G.7.3.9.1. General Coding unit syntax

|  |  |
| --- | --- |
| coding\_unit( x0, y0, log2CbSize , ctDepth) { | **Descriptor** |
| if( transquant\_bypass\_enable\_flag ) { |  |
| **cu\_transquant\_bypass\_flag** | ae(v) |
| } |  |
| if( slice\_type != I   && TextureModeDepth[ x0 >> Log2MinCbSize ][ y0 >> Log2MinCbSize ] = = -1) |  |
| **skip\_flag**[ x0 ][ y0 ] | ae(v) |
| if( skip\_flag[ x0 ][ y0 ] ) |  |
| prediction\_unit( x0, y0, log2CbSize ) |  |
| else { |  |
| if( TextureModeDepth[ x0 >> Log2MinCbSize ][ y0 >> Log2MinCbSize ] = = 1 ) { |  |
| nCbS = ( 1 << log2CbSize ) |  |
| if( slice\_type != I ) |  |
| **pred\_mode\_flag** | ae(v) |
| if( PredMode[ x0 ][ y0 ] != MODE\_INTRA | | log2CbSize = = Log2MinCbSize ) |  |
| **part\_mode** | ae(v) |
| if( PredMode[ x0 ][ y0 ] = = MODE\_INTRA ) { |  |
| if( PartMode = = PART\_2Nx2N && pcm\_enabled\_flag &&  log2CbSize >= Log2MinIPCMCUSize &&  log2CbSize <= Log2MaxIPCMCUSize ) |  |
| **pcm\_flag** | ae(v) |
| if( pcm\_flag ) { |  |
| **num\_subsequent\_pcm** | tu(3) |
| NumPCMBlock = num\_subsequent\_pcm + 1 |  |
| while( !byte\_aligned( ) ) |  |
| **pcm\_alignment\_zero\_bit** | f(1) |
| pcm\_sample( x0, y0, log2CbSize ) |  |
| } else { |  |
| pbOffset = ( PartMode = = PART\_NxN ) ? ( nCbS / 2 ) : 0 |  |
| if ( enable\_DMM\_flag && log2CbSize <= Log2DMMMaxSize ) { |  |
| for( j = 0; j <= pbOffset; j = j + pbOffset ) |  |
| for( i = 0; i <= pbOffset; i = i + pbOffset ) |  |
| **dmm\_flag[** x0 + i ][ y0+ j ] | ae(v) |
| for( j = 0; j <= pbOffset; j = j + pbOffset ) |  |
| for( i = 0; i <= pbOffset; i = i + pbOffset ) |  |
| if (dmm\_flag**[** x0 + i ][ y0+ j ] ) |  |
| **dmm\_mode[** x0 + i ][ y0+ j ] | ae(v) |
| for( j = 0; j <= pbOffset; j = j + pbOffset ) |  |
| for( i = 0; i <= pbOffset; i = i + pbOffset ) |  |
| if ( dmm\_flag**[** x0 + i ][ y0+ j ] && (  dmm\_mode**[** x0 + i ][ y0+ j ] = = MODE\_DMM\_WFULL ||   dmm\_mode**[** x0 + i ][ y0+ j ] = = MODE\_DMM\_WFULLDELTA ) ) |  |
| **wedge\_full\_tab\_idx[**x0 + i **][**y0 + i **]** | ae(v) |
| for( j = 0; j <= pbOffset; j = j + pbOffset ) |  |
| for( i = 0; i <= pbOffset; i = i + pbOffset ) |  |
| if( dmm\_flag**[** x0 + i ][ y0+ j ] && (  dmm\_mode**[** x0 + i ][ y0+ j ] = = MODE\_DMM\_WPREDDIR ||   dmm\_mode**[** x0 + i ][ y0+ j ] = = MODE\_DMM\_WPREDDIRDELTA ) ) |  |
| **dmm\_delta\_end\_flag[**x0 + i **][**y0 + i **]** | ae(v) |
| for( j = 0; j <= pbOffset; j = j + pbOffset ) |  |
| for( i = 0; i <= pbOffset; i = i + pbOffset ) |  |
| if ( dmm\_delta\_end\_flag**[**x0 + i **][**y0 + i **] )** |  |
| **dmm\_delta\_end\_abs\_minus1[**x0 + i **][**y0 + i **]** | ae(v) |
| for( j = 0; j <= pbOffset; j = j + pbOffset ) |  |
| for( i = 0; i <= pbOffset; i = i + pbOffset ) |  |
| if ( dmm\_delta\_end\_flag**[**x0 + i **][**y0 + i **] )** |  |
| **dmm\_delta\_end\_sign\_flag**[ x0 **][**y0 **]** | ae(v) |
| for( j = 0; j <= pbOffset; j = j + pbOffset ) |  |
| for( i = 0; i <= pbOffset; i = i + pbOffset ) |  |
| DmmDeltaFlag**[**x0 + i **][**y0 + i **]** = ( dmm\_flag**[** x0 + i ][ y0+ j ] && (  dmm\_mode**[**x0 + i **][**y0 + i **]**  = = MODE\_DMM\_WFULLDELTA ||   dmm\_mode**[**x0 + i **][**y0 + i **]**  = = MODE\_DMM\_WPREDDIRDELTA ||   dmm\_mode**[**x0 + i **][**y0 + i **]**  = = MODE\_DMM\_WPREDTEXDELTA ||   dmm\_mode**[**x0 + i **][**y0 + i **]**  = = MODE\_DMM\_CPREDTEXDELTA **) )** |  |
| for( j = 0; j <= pbOffset; j = j + pbOffset ) |  |
| for( i = 0; i <= pbOffset; i = i + pbOffset ) |  |
| if ( DmmDeltaFlag**[**x0 + i **][**y0 + i **] )** |  |
| **dmm\_dc\_1\_abs[**x0 + i **][**y0 + i **]** | ae(v) |
| for( j = 0; j <= pbOffset; j = j + pbOffset ) |  |
| for( i = 0; i <= pbOffset; i = i + pbOffset ) |  |
| if ( DmmDeltaFlag**[** x0 + i ][ y0+ j ] && dmm\_dc\_1\_abs**[**x0 + i **][**y0 + i **]** != 0 ) |  |
| **dmm\_dc\_1\_sign\_flag[**x0 + i **][**y0 + i **]** | ae(v) |
| for( j = 0; j <= pbOffset; j = j + pbOffset ) |  |
| for( i = 0; i <= pbOffset; i = i + pbOffset ) |  |
| if ( DmmDeltaFlag**[** x0 + i ][ y0+ j ] ) |  |
| **dmm\_dc\_2\_abs[**x0 + i **][**y0 + i **]** | ae(v) |
| for( j = 0; j <= pbOffset; j = j + pbOffset ) |  |
| for( i = 0; i <= pbOffset; i = i + pbOffset ) |  |
| if ( DmmDeltaFlag**[** x0 + i ][ y0+ j ] && dmm\_dc\_2\_abs**[**x0 + i **][**y0 + i **]** != 0 ) |  |
| **dmm\_dc\_2\_sign\_flag[**x0 + i **][**y0 + i **]** | ae(v) |
| } else { |  |
| for( j = 0; j <= pbOffset; j = j + pbOffset ) |  |
| for( i = 0; i <= pbOffset; i = i + pbOffset ) { |  |
| if( !dmm\_flag**[** x0 + i ][ y0+ j ] ) |  |
| **prev\_intra\_luma\_pred\_flag**[ x0 + i ][ y0+ j ] | ae(v) |
| } |  |
| for( j = 0; j <= pbOffset; j = j + pbOffset ) |  |
| for( i = 0; i <= pbOffset; i = i + pbOffset ) { |  |
| if( !dmm\_flag**[** x0 + i ][ y0+ j ] ) { |  |
| 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( rem\_intra\_luma\_pred\_mode[ x0 + i ][ y0+ j ] == 31 ) { |  |
| **edge\_intra\_flag**[ x0 + i ][ y0+ j ] | ae(v) |
| if( edge\_intra\_flag[ x0 + i ][ y0+ j ] ) { |  |
| **psip\_dir\_flag**[ x0 + i ][ y0+ j ] | ae(v) |
| edge\_dir = 0 |  |
|  |  |
|  |  |
| for( k = 0; k <pbOffset; k++ ) { |  |
| **psip\_code**[k] | ae(v) |
| if (psip\_code[k] > 0 ) edge\_dir = 0 |  |
| else if( psip\_code[k] < 0 ) edge\_dir = 1 |  |
| } |  |
| **edge\_dc\_flag**[ x0 + i ][ y0+ j ] | ae(v) |
| if( edge\_dc\_flag[ x0 + i ][ y0+ j ] ) { |  |
| **edge\_dc\_1\_abs**[ x0 + i ][ y0+ j ] | ae(v) |
| if( edge\_dc\_1\_abs[ x0 + i ][ y0+ j ] != 0 ) |  |
| **edge\_dc\_1\_sign\_flag**[ x0 + i ][ y0+ j ] | ae(v) |
| **edge\_dc\_2\_abs**[ x0 + i ][ y0+ j ] | ae(v) |
| if( edge\_dc\_2\_abs[ x0 + i ][ y0+ j ] != 0 ) |  |
| **edge\_dc\_2\_sign\_flag**[ x0 + i ][ y0+ j ] | ae(v) |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |
| **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 ( !depth\_flag && ViewId && multi\_view\_residual\_pred\_flag &&  ResidualCbfNonZero && PredMode != MODE\_INTRA) |  |
| **res\_pred\_flag** | ae(v) |
| if ( TextureModeDepth[ x0 >> Log2MinCbSize ][ y0 >> Log2MinCbSize ] = = ctDepth   && split\_coding\_unit\_flag[ x0 ][ y0 ] ) |  |
| coding\_tree( x0, y0, log2CbSize, ctDepth ) |  |
| if( !pcm\_flag ) { |  |
| if( PredMode[ x0 ][ y0 ] != MODE\_INTRA &&   !(PartMode = = PART\_2Nx2N && merge\_flag[x0][y0]) ) |  |
| **no\_residual\_syntax\_flag** | ae(v) |
| if( !no\_residual\_syntax\_flag ) { |  |
| MaxTrafoDepth = ( PredMode[ x0 ][ y0 ] = = MODE\_INTRA ?   max\_transform\_hierarchy\_depth\_intra + IntraSplitFlag :   max\_transform\_hierarchy\_depth\_inter ) |  |
| transform\_tree( x0, y0 x0, y0, log2CbSize, 0, 0 ) |  |
| } |  |
| } |  |
| } |  |
| } |  |

G7.4.9.1. General coding unit semantics

The specification in subclause F.7.4.9.1 apply with the following modifications and additions.

**dmm\_flag[**x0**][**y0**]** equal to 0 specifies that depth map model modes are not used. dmm\_flag[x0][y0] equal to 1 specifies that depth map model modes are used that the dmm\_mode syntax element is present. If the dmm\_flag[x0][y0] is not present in the bitstream, its value should be inferred to be equal to 0.

**dmm\_mode**[x0][y0]shall be one of the values shown in . dmm\_mode is used to derive the intra prediction mode when the dmm\_flag[x0][y0] is equal to 1.

**Table G-1 – Interpretation of dmm\_mode**

|  |  |
| --- | --- |
| **dmm\_mode** | **Method identifier** |
| **0** | **MODE\_DMM\_WFULL** |
| **1** | **MODE\_DMM\_WFULLDELTA** |
| **2** | **MODE\_DMM\_WPREDTEX** |
| **3** | **MODE\_DMM\_WPREDTEXDELTA** |
| **4** | **MODE\_DMM\_CPREDTEX** |
| **5** | **MODE\_DMM\_CPREDTEXDELTA** |
| **6** | **MODE\_DMM\_WPREDDIR** |
| **7** | **MODE\_DMM\_WPREDDIRDELTA** |

**wedge\_full\_tab\_idx[**x0**][**y0**]** specifies the index of the wedgelet pattern in the corresponding pattern list.

**dmm\_delta\_end\_flag[**x0**][**y0**]** equal to 0specifies that dmm\_delta\_end\_abs\_minus1[x0**]**[y0] and dmm\_delta\_end\_sign[x0][y0] syntax elements are not present. dmm\_delta\_end\_flag equal to 1 specifies that dmm\_delta\_end\_abs\_minus1[x0][y0] and dmm\_delta\_end\_sign**[**x0**][**y0**]** syntax elements are present.

**dmm\_delta\_end\_abs\_minus1[**x0**][**y0**]** and **dmm\_delta\_end\_sign\_flag[**x0**][**y0**]** are used to derive DmmDeltaEnd[ x0 ][ y0 ] as follows:

DmmDeltaEnd[x0][y0] = ( 1 − 2 \*dmm\_delta\_end\_sign\_flag[ x0 ][ y0 ] ) \*  
( dmm\_delta\_end\_abs\_minus1[ x0 ][ y0 ] + 1) (G‑17)

**dmm\_dc\_1\_abs[**x0**][**y0**]**, **dmm\_dc\_1\_sign\_flag[**x0**][**y0**]**, **dmm\_dc\_2\_abs[**x0**][**y0**]**, **dmm\_dc\_2\_sign\_flag[**x0**][**y0**]** are used to derive DmmQuantOffsetDC1[ x0 ][ y0 ] and DmmQuantOffsetDC2[ x0 ][ y0 ] values as follows:

DmmQuantOffsetDC1[x0][y0] = ( 1 − 2 \*dmm\_dc\_1\_sign\_flag[x0][y0] ) \* dmm\_dc\_1\_abs[x0][y0] (G‑18)  
DmmQuantOffsetDC2[x0][y0] = ( 1 − 2 \*dmm\_dc\_2\_sign\_flag[x0][y0] ) \* dmm\_dc\_2\_abs[x0][y0] (G‑19)

**edge\_intra\_flag[**x0**][**y0**]** equal to 0 specifies that plane-based intra prediction (psip) coding is not used. edge\_intra\_flag[ x0 ][ y0 ] equal to 1 specifies that psip coding is used.

**psip\_dir\_flag[ x0 ][ y0 ] equal to 0 specifies that the psip code is decoded in horizontal direction. psip\_dir\_flag[ x0 ][ y0 ] equal to 1 specifies that the psip code is decoded in vertical direction.**

**psip\_code[ i ] specifies the difference of pattern between two adjacent rows or columns.**

Table\_G-2 – Interpretation of psip\_code if edge\_dir = 0

|  |  |
| --- | --- |
| **psip\_code[i]** | **Pattern code difference** |
| **0** | **0** |
| **10** | **1** |
| **110** | **-1** |
| **1110** | **2** |
| **11110** | **-2** |
| **111110** | **3** |
| **111111** | **-3** |
| **…..** | **….** |

Table\_G-3 – Interpretation of psip\_code if edge\_dir = 1

|  |  |
| --- | --- |
| **psip\_code[i]** | **Pattern code difference** |
| **0** | **0** |
| **10** | **-1** |
| **110** | **1** |
| **1110** | **-2** |
| **11110** | **2** |
| **111110** | **-3** |
| **111111** | **3** |
| **…..** | **….** |



**edge\_dc\_flag[**x0**][**y0**]** equal to 0 specifies that de-quantized offset of region boundary chain coding is not used. edge\_dc\_flag[ x0 ][ y0 ] equal to 1 specifies that de-quantized offset of region boundary chain coding is used.

**edge\_dc\_1\_abs[**x0**][**y0**], edge\_dc\_1\_sign\_flag[**x0**][**y0**], edge\_dc\_2\_abs[**x0**][**y0**], edge\_dc\_2\_sign\_flag[**x0**][**y0**]** are used to derive EdgeQuantOffsetDC1[ x0 ][ y0 ] and EdgeQuantOffsetDC2[ x0 ][ y0 ] values as follows:

EdgeQuantOffsetDC1[ x0 ][y0] = ( 1 − 2 \*edge\_dc\_1\_sign\_flag[ x0 ][y0]) \* edge\_dc\_1\_abs[ x0 ][y0] (G‑)  
EdgeQuantOffsetDC2[ x0 ][y0] = ( 1 − 2 \*edge\_dc\_2\_sign\_flag[ x0 ][y0]) \* edge\_dc\_2\_abs[ x0 ][y0] (G‑)

**res\_pred\_flag** equal to 0 specifies that residual prediction is not used. res\_pred\_flag equal to 1 specifies that residual prediction is used. When res\_pred\_flag is not present in the bitstream, its value shall be inferred to be equal to 0.

The variable ResidualCbfNonZero is derived by values of cbf\_luma, cbf\_cb, cbf\_cr and PredMode of the corresponding blocks as follows: ResidualCbfNonZero is set to 1 if at least one of the corresponding blocks has both PredMode not equal to MODE\_INTRA and any of the values of cbf\_luma, cbf\_cb and cbf\_cr not equal to 0; otherwise, ResidualCbfNonZero is set equal to 0.The corresponding blocks are identified by the current PU and the disparity vector. All the corresponding blocks belong to transform units that are covered or partially covered by a corresponding rectangle area (of the current PU) in the inter-view reference view component, after shifting the PU location with a disparity vector.

G.8.4.2. 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,

– a variable log2PbSize specifying the size of the current luma prediction block.

specifies the value for the intra prediction mode and the associated names.

Table F‑ – Specification of intra prediction mode and associated names

|  |  |
| --- | --- |
| Intra prediction mode | 1. **Associated names** |
| 1. 0 | 1. Intra\_Planar |
| 1. 1 | 1. Intra\_DC |
| 1. 2..34 | 1. Intra\_Angular |
| 1. (35...42) | 1. Intra\_DepthPartition (used only for depth) |
| 1. Otherwise (43, 44) | 1. Intra\_PSIP (used only for depth) |

G.8.4.4.2.12. Specification of Intra\_PSIP (43, 44) prediction mode

Inputs to this process are:

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

– a variable nT specifying the transform size,

Output of this process is:

– predicted samples predSamples[ x, y ], with x, y = -1..nT−1.

This intra prediction mode is invoked when intraPredMode is equal to 43 or 44.

The value of representative value RepVal[i], with i=0,1 are derived as per the following ordered steps:

1. Let P = { p[ x, −1 ], with x=0..nT-1, p[ −1, y ], with y=0..nT-1 }
2. Let Th = ( Min(P) + Max(P) ) >> 1
3. Segment P into two group: P0, P1, where P0 = { x | x in P and x <= Th}, P1 = { y | y in P and y > Th }
4. If P0={}, then Set RepVal[0] = Min(P). Otherwise, set RepVal[0] = Average(P0)
5. If P1={}, then setRepVal[1] = Max(P). Otherwise, set RepVal[1] = Average(P1)
6. Reset Th = ( RepVal[0] + RepVal[1] ) >> 1

The values of the prediction samples predSamples[ x, y ], with x, y = -1..nT−1, are derived as per the following ordered steps:

* If psip\_dir\_flag is equal to 0,

1. If p[x,−1] <= Th, set predSamples [x,-1] = 0, with x=0..nT-1. Otherwise, set predSamples [x,-1] = 1.
2. For y=0…nT-1,

- if psip\_code[y] = 0, predSamples [x,y] = predSamples [x,y-1], with x=0...nT-1

- if psip\_code[y] > 0,

predSamples [x,y] = 1 – predSamples [(x-psip\_code[y]+nT)%nT,y-1], with x=0..psip\_code[y]-1

predSamples [x,y] = predSamples [(x-psip\_code[y]+nT)%nT,y-1], with x=psip\_code[y]..nT-1

- if psip\_code[y] < 0,

predSamples [x,y] = predSamples [(x-psip\_code[y]+nT)%nT,y-1], with x=0..nT-1+psip\_code[y]

predSamples [x,y] = 1 – predSamples [(x-psip\_code[y]+nT)%nT,y-1], with x=nT+psip\_code[y]..nT-1

1. The depth partition value derivation and assignment process as specified in subclause G.8.4.4.2.13 is invoked with p[ x, y ], EdgePattern[ x, y ], RepVal[i] and nT as inputs and the output is assigned to predSamples[ x, y ].

* If psip\_dir\_flag is equal to 1,

1. If p[−1,y] <= Th, set predSamples [-1,y] = 0, with y=0..nT-1. Otherwise, set predSamples [-1,y] = 1.
2. For x=0…nT-1,

- if psip\_code[x] = 0, predSamples [x,y] = predSamples [x-1,y], with y=0...nT-1

- if psip\_code[x] > 0,

predSamples [x,y] = 1 – predSamples [x-1,(y-psip\_code[x]+nT)%nT], with y=0..psip\_code[x]-1

predSamples [x,y] = predSamples [x-1,(y-psip\_code[x]+nT)%nT], with y=psip\_code[x]..nT-1

- if psip\_code[x] < 0,

predSamples [x,y] = predSamples [x-1,(y-psip\_code[x]+nT)%nT], with y=0..nT-1+psip\_code[x]

predSamples [x,y] = 1 – predSamples [x-1,(y-psip\_code[x]+nT)%nT], with y=nT+psip\_code[x]..nT-1

1. The depth partition value derivation and assignment process as specified in subclause G.8.4.4.2.13 is invoked with p[ x, y ], EdgePattern[ x, y ], RepVal[i], and nT as inputs and the output is assigned to predSamples[ x, y ].

G8.4.4.2.13. Depth partition value derivation and assignment process for region boundary chain coding mode

Inputs to this process are:

– binary pattern EdgePattern[ x, y ], with x, y =-1..nT−1,

representative values RepVal[ i ], with i =0, 1,

– a variable nT specifying the transform size,

Output of this process is:

– predicted samples predSamples[ x, y ], with x, y =0..nT−1.

The values of the prediction samples predSamples[ x, y ], with x, y = 0..nT−1, are derived as per the following ordered steps:

1. The predicted constant partition values EdgePredPartitionDC1 and EdgePredPartitionDC2 are set to RepVal[0] and RepVal[1], respectively.

(G‑)

1. The de-quantized offsets EdgeOffsetDC1 and EdgeOffsetDC2 are derived from EdgeQuantOffsetDC1[ x0 ][ y0 ] and EdgeQuantOffsetDC2[ x0 ][ y0 ] as follows.

EdgeOffsetDC1 = EdgeQuantOffsetDC1[ x0 ][ y0 ] \* Clip3( 1, ( 1 << BitDepthY ) − 1, 2 (QP’Y /10)- 2 ) (G‑)

EdgeOffsetDC2 = EdgeQuantOffsetDC2[ x0 ][ y0 ] \* Clip3( 1, ( 1 << BitDepthY ) − 1, 2 (QP’Y /10)- 2 ) (G‑)

1. The constant partition values are assigned to prediction values depending on the edge pattern value.

* If EdgePattern[ x, y ] is equal to 1, the following applies:

predSamples[ x, y ] = EdgePredPartitionDC2 + EdgeOffsetDC2, with x, y = 0..nT−1 (G‑)

* Otherwise, the following applies:

predSamples[ x, y ] = EdgePredPartitionDC1 + EdgeOffsetDC1, with x, y = 0..nT−1 (G‑)