#### Coding unit syntax

|  |  |
| --- | --- |
| coding\_unit( x0, y0, log2CbSize ) { | Descriptor |
| if( transquant\_bypass\_enabled\_flag ) |  |
| **cu\_transquant\_bypass\_flag** | ae(v) |
| if( slice\_type != I ) |  |
| **cu\_skip\_flag**[ x0 ][ y0 ] | ae(v) |
| nCbS = ( 1  <<  log2CbSize ) |  |
| if( cu\_skip\_flag[ x0 ][ y0 ] ) |  |
| prediction\_unit( x0, y0, nCbS, nCbS ) |  |
| else { |  |
| ~~if( intra\_block\_copy\_enabled\_flag )~~ |  |
| **~~intra\_bc\_flag~~**~~[ x0 ][ y0 ]~~ | ae(v) |
| if( slice\_type != I && ~~!intra\_bc\_flag[ x0 ][ y0 ]~~ intra\_block\_copy\_enabled\_flag) |  |
| **pred\_mode\_flag** | ae(v) |
| if( CuPredMode[ x0 ][ y0 ] != MODE\_INTRA ~~| | intra\_bc\_flag[ x0 ][ y0 ]~~  | |   log2CbSize = = MinCbLog2SizeY ) |  |
| **part\_mode** | ae(v) |
| if( CuPredMode[ x0 ][ y0 ] = = MODE\_INTRA ) { |  |
| if( PartMode = = PART\_2Nx2N && pcm\_enabled\_flag &&   ~~!intra\_bc\_flag[ x0 ][ y0 ] &&~~  log2CbSize >= Log2MinIpcmCbSizeY &&  log2CbSize <= Log2MaxIpcmCbSizeY ) |  |
| **pcm\_flag**[ x0 ][ y0 ] | ae(v) |
| if( pcm\_flag[ x0 ][ y0 ] ) { |  |
| while( !byte\_aligned( ) ) |  |
| **pcm\_alignment\_zero\_bit** | f(1) |
| pcm\_sample( x0, y0, log2CbSize ) |  |
| } ~~else if( intra\_bc\_flag[ x0 ][ y0 ] ) {~~ |  |
| ~~mvd\_coding( x0, y0, 2)~~ |  |
| ~~if( PartMode = = PART\_2NxN )~~ |  |
| ~~mvd\_coding( x0, y0 + ( nCbS / 2 ), 2)~~ |  |
| ~~else if( PartMode = = PART\_Nx2N )~~ |  |
| ~~mvd\_coding( x0 + ( nCbS / 2 ), y0, 2)~~ |  |
| ~~else if( PartMode = = PART\_NxN ) {~~ |  |
| ~~mvd\_coding( x0 + ( nCbS / 2 ), y0, 2)~~ |  |
| ~~mvd\_coding( x0, y0 + ( nCbS / 2 ), 2)~~ |  |
| ~~mvd\_coding( x0 + ( nCbS / 2 ), y0 + ( nCbS / 2 ), 2)~~ |  |
| ~~}~~ |  |
| } else { |  |
| pbOffset = ( PartMode = = PART\_NxN ) ? ( nCbS / 2 ) : nCbS |  |
| for( j = 0; j < nCbS; j = j + pbOffset ) |  |
| for( i = 0; i < nCbS; i = i + pbOffset ) |  |
| **prev\_intra\_luma\_pred\_flag**[ x0 + i ][ y0 + j ] | ae(v) |
| for( j = 0; j < nCbS; j = j + pbOffset ) |  |
| for( i = 0; i < nCbS; i = i + pbOffset ) |  |
| if( prev\_intra\_luma\_pred\_flag[ x0 + i ][ y0 + j ] ) |  |
| **mpm\_idx**[ x0 + i ][ y0 + j ] | ae(v) |
| else |  |
| **rem\_intra\_luma\_pred\_mode**[ x0 + i ][ y0 + j ] | ae(v) |
| if( ChromaArrayType = = 3 ) |  |
| for( j = 0; j < nCbS; j = j + pbOffset ) |  |
| for( i = 0; i < nCbS; i = i + pbOffset ) |  |
| **intra\_chroma\_pred\_mode**[ x0 + i ][ y0 + j ] | ae(v) |
| else if( ChromaArrayType != 0 ) |  |
| **intra\_chroma\_pred\_mode**[ x0 ][ y0 ] | ae(v) |
| } |  |
| } else { |  |
| if( PartMode = = PART\_2Nx2N ) |  |
| prediction\_unit( x0, y0, nCbS, nCbS ) |  |
| else if( PartMode = = PART\_2NxN ) { |  |
| prediction\_unit( x0, y0, nCbS, nCbS / 2 ) |  |
| prediction\_unit( x0, y0 + ( nCbS / 2 ), nCbS, nCbS / 2 ) |  |
| } else if( PartMode = = PART\_Nx2N ) { |  |
| prediction\_unit( x0, y0, nCbS / 2, nCbS ) |  |
| prediction\_unit( x0 + ( nCbS / 2 ), y0, nCbS / 2, nCbS ) |  |
| } else if( PartMode = = PART\_2NxnU ) { |  |
| prediction\_unit( x0, y0, nCbS, nCbS / 4 ) |  |
| prediction\_unit( x0, y0 + ( nCbS / 4 ), nCbS, nCbS \* 3 / 4 ) |  |
| } else if( PartMode = = PART\_2NxnD ) { |  |
| prediction\_unit( x0, y0, nCbS, nCbS \* 3 / 4 ) |  |
| prediction\_unit( x0, y0 + ( nCbS \* 3 / 4 ), nCbS, nCbS / 4 ) |  |
| } else if( PartMode = = PART\_nLx2N ) { |  |
| prediction\_unit( x0, y0, nCbS / 4, nCbS ) |  |
| prediction\_unit( x0 + ( nCbS / 4 ), y0, nCbS \* 3 / 4, nCbS ) |  |
| } else if( PartMode = = PART\_nRx2N ) { |  |
| prediction\_unit( x0, y0, nCbS \* 3 / 4, nCbS ) |  |
| prediction\_unit( x0 + ( nCbS \* 3 / 4 ), y0, nCbS / 4, nCbS ) |  |
| } else { /\* PART\_NxN \*/ |  |
| prediction\_unit( x0, y0, nCbS / 2, nCbS / 2 ) |  |
| prediction\_unit( x0 + ( nCbS / 2 ), y0, nCbS / 2, nCbS / 2 ) |  |
| prediction\_unit( x0, y0 + ( nCbS / 2 ), nCbS / 2, nCbS / 2 ) |  |
| prediction\_unit( x0 + ( nCbS / 2 ), y0 + ( nCbS / 2 ), nCbS / 2, nCbS / 2 ) |  |
| } |  |
| } |  |
| if( !pcm\_flag[ x0 ][ y0 ] ) { |  |
| if( CuPredMode[ x0 ][ y0 ] != MODE\_INTRA &&   !( PartMode = = PART\_2Nx2N && merge\_flag[ x0 ][ y0 ] ) | |   ~~( CuPredMode[ x0 ][ y0 ] = = MODE\_INTRA && intra\_bc\_flag[ x0 ][ y0 ] )~~ ) |  |
| **rqt\_root\_cbf** | ae(v) |
| if( rqt\_root\_cbf ) { |  |
| MaxTrafoDepth = ( CuPredMode[ x0 ][ y0 ] = = MODE\_INTRA ?   ( max\_transform\_hierarchy\_depth\_intra + IntraSplitFlag ) :   max\_transform\_hierarchy\_depth\_inter ) |  |
| transform\_tree( x0, y0, x0, y0, log2CbSize, 0, 0 ) |  |
| } |  |
| } |  |
| } |  |
| } |  |

#### 7.3.8.6 Prediction unit syntax

|  |  |
| --- | --- |
| prediction\_unit( x0, y0, nPbW, nPbH ) { | Descriptor |
| if( cu\_skip\_flag[ x0 ][ y0 ] ) { |  |
| if( MaxNumMergeCand > 1 ) |  |
| **merge\_idx**[ x0 ][ y0 ] | ae(v) |
| } else { /\* MODE\_INTER \*/ |  |
| if(slice\_type != I && intra\_block\_copy\_enabled\_flag) |  |
| **intra\_bc\_flag** | ae(v) |
| if( intra\_bc\_flag ) { |  |
| **symmetric\_ibc\_flag** | ae(v) |
| if(symmetric\_ibc\_flag) |  |
| **symmetric\_ibc\_type** | ae(v) |
| mvd\_coding( 0, 0, 2 ) |  |
| } else { |  |
| **merge\_flag**[ x0 ][ y0 ] | ae(v) |
| if( merge\_flag[ x0 ][ y0 ] ) { |  |
| if( MaxNumMergeCand > 1 ) |  |
| **merge\_idx**[ x0 ][ y0 ] | ae(v) |
| } else { |  |
| ... |  |
| } |  |
| } |  |

#### Motion vector difference syntax

|  |  |
| --- | --- |
| mvd\_coding( x0, y0, refList ) { | **Descriptor** |
| if(!symmetric\_ibc\_flag[x0][y0]**||** symmetric\_ibc\_type[x0][y0]!=1) |  |
| **abs\_mvd\_greater0\_flag**[ 0 ] | ae(v) |
| if(!symmetric\_ibc\_flag[x0][y0]**||** symmetric\_ibc\_type[x0][y0]!=0) |  |
| **abs\_mvd\_greater0\_flag**[ 1 ] | ae(v) |
| if( abs\_mvd\_greater0\_flag[ 0 ] ) |  |
| **abs\_mvd\_greater1\_flag**[ 0 ] | ae(v) |
| if( abs\_mvd\_greater0\_flag[ 1 ] ) |  |
| **abs\_mvd\_greater1\_flag**[ 1 ] | ae(v) |
| if( abs\_mvd\_greater0\_flag[ 0 ] ) { |  |
| if( abs\_mvd\_greater1\_flag[ 0 ] ) |  |
| **abs\_mvd\_minus2**[ 0 ] | ae(v) |
| **mvd\_sign\_flag**[ 0 ] | ae(v) |
| } |  |
| if( abs\_mvd\_greater0\_flag[ 1 ] ) { |  |
| if( abs\_mvd\_greater1\_flag[ 1 ] ) |  |
| **abs\_mvd\_minus2**[ 1 ] | ae(v) |
| **mvd\_sign\_flag**[ 1 ] | ae(v) |
| } |  |
| } |  |

#### Prediction unit semantics

…

**intra\_bc\_flag**[ x0 ][ y0 ] equal to 1 specifies that the current prediction unit is coded in intra block copying mode. intra\_bc\_flag[ x0 ][ y0 ] equal to 0 specifies that the current prediction unit is coded according to pred\_mode\_flag. When not present, the value of intra\_bc\_flag[ x0 ][ y0] is derived as the following:

* If intra\_block\_copy\_enabled\_flag is equal to 1, pred\_mode\_flag is equal to MODE\_INTER, and slice\_type is equal to I, intra\_bc\_flag[ x0 ][ y0 ] is set to be 1.
* Otherwise, intra\_bc\_flag[ x0 ][ y0 ] is set 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.

**symmeric\_ibc\_flag**[ x0 ][ y0 ] equal to 1 specifies that the current coding unit is coded in symmectic intra block copying mode. symmeric\_ibc\_flag [ x0 ][ y0 ] equal to 0 specifies that the current coding unit is not coded in symmectic intra block copying mode. When not present, the value of symmeric\_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.

**symmeric\_ibc\_type**[ x0 ][ y0 ] equal to 1 specifies that the current coding unit is coded in horizontal symmectic intra block copying mode. symmeric\_ibc\_flag [ x0 ][ y0 ] equal to 0 specifies that the current coding unit is coded in vertical symmectic intra block copying mode. When not present, the value of symmeric\_ibc\_type 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.

…

#### Motion vector difference semantics

**abs\_mvd\_greater0\_flag[** compIdx **]** specifies whether the absolute value of a motion vector component difference is greater than 0.

When abs\_mvd\_greater0\_flag [ compIdx ] is not present, it is inferred to be equal to 0.

### 8.4.4 Derivation process for block vector components in intra block copying prediction mode

…

–Otherwise, bvIntra[ xPb ][ yPb ][ compIdx ] is derived as follows:

bvIntra[ xPb ][ yPb ][ 0 ] = BvdIntra[ xPb ][ yPb ][ 0 ] + BvpIntra[ 0 ]

bvIntra[ xPb ][ yPb ][ 1 ] = BvdIntra[ xPb ][ yPb ][ 1 ] + BvpIntra[ 1 ]

If symmetric\_ibc\_flag[ xPb ][ yPb ] is equal to 1 and symmetric\_ibc\_type[ xPb ][ yPb ] is equal to 0, bvIntra[ xPb ][ yPb ][ 1 ] is set equal to 0.

If symmetric\_ibc\_flag[ xPb ][ yPb ] is equal to 1 and symmetric\_ibc\_type[ xPb ][ yPb ] is equal to 1, bvIntra[ xPb ][ yPb ][ 0 ] is set equal to 0.

**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 ] if symmeric\_ibc\_flag[xTbY][ yTbY] is equal to 0.

Otherwise (symmeric\_ibc\_flag[xTbY][ yTbY] is equal to 1), each sample at the location ( xRefCmp, yRefCmp ) is assigned to predSamples[ nTbS-1-x ][ y ] if symmeric\_ibc\_type [xTbY][ yTbY] is equal to 0; Otherwise (symmeric\_ibc\_type [xTbY][ yTbY] is lower equal to 1), each sample at the location ( xRefCmp, yRefCmp ) is assigned to predSamples[ x ][ nTbS-1-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 |
| symmetric\_ibc\_flag[ ][ ] | Table 9-34’ | 0…2 | 0…2 | 0…2 |
| symmetric\_ibc\_type[ ][ ] | Table 9-34’ | 0…2 | 0…2 | 0…2 |
| pred\_mode\_flag |  |  | 0 | 1 |
| … |  |  |  |  |

Table 9‑34’ – Values of initValue for ctxIdx of symmetric\_ibc\_flag and symmetic\_ibc\_type

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

**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 |
| symmetric\_ibc\_flag | FL | cMax = 1 |
| symmetric\_ibc\_type | 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 |
| symmetric\_ibc\_flag | 0 | na | na | na | na | na |
| symmetric\_ibc\_type | 0 | 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 ) |
| symmetric\_ibc\_flag[ x0 ][ y0 ] | symmetric\_ibc\_flag[ xNbL ][ yNbL ] | symmetric\_ibc\_flag [ xNbA ][ yNbA ] | ( condL  &&  availableL ) + ( condA  &&  availableA ) |
| symmetric\_ibc\_type[ x0 ][ y0 ] | symmetric\_ibc\_type[ xNbL ][ yNbL ] | symmetric\_ibc\_type [ xNbA ][ yNbA ] | ( condL  &&  availableL ) + ( condA  &&  availableA ) |