**Draft Text Specification**

The proposed text changes are based on the document JCTVC-P1005-v1.doc for the IntraBC vector prediction in SCCE1 Test 3.4. The changes are marked in yellow.

**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 ) |  |
| **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 ] ) |  |
| **pred\_mode\_flag** | ae(v) |
| 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 &&   ~~!intra\_bc\_flag[ x0 ][ y0 ] &&~~   log2CbSize >= Log2MinIpcmCbSizeY &&  log2CbSize <= Log2MaxIpcmCbSizeY ) |  |
| **pcm\_flag**[ x0 ][ y0 ] | ae(v) |
| if( pcm\_flag[ x0 ][ y0 ] ) { |  |
| while( !byte\_aligned( ) ) |  |
| **pcm\_alignment\_zero\_bit** | f(1) |
| pcm\_sample( x0, y0, log2CbSize ) |  |
| ~~} else if( intra\_bc\_flag[ x0 ][ y0 ] ) {~~ |  |
| ~~mvd\_coding( x0, y0, 2)~~ |  |
| ~~if( PartMode = = PART\_2NxN )~~ |  |
| ~~mvd\_coding( x0, y0 + ( nCbS / 2 ), 2)~~ |  |
| ~~else if( PartMode = = PART\_Nx2N )~~ |  |
| ~~mvd\_coding( x0 + ( nCbS / 2 ), y0, 2)~~ |  |
| ~~else if( PartMode = = PART\_NxN ) {~~ |  |
| ~~mvd\_coding( x0 + ( nCbS / 2 ), y0, 2)~~ |  |
| ~~mvd\_coding( x0, y0 + ( nCbS / 2 ), 2)~~ |  |
| ~~mvd\_coding( x0 + ( nCbS / 2 ), y0 + ( nCbS / 2 ), 2)~~ |  |
| ~~}~~ |  |
| } else { |  |
| pbOffset = ( PartMode = = PART\_NxN ) ? ( nCbS / 2 ) : nCbS |  |
| …… |  |

**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\*/ |  |
| mvd\_coding( x0, y0, 2) |  |
| **mvp\_l0\_flag** [ x0 ][ y0 ] | ae(v) |
| } else { /\* MODE\_INTER \*/ |  |
| …… |  |
| } |  |
| } |  |

**Semantics**

**Prediction unit semantics**

**Decoding Process**

**6.4.2 Derivation process for prediction block availability**

……

When availableN is equal to TRUE, ~~CuPredMode[ xNbY ][ yNbY ] is equal to MODE\_INTRA, availableN is set equal to FALSE.~~

–  If CuPredMode[ xPb][ yPb ] is equal to MODE\_INTER, and CuPredMode[ xNbY ][ yNbY ]is equal to MODE\_INTRA, availableN is set equal to FALSE.

– Otherwise, if intra\_bc\_flag[ xPb][ yPb ] is equal to 1, and intra\_bc\_flag[ xNbY ][ yNbY ] is equal to 0, availableN is set equal to FALSE.

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

Inputs to this process are:

* a luma location ( xCb, yCb ) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
* a variable log2CbSize specifying the size of the current luma coding block.

Output of this process is the (nCbS)x(nCbX) array of block vectors bvIntra.

1. ……
2. The following ordered steps apply, for the variable compIdx proceeding over the values 0..1:
   * The subclause 8.5.3.2.5 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, refidxL0 is set to the reference index, and the partition index blkIdx as inputs, and the block vector predictor BvpIntra[ xPb ][ yPb ] as the output.
   * The bvIntra[ xPb ][ yPb ][ compIdx ] is derived as follows:

bvIntra[ xPb ][ yPb ][ compIdx ] = BvdIntra[ xPb ][ yPb ][ compIdx ] + BvpIntra[ xPb ][ yPb ] [compIdx ]

* + ~~Depending upon the number of times this process has been invoked for the current coding tree unit, the following applies:~~
* ~~If this process is invoked for the first time for the current coding tree unit, bvIntra[ xPb ][ yPb ][ compIdx ] is derived as follows:~~

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

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

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

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

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

1. ~~The value of BvpIntra[ compIdx ] is updated to be equal to bvIntra[ xPb ][ yPb ][ compIdx ].~~
2. For use in derivation processes of variables invoked later in the decoding process, the following assignments are made for x = 0..nPbSw − 1 and y = 0..nPbSh − 1:

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

~~It is a requirement of bitstream conformance that all of the the following conditions are true:~~

~~– The value of bvIntra[ xPb ][ yPb ][ 0 ] shall be greater than or equal to – ( xPb % CtbSizeY + 64 ).~~

~~– The value of bvIntra[ xPb ][ yPb ][ 1 ] shall be greater than or equal to – ( yPb % CtbSizeY ).~~

– 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 shall be equal to TRUE.

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

– One or both of the following conditions shall be true:

– bvIntra[ xPb ][ yPb ][ 0 ] + nPbSw <= 0

– bvIntra[ xPb ][ yPb ][ 1 ] + nPbSh <= 0

##### **8.5.3.2.5 Derivation process for luma motion vector prediction**

The motion vector predictor mvpLX is derived in the following ordered steps:

1. …….
2. If both availableFlagLXA and availableFlagLXB are equal to 1 and mvLXA is not equal to mvLXB, or if intra\_bc\_flag[xPb][yPb] is equal to 1, availableFlagLXCol is set equal to 0. Otherwise, the derivation process for temporal luma motion vector prediction in subclause is invoked with luma prediction block location ( xPb, yPb ), the luma prediction block width nPbW, the luma prediction block height nPbH, and refIdxLX, with X being 0 or 1, as inputs, and with the output being the availability flag availableFlagLXCol and the temporal motion vector predictor mvLXCol.
3. ……
4. The motion vector predictor list is modified as follows:
   * + When mvLXA and mvLXB have the same value, mvLXB is removed from the list and the variable numMvpCandLX is set equal to the number of elements within the mvpListLX.
     + When numMvpCandLX is less than 2, the following applies repeatedly until numMvpCandLX is equal to 2:

If intra\_bc\_flag[xPb][yPb] is equal to 1

mvpListLX[ numMvpCandLX ][ 0 ] = (numMvpCandLX==0) ? -2\* nCbS : -nCbS (8‑144)

otherwise,

mvpListLX[ numMvpCandLX ][ 0 ] = 0 (8‑144)

mvpListLX[ numMvpCandLX ][ 1 ] = 0 (8‑145)

numMvpCandLX = numMvpCandLX + 1 (8‑146)

* + - When numMvpCandLX is greater than 2, all motion vector predictor candidates mvpListLX[ idx ] with idx greater than 1 are removed from the list.

1. …….

##### **8.2.3.6 Derivation process for motion vector predictor candidates**

…….

The motion vector mvLXA and the availability flag availableFlagLXA are derived in the following ordered steps:

1. ……
2. ……
3. ……
4. ……
5. ……
6. The following applies for ( xNbAk, yNbAk ) from ( xNbA0, yNbA0 ) to ( xNbA1, yNbA1 ):

* When availableAk is equal to TRUE and availableFlagLXA is equal to 0, the following applies:
* If PredFlagLX[ xNbAk ][ yNbAk ] is equal to 1 and DiffPicOrderCnt( RefPicListX[ RefIdxLX[ xNbAk ][ yNbAk ] ], RefPicListX[ refIdxLX ] ) is equal to 0, or if intra\_bc\_flag[xPb][ yPb ] is equal to 1, availableFlagLXA is set equal to 1 and the following applies:

mvLXA = MvLX[ xNbAk ][ yNbAk ] (8‑147)

* Otherwise, when PredFlagLY[ xNbAk ][ yNbAk ] (with Y = !X) is equal to 1 and DiffPicOrderCnt( RefPicListY[ RefIdxLY[ xNbAk ][ yNbAk ] ], RefPicListX[ refIdxLX ] ) is equal to 0, availableFlagLXA is set equal to 1 and the following applies:

mvLXA = MvLY[ xNbAk ][ yNbAk ] (8‑148)

1. When availableFlagLXA is equal to 0, and intra\_bc\_flag[xPb][yPb] is equal to 0, the following applies for ( xNbAk, yNbAk ) from ( xNbA0, yNbA0 ) to ( xNbA1, yNbA1 ) or until availableFlagLXA is equal to 1:

……

The motion vector mvLXB and the availability flag availableFlagLXB are derived in the following ordered steps:

1. The sample locations ( xNbB0, yNbB0 ), ( xNbB1, yNbB1 ), and ( xNbB2, yNbB2 ) are set equal to ( xPb + nPbW, yPb − 1 ), ( xPb + nPbW − 1, yPb − 1 ), and ( xPb − 1, yPb − 1 ), respectively.
2. The availability flag availableFlagLXB is set equal to 0 and the both components of mvLXB are set equal to 0. If intra\_bc\_flag[xPb][yPb] is equal to 1, isScaledFlagLX is set to 1.
3. The following applies for ( xNbBk, yNbBk ) from ( xNbB0, yNbB0 ) to ( xNbB2, yNbB2 ):

* The availability derivation process for a prediction block as specified in subclause  is invoked with the luma location ( xCb, yCb ), the current luma coding block size nCbS, the luma prediction block location ( xPb, yPb ), the luma prediction block width nPbW, the luma prediction block height nPbH, the luma location ( xNbY, yNbY ) set equal to ( xNbBk, yNbBk ), and the partition index partIdx as inputs, and the output is assigned to the prediction block availability flag availableBk.
* When availableBk is equal to TRUE and availableFlagLXB is equal to 0, the following applies:
* If PredFlagLX[ xNbBk ][ yNbBk ] is equal to 1, and DiffPicOrderCnt( RefPicListX[ RefIdxLX[ xNbBk ][ yNbBk ] ], RefPicListX[ refIdxLX ] ) is equal to 0, or intra\_bc\_flag[xPb][yPb] is equal to 1, availableFlagLXB is set equal to 1 and the following assignments are made:

mvLXB = MvLX[ xNbBk ][ yNbBk ] (8‑160)

refIdxB = RefIdxLX[ xNbBk ][ yNbBk ] (8‑161)

* Otherwise, when PredFlagLY[ xNbBk ][ yNbBk ] (with Y = !X) is equal to 1 and DiffPicOrderCnt( RefPicListY[ RefIdxLY[ xNbBk ][ yNbBk ] ], RefPicListX[ refIdxLX ] ) is equal to 0, availableFlagLXB is set equal to 1 and the following assignments are made:

mvLXB = MvLY[ xNbBk ][ yNbBk ] (8‑162)

refIdxB = RefIdxLY[ xNbBk ][ yNbBk ] (8‑163)

……