**Draft Text Specification**

The proposed text changes are based on the document JCTVC-S1005.doc for Intra BC deblocking in CE2 Test 4.2. The changes are marked in yellow.

### 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~~

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 ] + xPb − xCb is less than or equal to 0

– bvIntra[ xPb ][ yPb ][ 1 ] + yPb − yCb is less than or equal to 0

– It is a requirement of bitstream conformance that the following condition shall be true:

( xPb + bvIntra[ xPb ][ yPb ][ 0 ] + nPbSw − 1 ) / CtbSizeY − xCurr / CtbSizeY <= yCurr/CtbSizeY − ( yPb + bvIntra[ xPb ][ yPb ][ 1 ] + nPbSh − 1 ) / CtbSizeY

#### 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 predictor 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 and the last block vectors in subclause 8.4.4.2 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, the partition index partIdx and the last block vectors as LastBvIntra inputs, and the availability flags availableFlagN and the block vectors bvIntraN, with N being replaced by A0, A1, B0, B1, or B2, as output.
2. The derivation process for temporal block vector predictor in subclause 8.4.4.3 is invoked with the luma prediction block location ( xPb, yPb ), the luma prediction block width nPbW, and the luma prediction block height nPbH as the input, and the the availability flag availableFlagIntraCol and the block vector bvIntraCol as output.
3. The variables bvpIntraVirtual[ i ][ j ] (with i being equal to 0, 1, 2, 3, 4; j being equal to 0 or 1) specify five virtual block vector predictors, and are derived as follows:

bvpIntraVirtual[ 0 ][ 0 ] = -2 \* nPbW, bvpIntraVirtual[ 0 ][ 1 ] = 0.

bvpIntraVirtual[ 1 ][ 0 ] = -nPbW, bvpIntraVirtual[ 1 ][ 1 ] = 0.

bvpIntraVirtual[ 2 ][ 0 ] = 0, bvpIntraVirtual[ 2 ][ 1 ] = -2 \* nPbH.

bvpIntraVirtual[ 3 ][ 0 ] = 0, bvpIntraVirtual[ 3 ][ 1 ] = -nPbH.

bvpIntraVirtual[ 4 ][ 0 ] = -nPbW, bvpIntraVirtual[ 4 ][ 1 ] = -nPbH.

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

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

i = 0

if(availableFlagA1 )  
 bvpIntraList[ i++ ] = bvIntraA1  
if( availableFlagB1)  
 bvpIntraList[ i++ ] = bvIntraB1  
if( availableFlagB0)  
 bvpIntraList[ i++ ] = bvIntraB0  
if( availableFlagA0)  
 bvpIntraList[ i++ ] = bvIntraA0  
if( availableFlagB2)  
 bvpIntraList[ i++ ] = bvIntraB2  
if( availableFlagLast0 && (i < 5) )  
 bvpIntraList[ i++ ] = bvIntraLast0  
if( availableFlagLast1 && (i < 5))  
 bvpIntraList[ i++ ] = bvIntraLast1

if(i < 5)  
 for( j=0; j < 5; j++) for( j=0; j < 5; j++)  
 bvpIntraList[ i++ ] = bvpIntraVirtual[ j ]

1. The block vector predictor BvpIntra[ xPb ][ yPb ] is derived as follows.

BvpIntra [ xPb ][ yPb ] = bvpIntraList[ bvp\_flag[ xPb ][ yPb ]]

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

[Ed. (RLJ): It should be BvpIntra. Corrected in v3.]

#### 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.
* the last block vector LastBvIntra

Outputs of this process are (with N being replaced by A0, A1, B0, B1, B2, Last0, Last1):

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

The variables bvIntraA0[ compIdx ], bvIntraA0[ compIdx ], bvIntraB0[ compIdx ], bvIntraB1[ compIdx ], bvIntraB2[ compIdx ], bvIntraLast0 [ compIdx ] and bvIntraLast1 [ compIdx ] specify the spatial block vector predictor candidates and the last block vector predictor candidates, 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 A0, A1, B0, B1, B2, Last0 or Last1. 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 A0, A1, B0, B1, B2 Last0, or Last1;

availableFlagN is set equal to FALSE for N being equal to A0, A1, B0, B1, B2 Last0, or Last1.

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 ( xPb − 1 , yPb + nPbH − 1 ), and the partition index partIdx as inputs, if the output is equal to TRUE, availableFlagA1 is set equal to TRUE, and bvIntraA1 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 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 ( xPb + nPbW − 1 , yPb − 1 ), and the partition index partIdx as inputs, the output is assigned to the prediction block availability flag availableB1, bvIntraB1 is set equal to bvIntra[ xPb + nPbW − 1 ][ yPb − 1 ].

* If one or more of the following conditions are true, availableFlagB1 is set equal to FALSE, both components of bvIntraB1 are set equal to 0:
  + - availableB1 is equal to FALSE.
    - availableB1 is equal to TRUE, bvIntraB1 and bvIntraA1 have the same block vectors.
* Otherwise, availableFlagB1 is set equal to TRUE.

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 ( xPb + nPbW , yPb − 1 ), and the partition index partIdx as inputs, the output is assigned to the prediction block availability flag availableB0, bvIntraB0 is set equal to bvIntra[ xPb + nPbW ][ yPb − 1 ].

* If one or more of the following conditions are true, availableFlagB0 is set equal to FALSE, both components of bvIntraB0 are set equal to 0:
  + - availableB0 is equal to FALSE.
    - availableB0 is equal to TRUE, bvIntraB0 and bvIntraB1 have the same block vectors.
* Otherwise, availableFlagB0 is set equal to TRUE.

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 ( xPb – 1, yPb + nPbH ), and the partition index partIdx as inputs, the output is assigned to the prediction block availability flag availableA0, bvIntraA0 is set equal to [ xPb − 1 ][ yPb + nPbH ].

* If one or more of the following conditions are true, availableFlagA0 is set equal to FALSE, both components of bvIntraA0 are set equal to 0:
  + - availableA0 is equal to FALSE.
    - availableA0 is equal to TRUE, bvIntraA0 and bvIntraA1 have the same block vectors.
* Otherwise, availableFlagA0 is set equal to TRUE.

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 ( xPb – 1, yPb -1 ), and the partition index partIdx as inputs, the output is assigned to the prediction block availability flag availableB2, bvIntraB2 is set equal to [ xPb − 1 ][ yPb -1 ].

* If one or more of the following conditions are true, availableFlagB2 is set equal to FALSE, both components of bvIntraB2 are set equal to 0:
  + - availableB2 is equal to FALSE.
    - availableB2 is equal to TRUE, bvIntra B2 and bvIntraA1 have the same block vectors.
    - availableB2 is equal to TRUE, bvIntra B2 and bvIntraB1 have the same block vectors.
* Otherwise, availableFlagB2 is set equal to TRUE.

For the first last block vector predictor,

* If one or more of the following conditions are true, availableFlagLast0 is set equal to FALSE, both components of bvIntraLast0 are set equal to 0:
  + - both components of bvIntraLast0 are equal to 0.
    - bvIntraLast0 and bvIntraA1 have the same block vectors.
    - bvIntraLast0 and bvIntraB1 have the same block vectors.
* Otherwise, availableFlagB2 is set equal to TRUE.

For the second last block vector predictor,

* If one or more of the following conditions are true, availableFlagLast1 is set equal to FALSE, both components of bvIntraLast1 are set equal to 0:
  + - both components of bvIntraLast1 are equal to 0.
    - bvIntraLast1 and bvIntraA1 have the same block vectors.
    - bvIntraLast1 and bvIntraB1 have the same block vectors.

##### Derivation process for temporal block 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,

Outputs of this process are:

* the availability flag availableFlagIntraCol;
* the block vector prediction bvIntraCol.

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

The variables bvIntraCol and availableFlagIntraCol are derived as follows:

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

1. The bottom right collocated block vector is derived as follows:

xColBr = xPb + nPbW (8‑170)

yColBr = yPb + nPbH (8‑171)

* If yPb  >>  CtbLog2SizeY is equal to yColBr  >>  CtbLog2SizeY, yColBr is less than pic\_height\_in\_luma\_samples, and xColBr is less than pic\_width\_in\_luma\_samples, the following applies:
* The variable colPb specifies the prediction block covering the modified location given by ( ( xColBr  >>  4 )  <<  4, ( yColBr  >>  4 )  <<  4 ) inside the collocated picture specified by ColPic.
* The luma location ( xColPb, yColPb ) is set equal to the top-left sample of the collocated prediction block specified by colPb relative to the top-left luma sample of the collocated picture specified by ColPic.
* If the intra\_bc\_flag[ xColPb ][ yColPb ] inside the collocated picture is equal to 1, bvIntraCol are set equal to bvIntra[ xColPb ][ yColPb ] and availableFlagIntraCol is set equal to 1. Otherwise, both components of bvIntraCol are set equal to 0 and availableFlagIntraCol is set equal to 0.

1. When availableFlagIntraCol is equal to 0, the central collocated block vector is derived as follows:

xColCtr = xPb + ( nPbW  >>  1 ) (8‑172)

yColCtr = yPb + ( nPbH  >>  1 ) (8‑173)

* The variable colPb specifies the luma prediction block covering the modified location given by ( ( xColCtr  >>  4 )  <<  4, ( yColCtr  >>  4 )  <<  4 ) inside the collocated picture specified by ColPic.
* The luma location ( xColPb, yColPb ) is set equal to the top-left sample of the collocated luma prediction block specified by colPb relative to the top-left luma sample of the collocated picture specified by ColPic.
* If the intra\_bc\_flag[ xColPb ][ yColPb ] inside the collocated picture is equal to 1, bvIntraCol are set equal to bvIntra[ xColPb ][ yColPb ] and availableFlagIntraCol is set equal to 1. Otherwise, both components of bvIntraCol are set equal to 0 and availableFlagIntraCol is set equal to 0.

**Table 9‑38 – Syntax elements and associated binarizations**

|  |  |  |  |
| --- | --- | --- | --- |
| prediction\_unit( ) | merge\_flag[ ][ ] | FL | cMax = 1 |
| merge\_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 |
| bvp\_flag[ ][ ] | TR | cMax = 5 |

**Table 9‑43 – Assignment of ctxInc to syntax elements with context coded bins**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| bvp\_flag[ ][ ] | 0 | bypass | bypass | bypass | bypass | bypass |