the current coding unit.~~

~~Outputs of this process are (with N being replaced by A or B):~~

* ~~the block vectors bvIntraN of the neighbouring prediction units.~~
* ~~the availability flags availableFlagN of the neighbouring prediction units.~~

~~The variables bvIntraA[ compIdx ] specify the left neighbouring block vector predictor with compIdx being 0 or 1. The horizontal block vector component is assigned compIdx = 0 and the vertical block vector component is assigned compIdx = 1. The variable availableFlagN specifies the availability flags of the left and above neighbouring blocks, with N being equal to A or B. bvIntraN[ compIdx ] and availableFlagN are derived as follows:~~

~~bvIntraN[ compIdx ] is set equal to 0 for compIdx being equal to 0 and 1 and N being equal to A and B;~~

~~availableFlagN is set equal to FALSE for N being equal to A and B.~~

~~The availability derivation process for a prediction block as specified in subclause  is invoked with the luma location ( xCb, yCb ), the current luma coding block size nCbS, the luma prediction block location ( xPb, yPb ), the luma prediction block width nPbW, the luma prediction block height nPbH, the luma location ( xPb − 1 , yPb + nPbH − 1 ), and the partition index partIdx as inputs, if the output is equal to TRUE, availableFlagA is set equal to TRUE, and bvIntraA is set equal to bvIntra[ xPb − 1 ][ yPb + nPbH − 1 ]~~

~~[Ed. (GJS): Check usage of TRUE vs. true (generally).]~~

~~[Ed. (GJS): Check "is set to" versus "is set equal to" (generall).]~~

~~The availability derivation process for a prediction block as specified in subclause  is invoked with the luma location ( xCb, yCb ), the current luma coding block size nCbS, the luma prediction block location ( xPb, yPb ), the luma prediction block width nPbW, the luma prediction block height nPbH, the luma location ( xPb − nPbW − 1 , yPb − 1 ), and the partition index partIdx as inputs. If the output is equal to TRUE, and (yPb/CtbSizeY) is equal to ( ( yPb − 1 ) / CtbSizeY ), availableFlagB is set equal to TRUE, and bvIntraB is set equal to bvIntra[ xPb − nPbW − 1 ][ yPb − 1 ]~~

#### 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 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.

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:

( xTbY, yTbY ) = ( cIdx  = =  0 ) ? ( xTb0, yTb0 ) : ( xTb0 \* SubWidthC, yTb0 \* SubHeightC ) (8‑26)

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:
   * If either cIdx is equal to 0 or ChromaArrayType is not equal to 2, the following applies:
   * The variable xTb1 is set equal to xTb0 + ( 1  <<  ( log2TrafoSize − 1 ) ).
   * The variable yTb1 is set equal to yTb0 + ( 1  <<  ( log2TrafoSize − 1 ) ).
   * Otherwise (ChromaArrayType is equal to 2 and cIdx is greater than 0), the following applies:
   * The variable xTb1 is set equal to xTb0 + ( 1  <<  ( log2TrafoSize − 1 ) ).
   * The variable yTb1 is set equal to yTb0 + ( 2  <<  ( 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), for the variable blkIdx proceeding over the values 0..( cIdx > 0  &&  ChromaArrayType  = =  2 ? 1 : 0 ), the following ordered steps apply:

1. The variable nTbS is set equal to 1  <<  log2TrafoSize.
2. The variable yTbOffset is set equal to blkIdx \* nTbS.
3. The variable yTbOffsetY is set equal to yTbOffset \* SubHeightC.
4. When controlParaACT is not equal to 2, the variable residualDpcm is derived as follows:
   * If all of the following conditions are true, residualDpcm is set equal to 1.
   * implicit\_rdpcm\_enabled\_flag is equal to 1.
   * either transform\_skip\_flag[ xTbY ][ yTbY + yTbOffsetY ][ cIdx ] is equal to 1, or cu\_transquant\_bypass\_flag is equal to 1.
   * either predModeIntra is equal to 10, or predModeIntra is equal to 26.
   * Otherwise, residualDpcm is set equal to explicit\_rdpcm\_flag[ xTbY ][ yTbY + yTbOffsetY ][ cIdx ].
5. When controlParaACT is not equal to 1, ~~depending upon the value of predModeIntraBc,~~ the following applies:

– ~~When predModeIntraBc 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 (predModeIntraBc is equal to 1), the intra block copying process as specified in subclause  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.

##### ~~8.4.5.2.7 Specification of intra block copying 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 variable bvIntra specifying the block copying vector,~~

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

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

~~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:~~

~~( xTbY, yTbY ) = ( cIdx  = =  0 ) ? ( xTb0, yTb0 ) : ( xTb0 \* SubWidthC, yTb0 \* SubHeightC ) (8‑62)~~

~~Depending upon the values of trafoDepth, PartMode and nTbS, the following applies:~~

~~– If trafoDepth is equal to 0, PartMode is not equal to PART\_2Nx2N, and nTbS is greater than 4, the following applies, for the variable tempIdx proceeding over the values 0..3:~~

~~– The variable nTbS1 is set equal to nTbS / 2.~~

~~– The variable xTb1 is set equal to xTb0 + nTbS1 \* ( blkIdx % 2 ).~~

~~– The variable yTb1 is set equal to yTb0 + nTbS1 \* ( blkIdx / 2 ).~~

~~– The general intra block copying process as specified in this subclause is invoked with the location ( xTb1, yTb1 ), the variable nTbS set equal to nTbS1, the variable bvIntra, the variable trafoDepth is set equal to 1, and the variable cIdx as inputs, and the output is an (nTbS1) x (nTbS1) array predSamples. [Ed.: This should be tempSamples, and then copied into predSamples]~~

~~– Otherwise, the variable bv representing the block vector for prediction in full-sample units is derived as follows:~~

~~– If cIdx is not equal to 0, trafoDepth is equal to 0, and nTbS is equal to 4, the following applies:~~

~~– If ChromaArrayTyp is equal to 1, bv is set equal to bvIntra[ xTbY + 4 ][ yTbY + 4 ].~~

~~– Otherwise, if ChromaArrayType is equal to 2, bv is set equal to bvIntra[ xTbY + 4 ][ yTbY ].~~

~~– bv[ 0 ] = Max( −( xTbY % CtbSizeY + CtbSizeY ), bv[ 0 ] ) >> ( SubWidthC − 1 )~~

~~– bv[ 1 ] = Max( −( yTbY % CtbSizeY ), bv[ 1 ] ) >> ( SubHeightC − 1 )~~

~~– Otherwise, the following applies:~~

~~bv[ 0 ] = bvIntra[ xTbY ][ yTbY ][ 0 ] >> ( ( ( cIdx = = 0 ) ? 1 : SubWidthC ) − 1 ) (8‑63)~~

~~bv[ 1 ] = bvIntra[ xTbY ][ yTbY ][ 1 ] >> ( ( ( cIdx = = 0 ) ? 1 : SubHeightC ) − 1 ) (8‑64)~~

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

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

~~( 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 ]~~

##### 8.5.3.2.8 Derivation process for temporal luma motion vector prediction

Inputs to this process are:

* a luma location ( xPb, yPb ) specifying the top-left sample of the current luma prediction block relative to the top-left luma sample of the current picture,
* two variables nPbW and nPbH specifying the width and the height of the luma prediction block,
* a reference index refIdxLX, with X being 0 or 1.

Outputs of this process are:

* the motion vector prediction mvLXCol,
* the availability flag availableFlagLXCol.

The variable currPb specifies the current luma prediction block at luma location ( xPb, yPb ).

The variables mvLXCol and availableFlagLXCol are derived as follows:

* If slice\_temporal\_mvp\_enabled\_flag is equal to 0 or current slice type is I, both components of mvLXCol are set equal to 0 and availableFlagLXCol is set equal to 0.
* Otherwise, the following ordered steps apply:

……

#### 8.5.3.3 Decoding process for inter prediction samples

##### 8.5.3.3.1 General

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 luma location ( xBl, yBl ) specifying the top-left sample of the current luma prediction block relative to the top-left sample of the current luma coding block,

– a variable nCbS specifying the size of the current luma coding block,

– two variables nPbW and nPbH specifying the width and the height of the luma prediction block,

– the luma motion vectors mvL0 and mvL1,

– when ChromaArrayType is not equal to 0, the chroma motion vectors mvCL0 and mvCL1,

– the reference indices refIdxL0 and refIdxL1,

– the prediction list utilization flags, predFlagL0, and predFlagL1.

Outputs of this process are:

– an (nCbSL)x(nCbSL) array predSamplesL of luma prediction samples, where nCbSL is derived as specified below,

– when ChromaArrayType is not equal to 0, an (nCbSwC)x(nCbShC) array predSamplesCb of chroma prediction samples for the component Cb, where nCbSwC and nCbShC are derived as specified below,

– when ChromaArrayType is not equal to 0, an (nCbSwC)x(nCbShC) array predSamplesCr of chroma residual samples for the component Cr, where nCbSwC and nCbShC are derived as specified below.

If the reference picture in L0 is the current picture, it is a requirement of bitstream conformance that all of the the following conditions are true:

– When the derivation process for z-scan order block availability as specified in subclause  is invoked with ( xCurr, yCurr ) set equal to ( xCb, yCb ) and the neighbouring luma location ( xNbY, yNbY ) set equal to ( xBl + mvL0[ 0 ], yBl + mvL0[ 1 ] ) as inputs, the output shall be equal to TRUE.

– When the derivation process for z-scan order block availability as specified in subclause  is invoked with ( xCurr, yCurr ) set equal to ( xCb, yCb ) and the neighbouring luma location ( xNbY, yNbY ) set equal to ( xBl + mvL0[ 0 ] + nPbW − 1, yBl + mvL0[ 1 ] + nPbH – 1 ) as inputs, the output shall be equal to TRUE.

If the reference picture in L0 is the current picture, the mvL0 and mvCL0 are modified as mvL0 <<= 2, mvCL0 <<= 2.

The variable nCbSL is set equal to nCbS. When ChromaArrayType is not equal to 0, the variable nCbSwC is set equal to nCbS / SubWidthC and the variable nCbShC is set equal to nCbS / SubHeightC.

Let predSamplesL0L and predSamplesL1L be (nPbW)x(nPbH) arrays of predicted luma sample values and when ChromaArrayType is not equal to 0, predSamplesL0Cb, predSamplesL1Cb, predSamplesL0Cr, and predSamplesL1Cr be (nPbW / SubWidthC)x(nPbH / SubHeightC) arrays of predicted chroma sample values.

……

#### 9.3.2.2 Initialization process for context variables

……

In Table 9‑4, the ctxIdx for which initialization is needed for each of the three initialization types, specified by the variable initType, are listed. Also listed is the table number that includes the values of initValue needed for the initialization. For P and B slice types, the derivation of initType depends on the value of the cabac\_init\_flag syntax element. The variable initType is derived as follows:

if( slice\_type = = I && !intra\_block\_copy\_enabled\_flag )  
 initType = 0  
else if (slice\_type = = I && intra\_block\_copy\_enabled\_flag )  
 initType = 1  
else if( slice\_type = = P )  
 initType = cabac\_init\_flag ? 2 : 1 (‑)  
else  
 initType = cabac\_init\_flag ? 1 : 2

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 |  | 0 | 1 | 2 |
| sao\_type\_idx\_luma sao\_type\_idx\_chroma |  | 0 | 1 | 2 |
| coding\_quadtree( ) | split\_cu\_flag[ ][ ] |  | 0..2 | 3..5 | 6..8 |
| coding\_unit( ) | cu\_transquant\_bypass\_flag |  | 0 | 1 | 2 |
| cu\_skip\_flag |  |  | 0..2 | 3..5 |
| ~~intra\_bc\_flag[ ][ ]~~ |  | ~~0~~ | ~~1~~ | ~~2~~ |
| palette\_mode\_flag[ ][ ] |  | 0 | 1 | 2 |
| pred\_mode\_flag |  |  | 0 | 1 |
| part\_mode |  | 0 | 1..4 | 5..8 |
| prev\_intra\_luma\_pred\_flag[ ][ ] |  | 0 | 1 | 2 |
| intra\_chroma\_pred\_mode[ ][ ] |  | 0 | 1 | 2 |
| rqt\_root\_cbf |  |  | 0 | 1 |
| cu\_residual\_act\_flag |  | 0 | 1 | 2 |
| prediction\_unit( ) | merge\_flag[ ][ ] |  |  | 0 | 1 |
| merge\_idx[ ][ ] |  |  | 0 | 1 |
| intra\_bc\_flag[ ][ ] |  |  | 1 | 2 |
| inter\_pred\_idc[ ][ ] |  |  | 0..4 | 5..9 |
| ref\_idx\_l0[ ][ ], ref\_idx\_l1[ ][ ] |  |  | 0..1 | 2..3 |
| mvp\_l0\_flag[ ][ ], mvp\_l1\_flag[ ][ ], bvp\_flag[ ][ ] |  |  | 0 | 1 |

| Table 9‑38 – Syntax elements and associated binarizations | | | |
| --- | --- | --- | --- |
| **Syntax structure** | **Syntax element** | **Binarization** | |
| **Process** | **Input parameters** |
| coding\_quadtree( ) | split\_cu\_flag[ ][ ] | FL | cMax = 1 |
| coding\_unit( ) | cu\_transquant\_bypass\_flag | FL | cMax = 1 |
| cu\_skip\_flag | FL | cMax = 1 |
| ~~intra\_bc\_flag~~ | ~~FL~~ | ~~cMax = 1~~ |
| palette\_mode\_flag | FL | cMax = 1 |
| …… |  |  |
| palette\_coding( ) | previous\_palette\_entry\_flag[] | FL | cMax = 1 |
| …… |  |  |
| prediction\_unit( ) | merge\_flag[ ][ ] | FL | cMax = 1 |
| merge\_idx[ ][ ] | TR | cMax = MaxNumMergeCand − 1, cRiceParam = 0 |
| intra\_bc\_flag | FL | cMax = 1 |
| inter\_pred\_idc[ x0 ][ y0 ] |  | nPbW, nPbH |
| ref\_idx\_l0[ ][ ] | TR | cMax = num\_ref\_idx\_l0\_active\_minus1, cRiceParam = 0 |
| mvp\_l0\_flag[ ][ ] | FL | cMax = 1 |
| ref\_idx\_l1[ ][ ] | TR | cMax = num\_ref\_idx\_l1\_active\_minus1, cRiceParam = 0 |
| mvp\_l1\_flag[ ][ ] | FL | cMax = 1 |
| bvp\_flag[ ][ ] | FL | cMax = 1 |

Table 9‑38 – Values of initValue for intra\_bc\_flag

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of intra\_bc\_flag** | | |
| **~~0~~** | **1** | **2** |
| **initValue** | ~~185~~ | 197 | 197 |