**Syntax and semantics description for SCCE3.D2 (Independent Uniform Prediction mode)**

This section provides the changes to the SCCE3-base (JCTVC-P1005\_v4+Palette) syntax and semantics for the independent uniform prediction mode. The changes for IUP are identical to those made to JCTVC‑P1005\_v4 (without palette).

## 7.3.6 Slice segment header syntax

### 7.3.6.1 General slice segment header syntax

|  |  |
| --- | --- |
| slice\_segment\_header( ) { | Descriptor |
| … |  |
| if( sample\_adaptive\_offset\_enabled\_flag ) { |  |
| **slice\_sao\_luma\_flag** | u(1) |
| if( ChromaArrayType != 0 ) |  |
| **slice\_sao\_chroma\_flag** | u(1) |
| } |  |
| if( independent\_uniform\_pred\_enabled\_flag ) { |  |
| **num\_uniform\_colors** | u(v) |
| for( i = 0; i < num\_uniform\_colors; i++) |  |
| for( cIdx = 0; cIdx < ( ChromaArrayType != 0 ? 3 : 1 ); cIdx++ ) |  |
| **uniform\_colors**[ cIdx ][ i ] | ae(v) |
| } |  |
| if( slice\_type = = P | | slice\_type = = B ) { |  |
| **num\_ref\_idx\_active\_override\_flag** | u(1) |
| if( num\_ref\_idx\_active\_override\_flag ) { |  |
| **num\_ref\_idx\_l0\_active\_minus1** | ue(v) |
| if( slice\_type = = B ) |  |
| **num\_ref\_idx\_l1\_active\_minus1** | ue(v) |
| } |  |
| if( lists\_modification\_present\_flag && NumPocTotalCurr > 1 ) |  |
| ref\_pic\_lists\_modification( ) |  |
| if( slice\_type = = B ) |  |
| **mvd\_l1\_zero\_flag** | u(1) |
| if( cabac\_init\_present\_flag ) |  |
| **cabac\_init\_flag** | u(1) |
| if( slice\_temporal\_mvp\_enabled\_flag ) { |  |
| if( slice\_type = = B ) |  |
| **collocated\_from\_l0\_flag** | u(1) |
| if( ( collocated\_from\_l0\_flag && num\_ref\_idx\_l0\_active\_minus1 > 0 ) | |  ( !collocated\_from\_l0\_flag && num\_ref\_idx\_l1\_active\_minus1 > 0 ) ) |  |
| **collocated\_ref\_idx** | ue(v) |
| } |  |
| if( ( weighted\_pred\_flag && slice\_type = = P ) | |  ( weighted\_bipred\_flag && slice\_type = = B ) ) |  |
| pred\_weight\_table( ) |  |
| **five\_minus\_max\_num\_merge\_cand** | ue(v) |
| } |  |
| … |  |
| byte\_alignment( ) |  |
| } |  |

### 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( independent\_uniform\_pred\_enabled\_flag && num\_uniform\_colors > 0 ) |  |
| **independent\_uniform\_pred\_flag**[ x0 ][ y0 ] | ae(v) |
| if( independent\_uniform\_pred\_ flag && num\_uniform\_colors > 1 ) |  |
| **uniform\_color\_index**[ x0 ][ y0 ] | ae(v) |
| else { |  |
| if( intra\_block\_copy\_enabled\_flag ) |  |
| **intra\_bc\_flag**[ x0 ][ y0 ] | ae(v) |
| ... |  |
| } |  |
| 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 ] ) | | independent\_uniform\_pred\_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 ) |  |
| } |  |
| } |  |
| } |  |
| } |  |

### 7.3.8.10 Transform unit syntax

|  |  |
| --- | --- |
| transform\_unit( x0, y0, xBase, yBase, log2TrafoSize, trafoDepth, blkIdx ) { | **Descriptor** |
| log2TrafoSizeC = log2TrafoSize − ( ChromaArrayType = = 3 ? 0 : 1 ) |  |
| cbfLuma = cbf\_luma[ x0 ][ y0 ][ trafoDepth ] |  |
| cbfChroma =   cbf\_cb[ x0 ][ y0 ][ trafoDepth ] | |   cbf\_cr[ x0 ][ y0 ][ trafoDepth ] | |   ( ChromaArrayType = = 2 &&   ( cbf\_cb[ x0 ][ y0 + ( 1 << log2TrafoSizeC ) ][ trafoDepth ] | |   cbf\_cr[ x0 ][ y0 + ( 1 << log2TrafoSizeC ) ][ trafoDepth ] ) ) |  |
| if( cbfLuma | | cbfChroma ) { |  |
| if( cu\_qp\_delta\_enabled\_flag && !IsCuQpDeltaCoded ) { |  |
| **cu\_qp\_delta\_abs** | ae(v) |
| if( cu\_qp\_delta\_abs ) |  |
| **cu\_qp\_delta\_sign\_flag** | ae(v) |
| } |  |
| if( slice\_chroma\_qp\_adjustment\_enabled\_flag && cbfChroma &&  !cu\_transquant\_bypass\_flag && !IsCuChromaQpAdjustmentCoded ) { |  |
| **cu\_chroma\_qp\_adjustment\_flag** | ae(v) |
| if( cu\_chroma\_qp\_adjustment\_flag &&  chroma\_qp\_adjustment\_table\_size\_minus1 > 0 ) |  |
| **cu\_chroma\_qp\_adjustment\_idc** | ae(v) |
| } |  |
| if( cbfLuma ) |  |
| residual\_coding( x0, y0, log2TrafoSize, 0 ) |  |
| if( log2TrafoSize > 2 | | ChromaArrayType = = 3 ) { |  |
| if( cross\_component\_prediction\_enabled\_flag && cbfLuma &&   ( CuPredMode[ x0 ][ y0 ] = = MODE\_INTER | | intra\_bc\_flag[ x0 ][ y0 ] | | independent\_uniform\_pred\_ flag [ x0 ][ y0 ] | | intra\_chroma\_pred\_mode[ x0 ][ y0 ] = = 4) ) |  |
| cross\_comp\_pred( x0, y0, 0 ) |  |
| for( tIdx = 0; tIdx < ( ChromaArrayType = = 2 ? 2 : 1 ); tIdx++ ) |  |
| if( cbf\_cb[ x0 ][ y0 + ( tIdx << log2TrafoSizeC ) ][ trafoDepth ] ) |  |
| residual\_coding( x0, y0 + ( tIdx << log2TrafoSizeC ), log2TrafoSizeC, 1 ) |  |
| if( cross\_component\_prediction\_enabled\_flag && cbfLuma &&   ( CuPredMode[ x0 ][ y0 ] = = MODE\_INTER | | intra\_bc\_flag[ x0 ][ y0 ] | | independent\_uniform\_pred\_ flag [ x0 ][ y0 ] | |   intra\_chroma\_pred\_mode[ x0 ][ y0 ] = = 4) ) |  |
| cross\_comp\_pred( x0, y0, 1 ) |  |
| for( tIdx = 0; tIdx < ( ChromaArrayType = = 2 ? 2 : 1 ); tIdx++ ) |  |
| if( cbf\_cr[ x0 ][ y0 + ( tIdx << log2TrafoSizeC ) ][ trafoDepth ] ) |  |
| residual\_coding( x0, y0 + ( tIdx << log2TrafoSizeC ), log2TrafoSizeC, 2 ) |  |
| } else if( blkIdx = = 3 ) { |  |
| for( tIdx = 0; tIdx < ( ChromaArrayType = = 2 ? 2 : 1 ); tIdx++ ) |  |
| if( cbf\_cb[ xBase ][ yBase + ( tIdx << log2TrafoSizeC ) ][ trafoDepth ] ) |  |
| residual\_coding( xBase, yBase + ( tIdx << log2TrafoSize ), log2TrafoSize, 1 ) |  |
| for( tIdx = 0; tIdx < ( ChromaArrayType = = 2 ? 2 : 1 ); tIdx++ ) |  |
| if( cbf\_cr[ xBase ][ yBase + ( tIdx << log2TrafoSizeC ) ][ trafoDepth ] ) |  |
| residual\_coding( xBase, yBase + ( tIdx << log2TrafoSize ), log2TrafoSize, 2 ) |  |
| } |  |
| } |  |
| } |  |

### 7.3.8.11 Residual coding syntax

|  |  |
| --- | --- |
| residual\_coding( x0, y0, log2TrafoSize, cIdx ) { | Descriptor |
| if( transform\_skip\_enabled\_flag && !cu\_transquant\_bypass\_flag &&   ( log2TrafoSize <= Log2MaxTransformSkipSize ) ) |  |
| **transform\_skip\_flag**[ x0 ][ y0 ][ cIdx ] | ae(v) |
| if( ( CuPredMode[ x0 ][ y0 ] = = MODE\_INTER | | intra\_bc\_flag[ x0 ][ y0 ] | | independent\_uniform\_pred\_ flag [ x0 ][ y0 ] ) &&   explicit\_rdpcm\_enabled\_flag &&   ( transform\_skip\_flag[ x0 ][ y0 ][ cIdx ] | | cu\_transquant\_bypass\_flag ) ) { |  |
| **explicit\_rdpcm\_flag**[ x0 ][ y0 ][ cIdx ] | ae(v) |
| if( explicit\_rdpcm\_flag[ x0 ][ y0 ][ cIdx ] ) |  |
| **explicit\_rdpcm\_dir\_flag**[ x0 ][ y0 ][ cIdx ] | ae(v) |
| } |  |
| … |  |
| } |  |

### 7.4.3.3 Picture parameter set RBSP semantics

**constrained\_intra\_pred\_flag** equal to 0 specifies that intra prediction allows usage of residual data and decoded samples of neighbouring coding blocks coded using either intra or inter prediction modes. constrained\_intra\_pred\_flag equal to 1 specifies constrained intra prediction, in which case the general intra sample prediction process only uses residual data and decoded samples from neighbouring coding blocks coded using intra prediction modes.

NOTE 5 – Samples reconstructed from available intra blocks with intra\_bc\_flag equal to 1 or independent\_uniform\_pred\_ flag equal to 1 are considered as available for the purposes of constrained intra prediction. However, no restriction is placed on the origin of reference samples used by intra blocks with intra\_bc\_flag equal to 1. Encoders that employ constrained intra prediction for the purpose of error propagation control should, in regions where error propagation is to be controlled through the use of constrained intra prediction, ensure that intra blocks with intra\_bc\_flag equal to 1 only use - from neighbouring coding blocks coded using intra prediction modes.

**7.4.7.1 General slice segment header semantics**

**num\_uniform\_colors** specifies the total number of colors that may be used by the independent uniform prediction mode in in the current slice. When num\_uniform\_colors is not present, it is inferred to be equal to 0.

**uniform\_colors**[ cIdx ][ i ] specifies the i-th uniform color value for component cIdx. When uniform\_colors[ cIdx ][ i ] is not present, it is inferred to be equal to 0.

**7.4.9.5 Coding unit semantics**

**independent\_uniform\_pred\_flag**[ x0 ][ y0 ] equal to 1 specifies that the current coding unit is coded using the independent uniform prediction mode. independent\_uniform\_pred\_flag[ x0 ][ y0 ] equal to 0 specifies that the current coding unit is coded according to intra\_bc\_flag. When not present, the value of independent\_uniform\_pred\_flag[ x0 ][ y0 ] is inferred to be 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.

**uniform\_color\_index**[ x0 ][ y0 ] is an index to the color values that will be used in the independent uniform prediction mode for the current coding unit. When uniform\_color\_index[ x0 ][ y0 ] is not present, it 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.

**pred\_mode\_flag** equal to 0 specifies that the current coding unit is coded in inter prediction mode. pred\_mode\_flag equal to 1 specifies that the current coding unit is coded in intra prediction mode. The variable CuPredMode[ x ][ y ] is derived as follows for x = x0..x0 + nCbS − 1 and y = y0..y0 + nCbS − 1:

* If pred\_mode\_flag is equal to 0, CuPredMode[ x ][ y ] is set equal to MODE\_INTER.
* Otherwise (pred\_mode\_flag is equal to 1), CuPredMode[ x ][ y ] is set equal to MODE\_INTRA.

When pred\_mode\_flag is not present, the variable CuPredMode[ x ][ y ] is derived as follows for x = x0..x0 + nCbS − 1 and y = y0..y0 + nCbS − 1:

* If independent\_uniform\_pred\_flag[ x0 ][ y0 ] is equal to 1, CuPredMode[ x ][ y ] is inferred to be equal to MODE\_INTRA.
* Otherwise, if intra\_bc\_flag[ x0 ][ y0 ] is equal to 1, CuPredMode[ x ][ y ] is inferred to be equal to MODE\_INTRA.
* Otherwise, if slice\_type is equal to I, CuPredMode[ x ][ y ] is inferred to be equal to MODE\_INTRA.
* Otherwise (slice\_type is equal to P or B), when cu\_skip\_flag[ x0 ][ y0 ] is equal to 1, CuPredMode[ x ][ y ] is inferred to be equal to MODE\_SKIP.

**part\_mode** specifies partitioning mode of the current coding unit. The semantics of part\_mode depend on CuPredMode[ x0 ][ y0 ]. The variables PartMode and IntraSplitFlag are derived from the value of part\_mode as defined in Table 7‑10.

The value of part\_mode is restricted as follows:

* If CuPredMode[ x0 ][ y0 ] is equal to MODE\_INTRA, the following applies:
* If independent\_uniform\_pred\_flag[ x0 ][ y0 ] is equal to 1, part\_mode shall be equal to 0.
* Otherwise, if intra\_bc\_flag[ x0 ][ y0 ] is equal to 1, part\_mode shall be in the range of 0 to 3, inclusive.
* Otherwise (intra\_bc\_flag[ x0 ][ y0] is equal to 0 and independent\_uniform\_pred\_flag[ x0 ][ y0 ] is equal to 0), part\_mode shall be equal to 0 or 1.
* Otherwise (CuPredMode[ x0 ][ y0 ] is equal to MODE\_INTER), the following applies:
* If log2CbSize is greater than MinCbLog2SizeY and amp\_enabled\_flag is equal to 1, part\_mode shall be in the range of 0 to 2, inclusive, or in the range of 4 to 7, inclusive.
* Otherwise, if log2CbSize is greater than MinCbLog2SizeY and amp\_enabled\_flag is equal to 0, or log2CbSize is equal to 3, part\_mode shall be in the range of 0 to 2, inclusive.
* Otherwise (log2CbSize is greater than 3 and less than or equal to MinCbLog2SizeY), the value of part\_mode shall be in the range of 0 to 3, inclusive.

When part\_mode is not present, the variables PartMode and IntraSplitFlag are derived as follows:

* PartMode is set equal to PART\_2Nx2N.
* IntraSplitFlag is set equal to 0.

Table 7‑10 – Name association to prediction mode and partitioning type

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CuPredMode [ x0 ][ y0 ]** | **independent\_uniform\_pred\_flag [ x0 ][ y0 ]** | **intra\_bc\_flag [ x0 ][ y0 ]** | **part\_mode** | **IntraSplitFlag** | **PartMode** |
| MODE\_INTRA | 0 | 0 | 0 | 0 | PART\_2Nx2N |
| 1 | 1 | PART\_NxN |
| 1 | 0 | 0 | PART\_2Nx2N |
| 1 | 0 | PART\_2NxN |
| 2 | 0 | PART\_Nx2N |
| 3 | 0 | PART\_NxN |
| 1 | - | 0 | 0 | PART\_2Nx2N |
| MODE\_INTER | - | - | 0 | 0 | PART\_2Nx2N |
| 1 | 0 | PART\_2NxN |
| 2 | 0 | PART\_Nx2N |
| 3 | 0 | PART\_NxN |
| 4 | 0 | PART\_2NxnU |
| 5 | 0 | PART\_2NxnD |
| 6 | 0 | PART\_nLx2N |
| 7 | 0 | PART\_nRx2N |

**7.4.9.11 Residual coding semantics**

For intra prediction, different scanning orders are used. The variable scanIdx specifies which scan order is used where scanIdx equal to 0 specifies an up-right diagonal scan order, scanIdx equal to 1 specifies a horizontal scan order, and scanIdx equal to 2 specifies a vertical scan order. The value of scanIdx is derived as follows:

* If CuPredMode[ x0 ][ y0 ] is equal to MODE\_INTRA and one or more of the following conditions are true:
* log2TrafoSize is equal to 2.
* log2TrafoSize is equal to 3 and cIdx is equal to 0.

predModeIntra is derived as follows:

* If intra\_bc\_flag[ x0 ][ y0 ] is equal to 1 or independent\_uniform\_pred\_flag[ x0 ][ y0 ] is equal to 1, predModeIntra is undefined.
* If cIdx is equal to 0, predModeIntra is set equal to IntraPredModeY[ x0 ][ y0 ].
* Otherwise, predModeIntra is set equal to IntraPredModeC.

scanIdx is derived as follows:

* If predModeIntra is in the range of 6 to 14, inclusive, scanIdx is set equal to 2.
* Otherwise if predModeIntra is in the range of 22 to 30, inclusive, scanIdx is set equal to 1.
* Otherwise, scanIdx is set equal to 0.
* Otherwise, scanIdx is set equal to 0.

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

### 8.4.1 General decoding process for coding units coded in intra prediction mode

Inputs to this process are:

– a luma location ( xCb, yCb ) specifying 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 a modified reconstructed picture before deblocking filtering.

The derivation process for quantization parameters as specified in subclause 8.6.1 is invoked with the luma location ( xCb, yCb ) as input.

A variable nCbS is set equal to 1  <<  log2CbSize.

Depending on the values of pcm\_flag[ xCb ][ yCb ], palette\_mode\_flag[ xCb ][ yCb ], independent\_uniform\_pred\_flag[ xCb ][ yCb ] , and IntraSplitFlag, the decoding process for luma samples is specified as follows:

If pcm\_flag[ xCb ][ yCb ] is equal to 1, the reconstructed picture is modified as follows:

SL[ xCb + i ][ yCb + j ] =   
 pcm\_sample\_luma[ ( nCbS \* j ) + i ]  <<  ( BitDepthY − PcmBitDepthY ), with i, j = 0..nCbS − 1 (8‑12)

* Otherwise if (pcm\_flag[ xCb ][ yCb ] is equal to 0 and palette\_mode\_flag[ xCb ][ yCb ] is equal to 1), the decoding process is specified as follows:

For each i, j = 0..nCbS – 1,

SL[ xCb + i ][ yCb + j ] = sample\_array[ 0 ][ i ][ j ]

– 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. When independent\_uniform\_pred\_flag[ xCb ][ yCb ] is equal to 0 and intra\_bc\_flag[ xCb ][ yCb ] is equal to 0, the derivation process for the intra prediction mode as specified in subclause 8.4.2 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 in intra block copying prediction mode as specified in subclause 8.4.4 is invoked with the luma location ( xCb, yCb ) and variable log2CbSize as inputs, and the output being bvIntra.
3. The general decoding process for intra blocks as specified in subclause 8.4.4.1 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 predModeIntraBc set equal to intra\_bc\_flag[ xCb ][ yCb ], the variable predModeIntraIUP set equal to independent\_uniform\_pred\_flag[ xCb ][ yCb ], the variable bvIntra, and the variable cIdx set equal to 0 as inputs, and the output is a modified reconstructed picture before deblocking filtering.

…

When ChromaArrayType is not equal to 0, the following applies.

The variable log2CbSizeC is set equal to log2CbSize − ( ChromaArrayType  = =  3 ? 0 : 1 ).

Depending on the value of pcm\_flag[ xCb ][ yCb ], independent\_uniform\_pred\_flag[ xCb ][ yCb ], and IntraSplitFlag, the decoding process for chroma samples is specified as follows:

– If pcm\_flag[ xCb ][ yCb ] is equal to 1, the reconstructed picture is modified as follows:

SCb[ xCb / SubWidthC + i ][ yCb / SubHeightC + j ] =  pcm\_sample\_chroma[ ( nCbS / SubWidthC \* j ) + i ]  <<  
 ( BitDepthC − PcmBitDepthC ), with i = 0..nCbS / SubWidthC − 1, and j = 0..nS / SubHeightC − 1 (8‑13)

SCr[ xCb / SubWidthC + i ][ yCb / SubHeightC + j ] = pcm\_sample\_chroma[ ( nCbS / SubWidthC \* ( j + nCbS / SubHeightC ) ) + i ]  <<  
 ( BitDepthC − PcmBitDepthC ), with i = 0..nCbS / SubWidthC − 1, and j = 0..nS / SubHeightC − 1 (8‑14)

* Otherwise (pcm\_flag[ xCb ][ yCb ] is equal to 0, palette\_mode\_flag[ xCb ][ yCb ] is equal to 1), the decoding process is specified as follows

For each i = 0..nCbS/ SubWidthC – 1, j = 0..nCbS / SubHeightC – 1,

SCb[ xCb + i ][ yCb + j ] = sample\_array[ 1 ][ i ][ j ]

For each i = 0..nCbS/ SubWidthC – 1, j = 0..nCbS / SubHeightC – 1,

SCr[ xCb + i ][ yCb + j ] = sample\_array[ 2 ][ i ][ j ]

– Otherwise (pcm\_flag[ xCb ][ yCb ] is equal to 0, palette\_mode\_flag[ xCb ][ yCb ] is equal to 0), if IntraSplitFlag is equal to 0 or ChromaArrayType is not equal to 3, the following ordered steps apply:

1. When intra\_bc\_flag[ xCb ][ yCb ] is equal to 0 and independent\_uniform\_pred\_flag[ xCb ][ yCb ] is equal to 0, the derivation process for the chroma intra prediction mode as specified in 8.4.3 is invoked with the luma location ( xCb, yCb ) as input, and the output is the variable IntraPredModeC.
2. The general decoding process for intra blocks as specified in subclause 8.4.4.1 is invoked with the chroma location ( xCb / SubWidthC, yCb / SubHeightC ), the variable log2TrafoSize set equal to log2CbSizeC, the variable trafoDepth set equal to 0, the variable predModeIntra set equal to IntraPredModeC, the variable predModeIntraBc set equal to intra\_bc\_flag[ xCb ][ yCb ], the variable predModeIntraIUP set equal to independent\_uniform\_pred\_flag[ xCb ][ yCb ], the variable bvIntra, and the variable cIdx set equal to 1 as inputs, and the output is a modified reconstructed picture before deblocking filtering.
3. The general decoding process for intra blocks as specified in subclause 8.4.4.1 is invoked with the chroma location ( xCb / SubWidthC, yCb / SubHeightC ), the variable log2TrafoSize set equal to log2CbSizeC, the variable trafoDepth set equal to 0, the variable predModeIntra set equal to IntraPredModeC, the variable predModeIntraBc set equal to intra\_bc\_flag[ xCb ][ yCb ], the variable predModeIntraIUP set equal to independent\_uniform\_pred\_flag[ xCb ][ yCb ], the variable bvIntra, and the variable cIdx set equal to 2 as inputs, and the output is a modified reconstructed picture before deblocking filtering.

**8.4.2 Derivation process for luma intra prediction mode**

…

IntraPredModeY[ xPb ][ yPb ] is derived by the following ordered steps:

1. The neighbouring locations ( xNbA, yNbA ) and ( xNbB, yNbB ) are set equal to ( xPb − 1, yPb ) and ( xPb, yPb − 1 ), respectively.
2. For X being replaced by either A or B, the variables candIntraPredModeX are derived as follows:

* The availability derivation process for a block in z-scan order as specified in subclause 6.4.1 is invoked with the location ( xCurr, yCurr ) set equal to ( xPb, yPb ) and the neighbouring location ( xNbY, yNbY ) set equal to ( xNbX, yNbX ) as inputs, and the output is assigned to availableX.
* The candidate intra prediction mode candIntraPredModeX is derived as follows:
* If availableX is equal to FALSE, candIntraPredModeX is set equal to INTRA\_DC.
* Otherwise, if CuPredMode[ xNbX ][ yNbX ] is not equal to MODE\_INTRA or pcm\_flag[ xNbX ][ yNbX ] is equal to 1 or intra\_bc\_flag[ xNbX ][ yNbX ] is equal to 1 or independent\_uniform\_pred\_flag [ xNbX ][ yNbX ] is equal to 1, candIntraPredModeX is set equal to INTRA\_DC,
* Otherwise, if X is equal to B and yPb − 1 is less than ( ( yPb  >>  CtbLog2SizeY )  <<  CtbLog2SizeY ), candIntraPredModeB is set equal to INTRA\_DC.
* Otherwise, candIntraPredModeX is set equal to IntraPredModeY[ xNbX ][ yNbX ].

…

### 8.4.5 Decoding process for intra blocks

#### 8.4.5.1 General decoding process for intra blocks

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 log2TrafoSize specifying the size of the current transform block,

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

– a variable predModeIntra specifying the intra prediction mode,

– a variable predModeIntraBc specifying the intra block copying mode,

– a variable predModeIntraIUP specifying the independent uniform prediction mode,

– a variable bvIntra specifying the intra block copying vector,

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

Output of this process is a modified reconstructed picture before deblocking filtering.

…

1. Depending upon the value of predModeIntraBc and predModeIntraIUP, the following applies:

– When predModeIntraBc is equal to 0 and predModeIntraIUP is equal to 0, the general intra sample prediction process as specified in subclause 8.4.4.2.1 is invoked with the transform block location ( xTb0, yTb0 + yTbOffset ), the intra prediction mode predModeIntra, the transform block size nTbS, and the variable cIdx as inputs, and the output is an (nTbS)x(nTbS) array predSamples.

– Otherwise if predModeIntraIUP is equal to 0 and~~(~~ predModeIntraBc is equal to 1~~)~~, the intra block copying process as specified in subclause 8.4.4.2.7 is invoked with the transform block location ( xTb0, yTb0 + yTbOffset ), the transform block size nTbS, the variable trafoDepth, the variable bvIntra, and the variable cIdx as inputs, and the output is an (nTbS)x(nTbS) array predSamples.

– Otherwise (predModeIntraIUP is equal to 1 and predModeIntraBc is equal to 0), the independent uniform prediction process as specified in subclause 8.4.5.2.8 is invoked with the transform block location ( xTb0, yTb0 + yTbOffset ), the transform block size nTbS, the variable trafoDepth, the variable bvIntra, and the variable cIdx as inputs, and the output is an (nTbS)x(nTbS) array predSamples.

6. The scaling and transformation process as specified in subclause 8.6.2 is invoked with the luma location ( xTbY, yTbY + yTbOffsetY ), the variable trafoDepth, the variable cIdx, and the transform size trafoSize set equal to nTbS as inputs, and the output is an (nTbS)x(nTbS) array resSamples.

7. When residualDpcm is equal to 1, depending upon the value of predModeIntraBc and predModeIntraIUP, the following applies:

– When predModeIntraBc is equal to 0 and predModeIntraIUP is equal to 0, the directional residual modification process for blocks using a transform bypass as specified in subclause 8.6.5 is invoked with the variable mDir set equal to predModeIntra / 26, the variable nTbS, and the (nTbS)x(nTbS) array r set equal to the array resSamples as inputs, and the output is a modified (nTbS)x(nTbS) array resSamples.

– Otherwise, (predModeIntraBc is equal to 1 or predModeIntraIUP is equal to 1), the directional residual modification process for blocks using a transform bypass as specified in subclause 8.6.5 is invoked with the variable mDir set equal to explicit\_rdpcm\_dir\_flag[ xTbY ][ yTbY + yTbOffsetY ][ cIdx ], the variable nTbS, and the (nTbS)x(nTbS) array r set equal to the array resSamples as inputs, and the output is a modified (nTbS)x(nTbS) array resSamples.

**8.4.5.2.8 Specification of independent uniform 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 2-D variable uniform\_colors specifying the uniform colors,

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

– a variable uniform\_color\_index specifying the index of selected uniform color for the current block.

Output of this process is the predicted samples predSamples[ x ][ y ], with x, y = 0..nTbS − 1.

predSamples[ x ][ y ]= uniform\_colors[cIdx] [uniform\_color\_index[xTb0, yTb0]]

**8.6.2 Scaling and transformation process**

…

The variable rotateCoeffs is derived as follows:

– If all of the following conditions are true, rotateCoeffs is set equal to 1:

– transform\_skip\_rotation\_enabled\_flag is equal to 1

– nTbS is equal to 4

– CuPredMode[ xTbY ][ yTbY ] is not equal to MODE\_INTER

– intra\_bc\_flag[ xTbY ][ yTbY ] is equal to 0

– independent\_uniform\_pred\_flag [ xTbY ][ yTbY ] is equal to 0

– Otherwise, rotateCoeffs is set equal to 0

**9.3.2.2 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** |
| sao( ) | sao\_merge\_left\_flag sao\_merge\_up\_flag | Table 9‑5 | 0 | 1 | 2 |
| sao\_type\_idx\_luma sao\_type\_idx\_chroma | Table 9‑6 | 0 | 1 | 2 |
| coding\_quadtree( ) | split\_cu\_flag[ ][ ] | Table 9‑7 | 0..2 | 3..5 | 6..8 |
| coding\_unit( ) | cu\_transquant\_bypass\_flag | Table 9‑8 | 0 | 1 | 2 |
| cu\_skip\_flag | Table 9‑9 |  | 0..2 | 3..5 |
| independent\_uniform\_pred\_flag | Table 9-X1 | 0 | 1 | 2 |
| uniform\_color\_index | Table 9-X2 | 0 | 1 | 2 |
| intra\_bc\_flag[ ][ ] | Table 9‑33 | 0 | 1 | 2 |
| pred\_mode\_flag | Table 9‑10 |  | 0 | 1 |
| … |  |  |  |  |
| … |  |  |  |  |  |

Table 9‑X1 – Values of initValue for ctxIdx of independent\_uniform\_pred\_flag

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of independent\_uniform\_pred\_flag** | | |
| **0** | **1** | **2** |
| **initValue** | 185 | 197 | 197 |

Table 9‑X2 – Values of initValue for ctxIdx of uniform\_color\_index

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of uniform\_color\_index** | | |
| **0** | **1** | **2** |
| **initValue** | 185 | 197 | 197 |

### 9.3.3 Binarization process

#### 9.3.3.1 General

| **Table 9‑34 – Syntax elements and associated binarizations** | | | |
| --- | --- | --- | --- |
| **Syntax structure** | **Syntax element** | **Binarization** | |
| **Process** | **Input parameters** |
| coding\_unit( ) | … | … | … |
| independent\_uniform\_pred\_flag | FL | cMax = 1 |
| uniform\_color\_index | TR | cMax = num\_uniform\_colors, cRiceParam=0 |
| … | … | … |

#### 9.3.3.5 Binarization process for part\_mode

Table 9‑36 – Binarization for part\_mode

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **CuPredMode** [ xCb ][ yCb ] | **independent\_uniform\_pred\_flag** [ xCb ][ yCb ] | **intra\_bc\_flag** [ xCb ][ yCb ] | **part\_mode** | **PartMode** | **Bin string** | | | |
| log2CbSize >  MinCbLog2SizeY | | log2CbSize  = =  MinCbLog2SizeY | |
| !amp\_enabled\_flag | amp\_enabled\_flag | log2CbSize  = =  3 | log2CbSize > 3 |
| MODE\_INTRA | 0 | 0 | 0 | PART\_2Nx2N | - | - | 1 | 1 |
| 1 | PART\_NxN | - | - | 0 | 0 |
| 1 | 0 | PART\_2Nx2N | 1 | 1 | 1 | 1 |
| 1 | PART\_2NxN | 01 | 01 | 01 | 01 |
| 2 | PART\_Nx2N | 00 | 00 | 001 | 001 |
| 3 | PART\_NxN | - | - | 000 | 000 |
| 1 | - | 0 | PART\_2Nx2N | - | - | - | - |

**9.3.4.2 Derivation process for ctxTable, ctxIdx and bypassFlag**

**9.3.4.2.1 General**

**Table 9‑39 – Assignment of ctxInc to syntax elements with context coded bins**

| **Syntax element** | **binIdx** | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **>= 5** |
| Independent\_uniform\_pred\_flag | 0 | na | na | na | na | na |
| uniform\_color\_index | 0 | bypass | bypass | bypass | bypass | bypass |