**Draft Text Specification**

The proposed text changes are based on the document JCTVC-P1005-v1.doc for the combination of Intra BC prediction and BVD binarization in SCCE1 Test 5.2. 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 |  |
| 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 ) { |  |
| 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\*/ |  |
| intra\_bc\_bvd\_coding( x0, y0, 2) |  |
| **intra\_bc\_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) |
| } |  |
| } |  |
| } |  |
| } |  |

|  |  |
| --- | --- |
| intra\_bc\_bvd\_coding ( x0, y0, refList ) { | **Descriptor** |
| **intra\_bc\_abs\_bvd\_greater0\_flag[ 0 ]** | ae(v) |
| **intra\_bc\_abs\_bvd\_greater0\_flag[ 1 ]** | ae(v) |
| if(intra\_bc\_abs\_bvd\_greater0\_flag[ 0 ] ) { |  |
| **intra\_bc\_abs\_bvd\_minus1[ 0 ]** | ae(v) |
| **intra\_bc\_bvd\_sign\_flag[ 0 ]** | ae(v) |
| } |  |
| if(intra\_bc\_abs\_bvd\_greater0\_flag[ 1 ] ) { |  |
| **intra\_bc\_abs\_bvd\_minus1**[ 1 ] | ae(v) |
| **intra\_bc\_bvd\_sign\_flag**[ 1 ] | ae(v) |
| } |  |
| } |  |

**Semantics**

**Prediction unit semantics**

**intra\_bc\_abs\_bvd\_greater0\_flag**[ compIdx ] specifies whether the absolute value of a block vector component difference is greater than 0.

**intra\_bc\_abs\_bvd\_minus1**[ compIdx ] plus 1 specifies the absolute value of a block vector component difference.

When intra\_bc\_abs\_bvd\_minus1[ compIdx ] is not present, it is inferred to be equal to −1.

**intra\_bc\_bvd\_sign\_flag**[ compIdx ] specifies the sign of a block vector component difference as follows:

If intra\_bc\_bvd\_sign\_flag[ compIdx ] is equal to 0, the corresponding block vector component difference has a positive value.

Otherwise (intra\_bc\_bvd\_sign\_flag[ compIdx ] is equal to 1), the corresponding block vector component difference has a negative value.

When intra\_bc\_bvd\_sign\_flag[ compIdx ] is not present, it is inferred to be equal to 0.

The block vector difference BvdIntra [ compIdx ] for compIdx = 0..1 is derived as follows:

BvdIntra [ compIdx ] = intra\_bc\_abs\_bvd\_greater0\_flag [ compIdx ] \*  
 ( intra\_bc\_abs\_bvd\_minus1 [ compIdx ] + 1 ) \* ( 1 − 2 \* intra\_bc\_bvd\_sign\_flag [ compIdx ] )

**intra\_bc\_bvp\_flag**[ x0 ][ y0 ] equal to 1 specifies that above block vector is used as the block vector predictor. intra\_bc\_bvp\_flag[ x0 ][ y0 ] equal to 0 specifies that left block vector is used as the block vector predictor. When not present, the value of intra\_bc\_bvp\_flag[ x0 ][ y0 ] 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 coding block relative to the top-left luma sample of the picture.

**Decoding Process**

**6.4.2 Derivation process for prediction block availability**

Inputs to this process are:

* the 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,
* the 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 current luma prediction block,
* a variable partIdx specifying the partition index of the current prediction unit within the current coding unit,
* the luma location ( xNbY, yNbY ) covered by a neighbouring prediction block relative to the top-left luma sample of the current picture.

Output of this process is the availability of the neighbouring prediction block covering the location ( xNbY, yNbY ), denoted as availableN is derived as follows:

The variable sameCb specifying whether the current luma prediction block and the neighbouring luma prediction block cover the same luma coding block.

– If all of the following conditions are true, sameCb is set equal to TRUE:

– xCb is less than or equal than xNbY,

– yCb is less than or equal than yNbY,

– ( xCb + nCbS ) is greater than xNbY,

– ( yCb + nCbS ) is greater than yNbY.

– Otherwise, sameCb is set equal to FALSE.

The neighbouring prediction block availability availableN is derived as follows:

– If sameCb is equal to FALSE, the derivation process for z-scan order block availability as specified in subclause 6.4.1 is invoked with ( xCurr, yCurr ) set equal to ( xPb, yPb ) and the luma location ( xNbY, yNbY ) as inputs, and the output is assigned to availableN.

– Otherwise, if all of the following conditions are true, availableN is set equal to FALSE:

– ( nPbW << 1 ) is equal to nCbS,

– ( nPbH << 1 ) is equal to nCbS,

– partIdx is equal to 1,

– ( yCb + nPbH ) is less than or equal to yNbY,

– ( xCb + nPbW ) is greater than xNbY.

– Otherwise, availableN is set equal to TRUE.

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.

The variables nCbS, nCbSw, nCbSh 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 / 2 )
4. The following ordered steps apply, for the variable compIdx proceeding over the values 0..1:
5. Depending upon the number of times this process has been invoked for the current coding tree unit, the following applies:
   * The 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, 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 ]

1. ~~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 ).~~

 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 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 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

**8.4.4.1 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
* a variable partIdx specifying the index of the current prediction unit within the current coding unit.

Output of this process is the block vector prediction BvpIntra[ xPb ][ yPb ].

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

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

1. The derivation process for motion vector predictor candidates from neighbouring prediction unit partitions 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, and the partition index partIdx as inputs, and the availability flags availableFlagN and the block vectors bvIntraN, with N being replaced by A or B, as output.
2. The variables bvpIntraVirtual[ i ] (with i being equal to 0 or 1) specify two virtual block vector predictors, and they are derived as follows:

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

bvpIntraVirtual[ 1 ][0] = 2 \* nPbW, bvpIntraVirtual1[ 0 ][ 1 ] = 0;

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

i = 0   
if(availableFlagA )  
 bvpIntraList[ i++ ] = bvIntraA  
if(availableFlagB && bvIntraA!= bvIntraB)  
 bvpIntraList[ i++ ] = bvIntraB  
for( j=0; j<2 && i<2;j++) {  
 if( (j = =1 | | bvpIntraList[ 0 ]!= bvpIntraVirtual[ j ] )  
 bvpIntraList[ i++ ]= bvpIntraVirtual[ j ]  
}

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

for (i =0; i< 2 ; i++)  
BvpIntra [ xPb ][ yPb ][ i ]= bvpIntraVirtual[intra\_bc\_bvp\_flag[ xPb ][ yPb ][ i ]

**8.4.4.2 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.

Outputs of this process are (with N being replaced by A or B):

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

The variables bvIntraA[ compIdx ] specify the left neighboring block vector predictor 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 neighboring blocks, with N being equal to A or B. 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 A and B;

availableFlagN is set equal to FALSE for N being equal to A and B.

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, then availableFlagA is set to TRUE, and

bvIntraA = bvIntra[ xPb – 1 ][ yPb + nPbH – 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 ( xPb – nPbW - 1  , yPb – 1  ), and the partition index partIdx as inputs. If the output is equal to TRUE, and (yPb/CtbSizeY) is equal to ((yPb – 1)/CtbSizeY), then availableFlagB is set to TRUE, and

bvIntraB = bvIntra[ xPb – nPbW - 1 ][  yPb – 1 ]

| **Table 9‑34 – Syntax elements and associated binarizations** | | | |
| --- | --- | --- | --- |
| **Syntax structure** | **Syntax element** | **Binarization** | |
| **Process** | **Input parameters** |
| slice\_segment\_data( ) | end\_of\_slice\_segment\_flag | FL | cMax = 1 |
| end\_of\_sub\_stream\_one\_bit | FL | cMax = 1 |
| sao( ) | sao\_merge\_left\_flag | FL | cMax = 1 |
| sao\_merge\_up\_flag | FL | cMax = 1 |
| sao\_type\_idx\_luma | TR | cMax = 2, cRiceParam = 0 |
| sao\_type\_idx\_chroma | TR | cMax = 2, cRiceParam = 0 |
| sao\_offset\_abs[ ][ ][ ][ ] | TR | cMax = ( 1  <<  ( Min( bitDepth, 10 ) − 5 ) ) − 1, cRiceParam = 0 |
| sao\_offset\_sign[ ][ ][ ][ ] | FL | cMax = 1 |
| sao\_band\_position[ ][ ][ ] | FL | cMax = 31 |
| sao\_eo\_class\_luma | FL | cMax = 3 |
| sao\_eo\_class\_chroma | FL | cMax = 3 |
| 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 |
| pred\_mode\_flag | FL | cMax = 1 |
| part\_mode | 9.3.3.5 | ( 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[ ][ ] | 9.3.3.6 | - |
| rqt\_root\_cbf | FL | cMax = 1 |
| prediction\_unit( ) | merge\_flag[ ][ ] | FL | cMax = 1 |
| merge\_idx[ ][ ] | TR | cMax = MaxNumMergeCand − 1, cRiceParam = 0 |
| inter\_pred\_idc[ x0 ][ y0 ] | 9.3.3.7 | 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 |
| transform\_tree( ) | split\_transform\_flag[ ][ ][ ] | FL | cMax = 1 |
| cbf\_luma[ ][ ][ ] | FL | cMax = 1 |
| cbf\_cb[ ][ ][ ] | FL | cMax = 1 |
| cbf\_cr[ ][ ][ ] | FL | cMax = 1 |
| mvd\_coding( ) | abs\_mvd\_greater0\_flag[ ] | FL | cMax = 1 |
| abs\_mvd\_greater1\_flag[ ] | FL | cMax = 1 |
| abs\_mvd\_minus2[ ] | EG1 | - |
| mvd\_sign\_flag[ ] | FL | cMax = 1 |
| intra\_bc\_bvd\_coding ( ) | intra\_bc\_abs\_bvd\_greater0\_flag[ ] | FL | cMax = 1 |
| intra\_bc\_abs\_bvd\_minus1 [ ] | EG3 | - |
| intra\_bc\_bvd\_sign\_flag [ ] | FL | cMax = 1 |
| transform\_unit( ) | cu\_qp\_delta\_abs | 9.3.3.8 | - |
| cu\_qp\_delta\_sign\_flag | FL | cMax = 1 |
| cu\_chroma\_qp\_adjustment\_flag | FL | cMax = 1 |
| cu\_chroma\_qp\_adjustment\_idc | TR | cMax = chroma\_qp\_adjustment\_table\_size\_minus1, cRiceParam = 0 |
| cross\_comp\_pred( ) | log2\_res\_scale\_abs\_plus1 | TR | cMax = 4, cRiceParam = 0 |
| res\_scale\_sign\_flag | FL | cMax = 1 |
| residual\_coding( ) | transform\_skip\_flag[ ][ ][ ] | FL | cMax = 1 |
| explicit\_rdpcm\_flag[ ][ ][ ] | FL | cMax = 1 |
| explicit\_rdpcm\_dir\_flag[ ][ ][ ] | FL | cMax = 1 |
| last\_sig\_coeff\_x\_prefix | TR | cMax = ( log2TrafoSize << 1 ) − 1, cRiceParam = 0 |
| last\_sig\_coeff\_y\_prefix | TR | cMax = ( log2TrafoSize << 1 ) − 1, cRiceParam = 0 |
| last\_sig\_coeff\_x\_suffix | FL | cMax = ( 1  <<  ( ( last\_sig\_coeff\_x\_prefix  >>  1 ) − 1 ) − 1 ) |
| last\_sig\_coeff\_y\_suffix | FL | cMax = ( 1  <<  ( ( last\_sig\_coeff\_y\_prefix  >>  1 ) − 1 ) − 1 ) |
| coded\_sub\_block\_flag[ ][ ] | FL | cMax = 1 |
| sig\_coeff\_flag[ ][ ] | FL | cMax = 1 |
| coeff\_abs\_level\_greater1\_flag[ ] | FL | cMax = 1 |
| coeff\_abs\_level\_greater2\_flag[ ] | FL | cMax = 1 |
| coeff\_abs\_level\_remaining[ ] | 9.3.3.9 | current sub-block scan index i, baseLevel |
| coeff\_sign\_flag[ ] | FL | cMax = 1 |