Chroma QP adjustment spec, based upon N1005\_v3

#### Picture parameter set RBSP syntax

|  |  |
| --- | --- |
| pic\_parameter\_set\_rbsp( ) { | Descriptor |
| **...** | ... |
| **pps\_extension1\_flag** | u(1) |
| if( pps\_extension1\_flag ) { |  |
| if( transform\_skip\_enabled\_flag ) |  |
| **log2\_transform\_skip\_max\_size\_minus2** | ue(v) |
| **chroma\_tu\_restriction\_enabled\_flag** | u(1) |
| if( chroma\_tu\_restriction\_enabled\_flag ) |  |
| **diff\_cu\_chroma\_tu\_restriction\_depth** | ue(v) |
| **pps\_extension2\_flag** | u(1) |
| } |  |
| ... |  |
| } |  |

#### Picture parameter set RBSP semantics

**chroma\_tu\_restriction\_enabled\_flag** equal to 1 specifies that the cu\_chroma\_tu\_restriction\_flag may be present in the coding unit syntax. cu\_chroma\_tu\_restriction\_enabled\_flag equal to 0 specifies that the cu\_chroma\_tu\_restriction\_flag is not present in the coding unit syntax.

When ChromaArrayType is not equal to 3, it is a requirement of bitstream conformance that the value of cu\_chroma\_tu\_restriction\_enabled\_flag shall be equal to 0.

**diff\_cu\_chroma\_tu\_restriction\_depth** specifies the difference between the luma coding tree block size and the minimum luma coding block size of coding units that convey cu\_chroma\_tu\_restriction\_flag. The value of diff\_cu\_chroma\_tu\_restriction\_depth shall be in the range of 0 to log2\_diff\_max\_min\_luma\_coding\_block\_size, inclusive.

XXX: When not present, the value of diff\_cu\_chroma\_tu\_restriction\_depth is inferred to be equal to 0.

The variable Log2MinCuChromaTuRestrictionSize is derived as follows:

Log2MinCuQpChromaTuRestrictionSize = CtbLog2SizeY − diff\_cu\_chroma\_tu\_restriction\_depth (7‑31)

#### Coding quadtree syntax

|  |  |
| --- | --- |
| coding\_quadtree( x0, y0, log2CbSize, cqtDepth ) { | Descriptor |
| if( x0 + ( 1  <<  log2CbSize ) <= pic\_width\_in\_luma\_samples &&  y0 + ( 1  <<  log2CbSize ) <= pic\_height\_in\_luma\_samples &&  log2CbSize > MinCbLog2SizeY ) |  |
| **split\_cu\_flag**[ x0 ][ y0 ] | ae(v) |
| if( cu\_qp\_delta\_enabled\_flag && log2CbSize >= Log2MinCuQpDeltaSize ) { |  |
| IsCuQpDeltaCoded = 0 |  |
| CuQpDeltaVal = 0 |  |
| } |  |
| if( chroma\_tu\_restriction\_enabled\_flag && log2CbSize >= Log2MinCuQpChromaTuRestrictionSize ) { |  |
| IsCuQpChromaTuRestrictionCoded = 0 |  |
| CuQpChromaTuRestriction = 0 |  |
| } |  |
| ... |  |
| } |  |

#### 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( !intra\_bc\_flag[ x0 ][ y0 ] ) { |  |
| if( slice\_type != I ) |  |
| **pred\_mode\_flag** | ae(v) |
| if( CuPredMode[ x0 ][ y0 ] != MODE\_INTRA | |   log2CbSize = = MinCbLog2SizeY ) |  |
| **part\_mode** | ae(v) |
| } |  |
| if( CuPredMode[ x0 ][ y0 ] = = MODE\_INTRA ) { |  |
| if( PartMode = = PART\_2Nx2N && pcm\_enabled\_flag && !intra\_bc\_flag  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) |  |
| } else { |  |
| if( chroma\_tu\_restriction\_enabled\_flag && IntraSplitFlag && !IsCuQpChromaTuRestrictionCoded ) { |  |
| **cu\_chroma\_tu\_restriction\_flag** | ae(v) |
| } |  |
| 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 && IntraSplitFlag && !CuQpChromaTuRestriction ) |  |
| 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) |
| } |  |
| ... |  |
| } |  |

#### Coding unit semantics

**cu\_chroma\_tu\_restriction\_flag** equal to 1 specifies that a restriction to the minimum chroma transform unit size shall be enforced for this coding unit. cu\_chroma\_tu\_restriction\_flag equal to 0 specifies that no transform unit size restriction shall be enforced for this coding unit.

When cu\_chroma\_tu\_restriction\_flag is present, the variable IsCuQpChromaTuRestrictionCoded is set equal to 1, and the variable CuQpChromaTuRestriction is set equal to cu\_chroma\_tu\_restriction\_flag.

#### Transform unit syntax

|  |  |
| --- | --- |
| transform\_unit( x0, y0, xBase, yBase, log2TrafoSize, trafoDepth, blkIdx ) { | Descriptor |
| ... |  |
| if( cbf\_luma[ x0 ][ y0 ][ trafoDepth ] ) |  |
| residual\_coding( x0, y0, log2TrafoSize, 0 ) |  |
| if( log2TrafoSize > 2 | | ChromaArrayType = = 3 && !CuQpChromaTuRestriction ) { |  |
| 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 ) |  |
| 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 ][ trafoDepth ] ) |  |
| residual\_coding( xBase, yBase + ( tIdx << log2TrafoSize ), log2TrafoSize, 1 ) |  |
| for( tIdx = 0; tIdx < ( ChromaArrayType = = 2 ? 2 : 1 ); tIdx++) |  |
| if( cbf\_cr[ xBase ][ yBase ][ trafoDepth ] ) |  |
| residual\_coding( xBase, yBase + ( tIdx << log2TrafoSize ), log2TrafoSize, 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** |
| coding\_unit( ) | cu\_chroma\_tu\_restriction\_flag | XXX | 0 | 1 | 2 |

Table 9‑5 – Values of initValue for ctxIdx of cu\_chroma\_tu\_restriction\_flag

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of cu\_chroma\_tu\_restriction\_flag** | | |
| **0** | **1** | **2** |
| **initValue** | 153 | 153 | 153 |

### Binarization process

#### General

| Table 9‑34 – Syntax elements and associated binarizations | | | |
| --- | --- | --- | --- |
| **Syntax structure** | **Syntax element** | **Binarization** | |
| **Process** | **Input parameters** |
| coding\_unit( ) | cu\_chroma\_tu\_restriction\_flag | FL | cMax = 1 |

#### Derivation process for ctxTable, ctxIdx and bypassFlag

##### General

| Table 9‑39 – Assignment of ctxInc to syntax elements with context coded bins | | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **Syntax element** | **binIdx** | | | | | |
| **0** | **1** | **2** | **3** | **4** | **>= 5** |
| cu\_chroma\_tu\_restriction\_flag | 0 | na | na | na | na | na |