#### Coding unit syntax

|  |  |
| --- | --- |
| coding\_unit( x0, y0, log2CbSize ) { | Descriptor |
| … |  |
| } else if( intra\_bc\_flag[ x0 ][ y0 ] ) { |  |
| mvd\_coding( x0, y0, 2) |  |
| if( PartMode = = PART\_2NxN ) |  |
| mvd\_coding( x0, y0 + ( nCbS / 2 ), 2) |  |
| else if( PartMode = = PART\_Nx2N ) |  |
| mvd\_coding( x0 + ( nCbS / 2 ), y0, 2) |  |
| else if( PartMode = = PART\_NxN ) { |  |
| mvd\_coding( x0 + ( nCbS / 2 ), y0, 2) |  |
| mvd\_coding( x0, y0 + ( nCbS / 2 ), 2) |  |
| mvd\_coding( x0 + ( nCbS / 2 ), y0 + ( nCbS / 2 ), 2) |  |
| } |  |
| if ( PartMode = = PART\_2Nx2N ){ |  |
| **segmental\_ibc\_flag**[ x0 ][ y0 ] | ae(v) |
| if(segment\_ibc\_flag[ x0 ][ y0 ]){ |  |
| **segment\_num\_minus2**[ x0 ][ y0 ] | ae(v) |
| for( cIdx=0; cIdx<( ChromaArrayType = = 0 ? 1 : 3); cIdx++ ) |  |
| for( j = 0; j < nSegNum[ x0 ][ y0 ]; j++ ){ |  |
| **abs\_segment\_offset**[ x0 ][ y0 ][ j ][ cIdx ] | ae(v) |
| if(abs\_segment\_offset[ x0 ][ y0 ][ j ][ cIdx ]>0) |  |
| **segment\_offset\_sign\_flag**[ x0 ][ y0 ][ j ][ cIdx ]  } | ae(v) |
| } |  |
| } |  |
| }else{ |  |
| … |  |
| } |  |

#### Coding quadtree semantics

…

**segmental\_ibc\_flag**[ x0 ][ y0 ] equal to 1 specifies that the considered coding unit is coded in segmental intra block copy mode. segmental\_ibc\_flag [ x0 ][ y0 ] equal to 0 specifies that the considered coding unit is not coded in segmental intra block copying mode. When not present, the value of segmental\_ibc\_flag is inferred to be equal to 0. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

**segment\_num\_minus2**[ x0 ][ y0 ] plus 2 specifies the number of segments in the considered coding unit. When not present, the value of segment\_num\_minus2[ x0 ][ y0 ] is inferred to be equal to 0. The varaible nSegNum[ x0 ][ y0 ] is set equal to segment\_num\_minus2[ x0 ][ y0 ] + 2. 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.

**abs\_segment\_offset**[ x0 ][ y0 ][ j ][ cIdx ] specifies the absolute value of the segment offset for the jth segment in the considered coding unit. When not present, the value of abs\_segment\_offset[ x0 ][ y0 ][ j ][ cIdx ] 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. The array index cIdx specifies an indicator for the colour component; it is equal to 0 for luma, equal to 1 for Cb, and equal to 2 for Cr.

**segment\_offset\_sign**[ x0 ][ y0 ][ j ][ cIdx ] specifies the sign of the offset for the jth segment in the considered coding unit. When not present, the value of segment\_offset\_sign[ x0 ][ y0 ][ j ][ cIdx ] is inferred to be equal to 0. If segment\_offset\_sign[ x0 ][ y0 ][ j ][ cIdx ] is equal to 0, the value of the segment offset for the jth segment in the considered coding unit is positive.Otherwise (segment\_offset\_sign[ x0 ][ y0 ][ j ][ cIdx ] is equal to 1), the segment offset for the jth segment in the considered coding unit is negative. 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. The array index cIdx specifies an indicator for the colour component; it is equal to 0 for luma, equal to 1 for Cb, and equal to 2 for Cr.

The variable SegOffset[ x0 ][ y0 ][ j ][ cIdx ] for j = 0, … nSegNum[ x0 ][ y0 ], cIdx = 0,… ( ChromaArrayType = = 0 ? 1 : 3) is derived as follows:

SegOffset[ x0 ][ y0 ][ j ][ cIdx ]=abs\_segment\_offset[ x0 ][ y0 ][ j ][ cIdx ]\*(1-2\*segment\_offset\_sign[ x0 ][ y0 ][ j ][ cIdx ]).

…

**8.4.5.2.7 Specification of intra block copying prediction mode**

…

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

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

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

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

When segmental\_ibc\_flag[ xTb0 ][ yTb0 ] is equal to 1, the following applies in order:

1. A variable predAvg is derived as follows:

predAvg =,

where k = Log2( nTbS\*nTbS ).

1. A variable predMin is set equal to the minimum value for all predSamples[ x ][ y ] with x, y = 0, … nTbS-1. A variable predMax is set equal to the maximum value for all predSamples[ x ][ y ] with x, y = 0, … nTbS-1.
2. If nSegNum[ xTb0 ][ yTb0 ] is equal to 2, the following applies for all x, y = 0, … nTbS-1:
   * If predSamples[ x ][ y ] < predAvg, a variable segIdx[ x ][ y ] is set equal to 0;
   * Otherwise (predSamples[ x ][ y ] >= predAvg), segIdx[ x ][ y ] is set equal to 1.
3. Otherwise (nSegNum[ xTb0 ][ yTb0 ] is equal to 3), a variable thres0 is set equal to (predAvg + predMin + 1 ) >> 1, and a variable thres1 is set equal to (predAvg + predMax + 1 ) >> 1. Then the following applies for all x, y = 0, … nTbS-1:
   * If predSamples[ x ][ y ] < thres0, segIdx[ x ][ y ] is set equal to 0;
   * Othewise, if predSamples[ x ][ y ] > thres1, segIdx[ x ][ y ] is set equal to 2;
   * Otherwise, segIdx[ x ][ y ] is set equal to 1.
4. For j from 0 to nSegNum[ xTb0 ][ yTb0 ]-1, the following applies:
   * A variable segMin[ j ] is set equal to the minimum value for all predSamples[ x ][ y ] with x, y = 0, … nTbS-1 satisfying segIdx[ x ][ y ] is equal to j. A variable segMax[ j ] is set equal to the maximum value for all predSamples[ x ][ y ] with x, y = 0, … nTbS-1 satisfying segIdx[ x ][ y ] is equal to j.
   * A variable segPred[ j ] is set equal to (segMin[ j ] + segMax[ j ]+ 1)>>1.
   * A variable segValue[ j ] is set equal to Clip3( 0, ( 1  << BitDepthY ) − 1 ), .SegOffset[xTb0 ][ yTb0  ][ j ][ cIdx ]+ segPred[ j ]).
5. For all x, y = 0, … nTbS-1, predSamples[ x ][ y ] is set equal to segValue[segIdx[ x ][ y ]].

…

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** |
| coding\_unit( ) | … |  |  |  |  |
| intra\_bc\_flag[ ][ ] |  | 0 | 1 | 2 |
| segmental\_ibc\_flag [ ][ ] | Table 9-34’ | 0…2 | 0…2 | 0…2 |
| segment\_num\_minus2 [ ][ ] | Table 9-35’ | 0 | 0 | 0 |
| abs\_segment\_offset[ ][ ][ ][ ] | Table 9-36’ | 0…4 | 0…4 | 0…4 |
| segment\_offset\_sign[ ][ ][ ][ ] | Table 9-36’ | 0…4 | 0…4 | 0…4 |
| pred\_mode\_flag |  |  | 0 | 1 |
| … |  |  |  |  |

Table 9‑34’ – Values of initValue for ctxIdx of segmental\_ibc\_flag

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of segmental\_ibc\_flag** | | |
| **0** | **1** | **2** |
| **initValue** | 146 | 154 | 157 |

Table 9‑35’ – Values of initValue for ctxIdx of segment\_num\_minus2

|  |  |
| --- | --- |
| **Initialization variable** | **ctxIdx of** segment\_num\_minus2 |
| **0** |
| **initValue** | 154 |

Table 9‑36’ – Values of initValue for ctxIdx of abs\_segment\_offset and segment\_offset\_sign

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of** abs\_segment\_offset | | | | |
| **0** | **1** | **2** | **3** | **4** |
| **initValue** | 154 | 154 | 154 | 154 | 154 |

**Table 9‑34** – Syntax elements and associated binarizations

| **Syntax structure** | **Syntax element** | **Binarization** | |
| --- | --- | --- | --- |
| **Process** | **Input parameters** |
| … | … | … | … |
| coding\_unit( ) | … | … | … |
| cu\_skip\_flag | FL | cMax = 1 |
| intra\_bc\_flag | FL | cMax = 1 |
| segmental\_ibc\_flag | FL | cMax = 1 |
| segment\_num\_minus2 | FL | cMax = 1 |
| abs\_segment\_offset | EG0 | - |
| segment\_offset\_sign | FL | cMax = 1 |
| pred\_mode\_flag | FL | cMax = 1 |

| Table 9‑39 – Assignment of ctxInc to syntax elements with context coded bins | | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **Syntax element** | **binIdx** | | | | | |
| **0** | **1** | **2** | **3** | **4** | **>= 5** |
| … |  |  |  |  |  |  |
| intra\_bc\_flag | 0 | na | na | na | na | na |
| segmental\_ibc\_flag[ ][ ] | 0 | na | na | na | na | na |
| segment\_num\_minus2[ ][ ] | 0 | na | na | na | na | na |
| abs\_segment\_offset[ x0 ][ y0 ][ j ][cIdx ] | 2\*(nSegNum[ x0 ][ y0 ]– 2) + j | bypass | bypass | bypass | bypass | bypass |
| segment\_offset\_sign[ x0 ][ y0 ][ j ][cIdx ] | 2\*(nSegNum[ x0 ][ y0 ]– 2 ) + j | na | na | na | na | na |
| pred\_mode\_flag | 0 | na | na | na | na | na |
| … |  |  |  |  |  |  |

Table 9‑40 – Specification of ctxInc using left and above syntax elements

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