**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.

**Syntax**

#### 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\*/ |  |
| **intra\_bc\_flip\_or\_mask\_flag**[ x0 ][ y0 ] | ae(v) |
| if( intra\_bc\_flip\_or\_mask\_flag[ x0 ][ y0 ] ) { |  |
| **flipping\_ibc\_flag**[ x0 ][ y0 ] | ae(v) |
| if(flipping\_ibc\_flag[ x0 ][ y0 ]) |  |
| **flipping\_ibc\_direction**[ x0 ][ y0 ] | ae(v) |
| else { |  |
| **intra\_bc\_mask\_mode\_minus1**[ x0 ][ y0 ] | u(2) |
| **intra\_bc\_mask\_size\_flag1**[ x0 ][ y0 ] | u(1) |
| if( intra\_bc\_mask\_size\_flag1[ x0 ][ y0 ] ) |  |
| **intra\_bc\_mask\_size\_flag2**[ x0 ][ y0 ] | u(1) |
| } |  |
|  |  |
|  |  |
| } |  |
| 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) |
| } |  |
| } |  |
| } |  |
| } |  |

**Semantics**



#### Prediction unit semantics

**intra\_bc\_flip\_or\_mask\_flag**[ x0 ][ y0] equals to 1 specifies that either **flipping\_ibc\_flag**[ x0 ][ y0] or intra\_bc\_mask\_flag[ x0 ][ y0 ] of the current prediction unit will be set to 1.

**flipping\_ibc\_flag**[ x0 ][ y0] equals to 1 specifies that the current prediction unit is coded with flipped intra block copy mode. If the flipping\_ibc\_flag[ x0 ][ y0 ] equals to 0, it specifies that the current prediction unit is using normal intra block copy mode. 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 flipping\_ibc\_flag is inferred to be equal to 0.

**flipping\_ibc\_direction**[ x0 ][ y0 ] equal to 1 specifies that the current flipping direction is vertical. If the flipping\_ibc\_direction[ x0 ][ y0 ] equal to 0 specifies that the current flipping direction is horizontal. 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.

**intra\_bc\_mask\_flag**[ x0 ][ y0 ] equal to 1 specifies that the current prediction unit is coded in intra block copying mode with sample mask mode maskModeIntraBc larger than zero. intra\_bc\_mask\_flag [ x0 ][ y0 ] equal to 0 specifies that the current prediction unit is coded without a mask and maskModeIntraBc is set to zero. When not present, the value of intra\_bc\_mask\_flag is inferred to be equal to 0. 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.

**intra\_bc\_mask\_mode\_minus**[ x0 ][ y0 ] indicates the applied maskModeIntraBc shall be equal to intra\_bc\_mask\_mode\_minus [ x0 ][ y0 ] plus one. When not present, maskModeIntraBc is inferred to be equal to 0.

**intra\_bc\_mask\_size\_flag1**[ x0 ][ y0 ] equal to 0 indicates the width of the masking area for luminance component maskWidthIntraBcLuma is equal to two. intra\_bc\_mask\_size\_flag1[ x0 ][ y0 ] equal to 1 indicates that maskWidthIntraBcLuma is defined by intra\_bc\_mask\_size\_flag2[ x0 ][ y0 ].

**intra\_bc\_mask\_size\_flag2**[ x0 ][ y0 ] equal to 1 indicates the maskWidthIntraBcLuma is equal to one. intra\_bc\_mask\_size\_flag2[ x0 ][ y0 ] equal to 0 indicates maskWidthIntraBcLuma is equal to three.

**Decoding process**

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

## 8.4.1 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 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 and masking information in intra block copying prediction mode as specified in subclause is invoked with the luma location ( xCb, yCb ) and variable log2CbSize, variable maskModeIntraBc and variable maskWidthIntraBcLuma as inputs, and the output being bvIntra, ibcMask and ibcMaskLoc.
3. If cu\_residual\_act\_flag[ xCb ][ yCb ] is equal to 1, the following applies:

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

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

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

1. The general decoding process for intra blocks as specified in subclause is invoked with the luma 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 cIdx set equal to 0, and variable controlParaACT set to (cu\_residual\_act\_flag[ xCb ][ yCb ] ? 2 : 3 ), the variable ibcMask and the variable ibcMaskLoc as inputs, and the output is a modified reconstructed picture before deblocking filtering. [Ed. (GJS): Variable name controlParaACT seems to not follow editorial convention, by having several capitalized letters in a row. Do we do that elsewhere?]

……

## 8.4.4 Derivation process for block vector components and masking information 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.
* variables maskModeIntraBc and maskWidthIntraBcLuma.

Output of this process is the (nCbS)x(nCbX) array of block vectors bvIntra, the mask sample location array ibcMaskLoc and mask array ibcMask.

……

The variable maskWidthIntraBc is set equal to maskWidthIntraBcLuma / cSc, where:

cSc = ( cIdx == 0 ) ? 1 : ( maskModeIntraBc == 1 || maskModeIntraBc == 3 ) ? SubHeightC : SubWidthC

Each value of ibcMask[ xPb + x ][ yPb + y ] with x = 0.. nPbSw − 1, y = 0.. nPbSh – 1 are set to 0.

If maskModeIntraBc is larger than 0, the following additional steps apply:

– Variables maskFirstX, maskLastX, maskFirstY, maskLastY, maskRefX, maskRefY are set as follows:

firstX = maskModeIntraBc == 2 ? nPbSw - maskWidthIntraBc: 0;

firstY = maskModeIntraBc == 3 ? nPbSh - maskWidthIntraBc: 0;

lastX = maskModeIntraBc == 4 ? maskWidthIntraBc - 1 : nPbSw - 1;

lastY = maskModeIntraBc == 1 ? maskWidthIntraBc - 1 : nPbSh - 1;

– Variables maskRefX and maskRefY are defined as a function of variables maskModeIntraBc, firstX, firstY, lastX and lastY according to the table below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **maskModeIntraBc** | **1** | **2** | **3** | **4** |
| maskRefX | nPbSw / 2 | maskFirstX | nPbSw / 2 | maskLastX |
| maskRefY | maskLastY | nPbSh / 2 | maskFirstY | nPbSh / 2 |

– The components of the mask location array mlvIntra are set to:

ibcMaskLoc[ xPb + x ][ yPb + y ][ 0 ] =  xPb + maskRefX with x = 0.. nPbSw-1, y = 0.. nPbSh-1

ibcMaskLoc[ xPb + x ][ yPb + y ][ 1 ] =  yPb + maskRefY with x = 0.. nPbSw-1, y = 0.. nPbSh-1

* Each value of ibcMask[ xPb + x ][ yPb + y ] with x = maskFirstX..maskLastX, y = maskFirstY.. maskLastY are set to 1.

##### Specification of intra block copying prediction mode

Inputs to this process are:

– a sample location ( xTb0, yTb0 ) specifying the top-left sample of the current transform block relative to the top‑left sample of the current picture,

– a variable nTbS specifying the transform block size,

– a variable trafoDepth specifying the hierarchy depth of the current block relative to the coding unit,

– a variable bvIntra specifying the block copying vector,

– a variable ibcMaskLoc specifying the mask sample location,

– a variable ibcMask specifying the array of masked samples,

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

……

– Otherwise, the following applies:

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

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

– If maskModeIntraBc is larger than 0, the mask sample location ( xMaskCmp, yMaskCmp ) representing the mask sample location vector in full-sample units with x, y = 0..nTbS − 1, is derived as follows:

( xMaskCmp, yMaskCmp ) = ( ibcMaskLoc[ xTb0 + x ][ yTb0 + y ][ 0 ] + bv[ 0 ], ibcMaskLoc[ xTb0 + x ][ yTb0 + y ][ 1 ] + bv[ 1 ] ) (8‑xx)

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

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

( xRefCmp, yRefCmp ) = ( xTbCmp + x + bv[ 0 ], yTbCmp + y + bv[ 1 ] ) (8‑65)

~~Each sample at the location ( xRefCmp, yRefCmp ) is assigned to predSamples[ x ][ y ]~~

If flipping\_ibc\_flag[ xTbY ][ yTbY ] is equal to 0, the following applies:

– If ibcMask[ xTb0  + x ][ yTb0  + y ] is equal to 0, each sample at the location ( xRefCmp, yRefCmp ) is assigned to predSamples[ x ][ y ].

– Otherwise, each sample at the location ( xMaskCmp, yMaskCmp ) is assigned to predSamples[ x ][ y ].

Otherwise (flipping\_ibc\_flag[ xTbY ][ yTbY ] is equal to 1),

– If flipping\_ibc\_direction [ xTbY ][ yTbY ] is equal to 0, each sample at the location ( xRefCmp, yRefCmp ) is assigned to predSamples[ nTbS-1-x ][ y ];

– Otherwise (flipping\_ibc\_direction [ xTbY ][ yTbY ] is equal to 1), each sample at the location ( xRefCmp, yRefCmp ) is assigned to predSamples[ x ][ nTbS-1-y ].

**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 |  | 0 | 1 | 2 |
| sao\_type\_idx\_luma sao\_type\_idx\_chroma |  | 0 | 1 | 2 |
| coding\_quadtree( ) | split\_cu\_flag[ ][ ] |  | 0..2 | 3..5 | 6..8 |
| coding\_unit( ) | cu\_transquant\_bypass\_flag |  | 0 | 1 | 2 |
| cu\_skip\_flag |  |  | 0..2 | 3..5 |
| intra\_bc\_flag[ ][ ] |  | 0 | 1 | 2 |
| palette\_mode\_flag[ ][ ] |  | 0 | 1 | 2 |
| pred\_mode\_flag |  |  | 0 | 1 |
| part\_mode |  | 0 | 1..4 | 5..8 |
| prev\_intra\_luma\_pred\_flag[ ][ ] |  | 0 | 1 | 2 |
| intra\_chroma\_pred\_mode[ ][ ] |  | 0 | 1 | 2 |
| rqt\_root\_cbf |  |  | 0 | 1 |
| cu\_residual\_act\_flag |  | 0 | 1 | 2 |
| prediction\_unit( ) | merge\_flag[ ][ ] |  |  | 0 | 1 |
| merge\_idx[ ][ ] |  |  | 0 | 1 |
| inter\_pred\_idc[ ][ ] |  |  | 0..4 | 5..9 |
| ref\_idx\_l0[ ][ ], ref\_idx\_l1[ ][ ] |  |  | 0..1 | 2..3 |
| flipping\_ibc\_flag[ ][ ] | Table 9-38’ | 0…2 | 0…2 | 0…2 |
| flipping\_ibc\_direction[ ][ ] | Table 9-38’ | 0…2 | 0…2 | 0…2 |
| intra\_bc\_flip\_or\_mask\_flag[ ][ ] | Table 9-38'' | 0…2 | 0…2 | 0…2 |
| mvp\_l0\_flag[ ][ ], mvp\_l1\_flag[ ][ ], bvp\_flag[ ][ ] |  |  | 0 | 1 |

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

Table 9‑38’ – Values of initValue for ctxIdx of flipping\_ibc\_flag and flipping\_ibc\_direction

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of sao\_merge\_left\_flag and sao\_merge\_up\_flag** | | |
| **0** | **1** | **2** |
| **initValue** | 146 | 154 | 157 |

Table 9‑38’' – Values of initValue for ctxIdx of intra\_bc\_flip\_or\_mask\_flag

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of intra\_bc\_flip\_or\_mask\_flag** | | |
| **0** | **1** | **2** |
| **initValue** | 154 | 154 | 154 |

| Table ‑43 – Assignment of ctxInc to syntax elements with context coded bins | | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **Syntax element** | **binIdx** | | | | | |
| **0** | **1** | **2** | **3** | **4** | **>= 5** |
| … |  |  |  |  |  |  |
| intra\_bc\_flag | 0 | na | na | na | na | na |
| flipping\_ibc\_flag | 0 | na | na | na | na | na |
| flipping\_ibc\_direction | 0 | na | na | na | na | na |
| intra\_bc\_flip\_or\_mask\_flag | 0 | na | na | na | na | na |
| pred\_mode\_flag | 0 | na | na | na | na | na |
| … |  |  |  |  |  |  |

Table 9‑44 – Specification of ctxInc using left and above syntax elements

|  |  |  |  |
| --- | --- | --- | --- |
| **Syntax element** | **condL** | **condA** | **ctxInc** |
| split\_cu\_flag[ x0 ][ y0 ] | CtDepth[ xNbL ][ yNbL ] > cqtDepth | CtDepth[ xNbA ][ yNbA ] > cqtDepth | ( condL  &&  availableL ) + ( condA  &&  availableA ) |
| cu\_skip\_flag[ x0 ][ y0 ] | cu\_skip\_flag[ xNbL ][ yNbL ] | cu\_skip\_flag[ xNbA ][ yNbA ] | ( condL  &&  availableL ) + ( condA  &&  availableA ) |
| flipping\_ibc\_flag[ x0 ][ y0 ] | flipping\_ibc\_flag[ xNbL ][ yNbL ] | flipping\_ibc\_flag [ xNbA ][ yNbA ] | ( condL  &&  availableL ) + ( condA  &&  availableA ) |
| flipping\_ibc\_direction[ x0 ][ y0 ] | flipping\_ibc\_direction[ xNbL ][ yNbL ] | flipping\_ibc\_direction [ xNbA ][ yNbA ] | ( condL  &&  availableL ) + ( condA  &&  availableA ) |