#### 7.3.8.5 Coding unit syntax

|  |  |
| --- | --- |
| coding\_unit( x0, y0, log2CbSize ) { | Descriptor |
| if( transquant\_bypass\_enabled\_flag ) |  |
| **cu\_transquant\_bypass\_flag** | ae(v) |
| if( slice\_type != I ) |  |
| **cu\_skip\_flag**[ x0 ][ y0 ] | ae(v) |
| nCbS = ( 1  <<  log2CbSize ) |  |
| if( cu\_skip\_flag[ x0 ][ y0 ] ) |  |
| prediction\_unit( x0, y0, nCbS, nCbS ) |  |
| else { |  |
| if( intra\_block\_copy\_enabled\_flag ) { |  |
| **intra\_bc\_flag**[ x0 ][ y0 ] | ae(v) |
| if( intra\_bc\_flag[ x0 ][ y0 ] ) |  |
| **merge\_intra\_bc\_cu\_flag**[ x0 ][ y0 ] | ae(v) |
| } |  |
| if( slice\_type != I ) |  |
| **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 ] &&  !merge\_intra\_bc\_cu\_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 ) |  |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |

#### 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\*/ { |  |
| if( PartMode != PART\_2Nx2N ) |  |
| **merge\_intra\_bc\_flag**[x0][y0] | ae(v) |
| if( merge\_intra\_bc\_flag[x0][y0] && MaxNumMergeCand > 1 ) |  |
| **merge\_intra\_bc\_idx**[x0][y0] | ae(v) |
| else { |  |
| bvd\_coding( x0, y0, 2 ) |  |
| **bvp\_flag**[ x0 ][ y0 ] | ae(v) |
| **}** |  |
| } 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( 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) |
| } |  |
| } |  |
| } |  |
| } |  |

#### Coding unit semantics

**merge\_intra\_bc\_cu\_flag**[ x0 ][ y0 ] specifies whether the intra block copy prediction parameters for the current coding unit are inferred from a neighbouring intra block copy predicted partition. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture. When merge\_intra\_bc\_cu\_flag[ x0 ][ y0 ] is not present, it is inferred to be equal to 0.

#### 7.4.9.7 Prediction unit semantics

**merge\_intra\_bc\_flag**[ x0 ][ y0 ] specifies whether the intra block copy prediction parameters for the current prediction unit are inferred from a neighbouring intra block copy predicted partition. 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 merge\_intra\_bc\_flag[ x0 ][ y0 ] is not present, it is inferred to be equal to merge\_intra\_bc\_cu\_flag[ x0 ][ y0 ].

**merge\_intra\_bc\_idx**[ x0 ][ y0 ] specifies the intra block copy merging candidate index of the intra block copy merging candidate list where 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 merge\_intra\_bc\_idx[ x0 ][ y0 ] is not present, it is inferred to be equal to 0. merge\_intra\_bc\_index[ x0 ][ y0 ] shall be in the range of 0 to MaxNumMergeCand minus 1, inclusive.

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

### 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\* blkInc / 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

* + If intra\_bc\_merge\_flag[ xPb ][ yPb ] is equal to 1 and compIdx is equal to 0, the derivation process for luma block vectors for IntraBC merge mode as specified in subclause 8.4.4.3 is invoked with the luma coding block locations ( xCb, yCb ), the luma prediction block location ( xPb, yPb ), the coding block size nCbS, the luma prediction block width nPbW, the luma prediction block height nPbH, and the partition index blkIdx as inputs, and the luma block vectors bvIntra[ xPb ][ yPb ] as output.
  + Otherwise (if intra\_bc\_merge\_flag[ xPb ][ yPb ] is equal to 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 ] [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 6.4.1 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 6.4.1 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

##### Derivation process for luma block vectors for IntraBC merge mode

This process is only invoked when merge\_intra\_bc\_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 block vectors bvL2

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 ) = ( xPb, yPb ) (8‑xx)

nOrigPbW = nPbW (8‑xx)

nOrigPbH = nPbH (8‑xx)

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‑xx)

nPbW = nCbS (8‑xx)

nPbH = nCbS (8‑xx)

partIdx = 0 (8‑xx)

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 block vectors bvMerge are derived by the following ordered steps:

1. The derivation process for IntraBC merging candidates from neighbouring prediction unit partitions in subclause 8.4.4.3.1 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 nPbW, the luma prediction block height nPbH, and the partition index partIdx as inputs, and the number of spatial block vector candidates numCurrMergeCand and an array bvL2MergeCand containing the spatial block vector candidates as outputs.
2. When numCurrMergeCand is greater than 0, the block vector derivation process for merging candidates specified in subclause 8.4.4.3.2 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, the block vector array bvL2MergeCand, and numCurrMergeCand as inputs, and the number of merge candidates numDerivedCand and the derived block vectors array bvL2DerivedCand as outputs.
3. The array bvL2MergeCand is modified by appending the derived block vectors as follows:

for(i =0 ; i< numDerivedCand; i++) {

bvL2MergeCand [numCurrMergeCand][0] = bvL2DerivedCand[i][0]

bvL2MergeCand [numCurrMergeCand][1] = bvL2DerivedCand[i][1]

numCurrMergeCand++

}

1. The luma block vector bvL2 is derived as follows:

bvL2[0] = bvL2MergeCand[merge\_intra\_bc\_idx[ xOrigP ][ yOrigP ]][0] (8‑xx)

bvL2[1] = bvL2MergeCand[merge\_intra\_bc\_idx[ xOrigP ][ yOrigP ]] [1] (8‑xx)

##### Derivation process for spatial 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 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:

* the number of spatial block vector candidates numCurrMergeCand,
* the array of sptatial block vector candidates bvL2MergCand.

A list of availability flags availableFlagA0, availableFlagA1, availableFlagB0, availableFlagB1, and availableFlagB2 of the block vectors of the neighbouring prediction units, and a list of block vectors bvL2A0, bvL2A1, bvL2B0, bvL2B1, and bvL2B2 of the neighbouring prediction units are derived as follows.

For the derivation of availableFlagA1 and bvL2A1, the following applies:

– 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 xPb  >>  Log2ParMrgLevel is equal to xNbA1  >>  Log2ParMrgLevel and  yPb  >>  Log2ParMrgLevel is equal to yNbA1  >>  Log2ParMrgLevel, availableA1 is set equal to FALSE.

– The variables availableFlagA1, bvL2A1 are derived as follows:

* If availableA1 is equal to FALSE, availableFlagA1 is set equal to 0, both components of bvL2A1 are set equal to 0.
* Otherwise, availableFlagA1 is set equal to 1 and the following applies:

bvL2A1 = bvIntra[ xNbA1 ][ yNbA1 ] (8‑xx)

For the derivation of availableFlagB1 and bvL2B1, the following applies:

– 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 xPb  >>  Log2ParMrgLevel is equal to xNbB1  >>  Log2ParMrgLevel and yPb  >>  Log2ParMrgLevel is equal to yNbB1  >>  Log2ParMrgLevel, availableB1 is set equal to FALSE.

– The variables availableFlagB1 and bvL2B1 are derived as follows:

* If one or more of the following conditions are true, availableFlagB1 is set equal to 0, both components of bvL2B1 are set equal to 0:
  + - 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 block vectors.
* Otherwise, availableFlagB1 is set equal to 1 and the following assignments are made:

bvL2B1 = bvIntra[ xNbB1 ][ yNbB1 ] (8‑xx)

For the derivation of availableFlagB0 and bvL2B0, 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.

– The variables availableFlagB0 and bvL2B0 are derived as follows:

* If one or more of the following conditions are true, availableFlagB0 is set equal to 0, both components of bvL2B0 are set equal to 0:
  + - availableB0 is equal to FALSE.
    - availableA1 is equal to TRUE and the prediction units covering the luma locations ( xNbB0, yNbB0 ) and ( xNbA1, yNbA1 ) have the same block vectors
    - availableB1 is equal to TRUE and the prediction units covering the luma locations ( xNbB0, yNbB0 ) and ( xNbB1, yNbB1 ) have the same block vectors.
* Otherwise, availableFlagB0 is set equal to 1 and the following applies:

bvL2B0 = bvIntra[ xNbB0 ][ yNbB0 ] (8‑xxx)

For the derivation of availableFlagA0 and bvL2A0, 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.

– The variables availableFlagA0 and bvL2A0 are derived as follows:

* If one or more of the following conditions are true, availableFlagA0 is set equal to 0, both components of bvL2A0 are set equal to 0:
  + - availableA0 is equal to FALSE.
    - availableA1 is equal to TRUE and the prediction units covering the luma locations ( xNbA0, yNbA0 ) and ( xNbA1, yNbA1 ) have the same block vectors
    - availableB1 is equal to TRUE and the prediction units covering the luma locations ( xNbA0, yNbA0 ) and ( xNbB1, yNbB1 ) have the same block vectors
    - availableB0 is equal to TRUE and the prediction units covering the luma locations ( xNbA0, yNbA0 ) and ( xNbB0, yNbB0 ) have the same block vectors.
* Otherwise, availableFlagA0 is set equal to 1 and the following assignments are made:

bvL2A0 = bvIntra[ xNbA0 ][ yNbA0 ] (8‑xxx)

For the derivation of availableFlagB2 and bvL2B2, 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.

– The variables availableFlagB2 and bvL2B2 are derived as follows:

* If one or more of the following conditions are true, availableFlagB2 is set equal to 0, both components of bvL2B2 are set equal to 0:
  + - 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 block vectors.
    - availableB1 is equal to TRUE and the prediction units covering the luma locations ( xNbB1, yNbB1 ) and ( xNbB2, yNbB2 ) have the same block vectors.
    - availableB0 is equal to TRUE and prediction units covering the luma locations ( xNbB0, yNbB0 ) and ( xNbB2, yNbB2 ) have the same block vectors.
    - availableA0 is equal to TRUE and the prediction units covering the luma locations ( xNbA0, yNbA0 ) and ( xNbB2, yNbB2 ) have the same block vectors.
    - availableFlagA0 + availableFlagA1 + availableFlagB0 + availableFlagB1 is greater than 2.
* Otherwise, availableFlagB2 is set equal to 1, and the following applies:

bvL2B2 = bvIntra[ xNbB2 ][ yNbB2 ] (8‑xxx)

The outputs numCurrMergeCand specifying the number of spatial block vector merge candidates, and the array of spatial block vector candidates bvL2MergeCand are derived as follows:

numCurrMergeCand = 0  
if( availableFlagA1 )  
 bvL2MergeCand[ numCurrMergeCand ++ ] = bvL2A1  
if( availableFlagB1 )  
 bvL2MergeCand [ numCurrMergeCand ++ ] = bvL2B1if( availableFlagB0 )  
 bvL2MergeCand [ numCurrMergeCand ++ ] = bvL2B0 (8‑xx~~91~~)if( availableFlagA0 )  
 bvL2MergeCand [ numCurrMergeCand ++ ] = bvL2A0if( availableFlagB2 )  
 bvL2MergeCand [ numCurrMergeCand ++ ] = bvL2B2

##### 8.4.4.3.2 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,
* the array of block vector merge candidates bvL2MergeCand,
* the number of elements numCurrMergeCand in the array bvL2MergeCand,

Outputs of this process are:

* the number of derived block vectors numDerivedCand
* an array of block vectors bvL2DerivedCand containing the derived block vector candidates.

The variable numDerivedCand is set equal to 0.

When numCurrMergeCand 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 until derivedStop is equal to TRUE:

1. . The block vector bvL2X is set as follows:
   * + bvL2X[0] = bvL2MergeCand[derivedIdx][0]
     + bvL2X[1] = bvL2MergeCand[derivedIdx][0]
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 + bvL2X[0], yPb + bvL2X[1] )
3. The variable validDerivedBvFlag is set equal to 0.
4. When yNbR/CtbSizeY >= yCb/CtbSizeY-1, and if the prediction unit covering luma location ( xNbR, yNbR ) is predicted using IntraBC prediction mode, the following applies.

* The variable validDerivedBvFlag is set to 1, and the derived block vector bvL2Derived is derived as follows:
  + - bvL2Derived[0] = bvIntra[ xNbR ][ yNbR ][0] + bvL2X[0]
    - bvL2Derived[1] = bvIntra[ xNbR ][ yNbR ][1] + bvL2X[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 + bvL2Derived[0], yPb + bvL2Derived[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+bvL2Derived[0]+ nPbW−1,  yPb + bvL2Derived[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;
    - bvL2Derived[0] + nPbW > 0;
    - bvL2Derived[1] + nPbH > 0;
    - bvL2Derived[0] is equal to bvL2DerivedCand[k][0] and bvL2Derived[1] is equal to bvL2DerivedCand[k][1], for any value of k in the range of 0 to derivedIdx-1, inclusive;
    - bvL2Derived[0] is equal to bvL2MergeCand[k][0] and bvL2Derived[1] is equal to bvL2MergeCand[k][1], for any value of k in the range of 0 to numCurrMergeCand-1, inclusive.
* If validDerivedBvFlag is equal to 1, the following applies:

bvL2DerivedCand[ numDerivedCand ] [ 0 ] = bvL2Derived [ 0 ] (8‑xxx)

bvL2DerivedCand[ numDerivedCand ] [ 1 ] = bvL2Derived [ 1 ] (8‑xxx)

numDerivedCand = numDerivedCand + 1 (8‑xxx)

1. The variable derivedIdx is incremented by 1.
2. When derivedIdx is equal to numCurrMergeCand  or (numCurrMergeCand+ numDerivedCand) is equal to MaxNumMergeCand, derivedStop is set equal to TRUE.

#### Initialization process for context variables

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** |
| 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 |
| merge\_intra\_bc\_cu\_flag[][] | Table 9‑41 | 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\_intra\_bc\_flag[][] | Table 9‑41 | 0 | 1 | 2 |
| merge\_intra\_bc\_idx[][] | Table 9‑42 | 0 | 1 | 2 |
| merge\_flag[ ][ ] | Table 9‑15 |  | 0 | 1 |
| merge\_idx[ ][ ] | Table 9‑16 |  | 0 | 1 |
| 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[ ][ ] | Table 9‑19 |  | 0 | 1 |

Table 9‑41 – Value of initValue for ctxIdx of merge\_intra\_bc\_flag

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of merge\_intra\_bc\_flag** | | |
| **0** | **1** | **2** |
| **initValue** | 154 | 110 | 110 |

Table 9‑42 – Value of initValue for ctxIdx of merge\_intra\_bc\_ idx

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of merge\_intra\_bc\_idx** | | |
| **0** | **1** | **2** |
| **initValue** | 137 | 122 | 122 |

### Binarization process

| Table 9‑38 – Syntax elements and associated binarizations | | | |
| --- | --- | --- | --- |
| **Syntax structure** | **Syntax element** | **Binarization** | |
| **Process** | **Input parameters** |
| coding\_unit( ) | cu\_transquant\_bypass\_flag | FL | cMax = 1 |
| cu\_skip\_flag | FL | cMax = 1 |
| intra\_bc\_flag | FL | cMax = 1 |
| merge\_intra\_bc\_cu\_flag | FL | cMax = 1 |
| palette\_mode\_flag | FL | cMax = 1 |
| pred\_mode\_flag | FL | cMax = 1 |
| part\_mode | 9.3.3.6 | ( xCb, yCb ) = ( x0, y0), log2CbSize |
| pcm\_flag[ ][ ] | FL | cMax = 1 |
| prev\_intra\_luma\_pred\_flag[ ][ ] | FL | cMax = 1 |
| mpm\_idx[ ][ ] | TR | cMax = 2, cRiceParam = 0 |
| rem\_intra\_luma\_pred\_mode[ ][ ] | FL | cMax = 31 |
| intra\_chroma\_pred\_mode[ ][ ] | 9.3.3.7 | - |
| rqt\_root\_cbf | FL | cMax = 1 |
| cu\_residual\_act\_flag | FL | cMax = 1 |
| prediction\_unit( ) | merge\_flag[ ][ ] | FL | cMax = 1 |
| merge\_idx[ ][ ] | TR | cMax = MaxNumMergeCand − 1, cRiceParam = 0 |
| merge\_intra\_bc\_flag | FL | cMax = 1 |
| merge\_intra\_bc\_idx | TR | cMax = MaxNumMergeCand − 1, cRiceParam = 0 |
| 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 |