**Draft Text Specification**

The proposed text changes are based on the document JCTVC-R1005-v3.doc for the PU level IntraBC signalling in CE2 Test 2. The changes are marked in yellow. The text changes for CE2 Test3a and Test5b combined test is marked in blue.

**Syntax**

#### 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\_bc\_flag[ x0 ][ y0 ]~~ | | intra\_block\_copy\_enabled\_flag) |  |
| **pred\_mode\_flag** | ae(v) |
| if( palette\_enabled\_flag && ChromaArrayType = = 3 &&   CuPredMode[ x0 ][ y0 ] = = MODE\_INTRA ~~&& !intra\_bc\_flag[ x0 ][ y0 ]~~) |  |
| **palette\_mode\_flag**[ x0 ][ y0 ] | ae(v) |
| if( palette\_mode\_flag[ x0 ][ y0 ] ) |  |
| palette\_coding( x0, y0, nCbS ) |  |
| else { |  |
| if( CuPredMode[ x0 ][ y0 ] != MODE\_INTRA | | ~~intra\_bc\_flag[ x0 ][ y0 ] | |~~  log2CbSize = = MinCbLog2SizeY ) |  |
| **part\_mode** | ae(v) |
| if( CuPredMode[ x0 ][ y0 ] = = MODE\_INTRA ~~&& !intra\_bc\_flag[ x0 ][ y0 ]~~) { |  |
| if( PartMode = = PART\_2Nx2N && pcm\_enabled\_flag &&   log2CbSize >= Log2MinIpcmCbSizeY &&  log2CbSize <= Log2MaxIpcmCbSizeY ) |  |
| … |  |

#### 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 )~~ |  |
| **~~bvp\_flag~~**~~[ x0 ][ y0 ]~~ | ~~ae(v)~~ |
| } else { /\* MODE\_INTER \*/ |  |
| if( intra\_block\_copy\_enabled\_flag && slice\_type != I && ( PartMode < PART\_NxN | | MinCbLog2SizeY > 3 && PartMode = = PART\_NxN ))) |  |
| **intra\_bc\_flag**[ x0 ][ y0 ] | ae(v) |
| **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 && ( PartMode < PART\_NxN | | MinCbLog2SizeY > 3 && PartMode = = PART\_NxN )))~~ |  |
| **~~intra\_bc\_flag~~**~~[ x0 ][ y0 ]~~ | ~~ae(v)~~ |
| if(intra\_bc\_flag[ x0 ][ y0 ]){ /\* Intra BC\*/ |  |
| bvd\_coding( x0, y0, 2 ) |  |
| **bvp\_flag**[ x0 ][ y0 ] | ae(v) |
| }else{ |  |
| if( slice\_type = = B ) |  |
| **inter\_pred\_idc**[ x0 ][ y0 ] | ae(v) |
| if( inter\_pred\_idc[ x0 ][ y0 ] != PRED\_L1 ) { |  |
| if( num\_ref\_idx\_l0\_active\_minus1 > 0 ) |  |
| **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) |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |

**Semantics**

#### 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, i~~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.

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

The value of part\_mode is restricted as follows:

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

……

#### Prediction unit semantics

**intra\_bc\_flag**[ x0 ][ y0 ] equal to 1 specifies that the current prediction unit is coded in intra block copying mode. 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

**Decoding process**

## Decoding process for coding units coded in intra prediction mode

### General decoding process for coding units coded in intra prediction mode

……

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

1. The derivation process for the intra prediction mode as specified in subclause 8.4.2 is invoked with the luma location ( xCb, yCb ) as input.
2. ~~When intra\_bc\_flag[ xCb ][ yCb ] is equal to 1, the derivation process for block vector components in intra block copying prediction mode as specified in subclause 8.4.4 is invoked with the luma location ( xCb, yCb ) and variable log2CbSize as inputs, and the output being bvIntra.~~
3. If cu\_residual\_act\_flag[ xCb ][ yCb ] is equal to 1, the following applies:

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

* + The variable comp is set equal to (!cIdx ? L : (cIdx = =1 ? Cb : Cr). [Ed. (GJS): Suggest reformulating to avoid this usage formulation and this variable name.]
  + The general decoding process for intra blocks as specified in subclause 8.4.4.1 is invoked with the location ( xCb, yCb ), the variable log2TrafoSize set equal to log2CbSize, the variable trafoDepth set equal to 0, the variable predModeIntra set equal to IntraPredModeY[ xCb ][ yCb ], ~~the variable predModeIntraBc set equal to intra\_bc\_flag[ xCb ][ yCb ], the variable bvIntra,~~ the variable cIdx, and variable controlParaACT equal to 1 as inputs, and the output is the residual sample array resSamplescomp.

……

* Otherwise (pcm\_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. 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).
  + 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,~~ the variable cIdx, and variable controlParaACT set equal to 1 as inputs, and the output is the residual sample array resSamplescomp.

……

### 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 luma location ( xPb, yPb ) of the top-left sample of the current luma prediction block relative to the top-left sample of the current picture,
* the luma coding block size block nCbS,
* two variables nPbSw and nPbSh specifying the width and the height of the luma prediction block,a variable log2CbSize specifying the size of the current luma coding block.

Output of this process is ~~the (nCbS)x(nCbS)~~ (nPbSw)x(nPbSh) array 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 ):~~

* ~~The variable blkInc is set equal to ( PartMode = = PART\_2NxN ? 2 : 1 ).~~
* ~~The variable xPb is set equal to xCb + nPbSw \* ( blkIdx \* blkInc % 2 ).~~
* ~~The variable yPb is set equal to yCb + nPbSh \* ( blkIdx / 2 )~~

1. The following ordered steps apply, for the variable compIdx proceeding over the values 0..1:
2. 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 8.4.4.1 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 ]

……

### Decoding process for intra blocks

#### 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

……

1. 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 8.4.4.2.7 is invoked with the transform block location ( xTb0, yTb0 + yTbOffset ), the transform block size nTbS, the variable trafoDepth, the variable bvIntra, and the variable cIdx as inputs, and the output is an (nTbS)x(nTbS) array predSamples.~~

……

### Derivation process for motion vector components and reference indices

#### 8.5.3.2.1 General

……

For the derivation of the variables mvL0 and mvL1, refIdxL0 and refIdxL1, as well as predFlagL0 and predFlagL1, the following applies:

* If merge\_flag[ xPb ][ yPb ] is equal to 1, the derivation process for luma motion vectors for merge mode as specified in subclause 8.5.3.2.2 is invoked with the luma location ( xCb, yCb ), the luma location ( xPb, yPb ), the variables nCbS, nPbW, nPbH, and the partition index partIdx as inputs, and the output being the luma motion vectors mvL0, mvL1, the reference indices refIdxL0, refIdxL1, and the prediction list utilization flags predFlagL0 and predFlagL1.
* Otherwise, if intra\_bc\_flag[ xPb + x ][ yPb + y ] is equal to 1, The derivation process for block vector components as specified in subclause 8.4.4 is invoked with the luma coding block location ( xCb, yCb ), the luma prediction block location ( xPb, yPb ), the luma coding block size block nCbS, the luma prediction block width nPbW and the luma prediction block height nPbH as inputs, and the block vector array bvIntra as output.
* Otherwise, for X being replaced by either 0 or 1 in the variables predFlagLX, mvLX, and refIdxLX, in PRED\_LX, and in the syntax elements ref\_idx\_lX and MvdLX, the following applies:

……

When ChromaArrayType is not equal to 0 and predFlagLX, with X being 0 or 1, is equal to 1, the derivation process for chroma motion vectors in subclause 8.5.3.2.10 is invoked with mvLX as input, and the output being mvCLX.

~~If merge\_flag[ xPb ][ yPb ] is equal to 1 and the variable intra\_bc\_flag for the chosen merge candidate is equal to 1, intraBCFlag is set equal to 1; otherwise, intraBCFlag is set equal to 0. the following assignments are made for x = 0..nPbW − 1 and y = 0..nPbH − 1:~~

~~intra\_bc\_flag[ xPb + x ][ yPb + y ] = intraBCFlag (8‑xx)~~

~~If intraBCFlag is equal to 1,~~

~~bvIntra[ xPb + x ][ yPb + y ] = mvL0 (8‑xx)~~

##### Derivation process for luma motion vectors for merge mode

This process is only invoked when merge\_flag[ xPb ][ yPb ] is equal to 1, where ( xPb, yPb ) specify the top-left sample of the current luma prediction block relative to the top-left luma sample of the current picture.

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 luma location ( xPb, yPb ) of the top-left sample of the current luma prediction block relative to the top-left luma sample of the current picture,
* 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,
* a variable partIdx specifying the index of the current prediction unit within the current coding unit.

Outputs of this process are:

– the luma motion vectors mvL0 and mvL1,

– the reference indices refIdxL0 and refIdxL1,

– the prediction list utilization flags predFlagL0 and predFlagL1.

The location ( xOrigP, yOrigP ) and the variables nOrigPbW and nOrigPbH are derived to store the values of ( xPb, yPb ), nPbW, and nPbH as follows:

( xOrigP, yOrigP ) is set equal to ( xPb, yPb ) (8‑79)

nOrigPbW = nPbW (8‑80)

nOrigPbH = nPbH (8‑81)

When Log2ParMrgLevel is greater than 2 and nCbS is equal to 8, ( xPb, yPb ), nPbW, nPbH, and partIdx are modified as follows:

( xPb, yPb ) = ( xCb, yCb ) (8‑82)

nPbW = nCbS (8‑83)

nPbH = nCbS (8‑84)

partIdx = 0 (8‑85)

NOTE – When Log2ParMrgLevel is greater than 2 and nCbS is equal to 8, all the prediction units of the current coding unit share a single merge candidate list, which is identical to the merge candidate list of the 2Nx2N prediction unit.

The motion vectors mvL0 and mvL1, the reference indices refIdxL0 and refIdxL1, and the prediction utilization flags predFlagL0 and predFlagL1 are derived by the following ordered steps:

1. The derivation process for merging candidates from neighbouring prediction unit partitions in subclause 8.5.3.2.3 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 output being the availability flags availableFlagA0, availableFlagA1, availableFlagB0, availableFlagB1, and availableFlagB2, the reference indices refIdxLXA0, refIdxLXA1, refIdxLXB0, refIdxLXB1, and refIdxLXB2, the prediction list utilization flags predFlagLXA0, predFlagLXA1, predFlagLXB0, predFlagLXB1, and predFlagLXB2, and the motion vectors mvLXA0, mvLXA1, mvLXB0, mvLXB1, and mvLXB2, with X being 0 or 1.
2. If intra\_bc\_flag[xPb][yPb] is equal to 0, ~~The~~ the reference indices for the temporal merging candidate, refIdxLXCol, with X being 0 or 1, are set equal to 0; otherwise, refIdxL0Col is set to (num\_ref\_idx\_l0\_active\_minus1+1), refIdxL1Col is set to -1.
3. The derivation process for temporal luma motion vector prediction in subclause 8.5.3.2.8 is invoked with the luma location ( xPb, yPb ), the luma prediction block width nPbW, the luma prediction block height nPbH, and the variable refIdxL0Col as inputs, and the output being the availability flag availableFlagL0Col and the temporal motion vector mvL0Col.The variables availableFlagCol, predFlagL0Col and predFlagL1Col are derived as follows:

availableFlagCol = availableFlagL0Col (8‑86)

predFlagL0Col = availableFlagL0Col (8‑87)

predFlagL1Col = 0 (8‑88)

1. When slice\_type is equal to B and intra\_bc\_flag[xPb][yPb] is equal to 0, the derivation process for temporal luma motion vector prediction in subclause 8.5.3.2.8 is invoked with the luma location ( xPb, yPb ), the luma prediction block width nPbW, the luma prediction block height nPbH, and the variable refIdxL1Col as inputs, and the output being the availability flag availableFlagL1Col and the temporal motion vector mvL1Col. The variables availableFlagCol and predFlagL1Col are derived as follows:

availableFlagCol = availableFlagL0Col  | |  availableFlagL1Col (8‑89)

predFlagL1Col = availableFlagL1Col (8‑90)

1. The merging candidate list, mergeCandList, is constructed as follows:

i = 0  
if( availableFlagA1 )  
 mergeCandList[ i++ ] = A1  
if( availableFlagB1 )  
 mergeCandList[ i++ ] = B1if( availableFlagB0 )  
 mergeCandList[ i++ ] = B0 (8‑91)if( availableFlagA0 )  
 mergeCandList[ i++ ] = A0if( availableFlagB2 )  
 mergeCandList[ i++ ] = B2if( availableFlagCol )  
 mergeCandList[ i++ ] = Col

1. The variable numCurrMergeCand and numOrigMergeCand are set equal to the number of merging candidates in the mergeCandList.
2. When intra\_bc\_flag[xPb][yPb] is equal to 0 and slice\_type is equal to B, the derivation process for combined bi-predictive merging candidates specified in subclause 8.5.3.2.4 is invoked with mergeCandList, the reference indices refIdxL0N and refIdxL1N, the prediction list utilization flags predFlagL0N and predFlagL1N, the motion vectors mvL0N and mvL1N of every candidate N in mergeCandList, numCurrMergeCand, and numOrigMergeCand as inputs, and the output is assigned to mergeCandList, numCurrMergeCand, the reference indices refIdxL0combCandk and refIdxL1combCandk, the prediction list utilization flags predFlagL0combCandk and predFlagL1combCandk, and the motion vectors mvL0combCandk and mvL1combCandk of every new candidate combCandk being added into mergeCandList. The number of candidates being added, numCombMergeCand, is set equal to ( numCurrMergeCand − numOrigMergeCand ). When numCombMergeCand is greater than 0, k ranges from 0 to numCombMergeCand − 1, inclusive.
3. When intra\_bc\_flag[xPb][yPb] is equal to 0, ~~The~~the derivation process for zero motion vector merging candidates specified in subclause 8.5.3.2.5 is invoked with the mergeCandList, the reference indices refIdxL0N and refIdxL1N, the prediction list utilization flags predFlagL0N and predFlagL1N, the motion vectors mvL0N and mvL1N of every candidate N in mergeCandList, and numCurrMergeCand as inputs, and the output is assigned to mergeCandList, numCurrMergeCand, the reference indices refIdxL0zeroCandm and refIdxL1zeroCandm, the prediction list utilization flags predFlagL0zeroCandm and predFlagL1zeroCandm, and the motion vectors mvL0zeroCandm and mvL1zeroCandm of every new candidate zeroCandm being added into mergeCandList. The number of candidates being added, numZeroMergeCand, is set equal to ( numCurrMergeCand − numOrigMergeCand − numCombMergeCand ). When numZeroMergeCand is greater than 0, m ranges from 0 to numZeroMergeCand − 1, inclusive.
4. When intra\_bc\_flag[xPb][yPb] is equal to 1 and numOrigMergeCand is greater than 0, the derivation process for derived IntraBC merging candidates specified in subclause 8.5.3.2.11 is invoked with the luma coding block location ( xCb, yCb ), the luma prediction block location ( xPb, yPb ), the luma prediction block width nPbW, the luma prediction block height nPbH, mergeCandList, the reference indices refIdxL0N and refIdxL1N, the prediction list utilization flags predFlagL0N and predFlagL1N, the motion vectors mvL0N and mvL1N of every candidate N in mergeCandList, numCurrMergeCand as inputs, and the output is assigned to mergeCandList, numCurrMergeCand, the reference indices refIdxL0derivedCandk and refIdxL1derivedCandk, the prediction list utilization flags predFlagL0derivedCandk and predFlagL1derivedCandk, and the motion vectors mvL0derivedCandk and mvL1derivedCandk of every new candidate derivedCandk being added into mergeCandList. The number of candidates being added, numDerivedMergeCand, is set equal to ( numCurrMergeCand − numOrigMergeCand ). When numDerivedMergeCand is greater than 0, k ranges from 0 to numDerivedMergeCand − 1, inclusive.
5. The following assignments are made with N being the candidate at position merge\_idx[ xOrigP ][ yOrigP ] in the merging candidate list mergeCandList ( N = mergeCandList[ merge\_idx[ xOrigP ][ yOrigP ] ] ) and X being replaced by 0 or 1:

mvLX[ 0 ] = mvLXN[ 0 ] (8‑92)

mvLX[ 1 ] = mvLXN[ 1 ] (8‑93)

refIdxLX = refIdxLXN (8‑94)

predFlagLX = predFlagLXN (8‑95)

1. When predFlagL0 is equal to 1 and predFlagL1 is equal to 1, and ( nOrigPbW + nOrigPbH ) is equal to 12, the following applies:

refIdxL1 = −1 (8‑96)

predFlagL1 = 0 (8‑97)

##### Derivation process for spatial merging 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 as follows, with X being 0 or 1:

* the availability flags availableFlagA0, availableFlagA1, availableFlagB0, availableFlagB1, and availableFlagB2 of the neighbouring prediction units,
* the reference indices refIdxLXA0, refIdxLXA1, refIdxLXB0, refIdxLXB1, and refIdxLXB2 of the neighbouring prediction units,
* the prediction list utilization flags predFlagLXA0, predFlagLXA1, predFlagLXB0, predFlagLXB1, and predFlagLXB2 of the neighbouring prediction units,
* the motion vectors mvLXA0, mvLXA1, mvLXB0, mvLXB1, and mvLXB2 of the neighbouring prediction units.

For the derivation of availableFlagA1, refIdxLXA1, predFlagLXA1, and mvLXA1 the following applies:

– If intra\_bc\_flag[xPb][yPb] is equal to 0, ~~The~~the luma location ( xNbA1, yNbA1 ) inside the neighbouring luma coding block is set equal to ( xPb − 1,  yPb + nPbH − 1 ); otherwise the luma location ( xNbA1, yNbA1 ) inside the neighbouring luma coding block is set equal to ( xPb − 1,  yPb ).

– The availability derivation process for a prediction block as specified in subclause 6.4.2 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 ( xNbA1, yNbA1 ), and the partition index partIdx as inputs, and the output is assigned to the prediction block availability flag availableA1.

– When one or more of the following conditions are true, availableA1 is set equal to FALSE:

* xPb  >>  Log2ParMrgLevel is equal to xNbA1  >>  Log2ParMrgLevel and  yPb  >>  Log2ParMrgLevel is equal to yNbA1  >>  Log2ParMrgLevel.
* PartMode of the current prediction unit is equal to PART\_Nx2N, PART\_nLx2N, or PART\_nRx2N, and partIdx is equal to 1, and intra\_bc\_flag[xPb][yPb] is equal to 0.
* intra\_bc\_flag[xNbA1][ yNbA1] is not equal to intra\_bc\_flag[xPb][yPb].

– The variables availableFlagA1, refIdxLXA1, predFlagLXA1, and mvLXA1 are derived as follows:

* If availableA1 is equal to FALSE, availableFlagA1 is set equal to 0, both components of mvLXA1 are set equal to 0, refIdxLXA1 is set equal to −1 and predFlagLXA1 is set equal to 0, with X being 0 or 1.
* Otherwise, availableFlagA1 is set equal to 1 and the following assignments are made:

mvLXA1 = MvLX[ xNbA1 ][ yNbA1 ] (8‑98)

refIdxLXA1 = RefIdxLX[ xNbA1 ][ yNbA1 ] (8‑99)

predFlagLXA1 = PredFlagLX[ xNbA1 ][ yNbA1 ] (8‑100)

For the derivation of availableFlagB1, refIdxLXB1, predFlagLXB1, and mvLXB1 the following applies:

– If intra\_bc\_flag[xPb][yPb] is equal to 0, ~~The~~the luma location ( xNbB1, yNbB1 ) inside the neighbouring luma coding block is set equal to ( xPb + nPbW − 1,  yPb − 1 ); otherwise the luma location ( xNbB1, yNbB1 ) inside the neighbouring luma coding block is set equal to ( xPb,  yPb − 1 ).

– The availability derivation process for a prediction block as specified in subclause 6.4.2 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 ( xNbB1, yNbB1 ), and the partition index partIdx as inputs, and the output is assigned to the prediction block availability flag availableB1.

– When one or more of the following conditions are true, availableB1 is set equal to FALSE:

* xPb  >>  Log2ParMrgLevel is equal to xNbB1  >>  Log2ParMrgLevel and yPb  >>  Log2ParMrgLevel is equal to yNbB1  >>  Log2ParMrgLevel.
* PartMode of the current prediction unit is equal to PART\_2NxN, PART\_2NxnU, or PART\_2NxnD, and partIdx is equal to 1, and intra\_bc\_flag[xPb][yPb] is equal to 0.
* intra\_bc\_flag[xNbB1][ yNbB1] is not equal to intra\_bc\_flag[xPb][yPb].

– The variables availableFlagB1, refIdxLXB1, predFlagLXB1, and mvLXB1 are derived as follows:

* If one or more of the following conditions are true, availableFlagB1 is set equal to 0, both components of mvLXB1 are set equal to 0, refIdxLXB1 is set equal to −1, and predFlagLXB1 is set equal to 0, with X being 0 or 1:
  + - availableB1 is equal to FALSE.
    - availableA1 is equal to TRUE and the prediction units covering the luma locations ( xNbA1, yNbA1 ) and ( xNbB1, yNbB1 ) have the same motion vectors and the same reference indices.
* Otherwise, availableFlagB1 is set equal to 1 and the following assignments are made:

mvLXB1 = MvLX[ xNbB1 ][ yNbB1 ] (8‑101)

refIdxLXB1 = RefIdxLX[ xNbB1 ][ yNbB1 ] (8‑102)

predFlagLXB1 = PredFlagLX[ xNbB1 ][ yNbB1 ] (8‑103)

For the derivation of availableFlagB0, refIdxLXB0, predFlagLXB0, and mvLXB0 the following applies:

– The luma location ( xNbB0, yNbB0 ) inside the neighbouring luma coding block is set equal to ( xPb + nPbW,  yPb − 1 ).

– The availability derivation process for a prediction block as specified in subclause 6.4.2 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 ( xNbB0, yNbB0 ), and the partition index partIdx as inputs, and the output is assigned to the prediction block availability flag availableB0.

– When xPb  >>  Log2ParMrgLevel is equal to xNbB0  >>  Log2ParMrgLevel and yPb  >>  Log2ParMrgLevel is equal to yNbB0  >>  Log2ParMrgLevel, availableB0 is set equal to FALSE.

– When intra\_bc\_flag[xNbB0][ yNbB0] is not equal to intra\_bc\_flag[xPb][yPb], availableB0 is set equal to FALSE.

– The variables availableFlagB0, refIdxLXB0, predFlagLXB0, and mvLXB0 are derived as follows:

* If one or more of the following conditions are true, availableFlagB0 is set equal to 0, both components of mvLXB0 are set equal to 0, refIdxLXB0 is set equal to −1, and predFlagLXB0 is set equal to 0, with X being 0 or 1:
  + - availableB0 is equal to FALSE.
    - availableB1 is equal to TRUE and the prediction units covering the luma locations ( xNbB1, yNbB1 ) and ( xNbB0, yNbB0 ) have the same motion vectors and the same reference indices.
    - intra\_bc\_flag[xPb][yPb] is equal to 1 and availableA1 is equal to TRUE and the prediction units covering the luma locations ( xNbA1, yNbA1 ) and ( xNbB0, yNbB0 ) have the same motion vectors and the same reference indices.
* Otherwise, availableFlagB0 is set equal to 1 and the following assignments are made:

mvLXB0 = MvLX[ xNbB0 ][ yNbB0 ] (8‑104)

refIdxLXB0 = RefIdxLX[ xNbB0 ][ yNbB0 ] (8‑105)

predFlagLXB0 = PredFlagLX[ xNbB0 ][ yNbB0 ] (8‑106)

For the derivation of availableFlagA0, refIdxLXA0, predFlagLXA0, and mvLXA0 the following applies:

– The luma location ( xNbA0, yNbA0 ) inside the neighbouring luma coding block is set equal to ( xPb − 1,  yPb + nPbH ).

– The availability derivation process for a prediction block as specified in subclause 6.4.2 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 ( xNbA0, yNbA0 ), and the partition index partIdx as inputs, and the output is assigned to the prediction block availability flag availableA0.

– When xPb  >>  Log2ParMrgLevel is equal to xNbA0  >>  Log2ParMrgLevel and yPb  >>  Log2ParMrgLevel is equal to yA0  >>  Log2ParMrgLevel, availableA0 is set equal to FALSE.

– When intra\_bc\_flag[xNbA0][ yNbA0] is not equal to intra\_bc\_flag[xPb][yPb], availableA0 is set equal to FALSE.

– The variables availableFlagA0, refIdxLXA0, predFlagLXA0, and mvLXA0 are derived as follows:

* If one or more of the following conditions are true, availableFlagA0 is set equal to 0, both components of mvLXA0 are set equal to 0, refIdxLXA0 is set equal to −1, and predFlagLXA0 is set equal to 0, with X being 0 or 1:
  + - availableA0 is equal to FALSE.
    - availableA1 is equal to TRUE and the prediction units covering the luma locations ( xNbA1, yNbA1 ) and ( xNbA0, yNbA0 ) have the same motion vectors and the same reference indices.
    - intra\_bc\_flag[xPb][yPb] is equal to 1 and availableB1 is equal to TRUE and the prediction units covering the luma locations ( xNbB1, yNbB1 ) and ( xNbA0, yNbA0 ) have the same motion vectors and the same reference indices.
    - intra\_bc\_flag[xPb][yPb] is equal to 1 and availableB0 is equal to TRUE and the prediction units covering the luma locations ( xNbB0, yNbB0 ) and ( xNbA0, yNbA0 ) have the same motion vectors and the same reference indices.
* Otherwise, availableFlagA0 is set equal to 1 and the following assignments are made:

mvLXA0 = MvLX[ xNbA0 ][ yNbA0 ] (8‑107)

refIdxLXA0 = RefIdxLX[ xNbA0 ][ yNbA0 ] (8‑108)

predFlagLXA0 = PredFlagLX[ xNbA0 ][ yNbA0 ] (8‑109)

For the derivation of availableFlagB2, refIdxLXB2, predFlagLXB2, and mvLXB2 the following applies:

– The luma location ( xNbB2, yNbB2 ) inside the neighbouring luma coding block is set equal to ( xPb − 1, yPb − 1 ).

– The availability derivation process for a prediction block as specified in subclause 6.4.2 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 ( xNbB2, yNbB2 ), and the partition index partIdx as inputs, and the output is assigned to the prediction block availability flag availableB2.

– When xPb >>  Log2ParMrgLevel is equal to xNbB2  >>  Log2ParMrgLevel and yPb  >>  Log2ParMrgLevel is equal to yNbB2  >>  Log2ParMrgLevel, availableB2 is set equal to FALSE.

– When intra\_bc\_flag[xNbA0][ yNbA0] is not equal to intra\_bc\_flag[xPb][yPb], availableA0 is set equal to FALSE.

– The variables availableFlagB2, refIdxLXB2, predFlagLXB2, and mvLXB2 are derived as follows:

* If one or more of the following conditions are true, availableFlagB2 is set equal to 0, both components of mvLXB2 are set equal to 0, refIdxLXB2 is set equal to −1, and predFlagLXB2 is set equal to 0, with X being 0 or 1:
  + - availableB2 is equal to FALSE.
    - availableA1 is equal to TRUE and prediction units covering the luma locations ( xNbA1, yNbA1 ) and ( xNbB2, yNbB2 ) have the same motion vectors and the same reference indices.
    - availableB1 is equal to TRUE and the prediction units covering the luma locations ( xNbB1, yNbB1 ) and ( xNbB2, yNbB2 ) have the same motion vectors and the same reference indices.
    - intra\_bc\_flag[xPb][yPb] is equal to 1 and availableA0 is equal to TRUE and the prediction units covering the luma locations ( xNbA0, yNbA0 ) and ( xNbB2, yNbB2 ) have the same motion vectors and the same reference indices.
    - intra\_bc\_flag[xPb][yPb] is equal to 1 and availableB0 is equal to TRUE and the prediction units covering the luma locations ( xNbB0, yNbB0 ) and ( xNbB2, yNbB2 ) have the same motion vectors and the same reference indices.
    - intra\_bc\_flag[xPb][yPb] is 0 and availableFlagA0 + availableFlagA1 + availableFlagB0 + availableFlagB1 is equal to 3.
    - availableFlagA0 + availableFlagA1 + availableFlagB0 + availableFlagB1 is equal to 4.
* Otherwise, availableFlagB2 is set equal to 1, and the following assignments are made:

mvLXB2 = MvLX[ xNbB2 ][ yNbB2 ] (8‑110)

refIdxLXB2 = RefIdxLX[ xNbB2 ][ yNbB2 ] (8‑111)

predFlagLXB2 = PredFlagLX[ xNbB2 ][ yNbB2 ] (8‑112)

##### 8.5.3.2.11 Derivation process for derived IntraBC merging 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 luma location ( xPb, yPb ) of 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 merging candidate list mergeCandList,
* the reference indices refIdxL0N and refIdxL1N of every candidate N in mergeCandList,
* the prediction list utilization flags predFlagL0N and predFlagL1N of every candidate N in mergeCandList,
* the motion vectors mvL0N and mvL1N of every candidate N in mergeCandList,
* the number of elements numCurrMergeCand within mergeCandList,

Outputs of this process are:

* the merging candidate list mergeCandList,
* the number of elements numCurrMergeCand within mergeCandList,
* the reference indices refIdxL0derivedCandk and refIdxL1derivedCandk of every new candidate derivedCandk added into mergeCandList during the invocation of this process,
* the prediction list utilization flags predFlagL0derivedCandk and predFlagL1derivedCandk of every new candidate derivedCandk added into mergeCandList during the invocation of this process,
* the motion vectors mvL0derivedCandk and mvL1derivedCandk of every new candidate derivedCandk added into mergeCandList during the invocation of this process.

The variable numOrigMergeCand is set equal to numCurrMergeCand. When numOrigMergeCand is greater than 0 and less than MaxNumMergeCand, the variable derivedIdx is set equal to 0, the variable derivedStop is set equal to FALSE, and the following steps are repeated untilderivedStop is equal to TRUE:

1. The block vector bvL0 is set as follows, where X is set equal to mergeCandList[derivedIdx]:
   * + bvL0[0] = mvL0X[0]
     + bvL0[1] = mvL0X[1]
2. The luma sample location ( xNbR, yNbR ) specifying the top left luma sample location of the reference prediction block relative to the top left luma sample of the picture is set equal to ( xPb + bvL0[0], yPb + bvL0[1] ).
3. The variable validDerivedBvFlag is set equal to 0.
4. When yNbR/CtbSizeY >= yCb/CtbSizeY-1, and if the prediction unit covering luma sample location ( xNbR, yNbR ) is predicted using IntraBC prediction mode, the following applies.

* The variable validDerivedBvFlag is set to 1, and the derived block vector bvL0Derived is derived as follows:
  + - bvL0Derived[0] = MvL0[ xNbR ][ yNbR ][0] + bvL0[0]
    - bvL0Derived[1] = MvL0[ xNbR ][ yNbR ][1] + bvL0[1]
* The derivation process for z-scan order block availability as specified in subclause 6.4.1 is invoked with ( xCurr, yCurr ) set equal to ( xCb, yCb ) and the neighbouring luma location ( xNbY, yNbY ) set equal to ( xPb + bvL0Derived[0], yPb + bvL0Derived[1] ) as inputs, and the output is assigned to the prediction block availability flag availableD0.
* The derivation process for z-scan order block availability as specified in subclause 6.4.1 is invoked with ( xCurr, yCurr ) set equal to ( xCb, yCb ) and the neighbouring luma location ( xNbY, yNbY ) set equal to (xPb+bvL0Derived[0]+nPbW−1,  yPb + bvL0Derived[1] + nPbH – 1 ) as inputs, and the output is assigned to the prediction block availability flag availableD1.
* If one or more of the following conditions are true, validDerivedBvFlag is set equal to 0:
  + - availableD0 is equal to FALSE;
    - availableD1is equal to FALSE;
    - bvL0Derived[0] + xPb + nPbW - xCb > 0;
    - bvL0Derived[1] + yPb + nPbH – yCb > 0;
    - bvL0Derived[0] is equal to mvL0N[0] and bvL0Derived[1] is equal to mvL0N[1], where N is set equal to mergeCandList[k], for any value of k in the range of 0 to numCurrMergeCand-1, inclusive.
  + If validDerivedBvFlag is equal to 1, the candidate derivedCandk with k equal to ( numCurrMergeCand − numOrigMergeCand ) is added at the end of mergeCandList, i.e. mergeCandList[ numCurrMergeCand ] is set equal to derivedCandk, and the block vectors of derivedCandk are derived as follows and numCurrMergeCand is incremented by 1:

refIdxL0derivedCandk = num\_ref\_idx\_l0\_active\_minus1+1 (8‑xxx)

refIdxL1combCandk = -1 (8‑xxx)

predFlagL0combCandk = 1 (8‑xxx)

predFlagL1combCandk = 0 (8‑xxx)

mvL0deriedCandk[ 0 ] = bvL0Derived[ 0 ] (8‑xxx)

mvL0derivedCandk[ 1 ] = bvL0Derived[ 1 ] (8‑xxx)

mvL1combCandk[ 0 ] = 0 (8‑xxx)

mvL1combCandk[ 1 ] = 0 (8‑xxx)

numCurrMergeCand = numCurrMergeCand + 1 (8‑xxx)

1. The variable derivedIdx is incremented by 1.
2. When derivedIdx is equal to  numOrigMergeCand  or numCurrMergeCand is equal to MaxNumMergeCand, derivedStop is set equal to TRUE.

#### Decoding process for inter prediction samples

##### General

……

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.

* If intra\_bc\_flag [ xCb + xBl ][ yCb + yBl ] is equal to 1, the following applies:
  + The array prediction sample array for each colour component is derived by invoking the subclause 8.4.5.2.7 with the luma locations ( xCb + xBl, yCb + yBl ), the luma prediction block width nPbW, the luma prediction block height nPbH, the motion vectors bvIntra, and the current reconstructed picture array currPicL, the colour component index of the current block cIdx as inputs.
* Otherwise, for X being each of 0 and 1, when predFlagLX is equal to 1, the following applies:

……

**Parsing process**

Table 9‑4 – Association of ctxIdx and syntax elements for each initializationType in the initialization process

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Syntax structure** | **Syntax element** | **ctxTable** | **initType** | | |
| **0** | **1** | **2** |
| sao( ) | sao\_merge\_left\_flag sao\_merge\_up\_flag | Table 9‑5 | 0 | 1 | 2 |
| sao\_type\_idx\_luma sao\_type\_idx\_chroma | Table 9‑6 | 0 | 1 | 2 |
| coding\_quadtree( ) | split\_cu\_flag[ ][ ] | Table 9‑7 | 0..2 | 3..5 | 6..8 |
| coding\_unit( ) | cu\_transquant\_bypass\_flag | Table 9‑8 | 0 | 1 | 2 |
| cu\_skip\_flag | Table 9‑9 |  | 0..2 | 3..5 |
| ~~intra\_bc\_flag[ ][ ]~~ | ~~Table 9‑38~~ | ~~0~~ | ~~1~~ | ~~2~~ |
| palette\_mode\_flag[ ][ ] | Table 9‑39 | 0 | 1 | 2 |
| pred\_mode\_flag | Table 9‑10 |  | 0 | 1 |
| part\_mode | Table 9‑11 | 0 | 1..4 | 5..8 |
| prev\_intra\_luma\_pred\_flag[ ][ ] | Table 9‑12 | 0 | 1 | 2 |
| intra\_chroma\_pred\_mode[ ][ ] | Table 9‑13 | 0 | 1 | 2 |
| rqt\_root\_cbf | Table 9‑14 |  | 0 | 1 |
| cu\_residual\_act\_flag | Table 9‑40 | 0 | 1 | 2 |
| prediction\_unit( ) | merge\_flag[ ][ ] | Table 9‑15 |  | 0 | 1 |
| merge\_idx[ ][ ] | Table 9‑16 |  | 0 | 1 |
| intra\_bc\_flag[ ][ ] | Table 9‑38 |  | 1 | 2 |
| inter\_pred\_idc[ ][ ] | Table 9‑17 |  | 0..4 | 5..9 |
| ref\_idx\_l0[ ][ ], ref\_idx\_l1[ ][ ] | Table 9‑18 |  | 0..1 | 2..3 |
| mvp\_l0\_flag[ ][ ], mvp\_l1\_flag[ ][ ], bvp\_flag[ ][ ] | Table 9‑19 |  | 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 ] | 9.3.3.8 | 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 |

Table 9‑40 – Binarization for part\_mode

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