**Working draft modification for JCTVC-H0455**

All the modifications are highlighted by yellow.

### 7.3.2 Raw byte sequence payloads and RBSP trailing bits syntax

#### 7.3.2.1 Sequence parameter set RBSP syntax

|  |  |
| --- | --- |
| seq\_parameter\_set\_rbsp( ) { | Descriptor |
| **profile\_idc** | u(8) |
| **reserved\_zero\_8bits** /\* equal to 0 \***/** | u(8) |
| **chroma\_pred\_from\_luma\_enabled\_flag** | u(1) |
| **loop\_filter\_across\_slice\_flag** | u(1) |
| **sample\_adaptive\_offset\_enabled\_flag** | u(1) |
| **adaptive\_loop\_filter\_enabled\_flag** | u(1) |
| if ( pcm\_enabled\_flag ) |  |
| **pcm\_loop\_filter\_disable\_flag** | u(1) |
| **intra\_sdip\_enabled\_flag** | u(1) |
| **temporal\_id\_nesting\_flag** | u(1) |
| **inter\_4x4\_enabled\_flag** | u(1) |
| **num\_tile\_columns\_minus1** | ue(v) |
| **num\_tile\_rows\_minus1** | ue(v) |
| if ( num\_tile\_columns\_minus1 != 0 | | num\_tile\_rows\_minus1 != 0 ) { |  |
| **uniform\_spacing\_flag** | u(1) |
| if ( !uniform\_spacing\_flag ) { |  |
| for ( i = 0; i < num\_tile\_columns\_minus1; i++ ) |  |
| **column\_width[**i**]** | ue(v) |
| for ( i = 0; i < num\_tile\_rows\_minus1; i++ ) |  |
| **row\_height[**i**]** | ue(v) |
| } |  |
| **tile\_boundary\_independence\_flag** | u(1) |
| if ( tile\_boundary\_independence\_flag = = 1 ) |  |
| **loop\_filter\_across\_tile\_flag** | u(1) |
| } |  |
| rbsp\_trailing\_bits( ) |  |
| } |  |

### 7.3.6 Coding unit syntax

|  |  |
| --- | --- |
| coding\_unit( x0, y0, log2CUSize ) { | Descriptor |
| sdipEnabledFlag = intra\_sdip\_enabled\_flag && log2CUSize < 6 |  |
| if( slice\_type != I ) |  |
| **skip\_flag[** x0 **][** y0 **]** | ae(v) |
| if( skip\_flag[ x0 ][ y0 ] ) |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| else if( slice\_type != I | | log2CUSize = = Log2MinCUSize ) { |  |
| if( slice\_type != I ) |  |
| **pred\_mode\_flag** | ae(v) |
| if( PredMode != MODE\_INTRA | | log2CUSize = = Log2MinCUSize || sdipEnabledFlag) |  |
| **part\_mode** | ae(v) |
| x1 = x0 + ( ( 1 << log2CUSize ) >> 1 ) |  |
| y1 = y0 + ( ( 1 << log2CUSize ) >> 1 ) |  |
| x2 = x1 − ( ( 1 << log2CUSize ) >> 2 ) |  |
| y2 = y1 − ( ( 1 << log2CUSize ) >> 2 ) |  |
| x3 = x1 + ( ( 1 << log2CUSize ) >> 2 ) |  |
| y3 = y1 + ( ( 1 << log2CUSize ) >> 2 ) |  |
| if( PartMode == PART\_2Nx2N ) { |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| } else if( PartMode == PART\_2NxN ) { |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| prediction\_unit( x0, y1 , log2CUSize ) |  |
| } else if( PartMode == PART\_Nx2N ) { |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| prediction\_unit( x1, y0 , log2CUSize ) |  |
| } else if( PartMode == PART\_2NxnU ) { |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| prediction\_unit( x0, y2 , log2CUSize ) |  |
| } else if( PartMode == PART\_2NxnD ) { |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| prediction\_unit( x0, y3 , log2CUSize ) |  |
| } else if( PartMode == PART\_nLx2N ) { |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| prediction\_unit( x2, y0 , log2CUSize ) |  |
| } else if( PartMode == PART\_nRx2N ) { |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| prediction\_unit( x3, y0 , log2CUSize ) |  |
| } else { /\* PART\_NxN \*/ |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| prediction\_unit( x1, y0 , log2CUSize ) |  |
| prediction\_unit( x0, y1 , log2CUSize ) |  |
| prediction\_unit( x1, y1 , log2CUSize ) |  |
| } |  |
| if( !pcm\_flag ) { |  |
| transform\_tree( x0, y0, log2CUSize, log2CUSize, log2CUSize, 0, 0 ) |  |
| transform\_coeff( x0, y0, x0, y0, log2CUSize, log2CUSize, 0, 0 ) |  |
| } |  |
| } |  |
| } |  |

### 7.3.8 Transform tree syntax

|  |  |
| --- | --- |
| transform\_tree( x0, y0, log2CUSize, log2TrafoWidth, log2TrafoHeight, trafoDepth, blkIdx ) { | Descriptor |
| if( trafoDepth = = 0 && IntraSplitFlag = = 0 && PredMode != MODE\_INTRA &&   !(PartMode = = PART\_2Nx2N && merge\_flag[x0][y0]) ) |  |
| **no\_residual\_data\_flag** | ae(v) |
| if( !no\_residual\_data\_flag ) { |  |
| log2TrafoSize = ( log2TrafoWidth + log2TrafoHeight ) >> 1 |  |
| intraSdipFlag = (PredMode == MODE\_INTRA && TUSplitDirection!=2) |  |
| intraSplitFlag = ( IntraSplitFlag && trafoDepth = = 0 ? 1 : 0 ) |  |
| interSplitFlag = ( max\_transform\_hierarchy\_depth\_inter = =0 &&  PredMode = = MODE\_INTER && PartMode != PART\_2Nx2N &&  trafoDepth = = 0 ) |  |
| maxDepth = ( PredMode = = MODE\_INTRA ?   max\_transform\_hierarchy\_depth\_intra + IntraSplitFlag :   max\_transform\_hierarchy\_depth\_inter + InterSplitFlag ) |  |
| if(TUSplitDirection==0 && intraSdipFlag){ |  |
| xBase = x0 |  |
| yBase = y0 − ( y0 & ( ( 1 << (log2TrafoHeight + 2)) − 1 ) ) |  |
| }else if(intraSdipFlag){ |  |
| xBase = x0 − ( x0 & ( ( 1 << (log2TrafoWidth + 2)) − 1 ) ) |  |
| yBase = y0 |  |
| }else{ |  |
| xBase = x0 − ( x0 & ( 1 << log2TrafoWidth ) ) |  |
| yBase = y0 − ( y0 & ( 1 << log2TrafoHeight ) ) |  |
| } |  |
| if( log2TrafoSize <= Log2MaxTrafoSize &&   log2TrafoSize > Log2MinTrafoSize &&  trafoDepth < maxDepth && !intraSplitFlag && !interSplitFlag ) |  |
| **split\_transform\_flag**[ x0 ][ y0 ][ trafoDepth ] | ae(v) |
| if( PredMode != MODE\_INTRA &&  log2TrafoSize <= Log2MaxTrafoSize ) { |  |
| firstChromaCbf = ( log2TrafoSize = = Log2MaxTrafoSize | |  trafoDepth = = 0 ) ? 1 : 0 |  |
| if( firstChromaCbf | | log2TrafoSize > Log2MinTrafoSize ) {  [Ed. (WJ): Log2MinTrafoSize or 2?] |  |
| if( firstChromaCbf | | cbf\_cb[ xBase ][ yBase ][ trafoDepth − 1 ] ) { |  |
| readCbf = true |  |
| if( blkIdx = = 3 && log2TrafoSize < Log2MaxTrafoSize ) |  |
| readCbf = cbf\_cb[ xBase ][ yBase ][ trafoDepth ] | |   cbf\_cb[ xBase + ( 1 << log2TrafoWidth ) ][ yBase ][ trafoDepth ] | |   cbf\_cb[ xBase ][ yBase + ( 1 << log2TrafoHeight ) ][ trafoDepth ] |  |
| if ( !readCbf ) |  |
| cbf\_cb[ x0 ][ y0 ][ trafoDepth ] = 1 |  |
| else |  |
| **cbf\_cb**[ x0 ][ y0 ][ trafoDepth ] | ae(v) |
| } |  |
| if( firstChromaCbf | | cbf\_cr[ xBase ][ yBase ][ trafoDepth − 1 ] ) { |  |
| readCbf = true |  |
| if( blkIdx = = 3 && log2TrafoSize < Log2MaxTrafoSize ) |  |
| readCbf = cbf\_cr[ xBase ][ yBase ][ trafoDepth ] | |   cbf\_cr[ xBase + ( 1 << log2TrafoWidth ) ][ yBase ][ trafoDepth ] | |   cbf\_cr[ xBase ][ yBase + ( 1 << log2TrafoHeight ) ][ trafoDepth ] |  |
| if ( !readCbf ) |  |
| cbf\_cr[ x0 ][ y0 ][ trafoDepth ] = 1 |  |
| else |  |
| **cbf\_cr**[ x0 ][ y0 ][ trafoDepth ] | ae(v) |
| } |  |
| } |  |
| } |  |
| if( split\_transform\_flag[ x0 ][ y0 ][ trafoDepth ] ) { |  |
| if(TUSplitDirection = = 2 ) { |  |
| x1 = x0 + ( ( 1 << log2TrafoWidth ) >> 1 ) |  |
| y1 = y0 |  |
| x2 = x0 |  |
| y2 = y0 + ( ( 1 << log2TrafoHeight ) >> 1 ) |  |
| x3 = x1 |  |
| y3 = y2 |  |
| } else { |  |
| x1 = x0 + ( ( 1 << log2TrafoWidth ) >> 2 ) \* TUSplitDirection |  |
| y1 = y0 + ( ( 1 << log2TrafoHeight) >> 2 ) \* ( 1 − TUSplitDirection ) |  |
| x2 = x1 + ( ( 1 << log2TrafoWidth ) >> 2 ) \* TUSplitDirection |  |
| y2 = y1 + ( ( 1 << log2TrafoHeight) >> 2 ) \* ( 1 − TUSplitDirection ) |  |
| x3 = x2 + ( ( 1 << log2TrafoWidth ) >> 2 ) \* TUSplitDirection |  |
| y3 = y2 + ( ( 1 << log2TrafoHeight) >> 2 ) \* ( 1 − TUSplitDirection ) |  |
| log2TrafoHeight = log2TrafoHeight + 2 \*TUSplitDirection − 1 |  |
| log2TrafoWidth = log2TrafoWidth − 2 \* TUSplitDirection + 1 |  |
| } |  |
| transform\_tree( x0, y0, log2CUSize, log2TrafoWidth − 1, log2TrafoHeight − 1,   trafoDepth + 1, 0 ) |  |
| transform\_tree( x1, y1, log2CUSize, log2TrafoWidth − 1, log2TrafoHeight − 1,   trafoDepth + 1, 1 ) |  |
| transform\_tree( x2, y2, log2CUSize, log2TrafoWidth − 1, log2TrafoHeight − 1,   trafoDepth + 1, 2 ) |  |
| transform\_tree( x3, y3, log2CUSize, log2TrafoWidth − 1, log2TrafoHeight − 1,   trafoDepth + 1, 3 ) |  |
| } else { |  |
| if( PredMode = = MODE\_INTRA | | trafoDepth != 0 | |  cbf\_cb[ x0 ][ y0 ][ trafoDepth ] | |  cbf\_cr[ x0 ][ y0 ][ trafoDepth ] ) { |  |
| readCbf = true |  |
| if( blkIdx = = 3 && PredMode != MODE\_INTRA &&   ( ( log2CUSize <= Log2MaxTrafoSize+1 ) || ( log2TrafoSize < Log2MaxTrafoSize ) ) |  |
| readCbf = cbf\_luma[ xBase ][ yBase ][ trafoDepth ] | |   cbf\_luma[ xBase + ( 1 << log2TrafoWidth ) ][ yBase ][ trafoDepth ] | |   cbf\_luma[ xBase ][ yBase + ( 1 << log2TrafoHeight ) ][ trafoDepth ] | |   cbf\_cb[ xBase ][ yBase ][ trafoDepth – 1] | |   cbf\_cr[ xBase ][ yBase ][ trafoDepth – 1] |  |
| if ( !readCbf ) |  |
| cbf\_luma[ x0 ][ y0 ][ trafoDepth ] = 1 |  |
| else |  |
| **cbf\_luma**[ x0 ][ y0 ][ trafoDepth ] | ae(v) |
| } |  |
| if( PredMode = = MODE\_INTRA && ( ! intraSdipFlag | | trafoDepth  == 0) ) |  |
| if( log2TrafoSize > Log2MinTrafoSize ) { [Ed. (WJ): Log2MinTrafoSize or 2?] |  |
| **cbf\_cb**[ x0 ][ y0 ][ trafoDepth ] | ae(v) |
| **cbf\_cr**[ x0 ][ y0 ][ trafoDepth ] | ae(v) |
| } else if( blkIdx = = 0 ) { |  |
| **cbf\_cb**[ x0 ][ y0 ][ trafoDepth − 1 ] | ae(v) |
| **cbf\_cr**[ x0 ][ y0 ][ trafoDepth − 1 ] | ae(v) |
| } |  |
| } |  |
| } |  |
| } |  |

### 7.3.9 Transform coefficient syntax

|  |  |
| --- | --- |
| transform\_coeff( x0, y0, xC, yC, log2TrafoWidth, log2TrafoHeight, trafoDepth, blkIdx ) { | Descriptor |
| if( cbf\_luma[ x0 ][ y0 ][ trafoDepth ] | | cbf\_cb[ x0 ][ y0 ][ trafoDepth ] | |  cbf\_cr[ x0 ][ y0 ][ trafoDepth ] { |  |
| if( cu\_qp\_delta\_enabled\_flag && !IsCuQpDeltaCoded ) { |  |
| **cu\_qp\_delta** | ae(v) |
| IsCuQpDeltaCoded = 1 |  |
| } |  |
| log2TrafoSize = (log2TrafoWidth + log2TrafoHeight) >> 1 |  |
| if( split\_transform\_flag[ x0 ][ y0 ][ trafoDepth ] ) { |  |
| if(TUSplitDirection = = 2 ) { |  |
| x1 = x0 + ( ( 1 << log2TrafoWidth ) >> 1 ) |  |
| y1 = y0 |  |
| x2 = x0 |  |
| y2 = y0 + ( ( 1 << log2TrafoHeight ) >> 1 ) |  |
| x3 = x1 |  |
| y3 = y2 |  |
| } else { |  |
| x1 = x0 + ( ( 1 << log2TrafoWidth ) >> 2 ) \* TUSplitDirection |  |
| y1 = y0 + ( ( 1 << log2TrafoHeight) >> 2 ) \* ( 1 − TUSplitDirection ) |  |
| x2 = x1 + ( ( 1 << log2TrafoWidth ) >> 2 ) \* TUSplitDirection |  |
| y2 = y1 + ( ( 1 << log2TrafoHeight) >> 2 ) \* ( 1 − TUSplitDirection ) |  |
| x3 = x2 + ( ( 1 << log2TrafoWidth ) >> 2 ) \* TUSplitDirection |  |
| y3 = y2 + ( ( 1 << log2TrafoHeight) >> 2 ) \* ( 1 − TUSplitDirection ) |  |
| log2TrafoHeight = log2TrafoHeight + 2 \* TUSplitDirection − 1 |  |
| log2TrafoWidth = log2TrafoWidth − 2 \* TUSplitDirection + 1 |  |
| } |  |
| transform\_coeff( x0, y0, x0, y0, log2TrafoWidth − 1, log2TrafoHeight − 1, trafoDepth + 1, 0 ) |  |
| transform\_coeff( x1, y1, x0, y0, log2TrafoWidth − 1, log2TrafoHeight − 1, trafoDepth + 1, 1 ) |  |
| transform\_coeff( x2, y2, x0, y0, log2TrafoWidth − 1, log2TrafoHeight − 1, trafoDepth + 1, 2 ) |  |
| transform\_coeff( x3, y3, x0, y0, log2TrafoWidth − 1, log2TrafoHeight − 1, trafoDepth + 1, 3 ) |  |
| } else { |  |
| log2TrafoSize = ( ( log2TrafoWidth + log2TrafoHeight ) >> 1 ) |  |
| log2TrafoSizeC = ( ( log2TrafoSize = = Log2MinTrafoSizeC ) ?  log2TrafoSize : log2TrafoSize – 1 ) |  |
| if ( PredMode = = MODE\_INTRA ) { |  |
| scanIdx = ScanType[ log2TrafoSize – 2 ][ IntraPredMode ] |  |
| scanIdxC = ScanType[ log2TrafoSize – 2 ][ IntraPredModeC ] |  |
| } else { |  |
| scanIdx = 0 |  |
| scanIdxC = 0 |  |
| } |  |
| if ( cbf\_luma[ x0 ][ y0 ][ trafoDepth ] ) |  |
| residual\_coding( x0, y0, log2TrafoWidth, log2TrafoHeight, scanIdx, 0 ) |  |
| if ( log2TrafoSize > Log2MinTrafoSize ) { [Ed. (WJ): Log2MinTrafoSize or 2?] |  |
| if ( cbf\_cb[ x0 ][ y0 ][ trafoDepth ] ) |  |
| residual\_coding( x0, y0, log2TrafoSizeC, trafoDepth, scanIdxC, 1 ) |  |
| if ( cbf\_cr[ x0 ][ y0 ][ trafoDepth ] ) |  |
| residual\_coding( x0, y0, log2TrafoSizeC, trafoDepth, scanIdxC, 2 ) |  |
| } else if ( blkIdx == 3 ) { |  |
| if ( cbf\_cb[ x0 ][ y0 ][ trafoDepth ] ) |  |
| residual\_coding( xC, yC, log2TrafoSizeC, trafoDepth, scanIdxC, 1 ) |  |
| if ( cbf\_cr[ x0 ][ y0 ][ trafoDepth ] ) |  |
| residual\_coding( xC, yC, log2TrafoSizeC, trafoDepth, scanIdxC, 2 ) |  |
| } |  |
| } |  |
| } |  |
| } |  |

## 7.4 Semantics

**pcm\_loop\_filter\_disable\_flag** specifies whether the loop filter process is disabled on reconstructed pixels of I\_PCM blocks. If the pcm\_loop\_filter\_disable\_flag value is equal to 1, both deblocking and adaptive loop filter processes on the reconstructed pixels of I\_PCM blocks are disabled; otherwise the pcm\_loop\_filter\_disable\_flag value is equal to 0, both deblocking and adaptive loop filter processes on the reconstructed pixels of I\_PCM blocks are not disabled.

[Ed. (WJ): select one expression – enabled\_flag or disable\_flag]

**intra\_sdip\_enabled\_flag** equal to 1 specifies that the short distance intra prediction process is applied according to the intra luma prediction mode.

**temporal\_id\_nesting\_flag** specifies whether inter prediction is additionally restricted for the coded video sequence.

Dependent on temporal\_id\_nesting\_flag, the following applies.

– If temporal\_id\_nesting\_flag is equal to 0, additional constraints may not be obeyed.

– Otherwise (temporal\_id\_nesting\_flag is equal to 1), the following applies.

– For each access unit auA with temporal\_id equal to tIdA, an access unit auB with temporal\_id equal to tIdB and tIdB less than or equal to tIdA shall not be referenced by inter prediction when there exists an access unit auC with temporal\_id equal to tIdC and tIdC less than tIdB, which follows the access unit auB and precedes the access unit auA in decoding order.

NOTE – The syntax element temporal\_id\_nesting\_flag is used to indicate that temporal up-switching, i.e., switching from decoding of up to a specific temporal\_id tIdN to decoding up to a temporal\_id tIdM > tIdN, is always possible.

### 7.4.8 Transform tree semantics

The variable TUSplitDirection defines how a transform block is split into four blocks with smaller horizontal or vertical size for the purpose of transform coding. The blocks are half horizontal and vertical size when TUSplitDirection is equal to 2, full horizontal and quarter vertical size when TUSplitDirection is equal to 0, quarter horizontal and full vertical size when TUSplitDirection is equal to 1. TUSplitDirection is specified as follows.

if( ( ( log2TrafoSize = = Log2MaxTrafoSize  
 | | ( log2TrafoSize < Log2MaxTrafoSize && trafoDepth = = 0 ) )   
 && log2TrafoSize > ( Log2MinTrafoSize + 1 )  
 && ( PartMode = = PART\_2NxN  
 | | PartMode = = PART\_2NxnU | | PartMode = = PART\_2NxnD ) )  
 | | ( log2TrafoSize = = ( Log2MinTrafoSize + 1 )  
 && log2TrafoWidth < log2TrafoHeight ) )   
{  
 TUSplitDirection = 0  
}  
else if( ( ( log2TrafoSize = = Log2MaxTrafoSize  
 | | ( log2TrafoSize < Log2MaxTrafoSize && trafoDepth = = 0 ) )   
 && log2TrafoSize > ( Log2MinTrafoSize + 1 )  
 && ( PartMode = = PART\_Nx2N  
 | | PartMode = = PART\_nLx2N | | PartMode = = PART\_nRx2N ) )  
 | | ( log2TrafoSize = = ( Log2MinTrafoSize + 1 )  
 && log2TrafoWidth > log2TrafoHeight ) )   
{  
 TUSplitDirection = 1  
}  
else   
{  
 TUSplitDirection = 2  
}

Table 7‑11 ‑ Name association to prediction mode and partitioning type

|  |  |  |  |
| --- | --- | --- | --- |
| **PredMode** | **part\_mode** | **IntraSplitFlag** | **PartMode** |
| MODE\_INTRA | 0 | 0 | PART\_2Nx2N |
| 1 | 1 | PART\_2NxN |
| 2 | 1 | PART\_Nx2N |
| 3 | 1 | PART\_NxN |
| 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 |

# Decoding process

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

Inputs to this process are:

– a luma location ( xB, yB ) specifying the top-left luma sample of the current coding unit relative to the top‑left luma sample of the current picture,

– a variable log2CUSize specifying the size of the current coding unit.

Output of this process is:

– a modified reconstructed picture before deblocking filtering.

A variable nS is set equal to ( 1 << log2CUSize ).

Depending on pcm\_flag and IntraSplitFlag and PartMode, the decoding process for luma samples is specified as follows.

– If pcm\_flag is equal to 1, the reconstucted picture is modified as follows:

recSamplesL[ xB + i, yB + j ] =   
 pcm\_sample\_luma[ ( nS \* j ) + i ] << ( BitDepthY – PCMBitDepthY ), with i, j = 0..nS-1 (8‑13)

– Otherwise (pcm\_flag 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 8.3.1 is invoked with the luma location ( xB, yB ) and the variable log2PUSize set equal to log2CUSize as well as IntraPredMode that is previously (in decoding order) derived for adjacent coding units as the input and the output is the variable IntraPredMode[ xB ][ yB ].
2. The decoding process for intra blocks as specified in subclause 8.3.3 is invoked with the luma location ( xB, yB ), the variable log2TrafoSize set equal to log2CUSize, the variable trafoDepth set equal to 0, the luma intra prediction mode IntraPredMode[ xB ][ yB ] and the variable cIdx set equal to 0 as the inputs and the output is a modified reconstructed picture before deblocking filtering.

– Otherwise (pcm\_flag is equal to 0 and IntraSplitFlag is equal to 1), for the variable blkIdx (proceeding over the values 0.1 for PartMode PART\_2NxN and PART\_Nx2N ,0..3 for PartMode PART\_NxN ), the following ordered steps apply:

1. Depending on PartMode, the variable xBS is set equal to the values as follows

xB + ( (1 << log2CUSize ) >> 1 ) \* ( blkIdx % 2 ), when PartMode is equal to PART\_NxN

xB, when PartMode is equal to PART\_2NxN

xB + ( (1 << log2CUSize ) >> 2) \*  blkIdx\*2, when PartMode is equal to PART\_Nx2N

1. Depending on PartMode, the variable yBS is set equal to the values as follows

yB + ( (1 << log2CUSize ) >> 1 ) \* ( blkIdx / 2 ), when PartMode is equal to PART\_NxN or PART\_2Nx2N

yB + ( (1 << log2CUSize ) >> 2 ) \*  blkIdx\*2, when PartMode is equal to PART\_2NxN

yB, when PartMode is equal to PART\_Nx2N

1. Depending on PartMode, the variables log2TrafoWidth and log2TrafoHeight are set equal to the values as follows
   * When PartMode is equal to PART\_NxN or PART\_2Nx2N

log2TrafoWidth = log2CUSize – 1

log2TrafoHeight= log2CUSize – 1

* + When PartMode is equal to PART\_2NxN

log2TrafoWidth = log2CUSize, log2PUWidth= log2TrafoWidth

log2TrafoHeight= log2CUSize – 2, log2PUHeight= log2TrafoHeight+1

* + When PartMode is equal to PART\_Nx2N

log2TrafoWidth = log2CUSize – 2, log2PUWidth= log2TrafoWidth+1

log2TrafoHeight= log2CUSize , log2PUHeight= log2TrafoHeight

1. The derivation process for the intra prediction mode as specified in subclause 8.3.1 is invoked with the luma location ( xBS, yBS ) and the variables log2PU Width, log2PUHeight are set equal to log2CUSize  or log2CUSize – 1 respectively and the variable blkIdx as well as IntraPredMode that is previously (in decoding order) derived for adjacent coding units as the input and the output is the variable IntraPredMode[ xBS ][ yBS ].

The decoding process for intra blocks as specified in subclause is invoked with the luma location ( xB, yB ), the variables log2TrafoWidth and log2TrafoHeight, the variable trafoDepth set equal to 1, the prediction partition mode PartMode, the luma intra prediction mode IntraPredMode[ xBS ][ yBS ] and the variable cIdx set equal to 0 as the inputs and the output is a modified reconstructed picture before deblocking filtering.

Depending on pcm\_flag, the decoding process for chroma samples is specified as follows:

– If pcm\_flag is equal to 1, the reconstucted picture is modified as follows:

recSamplesCb[ xB/2 + i, yB/2 + j ] =   
 pcm\_sample\_chroma[ ( nS/2 \* j ) + i ] << ( BitDepthC – PCMBitDepthC ) with i, j = 0..nS/2-1 (8‑13)

recSamplesCr[ xB/2 + i, yB/2 + j ] =   
 pcm\_sample\_chroma[ ( nS/2 \* ( j + nS ) ) + i ] << ( BitDepthC – PCMBitDepthC ) with i, j = 0..nS/2-1 (8‑13)

– Otherwise (pcm\_flag is equal to 0), the following ordered steps apply:

1. The derivation process for the chroma intra prediction mode as specified in 8.3.2 is invoked with the luma location ( xB, yB ) as input and the output is the variable IntraPredModeC.
2. The decoding process for intra blocks as specified in subclause 8.3.3 is invoked with the chroma location ( xB/2, yB/2 ), the variable log2TrafoWidth set equal to log2CUSize-1, the variable log2TrafoHeight set equal to log2CUSize-1, the variable trafoDepth set equal to 0, the chroma intra prediction mode IntraPredModeC, and the variable cIdx set equal to 1 as the inputs and the output is a modified reconstructed picture before deblocking filtering.
3. The decoding process for intra blocks as specified in subclause 8.3.3 is invoked with the chroma location ( xB/2, yB/2 ), the variable log2TrafoWidth set equal to log2CUSize-1, the variable log2TrafoHeight set equal to log2CUSize-1, the variable trafoDepth set equal to 0, the prediction partition mode PartMode, the chroma intra prediction mode IntraPredModeC, and the variable cIdx set equal to 2 as the inputs and the output is a modified reconstructed picture before deblocking filtering.

### Derivation process for luma intra prediction mode

Inputs to this process are:

– a luma location ( xB, yB ) specifying the top-left luma sample of the current block relative to the top‑left luma sample of the current picture,

– two variables log2PUWidth and log2PUHeight specifying the width and height of the current prediction unit,

– variable arrays IntraPredMode (If available) that are previously (in decoding order) derived for adjacent coding units.

Output of this process is the variable IntraPredMode[ xB ][ yB ].

The variable log2PUSize is set equal to (log2PUWidth + log2PUHeight+1)>>1.

Table 8‑5 specifies the value for the intra prediction mode and the associated names.

Table 8‑1 – Specification of intra prediction mode and associated names

|  |  |
| --- | --- |
| **Intra prediction mode** | **Associated names** |
| 0 | Intra\_Planar |
| 1 | Intra\_Vertical |
| 2 | Intra\_Horizontal |
| 3 | Intra\_DC |
| Otherwise (4..34) | Intra\_Angular |
| 35 | Intra\_FromLuma (used only for chroma) |

Table 8‑1 specifies the number of luma intra prediction modes intraPredModeNum depending on log2PUSize.

Table 8‑2 – Specification of intraPredModeNum

|  |  |
| --- | --- |
| **log2PUSize** | **intraPredModeNum** |
| 2 | 18 |
| 3 | 35 |
| 4 | 35 |
| 5 | 35 |
| 6 | 35 |

Table 8‑2 specifies the mapping table used for converting the number of intra prediction modes.

IntraPredMode[ xB ][ yB ] labelled 0, 1, 2, .., 34 represents directions of predictions as illustrated in Figure 8‑1.



Figure 8‑1 – Intra prediction mode directions (informative)

IntraPredMode[ xB ][ yB ] is derived as the following ordered steps. [Ed. (WJ): proponent suggests to move this part to the syntax since the other syntax elements utilize IntraPredMode. But it seems too complex to move all the following process to the syntax table. Maybe it’s better to move this part to the semantics section or simply avoid the use of IntraPredMode to parse the syntax item]

1. The derivation process for neighbouring treeblocks specified in subclause XXX with ( xB,  yB ) given as input and the output is assigned to tbAddrA and tbAddrB specifying the treeblock addresses of treeblocks covering ( xBA,  yBA ) and ( xBB, yBB ) respectively where ( xBA,  yBA ) is set equal to ( xB-1,  yB ) and ( xBB,  yBB ) is set equal to ( xB,  yB-1 ).
2. For N being either replaced A or B, the variables intraPredModeN are derived as follows.

* If the treeblock with address tbAddrN is not available, intraPredModeN is set equal to Intra\_Planar.
* Otherwise, if the coding unit covering ( xBN,  yBN ) is not coded as intra mode, intraPredModeN is set equal to Intra\_Planar,
* Otherwise, if yB-1 is smaller than YLCU, intraPredModeA is set equal to IntraPredMode[ xBA ][ yBA ] and intraPredModeB is set equal to Intra\_Planar.
* Otherwise, intraPredModeN is set equal to IntraPredMode[ xBN ][ yBN ], where IntraPredMode is the variable array assigned to the coding unit covering the luma location ( xBN, yBN ).

1. For N being either replaced A or B, the variables candIntraPredModeN are derived as follows.

* If intraPredModeN is greater than or equal to intraPredModeNum, candIntraPredModeN is set equal to Intra\_Planar.
* Otherwise, candIntraPredModeN is set equal to intraPredModeN

1. If candIntraPredModeA is equal to candIntraPredModeB, the candIntraPredModeA is modified as follows:

* If candIntraPredModeA is not equal to Intra\_Planar, candIntraPredModeA is set equal to Intra\_Planar
* Otherwise, candIntraPredModeA is set equal to Intra\_DC

1. The candModeList[x] is derived as follows:

* If the PartMode is equal to PART\_2Nx2N or PART\_NxN

candModeList[0] = Min( candIntraPredModeA, candIntraPredModeB ) (8‑13)  
candModeList[1] = Max( candIntraPredModeA, candIntraPredModeB ) (8‑13)

* else

candModeList[0] = Intra\_Plana

candModeList[1] = IntraPredMode[ xBB ][ yBB ] when PartMode is equal to PART\_2NxN, else candModeList[1] = IntraPredMode[ xBA ][ yBA ], and set candModeList[1] equal to Intra\_DC if candModeList[1] is equal to Intra\_Planar;

1. IntraPredMode[ xB ][ yB ] is derived by applying the following procedure:

* If prev\_intra\_pred\_flag[ xB ][ yB ] is true, the IntraPredMode[ xB ][ yB ] is set equal to candModeList[ mpm\_flag ][ xB ][ yB ]]
* Otherwise IntraPredMode[ xB ][ yB ] is derived by applying the following ordered steps:
  1. IntraPredMode[ xB ][ yB ] = rem\_intra\_luma\_pred\_mode
  2. When IntraPredMode[ xB ][ yB ] is equal or greater than candModeList[ 0 ], the value of IntraPredMode[ xB ][ yB ] is increased by one
  3. When IntraPredMode[ xB ][ yB ] is equal or greater than candModeList[ 1 ], the value of IntraPredMode[ xB ][ yB ] is increased by one

### Derivation process for chroma intra prediction mode

[Ed.: (WJ) this subclause may be moved to the semantics of intra\_chroma\_pred\_mode syntax]

Inputs to this process are:

– a luma location ( xB, yB ) specifying the top-left luma sample of the current block relative to the top‑left luma sample of the current picture,

– a variable log2TrafoSize specifying the size of the current coding unit.

Output of this process is the variable IntraPredModeC.

Depending on PartMode, the chroma intra prediction mode IntraPredModeC is derived as follows:

– If PartMode is equal to PART\_2Nx2N or PART\_NxN, the chroma intra prediction mode IntraPredModeC is derived as specifed in Table 8‑3 or Table 8‑4 with intra\_chroma\_pred\_mode, IntraPredMode[ xB ][ yB ] and chroma\_pred\_from\_luma\_enabled\_flag as inputs.

Table 8‑3 – Specification of IntraPredModeC according to the values of intra\_chroma\_pred\_mode and IntraPredMode[ xB ][ yB ] when chroma\_pred\_from\_luma\_enabled\_flag is equal to 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **intra\_chroma\_pred\_mode** | **IntraPredMode[ xB ][ yB ]** | | | | |
| **0** | **1** | **2** | **3** | **X ( 0 <= X < 35 )** |
| 0 | 7 | 0 | 0 | 0 | 0 |
| 1 | 1 | 7 | 0 | 0 | 1 |
| 2 | 2 | 2 | 7 | 1 | 2 |
| 3 | 3 | 3 | 3 | 7 | 3 |
| 4 | LM | LM | LM | LM | LM |
| 5 | 0 | 1 | 2 | 3 | X |

Table 8‑4 – Specification of IntraPredModeC according to the values of intra\_chroma\_pred\_mode and IntraPredMode[ xB ][ yB ] when chroma\_pred\_from\_luma\_enabled\_flag is equal to 0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **intra\_chroma\_pred\_mode** | **IntraPredMode[ xB ][ yB ]** | | | | |
| **0** | **1** | **2** | **3** | **X ( 0 <= X < 35 )** |
| 0 | 7 | 0 | 0 | 0 | 0 |
| 1 | 1 | 7 | 1 | 1 | 1 |
| 2 | 2 | 2 | 7 | 2 | 2 |
| 3 | 3 | 3 | 3 | 7 | 3 |
| 4 | 0 | 1 | 2 | 3 | X |

– Otherwise (PartMode is equal to PART\_2NxN or PART\_Nx2N), IntraPredModeC is derived by following procedures with intra\_chroma\_pred\_mode and the two intra prediction modes of the current coding unit as inputs.

1. The variables xOff, yOff are set as follows:

if(PartMode == PART\_2NxN){

xOff = 0;

yOff = (1<< log2TrafoSize)>>2; }

else if(PartMode == PART\_ Nx2N){

xOff = (1<< log2TrafoSize)>>2;

yOff = 0;}

1. mapIntraPredModeC[0] is derived as follows:

j = 0;

mapIntraPredModeC[j++] = 35;

1. The table IntraPredModeCnt with a size of 34 recording the two intra prediction modes of the current coding unit is given by

IntraPredModeCnt[34] = {0};

for(blkIdx = 0; blkIdx < 4; blkIdx+=2)

IntraPredModeCnt[ IntraPredMode[ xB+xOff\*blkIdx][yB+yOff\*blkIdx] ]++;

1. mapping table mapIntraPredModeC with a size of 6 is derived as follows:

for( n = 0; n < 34; n++ )

if( IntraPredModeCnt[n] >1){

mapIntraPredModeC[j++] = n;}

If(log2TrafoSize > 3)

mapIntraPredModeC[j++] = 36;

for( n = 0; n < 4; n++ )

if(IntraPredModeCnt[n] < 2 && j < 6)

mapIntraPredModeC[j++] = n;

if(j < 6)

mapIntraPredModeC[j] = 7;

1. IntraPredModeC is then derived by

* IntraPredModeC = mapIntraPredModeC[ intra\_chroma\_pred\_mode ].

### Decoding process for intra blocks

Inputs to this process are:

– a sample location ( xB, yB ) specifying the top-left sample of the current block relative to the top‑left sample of the current picture,

– two variables log2TrafoWidth and log2TrafoHeight specifying the width and height of the current block,

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

– a variable PartMode specifying prediction partition mode of the current coding unit.

– a variable intraPredMode specifying the intra prediction mode.

– a variable cIdx specifying the chroma component of the current block,

Output of this process is:

– a modified reconstructed picture before deblocking filtering.

Depending split\_transform\_flag[ xB ][ yB ][ trafoDepth ], log2TrafoWidth, log2TrafoHeight and PartMode, the following applies:

– If split\_transform\_flag[ xB ][ yB ][ trafoDepth ] is equal to 1, and PartMode is equal to PART\_2Nx2N or PART\_ NxN,the following ordered steps apply:

1. The variable xB1 is set equal to xB + ( ( 1 << log2TrafoSize ) >> 1 ).
2. The variable yB1 is set equal to yB + ( ( 1 << log2TrafoSize ) >> 1 ).
3. The decoding process for intra blocks as specified in this subclause is invoked with the location ( xB, yB ), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the intra prediction mode intraPredMode, and the variable cIdx as the inputs and the output is a modified reconstructed picture before deblocking filtering.
4. The decoding process for intra blocks as specified in this subclause is invoked with the location ( xB1, yB ), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the intra prediction mode intraPredMode, and the variable cIdx as the inputs and the output is a modified reconstructed picture before deblocking filtering.
5. The decoding process for intra blocks as specified in this subclause is invoked with the location ( xB, yB1 ), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the intra prediction mode intraPredMode, and the variable cIdx as the inputs and the output is a modified reconstructed picture before deblocking filtering.
6. The decoding process for intra blocks as specified in this subclause is invoked with the location ( xB1, yB1 ), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the intra prediction mode intraPredMode, and the variable cIdx as the inputs and the output is a modified reconstructed picture before deblocking filtering.

– Otherwise, if split\_transform\_flag[ xB ][ yB ][ trafoDepth ] is equal to 1, and log2TrafoWidth is not equal to log2TrafoHeight, the following ordered steps apply:

1. If log2TrafoWidth is greater than log2TrafoHeight, the variables xB1, xB2, xB3 are set equal to xB. Otherwise, the variable xB1 is set equal to xB + ( ( 1 << log2TrafoWidth ) >> 2 ), the variable xB2 is set equal to xB1 + ( ( 1 << log2TrafoWidth ) >> 2 ), the variable xB3 is set equal to xB2 + ( ( 1 << log2TrafoWidth ) >> 2 ).
2. If log2TrafoWidth is greater than log2TrafoHeight, the variable yB1 is set equal to yB + ( ( 1 << log2TrafoHeight ) >> 2 ), the variable yB2 is set equal to yB1 + ( ( 1 << log2TrafoHeight ) >> 2 ), the variable yB3 is set equal to yB2 + ( ( 1 << log2TrafoHeight ) >> 2 ). Otherwise, the variables yB1, yB2, yB3 are set equal to yB.
3. If log2TrafoWidth is greater than log2TrafoHeight, the variable log2TrafoHeight is set equal to log2TrafoHeight - 2. Otherwise, the variable log2TrafoWidth is set equal to log2TrafoWidth - 2.
4. The decoding process for intra blocks as specified in this subclause is invoked with the location ( xB, yB ), the variables log2TrafoWidth and log2TrafoHeight, the variable trafoDepth set equal to trafoDepth + 1, the intra prediction mode intraPredMode, and the variable cIdx as the inputs and the output is a modified reconstructed picture before deblocking filtering.
5. The decoding process for intra blocks as specified in this subclause is invoked with the location ( xB1, yB1 ), the variables log2TrafoWidth and log2TrafoHeight, the variable trafoDepth set equal to trafoDepth + 1, the intra prediction mode intraPredMode, and the variable cIdx as the inputs and the output is a modified reconstructed picture before deblocking filtering.
6. The decoding process for intra blocks as specified in this subclause is invoked with the location ( xB2, yB2 ), the variables log2TrafoWidth and log2TrafoHeight, the variable trafoDepth set equal to trafoDepth + 1, the intra prediction mode intraPredMode, and the variable cIdx as the inputs and the output is a modified reconstructed picture before deblocking filtering.
7. The decoding process for intra blocks as specified in this subclause is invoked with the location ( xB3, yB3 ), the variables log2TrafoWidth and log2TrafoHeight, the variable trafoDepth set equal to trafoDepth + 1, the intra prediction mode intraPredMode, and the variable cIdx as the inputs and the output is a modified reconstructed picture before deblocking filtering.

* Otherwise (split\_transform\_flag[ xB ][ yB ][ trafoDepth ] is equal to 0), the following ordered steps apply:

1. The variable nW is set equal to 1 << log2TrafoWidth, the variable nH is set equal to 1 << log2TrafoHeight.
2. The intra sample prediction process as specified in subclause 8.3.3.1 is invoked with the location ( xB, yB ), the intra prediction mode intraPredMode, the prediction width nW, the prediction height nH and the variable cIdx as the inputs and the output is a (nW)x(nH) array predSamples.
3. The scaling and transformation process as specified in subclause 8.5.1 is invoked with the location ( xB, yB ), the variable trafoDepth, the variable cIdx, and the transform width trafoWidth set equal to nW, and the transform height trafoHeight set equal to nH as the inputs and the output is a (nW)x(nH) array resSamples.
4. The residual signal accumulation process as specified in subclause XXX is invoked with the variable arrayWidth set equal to nW, the variable arrayHeight set equal to nH, the (nW)x(nH) array predSamples, and the (nW)x(nH) array resSamples as the inputs and the output is a (nW)x(nH) array recSamples.
5. The picture reconstruction process for a component before deblocking filtering as specified in subclause XXX is invoked with the location ( xB, yB ), the variable arrayWidth set equal to nW, the variable arrayHeight set equal to nH, the variable cIdx set equal to 0, and the (nW)x(nH) array recSamples as the inputs and the output is a modified reconstructed picture before deblocking filtering.

#### Intra sample prediction

Inputs to this process are:

– a sample location ( xB, yB ) specifying the top-left sample of the current block relative to the top‑left sample of the current picture,

– a variable intraPredMode specifying the intra prediction mode,

– two variables nW and nH specifying the prediction width and height.

– a variable cIdx specifying the chroma component of the current block,

Output of this process is:

– the predicted samples predSamples[ x, y ], with x, y =0..nW-1 and y = 0...nH-1.

The variable nS is set as (nW+nH)>>1, the nS\*4+1 neighbouring samples p[ x, y ] that are constructed samples prior to the deblocking filter process, with x = -1, y = -1..nS\*2-1 and x = 0..nS\*2-1, y=-1, are derived as follows.

– The luma location (xBN, yBN ) is specified by

xBN = xB + x  (8‑14)

yBN = yB +y  (8‑15)

– Each sample p[ x, y ] with x = -1, y= -1..nS\*2-1  and x = 0..nS\*2-1, y = -1 is derived as follows

* If any of the following condition is true, the sample p[ x, y ] is marked as “not available for intra prediction”
  1. the coding unit covering ( xBN, yBN ) is not available
  2. the coding unit covering ( xBN, yBN ) is not coded as intra mode and constrained\_intra\_pred\_flag is equal to 1
* Otherwise, the sample p[ x, y ] is marked as “available for intra prediction” and the sample at the location ( xBN, yBN ) inside the treeblock tbAddrN is assigned to p[ x, y ].

When chroma\_pred\_from\_luma\_enabled\_flag is equal to 1 and cIdx is equal to 0, the nS\*4+1 neighbouring samples PLM[ x, y ] that are constructed luma samples for Intra\_FromLuma prediction mode, with x = -1, y = -1..nS\*2-1 and x = 0..nS\*2-1, y=-1, are derived as following ordered steps:

1. If the sample p[ x, y ] is marked as “not available for intra prediction”, the sample PLM[ x, y ] is marked as “not available for intra prediction”, otherwise, the sample PLM[ x, y ] is marked as “available for intra prediction”, with x = -1, y = -1..nS\*2-1 and x = 0..nS\*2-1, y=-1.
2. For x = -1, y= 0..nS\*2-1 , if the sample PLM[ x, y ] is marked as “available for intra prediction”, the sample location at the location ( xB + x – 1, yB + y ) inside the treeblock tbAddrN is assigned to PLM[ x, y ].
3. For x = -1..nS\*2-1, y=-1 , if the sample PLM[ x, y ] is marked as “available for intra prediction”, the sample location at the location ( xB + x , yB + y ) inside the treeblock tbAddrN is assigned to PLM[ x, y ].

When at least one sample p[ x, y ] with x = -1, y = -1..nS\*2 1 and x = 0..nS\*2 1, y = -1 is marked as “not available for intra prediction,” the reference sample substitution process for intra sample prediction in subclause 8.3.3.1.1 is invoked with the samples p[ x, y ] with x = -1, y = -1..nS\*2 1 and x = 0..nS\*2 1, y = -1 as input and the modified samples p[ x, y ] with x = -1, y = -1..nS\*2 1 and x = 0..nS\*2 1, y = -1 as output.

When chroma\_pred\_from\_luma\_enabled\_flag is equal to 1, cIdx is equal to 0, and at least one sample PLM[ x, y ] with x = -1, y = -1..nS\*2 1 and x = 0..nS\*2 1, y = -1 is marked as “not available for intra prediction,” the reference sample substitution process for intra sample prediction in subclause 8.3.3.1.1 is invoked with the samples PLM[ x, y ] with x = -1, y = -1..nS\*2 1 and x = 0..nS\*2 1, y = -1 as input and the modified samples PLM[ x, y ] with x = -1, y = -1..nS\*2 1 and x = 0..nS\*2 1, y = -1 as output.

Depending on intraPredMode, the following ordered steps apply:

1. When cIdx is equal to 0, filtering process of neighbouring samples specified in 8.3.3.1.2 is invoked with the sample array p and the prediction width nW and height nH as the inputs and the output is reassigned to the sample array p.
2. Intra sample prediction process according to intraPredMode applies as follows:
   1. One of the intra prediction modes specified in subclause 8.3.3.1.3 to 8.3.3.1.8 is invoked with the sample location ( xB, yB ), the sample array p, the prediction width nW and height nH and the chroma component index cIdx as the inputs and the output is the predicted sample array predSamples according to intraPredMode.

[Ed. (WJ): some functions do not use some input parameters. Above sentence should be improved]

##### Reference sample substitution process for intra sample prediction

Inputs to this process are the reference samples p[ x, y ] with x = ‑1, y = ‑1..nS\*2‑1 and x = 0..nS\*2‑1, y = ‑1 for intra sample prediction.

Outputs of this process are the modified reference samples p[ x, y ] with x = ‑1, y = ‑1..nS\*2‑1 and x = 0..nS\*2‑1, y = ‑1 for intra sample prediction.

The values of the samples p[ x, y ] with x = ‑1, y = ‑1..nS\*2‑1 and x = 0..nS\*2‑1, y = ‑1 are modified as follows:

* If all samples p[ x, y ] with x = ‑1, y = ‑1..nS\*2‑1 and x = 0..nS\*2‑1, y = ‑1 are marked as “not available for intra prediction,” the value ( 1 << ( BitDepthY ‑ 1 ) ) is substituted for the values of all samples p[ x, y ].
* Otherwise (at least one but not all samples p[ x, y ] are marked as “not available for intra prediction”), the following ordered steps are performed:

1. If p[ -1, nS\*2-1 ] is marked as “not available for intra prediction”, searching sequentially starting from x = -1, y = nS\*2-1 to x = -1, y = -1, then from x = 0,y = -1 to x = nS\*2-1 ,y = -1. As soon as a sample p[ x, y ] marked as “available for intra prediction” is found, the search is terminated and the value of p[ x, y ] is assigned to p[ -1, nS\*2-1 ].
2. For x = -1, y = nS\*2-2...-1, if p[ x, y ] is marked as “not available for intra prediction”, the value of p[ x, y+1 ] is substituted for the value of p[ x, y].
3. For x = 0..nS\*2-1, y = -1, if p[ x, y ] is marked as “not available for intra prediction”, the value of p[ x-1, y ] is substituted for the value of p[ x, y ].

All samples p[ x, y ] with x = ‑1, y = ‑1..nS\*2‑1 and x = 0..nS\*2‑1, y = ‑1 are marked as “available for intra prediction.”

##### Filtering process of neighbouring samples

Inputs to this process are:

– neighbouring samples p[ x, y ], with x, y = -1..2\*nS-1,

– two variables nW and nH specifying the prediction width and height.

When prediction width nW is not equal to prediction height nH, and nW <= 4, intraFilterType[nW][IntraPredMode] = 0; Otherwise, intraFilterType[nW][IntraPredMode] is as defined in table 8-5.

Output of this process is:

– filtered samples pF[ x, y ],. with x, y = -1..2\*nS-1.

Table 8‑5 – Specification of intraFilterType[ nS ][ IntraPredMode ] for various prediction unit sizes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **IntraPredMode** | **intraFilterType**  **for nS = 4** | **intraFilterType**  **for nS = 8** | **intraFilterType**  **for nS = 16** | **intraFilterType**  **for nS = 32** | **intraFilterType**  **for nS = 64** |
| 0 | 0 | 1 | 1 | 1 | 0 |
| 1-2 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 1 | 1 | 1 | 0 |
| 4, 5 | 0 | 0 | 1 | 1 | 0 |
| 6 | 0 | 1 | 1 | 1 | 0 |
| 7, 8 | 0 | 0 | 1 | 1 | 0 |
| 9 | 0 | 1 | 1 | 1 | 0 |
| 10-20 | 0 | 0 | 1 | 1 | 0 |
| 21, 22 | 0 | 0 | 0 | 1 | 0 |
| 23-28 | 0 | 0 | 1 | 1 | 0 |
| 29, 30 | 0 | 0 | 0 | 1 | 0 |
| 31-33 | 0 | 0 | 1 | 1 | 0 |
| 34 | n/a | n/a | n/a | n/a | n/a |

Filtered sample array pF[ x, y ] with x = -1..nS\*2-1 and y = -1..nS\*2-1  are derived as follows:

– When intraFilterType[ nS ][ IntraPredMode ] is equal to 1, the following applies:

pF[ -1, nS\*2-1 ] = p[ -1, nS\*2-1 ] (8‑17)

pF[ nS\*2-1, -1 ] = p[ nS\*2-1, -1 ] (8‑18)

pF[ -1, y ] = ( p[ -1, y+1 ] + 2\*p[ -1, y ] + p[ -1, y-1 ] + 2 ) >> 2 for y = nS\*2-2..0 (8‑19)

pF[ -1, -1] = ( p[ -1, 0 ] + 2\*p[ -1, -1] + p[ 0, -1 ] + 2) >> 2 (8‑20)

pF[ x, -1 ] = ( p[ x-1, -1 ] + 2\*p[ x, -1 ] + p[ x+1, -1 ] + 2 ) >> 2 for x = 0..nS\*2-2 (8‑21)

##### Specification of Intra\_Vertical prediction mode

Inputs to this process are:

– neighbouring samples p[ x, y ], with x, y = -1..2\*nS-1,

– two variables nW and nH variable nS specifying the prediction width and height

– a variable cIdx specifying the chroma component of the current block

Output of this process is:

– predicted samples predSamples[ x, y ], with x =0..nW-1 and y = 0..nH-1.

This intra prediction mode is invoked when intraPredMode is equal to 0.

The values of the prediction samples predSamples[ x, y ], with x =0..nW-1 and y = 0..nH-1, are derived as follows:

– If cIdx is equal to 0,

predSamples[ x, y ] = p[ x, -1 ], with x =0..nW-1 and y = 0..nH-1 (8‑30)

predSamples[ x, y ] = Clip1Y( p[ x, -1 ] + ( ( d[ y ] + ( d[ y ] < 0 ? 1 : 0 ) )>> 1 ) ), with x = 0, y = 0..nH-1

where d[ y ] = p[ -1, y ] – p[ -1, -1 ] (8‑30)

– Otherwise,

predSamples[ x, y ] = p[ x, -1 ], with x =0..nW-1 and y = 0..nH-1 (8‑30)

##### Specification of Intra\_Horizontal prediction mode

Inputs to this process are:

– neighbouring samples p[ x, y ], with x, y = -1..2\*nS-1,

– two variables nW and nH specifying the prediction block width and height.

– a variable cIdx specifying the chroma component of the current block

Output of this process is:

– predicted samples predSamples[ x, y ], with x =0..nW-1 and y = 0..nH-1.

This intra prediction mode is invoked when intraPredMode is equal to 1.

The values of the prediction samples predSamples[ x, y ], with x =0..nW-1 and y = 0..nH-1, are derived as follows:

– If cIdx is equal to 0,

predSamples[ x, y ] = p[ -1, y ], with x =0..nW-1 and y = 0..nH-1 (8‑31)

predSamples[ x, y ] = Clip1Y( p[ -1, y ] + ( ( d[ x ] + ( d[ x ] < 0 ? 1 : 0 ) ) >> 1 ), with x =0..nW-1, y = 0

where d[ x ] = p[ x, -1 ] – p[ -1, -1 ] (8‑31)

– Otherwise,

predSamples[ x, y ] = p[ -1, y ], with x =0..nW-1 and y = 0..nH-1 (8‑31)

##### Specification of Intra\_DC prediction mode

Inputs to this process are:

– neighbouring samples p[ x, y ], with x, y = -1..2\*nS-1,

– two variables nW and nH specifying the prediction block width and height.

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

Output of this process is:

– predicted samples predSamples[ x, y ], with x =0..nW-1 and y = 0..nH-1.

This intra prediction mode is invoked when intraPredMode is equal to 2.

The values of the prediction samples predSamples[ x, y ], with x =0..nW-1 and y = 0..nH-1, are derived as the following ordered steps:

1. A variable DCVal is derived as:

If nW is equal to nH,

DCVal = , with x, y = 0..nS-1 (8‑32)  
where k=log2(nW)

Otherwise, if nW is greater than nH

DCVal = , with x = 0.. nW-1 (8‑32)  
where k=log2(nW)

Otherwise, if nW is less than nH

DCVal = , with y = 0.. nH-1 (8‑32)  
where k=log2(nH)

1. The variable nS is set equal to Max(nW,nH), depending on the chroma component index cIdx, the following applies.

* If cIdx is equal to 0, the following applies.

predSamples[ 0, 0 ] = ( 1\*p[ -1, 0 ] + 2\*DCVal + 1\*p[ 0, -1 ] + 2 ) >> 2 (8‑32)  
predSamples[ x, 0 ] = ( 1\*p[ x, -1 ] + 3\*DCVal + 2 ) >> 2, with x = 1..nW-1 (8‑32)  
predSamples[ 0, y ] = ( 1\*p[ -1, y ] + 3\*DCVal + 2 ) >> 2, with y = 1..nH-1 (8‑32)  
predSamples[ x, y ] = DCVal, with x = 1.. nW-1, y = 1.. nH-1 (8‑32)

* Otherwise, the prediction samples predSamples[ x, y ] are derived as

predSamples[ x, y ] = DCVal, with x = 0.. nW-1, y = 0.. nH-1 (8‑32)

##### Specification of Intra\_Angular prediction mode

Inputs to this process are:

– neighbouring samples p[ x, y ], with x, y = -1..2\*nS-1,

– two variables nW and nH variable nS specifying the prediction width and height.

Output of this process is:

– predicted samples predSamples[ x, y ], with x =0..nW-1 and y = 0..nH-1.

This intra prediction mode is invoked when intraPredMode is in the range of 3..33.

Table 8‑6 specifies the mapping table between intraPredMode and the rearranged intra prediction order intraPredOrder.

Table 8‑6 – Specification of intraPredOrder

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **intraPredMode** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** |
| **intraPredOrder** | - | - | - | - | 1 | 5 | 13 | 17 | 21 | 29 | 33 | 3 | 7 | 11 | 15 | 19 | 23 | 27 |
| **intraPredMode** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** | **32** | **33** | **34** |  |
| **intraPredOrder** | 31 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 | 22 | 24 | 26 | 28 | 30 | 32 |  |

Figure 8‑2 illustrates the total 34 intra angles and Table 8‑7 specifies the mapping table between intraPredOrder and the angle parameter intraPredAngle.



Figure 8‑2 – Intra prediction angle definition (informative)

Table 8‑7 – Specification of intraPredAngle

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **intraPredOrder** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** |
| **intraPredAngle** | - | -32 | -26 | -21 | -17 | -13 | -9 | -5 | -2 | - | 2 | 5 | 9 | 13 | 17 | 21 | 26 |
| **intraPredOrder** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** | **32** | **33** |
| **intraPredAngle** | 32 | -26 | -21 | -17 | -13 | -9 | -5 | -2 | - | 2 | 5 | 9 | 13 | 17 | 21 | 26 | 32 |

Table 8‑8 further specifies the mapping table between intraPredOrder and the inverse angle parameter invAngle.

Table 8‑8 – Specification of invAngle

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **intraPredOrder** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| **invAngle** | -256 | -315 | -390 | -482 | -630 | -910 | -1638 | -4096 |
| **intraPredOrder** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** |
| **invAngle** | -315 | -390 | -482 | -630 | -910 | -1638 | -4096 | - |

The reference pixel array refMain[ x ], with x=- Max(nW,nH)..2\*nS is specified as follows.

– If intraPredOrder is less than 18,

refMain[ x ] = p[ -1+x, -1 ], with x=0..nW (8‑36)

* If intraPredAngle is less than 0,

refMain[ x ] = p[ -1, -1+( ( x\*invAngle+128 )>>8 ) ], with x=( nH\*intraPredAngle ) >>5..-1 (8‑37)

* Otherwise,

refMain[ x ] = p[ -1+x, -1 ], with x=nW+1..2\*nS (8‑38)

Otherwise,

refMain[ x ] = p[ -1, -1+x ], with x=0..nH (8‑39)

* If intraPredAngle is less than 0,

refMain[ x ] = p[ -1+( ( x\*invAngle+128 )>>8 ), -1 ], with x=( nW\*intraPredAngle ) >>5..-1 (8‑40)

* Otherwise,

refMain[ x ] = p[ -1, -1+x ], with x=nH+1..2\*nS (8‑41)

The values of the prediction samples predSamples[ x, y ], with x =0..nW-1 and y = 0..nH-1 are derived by the following procedures.

– The index variable iIdx and the multiplication factor iFact are derived by

iIdx = ( ( y + 1 )\*intraPredAngle ) >> 5 (8‑42)

iFact = ( ( y + 1 )\*intraPredAngle ) && 31 (8‑43)

– Depending on the value of iFact, the following applies.

* If iFact is not equal to 0, the value of the prediction samples predSamples[ x, y ] is derived by

predSamples[ x, y ] = ( ( 32 – iFact )\*refMain[ x+iIdx+1 ] + iFact\*refMain[ x+iIdx+2] + 16 ) >> 5 (8‑44)

* Otherwise, the value of the prediction samples predSamples[ x, y ] is derived by

predSamples[ x, y ] = refMain[ x+iIdx+1 ] (8‑45)

##### Specification of Intra\_Planar prediction mode

Inputs to this process are:

– neighbouring samples p[ x, y ], with x, y = -1..2\*nS-1,

– two variables nW and nH specifying the prediction width and height,

Output of this process is:

– predicted samples predSamples[ x, y ], with x = 0.. nW-1, y = 0.. nH-1

This intra prediction mode is invoked when intraPredMode is equal to 34.

The values of the prediction samples predSamples[ x, y ], with x = 0.. nW-1, y = 0.. nH-1 are derived by

predSamples[ x, y ] = (  
 (( ( nW – 1 – x ) \* p[ -1, y ] + ( x + 1 ) \* p[ nW, -1 ] + nW/2 ) >> m ) +   
 (( ( nH – 1 – y ) \* p[ x ,-1 ] + ( y + 1 ) \* p[ -1, nH ] + nH/ 2 ) >> n )+1 ) >> 1 (8‑45)  
with x = 0.. nW-1, y = 0.. nH-1 where m = log2( nW ), n = log2( nH )

##### Specification of Intra\_FromLuma prediction mode

### Transformation process for scaled transform coefficients

Inputs of this process are:

– a variable nW specifying the width of the current transform unit,

– a variable nH specifying the height of the current transform unit,

– a (nW)x(nH) array d of scaled transform coefficients with elements dij.

– a variable cIdx specifying the chroma component of the current block,

Output of this process is residual samples as a (nW)x(nH) array r with elements rij.

Depending on PredMode and IntraPredMode, the following applies:

– If PredMode is equal to MODE\_INTRA and cIdx is equal to 0, if nW is equal to 4, the variables horizTrType is specified as with IntraPredMode as input and if nH is equal to 4, the variables vertTrType is specified as with IntraPredMode as input. [Ed. (WJ): DST is applied only for luma 4x4 block]

– Otherwise, the variables horizTrType and vertTrType are set equal to 0.

Table 8‑13 – Specification of horizTrType and vertTrType

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **IntraPredMode** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| **horizTrType** | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| **vertTrType** | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **IntraPredMode** | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |
| **horizTrType** | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| **vertTrType** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |

The constructed residual samples are derived as specified in the following ordered steps.

(8‑427)

## In-loop filter process

[Ed: (WJ) overall process description need to be inserted]

### Deblocking filter process

A conditional filtering process shall be performed on a treeblock basis after the completion of the picture construction process prior to deblocking filter process for the entire decoded picture (as specified in subclauses XXX and YYY) [Ed.: (WJ) those subclauses seem not defined yet], with all treeblocks in a picture processed in order of increasing treeblock addresses.

Each treeblock is processed on a coding unit basis with the same order as decoding process. For each coding unit, vertical edges are filtered first, starting with the edge on the left-hand side of the coding unit proceeding through the edges towards the right-hand side of the coding unit in their geometrical order. Then, the horizontal edges are filtered starting with the edge on the top of the coding unit proceeding through the edges towards the bottom of the coding unit in their geometrical order.

Sample values above of the current coding unit that may have already been modified by the filtering of horizontal edges of deblocking filter process operation on previous coding unit shall be used as inputs to the deblocking filter process on the current coding unit and may be further modified during the filtering of the current coding unit. Sample values to the left of the current coding unit shall be used as inputs to the deblocking filter process on the current coding unit and may be further modified during the filtering of the current coding unit. Sample values to the left of the current coding unit may be modified by the filtering of vertical edge and may be further modified by the filtering of horizontal edges.

Sample values modified during filtering of vertical edges are used as input for the filtering of the horizontal edges. For sample values modified by both filtering of horizontal edges and filtering of vertical edges, filtering of horizontal edges is applied after filtering of vertical edges.

The deblocking filter process shall be applied to all prediction unit edges and transform unit edges of a picture, except edges at the boundary of the picture, any edges for which the deblocking filter process is disabled by disable\_deblocking\_filter\_flag, any edges coinside with tile boundaries when loop\_filter\_across\_tile\_flag is equal to 0, and any edges coinside with slice boundaries when loop\_filter\_across\_slice\_flag is equal to 0. For the transform units and prediction units with edges smaller than 8 samples in either vertical or horizontal direction, only the edges lying on the 8x8 sample grid are filtered.

When disable\_deblocking\_filter\_flag is not equal to 1, the deblocking filter process is invoked as the following ordered steps for each coding unit with the same order as decoding process.

1. The coding unit size nS is set equal to 1 << log2CUSize.
2. The variables FilterInternalEdgesFlag, FilterLeftCuEdgeFlag and FilterTopCuEdgeFlag are derived as follows.

* The variable FilterInternalEdges is set equal to 1.
* If the left boundary of current coding unit is the left boundary of the picture, or if the left boundary of current coding unit is the left boundary of the tile and loop\_filter\_across\_tile\_flag is equal to 0, or if the left boundary of current coding unit is the left boundary of the slice and loop\_filter\_across\_slice\_flag is equal to 0, the variable FilterLeftCuEdgeFlag is set equal to 0, otherwise set equal to 1.
* If the top boundary of current coding unit is the top boundary of the picture, or if the top boundary of current coding unit is the top boundary of the tile and loop\_filter\_across\_tile\_flag is equal to 0, or if the top boundary of current coding unit is the top boundary of the slice and loop\_filter\_across\_slice\_flag is equal to 0, the variable FilterTopCuEdgeFlag is set equal to 0, otherwise set equal to 1.

1. All elements of two-dimensional array of size (nS)x(nS), horEdgeFlags and verEdgeFlags are initialized to zero.
2. The derivation process of transform unit boundary specified in subclause are invoked with the luma location ( xB, yB ) set equal to ( 0, 0 ), the transform unit width log2TrafoWidth set equal to log2CUSize, the transform unit height log2TrafoHeight set equal to log2CUSize, the prediction partition mode PartMode and the variable trafoDepth set equal to 0 as the inputs and the modified horEdgeFlags and verEdgeFlags as outputs.
3. The derivation process of prediction unit boundary specified in subclause are invoked with the coding unit size log2CUSize and the prediction partition mode PartMode as inputs, and the modified horEdgeFlags and verEdgeFlags as outputs.
4. The derivation process of the boundary filtering strength specified in subclause 8.6.1.3 is invoked with the luma location ( xC, yC ), the coding unit size log2CUSize, horEdgeFlags and verEdgeFlags as inputs and an array of size (2)x(nS)x(nS), bS as output.
5. The filtering process for coding unit specified in subclause 8.6.1.4 are invoked with the luma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture, the coding unit size log2CUSize and the array bS as inputs and the modified reconstructued picture as output.

#### Derivation process of transform unit boundary

Inputs of this process are:

– a luma location ( xB, yB ) specifying the top-left luma sample of the current block relative to the top‑left luma sample of the current coding unit,

– a variable log2TrafoWidth specifying the width of the current block,

– a variable log2TrafoHeight specifying the height of the current block,

– a variable trafoDepth.

Outputs of this process are:

– two-dimensional arrays of (nS)x(nS), horEdgeFlags and verEdgeFlags.

Depending on split\_transform\_flag[ xB ][ yB ][ trafoDepth ] and PartMode, the following applies:

– If split\_transform\_flag[ xB ][ yB ][ trafoDepth ] is equal to 1, the following ordered steps apply:

1. The variables xB1, yB1, xB2, yB2, xB3 and yB3 are derived as follows.
   * If TUSplitDirection is equal to 2, the following applies.
   * The variable xB1 is set equal to xB + ( ( 1 << log2TrafoWidth ) >> 1 ).
   * The variable yB1 is set equal to yB.
   * The variable xB2 is set equal to xB.
   * The variable yB1 is set equal to yB + ( ( 1 << log2TrafoHeight ) >> 1 ).
   * The variable xB3 is set equal to xB1.
   * The variable yB3 is set equal to yB2.
   * The variable log2TrafoWidth1 is set equal to log2TrafoWidth − 1.
   * The variable log2TrafoHeight1 is set equal to log2TrafoHeight − 1.
   * Otherwise (TUSplitDirection is equal to 0 or 1), the following applies.
   * The variable xB1 is set equal to xB + ((1 << (log2TrafoWidth)) >> 2) \* TUSplitDirection.
   * The variable yB1 is set equal to yB + ((1 << (log2TrafoHeight)) >> 2) \* (1 − TUSplitDirection).
   * The variable xB2 is set equal to xB1 + ((1 << (log2TrafoWidth)) >> 2) \* TUSplitDirection.
   * The variable yB2 is set equal to yB1 + ((1 << (log2TrafoHeight)) >> 2) \* (1 − TUSplitDirection).
   * The variable xB3 is set equal to xB2 + ((1 << (log2TrafoWidth)) >> 2) \* TUSplitDirection.
   * The variable yB3 is set equal to yB2 + ((1 << (log2TrafoHeight)) >> 2) \* (1 − TUSplitDirection).
   * The variable log2TrafoWidth1 is set equal to (log2TrafoWidth − 2) \* TUSplitDirection.
   * The variable log2TrafoHeight1 is set equal to (log2TrafoHeight − 2) \* (1 − TUSplitDirection).
2. The deriviation process of transform unit boundary as specified in this subclause is invoked with the luma location ( xB, yB ), the variable log2TrafoWidth set equal to log2TrafoWidth1, the variable log2TrafoHeight set equal to log2TrafoHeight1 and the variable trafoDepth1 set equal to trafoDepth + 1 as inputs and the outputs are the modified versions of two arrays, horEdgeFlags and verEdgeFlags.
3. The deriviation process of transform unit boundary as specified in this subclause is invoked with the luma location ( xB1, yB1 ), the variable log2TrafoWidth set equal to log2TrafoSizeWidth1, the variable log2TrafoHeight set equal to log2TrafoHeight1 and the variable trafoDepth1 set equal to trafoDepth + 1 as inputs and the outputs are the modified versions of two arrays, horEdgeFlags and verEdgeFlags.
4. The deriviation process of transform unit boundary as specified in this subclause is invoked with the luma location ( xB2, yB2 ), the variable log2TrafoWidth set equal to log2TrafoSizeWidth1, the variable log2TrafoHeight set equal to log2TrafoHeight1 and the variable trafoDepth1 set equal to trafoDepth + 1 as inputs and the outputs are the modified versions of two arrays, horEdgeFlags and verEdgeFlags.
5. The deriviation process of transform unit boundary as specified in this subclause is invoked with the luma location ( xB3, yB3 ), the variable log2TrafoWidth1 set equal to log2TrafoSizeWidth1, the variable log2TrafoHeight set equal to log2TrafoHeight1 and the variable trafoDepth1 set equal to trafoDepth + 1 as inputs and the outputs are the modified versions of two arrays, horEdgeFlags and verEdgeFlags.

– Otherwise (split\_transform\_flag[ xB ][ yB ][ trafoDepth ] is equal to 0), the following applies:

* If yB is equal to zero, horEdgeFlags[ xB + k ][ yB ] is set equal to FilterTopCuEdgeFlag, otherwise horEdgeFlags[ xB + k ][ yB ] is set equal to FilterInternalEdgesFlag for k = 0.. ( 1 << log2TrafoWidth ) – 1.
* If xB is equal to zero, verEdgeFlags[ xB ][ yB + k ] is set equal to FilterLeftCuEdgeFlag, otherwise verEdgeFlags[ xB ][ yB + k ] is set equal to FilterInternalEdgesFlag for k = 0.. ( 1 << log2TrafoHeight ) – 1.

#### Derivation process of prediction unit boundary

Inputs of this process are:

– a variable log2CUSize specifying the coding unit size,

– a prediction partition mode PartMode.

Outputs of this process are:

– two-dimensional arrays of (nS)x(nS), horEdgeFlags and verEdgeFlags.

Depending on PartMode, the following applies:

– If PartMode is equal to PART\_2NxN or PART\_NxN, horEdgeFlags[ k ][ 1 << ( log2CUSize – 1 ) ] is set equal to FilterInternalEdgesFlag for k = 0.. ( 1 << log2CUSize ) – 1.

– If PartMode is equal to PART\_Nx2N or PART\_NxN, verEdgeFlags[ 1 << ( log2CUSize – 1 ) ][ k ] is set equal to FilterInternalEdgesFlag for k = 0.. ( 1 << log2CUSize ) – 1.

– If PartMode is equal to PART\_2NxnU, horEdgeFlags[ k ][ (1 << ( log2CUSize – 1 )) – (1 << ( log2CUSize – 2 )) ] is set equal to FilterInternalEdgesFlag for k = 0.. ( 1 << log2CUSize ) – 1.

– If PartMode is equal to PART\_2NxnD, horEdgeFlags[ k ][ (1 << ( log2CUSize – 1 )) + (1 << ( log2CUSize – 2 )) ] is set equal to FilterInternalEdgesFlag for k = 0.. ( 1 << log2CUSize ) – 1.

– If PartMode is equal to PART\_nLx2N, verEdgeFlags[ (1 << ( log2CUSize – 1 )) – (1 << ( log2CUSize – 2 )) ][ k ] is set equal to FilterInternalEdgesFlag for k = 0.. ( 1 << log2CUSize ) – 1.

– If PartMode is equal to PART\_nRx2N, verEdgeFlags[ (1 << ( log2CUSize – 1 )) + (1 << ( log2CUSize – 2 )) ][ k ] is set equal to FilterInternalEdgesFlag for k = 0.. ( 1 << log2CUSize ) – 1.

– If PartMode is equal to PART\_2NxN and predtype is intra, horEdgeFlags[ k ][ 1 << ( log2CUSize – 2 ) ], horEdgeFlags[ k ][ (1 << ( log2CUSize – 2 )) \* 2 ], horEdgeFlags[ k ][ (1 << ( log2CUSize – 2 )) \* 3 ] are set equal to FilterInternalEdgesFlag for k = 0.. ( 1 << log2CUSize ) – 1.

– If PartMode is equal to PART\_Nx2N and predtype is intra, verEdgeFlags [ 1 << ( log2CUSize – 2 ) ][ k ], verEdgeFlags[ (1 << ( log2CUSize – 2 )) \* 2 ] [ k ], verEdgeFlags[ (1 << ( log2CUSize – 2 )) \* 3 ] [ k ] are set equal to FilterInternalEdgesFlag for k = 0.. ( 1 << log2CUSize ) – 1

#### Derivation process of boundary filtering strength

#### Binarization process for part\_mode

Input to this process is a request for a binarization for the syntax element part\_mode and a variable cLog2CUSize specifying the current CU size.

Output of this process is the binarization of the syntax element.

The binarization for part\_mode is given by Table 9‑32 depending on PredMode, cLog2CUSize and

inter\_4x4\_enabled\_flag.

Table 9‑33 – Binarization for part\_mode

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **PredMode** | **Value of part\_mode** | **PartMode** | **Bin string** | | |
| cLog2CUSize >  Log2MinCUSize | cLog2CUSize = = Log2MinCUSize | |
| cLog2CUSize = = 3 &&  !inter\_4x4\_enabled\_flag | cLog2CUSize > 3 | |  inter\_4x4\_enabled\_flag |
| MODE\_INTRA | 0 | PART\_2Nx2N | 1 | 1 | 1 |
| 1 | PART\_2NxN | 01 | 001 | 001 |
| 2 | PART\_Nx2N | 00 | 000 | 000 |
| 3 | PART\_NxN | - | 01 | 01 |
| MODE\_INTER | 0 | PART\_2Nx2N | 1 | 1 | 1 |
| 1 | PART\_2NxN | 011 | 01 | 01 |
| 2 | PART\_Nx2N | 001 | 00 | 001 |
| 3 | PART\_NxN | - | - | 000 |
| 4 | PART\_2NxnU | 0100 | - | - |
| 5 | PART\_2NxnD | 0101 | - | - |
| 6 | PART\_nLx2N | 0000 | - | - |
| 7 | PART\_nRx2N | 0001 | - | - |