H.7.3.9.3 Depth mode parameter syntax

|  |  |
| --- | --- |
| depth\_mode\_parameters( x0 , y0 ) { | **Descriptor** |
| **depth\_intra\_mode**[ x0 ][ y0 ] | ae(v) |
| if ( depth\_intra\_mode[ x0 ][ y0 ] = = INTRA\_DEP\_DMM\_WFULL  | |    depth\_intra\_mode[ x0 ][ y0 ] = = INTRA\_DEP\_SDC\_DMM\_WFULL ) |  |
| **wedge\_full\_tab\_idx**[ x0  ][ y0 ] | ae(v) |
|  |  |
|  | ae(v) |
| else if( depth\_intra\_mode[ x0 ][ y0 ] = = INTRA\_DEP\_DMM\_WPREDDIR ) { |  |
| **dmm\_delta\_end\_flag**[ x0 ][ y0 ] | ae(v) |
| if ( dmm\_delta\_end\_flag[ x0 ][ y0 ] ) { |  |
| **dmm\_delta\_end\_abs\_minus1**[ x0 ][ y0 ] | ae(v) |
| **dmm\_delta\_end\_sign\_flag**[ x0 ][ y0 ] | ae(v) |
| } |  |
| } else if( depth\_intra\_mode[ x0 ][ y0 ] = = INTRA\_DEP\_CHAIN ) { |  |
| **edge\_start\_left\_flag**[ x0 ][ y0 ] | ae(v) |
| **edge\_start\_position**[ x0 ][ y0 ] | ae(v) |
| **num\_edge\_codes\_minus1**[ x0 ][ y0 ] | ae(v) |
| for( k = 0; k <= num\_edge\_codes\_minus1; k++ ) |  |
| **edge\_code**[ k ] | ae(v) |
| } |  |
| if( DmmFlag[ x0 ][ y0 ] ) { |  |
| **dmm\_dc\_flag**[ x0 ][ y0 ] | ae(v) |
| if ( dmm\_dc\_flag[ x0 ][ y0 ] ) |  |
| for( i = 0; i < 2; i ++ ) { |  |
| **dmm\_dc\_abs**[ x0 ][ y0 ][ i ] | ae(v) |
| if ( dmm\_dc\_abs[ x0 ][ y0 ][ i ]) |  |
| **dmm\_dc\_sign\_flag**[ x0 ][ y0 ][ i ] | ae(v) |
| } |  |
| } |  |
| else if( depth\_intra\_mode[ x0 ][ y0 ] = = INTRA\_DEP\_CHAIN ) { |  |
| **edge\_dc\_flag**[ x0 ][ y0 ] | ae(v) |
| if( edge\_dc\_flag[ x0 ][ y0 ] ) { |  |
| for( i = 0; i < 2; i++ ) { |  |
| **edge\_dc\_abs**[ x0 ][ y0 ][ i ] | ae(v) |
| if( edge\_dc\_abs[ x0 ][ y0 ][ i ] != 0 ) |  |
| **edge\_dc\_sign\_flag**[ x0 ][ y0 ][ i ] | ae(v) |
| } |  |
| } |  |
| } |  |
| else if( SdcFlag[ x0 ][ y0 ] ) { |  |
| sdcNumSegments =   ( depth\_intra\_mode[ x0  ][ y0 ] = = INTRA\_DEP\_SDC\_DMM\_WFULL ) ? 2 : 1 |  |
| for( i = 0; i < sdcNumSegments; i++ ) { |  |
| **sdc\_residual\_flag**[ x0 ][ y0 ][ i ] | ae(v) |
| if( sdc\_residual\_flag[ x0 ][ y0 ][ i ])  { |  |
| **sdc\_residual\_sign\_flag**[ x0 ][ y0 ][ i ] | * + 1. ae(v) |
| **sdc\_residual\_abs\_minus1**[ x0 ][ y0 ][ i ] | * + 1. ae(v) |
| } |  |
| } |  |
| } |  |
| } |  |

H.7.4.9.3 Depth mode parameter semantics

The variable Log2MaxDmmCbSize is set equal to 5.

**depth\_intra\_mode**[ x0 ][ y0 ] specifies the depth intra mode of the current prediction unit. Table H‑2 specifies the value for the depth intra mode and the associated names.

The variable SdcFlag[ x0 ][ y0 ] is derived as specified in the following:

* 1. SdcFlag[ x0 ][ y0 ] = ( depth\_intra\_mode[ x0 ][ y0 ]  = =  INTRA\_DEP\_SDC\_PLANAR )  | |   (H‑16)  
      ( depth\_intra\_mode[ x0 ][ y0 ]  = =  INTRA\_DEP\_SDC\_DMM\_WFULL )  | |    
      ( depth\_intra\_mode[ x0 ][ y0 ]  = =  INTRA\_DEP\_SDC\_DC )

The variable DmmFlag[ x0 ][ y0 ] is derived as specified in the following:

* 1. DmmFlag[ x0 ][ y0 ] = ( depth\_intra\_mode[ x0 ][ y0 ]  = =  INTRA\_DEP\_DMM\_WFULL )  | |   (H‑17)  
        
      ( depth\_intra\_mode[ x0 ][ y0 ]  = =  INTRA\_DEP\_DMM\_WPREDDIR )

Table ‑2 – Specification of depth\_intra\_mode and associated names

|  |  |
| --- | --- |
| **depth\_intra\_mode** | **DepthIntraMode** |
| 0 | INTRA\_DEP\_SDC\_PLANAR |
| 1 | INTRA\_DEP\_NONE |
| 2 | INTRA\_DEP\_SDC\_DMM\_WFULL |
| 3 | INTRA\_DEP\_DMM\_WFULL |
|  |  |
|  |  |
| 4 | INTRA\_DEP\_SDC\_DC |
| 5 | INTRA\_DEP\_DMM\_WPREDDIR |
| 6 | INTRA\_DEP\_CHAIN |

**wedge\_full\_tab\_idx**[ x0 ][ y0 ]specifies the index of the wedgelet pattern in the corresponding pattern list when depth\_intra\_mode[ x0 ][ y0 ] is equal to INTRA\_DEP\_DMM\_WFULL.

**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:

* 1. DmmDeltaEnd[ x0 ][ y0 ] =   
     ( 1 − 2 \*dmm\_delta\_end\_sign\_flag[ x0 ][ y0 ] ) \* ( dmm\_delta\_end\_abs\_minus1[ x0 ][ y0 ] + 1) (H‑18)

**edge\_start\_left\_flag**[ x0 ][ y0 ] equal to 0 specifies that the start point of region boundary chain coding is located on the top row boundary of the current block. edge\_start\_left\_flag[ x0 ][ y0 ] equal to 1 specifies that the start point of region boundary chain coding is located on the left column boundary of the current block.

**edge\_start\_position**[ x0 ][ y0 ] specifies the column position of the start point of region boundary chain coding when edge\_start\_left\_flag[ x0 ][ y0 ] is equal to 0 and specifies the row position of the start point of region boundary chain coding when edge\_start\_left\_flag[ x0 ][ y0 ] is equal to 1.

**num\_edge\_codes\_minus1**[ x0 ][ y0 ] +1 specifies the number of edges within the current block.

**edge\_code**[ x0 ][ y0 ][ k ] shall be one of the values shown in . edge\_code[ x0 ][ y0 ][ k ] is used to derive the edge direction of the k-th edge when edge\_intra\_flag[ x0 ][ y0 ] is equal to 1.

Table H‑3 – Interpretation of edge\_code[ x0 ][ y0 ][ k ]

|  |  |
| --- | --- |
| **edge\_code**[ x0 ][ y0 ]**[**k **]** | **edge direction** |
| 0 | 0° |
| 1 | 45° |
| 2 | −45° |
| 3 | 90° |
| 4 | −90° |
| 5 | 135° |
| 6 | −135° |

**dmm\_dc\_flag**[ x0 ][ y0 ] equal to 1 specifies that dmm\_dc\_abs[ x0 ][ y0 ][ i ] and dmm\_dc\_sign\_flag[ x0 ][ y0 ][ i ] are present, dmm\_dc\_flag[ x0 ][ y0 ] equal to 0 specifies that dmm\_dc\_abs[ x0 ][ y0 ][ i ] and dmm\_dc\_sign\_flag[ x0 ][ y0 ][ i ] are not present.

**dmm\_dc\_abs**[ x0][ y0 ][ i ], **dmm\_dc\_sign\_flag**[ x0 ][ y0 ][ i ]are used to derive DcOffset[ x0 ][ y0 ][ i ] as follows:

* 1. DcOffset[ x0 ][ y0 ][ i ] =   
     ( 1 − 2 \*dmm\_dc\_sign\_flag[ x0 ][ y0 ][ i ] ) \* dmm\_dc\_abs[ x0 ][ y0 ][ i ] (H‑19)

**edge\_dc\_flag**[ x0 ][ y0 ] equal to 1 specifies that edge\_dc\_abs[ x0 ][ y0 ][ i ] and edge\_dc\_sign\_flag[ x0 ][ y0 ][ i ] are present, edge\_dc\_flag[ x0 ][ y0 ] equal to 0 specifies that edge\_dc\_abs[ x0 ][ y0 ][ i ] and edge\_dc\_sign\_flag[ x0 ][ y0 ][ i ] are not present.

**edge\_dc\_abs**[ x0][ y0 ][ i ], **edge\_dc\_sign\_flag**[ x0 ][ y0 ][ i ]are used to derive DcOffset[ x0 ][ y0 ][ i ] as follows:

* 1. DcOffset[ x0 ][ y0 ][ i ] =   
     ( 1 − 2 \*edge\_dc\_sign\_flag[ x0 ][ y0 ][ i ] ) \* edge\_dc\_abs[ x0 ][ y0 ][ i ] (H‑20)

**sdc\_residual\_flag**[ x0 ][ y0 ][ i ] equal to 0 specifies that the residual is zero for segment i. sdc\_residual\_flag equal to 1 specifies that the residual is non-zero and the sdc\_residual\_sign\_flag and sdc\_residual\_abs[ i ] syntax elements are present for segment i.

**sdc\_residual\_abs\_minus1**[ x0 ][ y0 ][ i ]and **sdc\_residual\_sign\_flag**[ x0 ][ y0 ][ i ]are used to derive SdcResidual[ x0 ][ y0 ][ i ] for segment i as follows:

* 1. SdcResidual[ x0 ][ y0 ][i] =   
     ( 1 − 2 \*sdc\_residual\_sign\_flag[ x0 ][ y0 ][ i ] ) \* ( sdc\_residual\_mag\_minus1[ x0 ][ y0 ][ i ] + 1) (H‑21)

H.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 H‑4 – Specification of intra prediction mode and associated names

|  |  |
| --- | --- |
| **Intra prediction mode** | **Associated names** |
| 0 | Intra\_Planar |
| 1 | Intra\_DC |
| 2..34 | Intra\_Angular |
| 35…38 | Intra\_DepthPartition (used only for depth) |
| Otherwise (39, 40) | Intra\_Chain (used only for depth) |

[Ed. (GT): Since the Intra\_FromLuma mode has been removed in the HEVC text spec used as base for this document, the dmm mode and intra chain numbers are here decremented by 1 compared to HTM-7.0.]

[Ed. (GT): Consider reducing number of possible IntraPredMode values by using dmm\_dc\_flag and edge\_dc\_flag explicitly. ]

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

* If depth\_intra\_mode[ xB ][ yB ] is equal to INTRA\_DEP\_SDC\_PLANAR, IntraPredMode[ xB ][ yB ] is set equal to Intra\_Planar.
* Otherwise, if depth\_intra\_mode[ xB ][ yB ] is equal to INTRA\_DEP\_SDC\_DMM\_WFULL, IntraPredMode[ xB ][ yB ] is set equal to Intra\_DepthPartition( 35 ).
* Otherwise if sdc\_pred\_mode[ xB ][ yB ] is equal to INTRA\_DEP\_SDC\_DC, IntraPredMode[ xB ][ yB ] is set equal to Intra\_DC.
* Otherwise, if depth\_intra\_mode[ xB ][ yB ] is equal to INTRA\_DEP\_DMM\_WFULL, IntraPredMode[ xB ][ yB ] is set equal to Intra\_DepthPartition( 35 + dmm\_dc\_flag[ xB ][ yB ] ) .
* Otherwise, if depth\_intra\_mode[ xB ][ yB ] is equal to INTRA\_DEP\_DMM\_WPREDDIR, IntraPredMode[ xB ][ yB ] is set equal to Intra\_DepthPartition( 37 + dmm\_dc\_flag[ xB ][ yB ] ) .
* Otherwise if depth\_intra\_mode[ xB ][ yB ] is equal to INTRA\_DEP\_CHAIN, IntraPredMode[ xB ][ yB ] is set equal to Intra\_Chain( 39 + edge\_dc\_flag[ xB ][ yB ] ).
* Otherwise ( depth\_intra\_mode[ xB ][ yB ] is equal to INTRA\_DEP\_NONE ), IntraPredMode[ xB ][ yB ] is derived as the following ordered steps.
  1. The neighbouring locations ( xBA, yBA ) and ( xBB, yBB ) are set equal to ( xB−1, yB ) and ( xB, yB−1 ), respectively.
  2. For N being either replaced A or B, the variables candIntraPredModeN are derived as follows.
     + The availability derivation process for a block in z-scan order as specified in subclause 6.4.1 is invoked with the location ( xCurr, yCurr ) set equal to ( xB, yB ) and the neighbouring location ( xN, yN ) set equal to ( xBN, yBN ) as the input and the output is assigned to availableN.
     + The candidate intra prediction mode candIntraPredModeN is derived as follows.
       - If availableN is equal to FALSE, candIntraPredModeN is set equal to Intra\_DC.
       - Otherwise, if PredMode[ xBN ][ yBN ] is not equal to MODE\_INTRA, candIntraPredModeN is set equal to Intra\_DC,
       - Otherwise, if N is equal to B and yB−1 is less than (( yB >> Log2CtbSizeY ) << Log2CtbSizeY), intraPredModeB is set equal to Intra\_DC.
       - Otherwise, if candIntraPredModeN is larger than 34, candIntraPredModeN is set equal to Intra\_DC.
       - Otherwise, candIntraPredModeN is set equal to IntraPredMode[ xBN ][ yBN ].
  3. The candModeList[ x ] with x=0..2 is derived as follows:
     + If candIntraPredModeB is equal to candIntraPredModeA, the following applies:
       - If candIntraPredModeA is less than 2 (either Intra\_Planar or Intra\_DC), candModeList[ x ] with x=0..2 is derived as:

candModeList[0] = Intra\_Planar (H‑22)  
candModeList[1] = Intra\_DC (H‑23)  
candModeList[2] = Intra\_Angular (26) (H‑24)

* + - * Otherwise, candModeList[ x ] with x=0..2 is derived as:

candModeList[0] = candIntraPredModeA (H‑25)  
candModeList[1] = 2 + ( ( candIntraPredModeA + 29 ) % 32 ) (H‑26)  
candModeList[2] = 2 + ( ( candIntraPredModeA − 2 + 1 ) % 32 ) (H‑27)

* + - Otherwise (candIntraPredModeB is not equal to candIntraPredModeA), the following applies:
      * candModeList[0] and candModeList[1] are derived as follows:

candModeList[0] = candIntraPredModeA (H‑28)  
candModeList[1] = candIntraPredModeB (H‑29)

* + - * If none of candModeList[0] and candModeList[1] is equal to Intra\_Planar, candModeList[2] is set equal to Intra\_Planar,
      * Otherwise, if none of candModeList[0] and candModeList[1] is equal to Intra\_DC, candModeList[2] is set equal to Intra\_DC,
      * Otherwise, candModeList[2] is set equal to Intra\_Angular (26).
  1. IntraPredMode[ xB ][ yB ] is derived by applying the following procedure:
     + If prev\_intra\_luma\_pred\_flag[ xB ][ yB ] is equal to 1, the IntraPredMode[ xB ][ yB ] is set equal to candModeList[ mpm\_idx ].
     + Otherwise IntraPredMode[ xB ][ yB ] is derived by applying the following ordered steps:

1. The array candModeList[x], x=0..2 is modified as the following ordered steps:
   1. When candModeList[0] is greater than candModeList[1], both values are swapped as follows.

( candModeList[0], candModeList[1] ) = Swap( candModeList[0], candModeList[1] ) (H‑30)

* 1. When candModeList[0] is greater than candModeList[2], both values are swapped as follows.

( candModeList[0], candModeList[2] ) = Swap( candModeList[0], candModeList[2] ) (H‑31)

* 1. When candModeList[1] is greater than candModeList[2], both values are swapped as follows.

( candModeList[1], candModeList[2] ) = Swap( candModeList[1], candModeList[2] ) (H‑32)

1. IntraPredMode[xB][yB] is derived as the following ordered steps:
   1. IntraPredMode[ xB ][ yB ] = rem\_intra\_luma\_pred\_mode[ xB ][ yB ]
   2. When IntraPredMode[ xB ][ yB ] is greater than or equal to candModeList[ 0 ], the value of IntraPredMode[ xB ][ yB ] is increased by one
   3. When IntraPredMode[ xB ][ yB ] is greater than or equal to candModeList[ 1 ], the value of IntraPredMode[ xB ][ yB ] is increased by one

When IntraPredMode[ xB ][ yB ] is greater than or equal to candModeList[ 2 ], the value of IntraPredMode[ xB ][ yB ] is increased by one

H.8.4.4.2 Intra sample prediction

H.8.4.4.2.1 General intra sample prediction

The specification in subclause 8.4.4.2.1 with the following paragraphs added to the end of subclause apply:

* Otherwise, if intraPredMode is equal to Intra\_DepthPartition(35,36), the corresponding intra prediction mode specified in subclause H.8.4.4.2.7 is invoked with the location ( xB0, yB0 ), the intra prediction mode intraPredMode, the sample array p and the transform block size nT as the inputs and the output are the predicted sample array predSamples.
* Otherwise, if intraPredMode is equal to Intra\_DepthPartition(37,38), the corresponding intra prediction mode specified in subclause H.8.4.4.2.10 is invoked with the location ( xB0, yB0 ), the intra prediction mode intraPredMode, with the sample array p and the transform block size nT as the inputs and the output are the predicted sample array predSamples.
* Otherwise, if intraPredMode is equal to Intra\_Chain(39,40), the corresponding intra prediction mode specified in subclause 8 is invoked with the location ( xB0, yB0 ), the intra prediction mode intraPredMode, with the sample array p and the transform block size nT as the inputs and the output are the predicted sample array predSamples.

H.8.4.4.2.7 Specification of Intra\_DepthPartition (35, 36) prediction mode

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,
* the 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 =0..nT−1.

This intra prediction mode is invoked when intraPredMode is equal to 35 or 36.

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

* 1. The variable wedgePattern[ x ][ y ] with x, y =0..nT−1, specifying a binary partition pattern is derived as.
     + 1. wedgePattern = WedgePatternTable[ Log2( nT) ][ wedge\_full\_tab\_idx[ xB ][ yB ] ] (H‑33)
  2. The depth partition value derivation and assignment process as specified in subclause H.8.4.4.2.12 is invoked with the neighbouring samples p[ x ][ y ], the binary pattern wedgePattern [ xB ][ yB ], the transform size nT, the dcOffsetAvailFlag set equal to ( intraPredMode = = Intra\_DepthPartition(36) ), intraChainFlag set equal to 0, and the DC Offsets DcOffset[ xB ][ yB ][ 0 ], and DcOffset[ xB ][ yB ][ 1 ] as inputs and the output is assigned to predSamples[ x ][ y ].
  3. For x, y = 0..nT − 1, inclusive the following applies:
     + WedgeIdx[ xB +x ][ yB +y] is set equal to wedge\_full\_tab\_idx[ xB ][ yB ].

H.8.4.4.2.10 Specification of Intra\_DepthPartition (37, 38) prediction mode

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,
* the 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 =0..nT−1.

This intra prediction mode is invoked when intraPredMode is equal to 41 or 42.

Sample location ( xRef, yRef ) specifies the top-left sample of a reference block relative to the top‑left sample of the current picture.

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

* 1. The flag refAvailableFlag specifying whether a neighbouring block is available for prediction is set equal to 0.
  2. The luma location ( xRefA, yRefA ) is set equal ( xB, yB − 1 ) and the luma location ( xRefL, yRefL ) is set equal to the ( xB − 1, yB ).
  3. For N being replace by A and L, the following applies:
     + The availability derivation process for a block in z-scan order as specified in subclause 6.4.1 is invoked with the location of the current block ( xB, yB ) and the location ( xRefN, yRefN) as the input and the output is assigned to flag availableFlagN.
     + When refAvailableFlag is equal to 0, availableFlagN is equal to 1 and PredMode[ xRefN ][ yRefN ] is equal to MODE\_INTRA and IntraPredMode[ xRefN ][ yRefN ] is in the range of 35 to 38 or 41 to 42, the following applies:
       - The flag refAboveFlag is set equal to ( N = = A )
       - The variable refWedgeIdx is set to the Wedgelet pattern index value WedgeIdx[ xRefN ][ yRefN ].
       - The Wedgelet direction derivation process as specified in subclause H.8.4.4.2.10.1 is invoked with sample location ( xB, yB ), the transform size nT, the flag refAboveFlag and the reference Wedgelet index refWedgeIdx as inputs and the output is assigned to refAvailableFlag, refEndPos, refDeltaHor and refDeltaVer.
  4. When refAvailableFlag is equal to 0, the following applies:.
     + The reference partition end position refEndPos and the flag refAboveFlag are derived as specified in the following:

maxSlopeAbove = 0   
 posMaxSlopeAbove = 0  
 maxSlopeLeft = 0  
 posMaxSlopeLeft = 0  
 for( k = 0; k < nT − 1; k++ ) {  
 if( Abs( p[ k+1 ][ −1 ] − p[ k ][ −1 ] ) > maxSlopeAbove ) {  
 maxSlopeAbove = Abs( p[ k+1 ][ −1 ] − p[ k ][ −1 ] )  
 posMaxSlopeAbove = Min( k+1, nT−2 )  
 }  
 if( Abs( p[ −1 ][ k+1 ] ‑ p[ −1 ][ k ] ) > maxSlopeLeft ) {  
 maxSlopeLeft = Abs( p[ −1 ][ k+1 ] ‑ p[ −1 ][ k ] )  
 posMaxSlopeLeft = Min( k+1, nT−2 )  
 }  
 }  
 refEndPos = ( maxSlopeAbove > maxSlopeLeft ) ? posMaxSlopeAbove : posMaxSlopeLeft  
 refAboveFlag = ( maxSlopeAbove > maxSlopeLeft ) ? 1 : 0

* + - When availableFlagA is equal to 1 or availableFlagL is equal to 1, the following applies:
      * The luma position ( x, y ) is set equal to ( availableFlagA ? ( xRefA , yRefA ) : ( xRefL , yRefL ).
      * When PredMode[ x ][ y ] is equal to MODE\_INTRA and IntraPredMode[ x ][ y ] is in the range of 2 to 34, the following applies:
        + The flag refAvailableFlag is set equal to 1.
        + The variable predModeIntra is set equal to IntraPredMode[ x ][ y ].
        + Depending on predModeIntra, the variable intraPredAng is derived as specified in 8-4.
        + Depending on predModeIntra, refAboveFlag and intraPredAng, the variables refDeltaHor and refDeltaVer are derived as specified in Table H‑5.

Table H‑5 – Specification of refDeltaHor, refDeltaVer   
depending on predModeIntra, refAboveFlag, intraPredAng

|  |  |  |  |
| --- | --- | --- | --- |
| **predModeIntra < 18** | **refAboveFlag** | **refDeltaHor** | **refDeltaVer** |
| 1 | 1 | ( intraPredAng > 0 ) ? −32 : 32 | Abs( intraPredAng ) |
| 0 | 32 | − intraPredAng |
| 0 | 1 | − intraPredAng | 32 |
| 0 | Abs( intraPredAng ) | ( intraPredAng>0 ) ? −32 : 32 |

* 1. The variable resShift specifying a shift for wedgelet sampling, the variable wedgeOri specifying a wedgelet orientation. and the xS, yS, xE and yE specifying the horizontal and vertical position of the start and end of the wedgelet partition line are derived as specified in the following:
     + The variable resShift is set as specified in Table H−7 with log2BlkSize equal to Log2( nT )**.**
     + If refAvailableFlag is equal to 1, the wedgelet direction prediction process specified in subclause H.8.4.4.2.10.2 is invoked with sample location ( xB, yB ), the transform block size nT, the reference block direction flag refAboveFlag, reference partition end position refEndPos, and reference partition horizontal and vertical slope refDeltaHor and refDeltaVer as inputs and the output is assigned to wedgeOri, xS, yS, xE and yE.
     + Otherwise (refAvailableFlag is not equal to 1), the variables wedgeOri, xS, yS, xE and yE are set equal to 0.
  2. The Wedgelet pattern generation process as specified in subclause H.8.4.4.2.13.1 is invoked with the transform size nT, the variable, resShift, the Wedgelet orientation identifier wedgeOri, the line start horizontal position xS, the line start vertical position yS, the line end horizontal position xE and the line end vertical position yE as inputs and the output is assigned to wedgePattern[ x ][ y ]. [ Ed. (PM) Software uses wedgelet lookup list for partition derivation. (#2) ]
  3. The depth partition value derivation and assignment process as specified in subclause H.8.4.4.2.12 is invoked with the neighbouring samples p[ x ][ y ], the binary pattern wedgePattern[ x ][ y ], the transform size nT, the dcOffsetAvailFlag set equal to ( intraPredMode = = Intra\_DepthPartition(42) ), intraChainFlag set equal to 0, and the DC Offsets DcOffset[ xB ][ yB ][ 0 ], and DcOffset[ xB ][ yB ][ 1 ] as inputs and the output is assigned to predSamples[ x ][ y ].
  4. The Wedgelet pattern index WedgeIdx[ xB ][ yB ] is derived as specified in the following:
     + For k = 0..NumWedgePattern[ Log2( nT ) ] − 1, inclusive, the following applies:
       - The flags patIdenticalFlag and patInvIdenticalFlag are set equal to 1.
         * For x, y = 0..nT − 1, inclusive the following applies.

If wedgePattern[ x ][ y ] is not equal to WedgePatternTable[ Log2( nT ) ][ k ][ x ][ y ], patIdenticalFlag is set to 0.

Otherwise ( wedgePattern[ x ][ y ] is equal to WedgePatternTable[ Log2( nT ) ][ k ][ x ][ y ] ), patInvIdenticalFlag is set to 0.

* + - * When patIdenticalFlag is equal to 1 or patInvIdenticalFlag is equal to 1, the following applies
        + For x, y = 0..nT − 1, inclusive the following applies:

WedgeIdx[ xB +x ][ yB +y] is set equal to k.

H.8.4.4.2.10.1 Wedgelet direction derivation process

Inputs to this process are:

* a sample location ( xN, yN ) specifying a position in the neighbouring block relative to the top‑left sample of the current picture,
* a variable nT specifying the transform block size,
* a flag refAboveFlag specifying whether the above or left neighbouring block is used for prediction,
* a variable refWedgeIdx specifying the wedgelet pattern list index of the reference block.

Outputs of this process are:

* a flag refValidFlag specifying whether prediction from the reference block is possible or not,
* a variable refEndPos specifying the end position of the reference block partition,
* variables refDeltaHor and refDeltaVer specifying the horizontal and vertical slope of the reference block partition direction.

The flag refValidFlag and the variables refOffset, refEndPos, refDeltaHor and refDeltaVer are set equal to 0.

1. The luma location ( xN, yN ) is set equal to ( refAboveFlag ? ( xB, yB − 1 ) : ( xB − 1, yB ) ).
2. The variable refBlkSize is set equal to ( CbSize[ xN ][ yN ] >> ( ( CtPartMode[ xN ][ yN ] = = PART\_NxN ) ? 1 :0 ) ) and the variable refOffset is derived as specified in the following: .
   1. xRef = CbPosX[ xN ][ yN ] + ( ( xB – CbPosX[ xN ][ yN ] ) / refBlkSize ) \* refBlkSize (H‑40)  
      yRef = CbPosY[ xN ][ yN ] + ( ( yB – CbPosY[ xN ][ yN ] ) / refBlkSize ) \* refBlkSize (H‑41)  
      refOffset = refAboveFlag ? Max( 0, xB – xRef ) : Max( 0, yB – yRef ) (H‑42)

The array refWedgePattern[ x ][ y ] of size (refBlkSize)x(refBlkSize), specifying a binary partition pattern is derived as follows:

* 1. refWedgePattern = WedgePatternTable[ Log2( refBlkSize ) ][ refWedgeIdx ] (H‑43)

Depending of refWedgePattern the value of the variable wedgeOri specifying a wedgelet orientation is derived as specified in Table H‑6. [Ed. (PM): Software uses wedgelet lookup list for deriving orientation.]

Table H‑6 – Specification of wedgeOri depending on refWedgePattern

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **refWedgePattern[ 0 ][ 0 ]** | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| **refWedgePattern[ refBlkSize –1 ][ 0 ]** | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| **refWedgePattern[ 0 ][ refBlkSize –1 ]** | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| **refWedgePattern[ refBlkSize –1 ][ refBlkSize –1 ]** | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| **wedgeOri** | 0 | 1 | 2 | 3 | 4 | | 5 | |

Depending refBlkSize and wedgeOri, the variables xPosS, yPosS, xPosE, yPosE, xIncS, yIncE, xIncE, yIncE are derived as specified in Table H−8, with wBlkSize being equal to refBlkSize.

1. The values of xRefS, yRefS, and xRefE, yRefE are derived as specified in the following:
2. [Ed. (PM): Software uses wedgelet lookup list for deriving start and end position.(#2)]
   * For C being replaced by S and E, the following applies:

for( k = 0; k < refBlkSize − 1; k++ ) {  
 x0 = xPosC + k \* xIncC   
 y0 = yPosC + k \* yIncC   
 x1 = xPosC + (k + 1)\* xIncC   
 y1 = yPosC + (k + 1)\* yIncC   
 if( refWedgePattern[ x0 ][ y0 ] ! = refWedgePattern[ x1 ][ y1 ] ) {  
 xRefC = ( refWedgePattern[ x0 ][ y0 ] = = 1 ) ? x0 : x1  
 yRefC = ( refWedgePattern[ x0 ][ y0 ] = = 1 ) ? y0 : y1  
 }  
 }

1. The value of refValidFlag, refEndPos, refDeltaHor and refDeltaVer are set equal to 0 and the following applies:

if( ( refAboveFlag && ( wedgeOri = = 2 | | wedgeOri = = 3 | | wedgeOri = = 4) ) | |   
 ( !refAboveFlag && ( wedgeOri = = 1 | | wedgeOri = = 2 | | wedgeOri = = 5 ) ) ) {  
 if( ( refAboveFlag && 2 = = wedgeOri) | | ( !refAboveFlag && ( 1 = = wedgeOri | | 5 = = wedgeOri ) ) ) {  
 ( xRefE, xRefS ) = Swap( xRefE, xRefS )  
 ( yRefE, yRefS ) = Swap( yRefE, yRefS )  
 }  
 refDeltaHor = xRefE − xRefS  
 refDeltaVer = yRefE − yRefS  
 refEndPos = ( refAboveFlag ? xRefE : yRefE ) − refOffset  
 if( refEndPos > 0 && refEndPos < ( nT − 1 ) ) {  
 refValidFlag = 1  
 }  
}

H.8.4.4.2.10.2 Wedgelet direction prediction process

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 nT specifying the block size,
* a flag refAboveFlag specifying whether the above or left neighbouring block is used for prediction,
* a variable refEndPos specifying the end position of the reference block partition,
* variables refDeltaHor and refDeltaVer specifying the horizontal and vertical slope of the reference block partition direction.

Outputs of this process are:

* a variable wedgeOri specifying the orientation identifier of the Wedgelet pattern,
* a variable xS specifying the partition line start horizontal position,
* a variable yS specifying the partition line start vertical position,
* a variable xE specifying the partition line end horizontal position,
* a variable yE specifying the partition line end vertical position.

The variable log2BlkSize is set equal to Log2( nT ).

Depending on log2BlkSize, the variable resShift specifying the re-sampling of the wedgelet partition for the current block is derived as specified in Table H−7.

The variable blkMax specifies the maximum value for partition line start and end positions in the resampled domain is derived as follows:

* 1. blkMax = ( ( resShift >= 0 ) ? ( nT << resShift ) : ( nT >> resShift ) ) – 1 (H‑44)

The variable scRefEndPos specifying the reference end position in the resampled domain is derived as follows:

* 1. scRefEndPos = ( resShift >= 0 ) ? ( refEndPos << resShift ) : ( refEndPos >> resShift ) (H‑45)

The variables deltaM, deltaN and offsetEndPos are derived as specified in the following:

* 1. deltaM = refAboveFlag ? refDeltaHor : refDeltaVer (H‑46)  
     deltaN = refAboveFlag ? refDeltaVer : refDeltaHor (H‑47)  
     offsetEndPos = DmmDeltaEnd[ xB ][ yB ] \* ( refAboveFlag ? 1 : –1 ) (H‑48)

The variables mS and nS are derived as specified in the following:

nS = 0  
mS = scRefEndPos  
if( abs( deltaM ) >= abs( deltaN ) && deltaM ! = 0 )   
 mS + = ( deltaM > 0 ) ? 1 : –1

Depending on deltaN, the variables wedgeOri, mE and nE are derived as specified in the following:

* + When deltaN is equal to 0, the following applies:

mE = ( deltaM < 0 ) ? 0 : blkMax  
 nE = Min( Max( offsetEndPos \* ( ( deltaM < 0 ) ? 1 : –1 ), 0 ), blkMax )  
 wedgeOri = ( deltaM < 0 ) ? 0 : ( refAboveFlag ? 1 : 3 )

* + When deltaN is greater than 0, the following applies:

mVirtualEnd = mS + Round( ( blkMax \* deltaM )  deltaN )  
 if( mVirtualEnd < 0 ) {  
 iNe = Round( ( –mS \* deltaN )  deltaM ) + offsetEndPos  
 nE = Clip3( 0, blkMax, iNe )  
 mE = ( iNe > blkMax ) ? Min( ( iNe – blkMax ), blkMax ) : 0  
 wedgeOri = ( iNe > blkMax ) ? ( refAboveFlag ? 4 : 5 ) : 0  
 } else if( mVirtualEnd > blkMax ) {  
 iNe = Round( ( ( blkMax – mS ) \* deltaN )  deltaM ) – offsetEndPos  
 nE = Clip3( 0, blkMax, iNe )  
 mE = ( iNe > blkMax ) ? Max( ( blkMax – ( iNe – blkMax ) ), 0 ) : blkMax  
 wedgeOri = ( iNe > blkMax ) ? ( refAboveFlag ? 4 : 5 ) : ( ( refAboveFlag ) ? 1 : 3 )  
 } else {   
 iMe = mVirtualEnd + offsetEndPos  
 mE = Clip3( 0, blkMax, iMe )  
 if( iMe < 0 ) {  
 nE = Max( ( blkMax + iMe ), 0 )  
 wedgeOri = 0   
 }else if( iMe > blkMax ) {  
 nE = Max( ( blkMax – ( iMe – blkMax ) ), 0 )  
 wedgeOri = ( refAboveFlag ? 1 : 3 )  
 }else {  
 nE = blkMax  
 wedgeOri = ( refAboveFlag ? 4 : 5 )  
 }  
 }

Depending on refAboveFlag and wedgeOri, the variables mS, nS, mE and nE are modified as specified in the following:

if( ( refAboveFlag && wedgeOri = = 1 ) | | ( !refAboveFlag && ( wedgeOri = = 0 | | wedgeOri = = 5 ) ) ) {  
 ( mE, mS ) = Swap( mE, mS )  
 ( nE, nS ) = Swap( nE, nS )  
}

Depending on refAboveFlag, the variables yS, yS, yE and yE are derived as specified in the following:

* 1. ( xS, yS ) = refAboveFlag ? ( mS, nS ) : ( nS, mS ) (H‑49)  
     ( xE, yE ) = refAboveFlag ? ( mE, nE ) : ( nE, mE ) (H‑50)

H.8.4.4.2.11 Specification of Intra\_Chain (39, 40) prediction mode

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,
* the 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 = 0..nT−1.

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

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

* 1. [Ed. (GT) The process of edge reconstruction (items 1 - 3) need to be specified more in detail. In current state the specification is not precise enough. (#3)]
  2. The start point of the chain in the current prediction unit is derived from edge\_start\_left\_flag[ xB ][ yB ] and edge\_start\_position[ xB ][ yB ].
* If edge\_start\_left\_flag[ xB ][ yB ] is equal to 1, the start point is set as s[ 0 ][ yS ] where yS is equal to edge\_start\_position[ xB ][ yB ].
* Otherwise, the start point is set as s[ xS ][ 0 ] where xS is equal to edge\_start\_position[ xB ][ yB ].
  1. The number of edges and the direction of each edge are derived from num\_edge\_codes\_minus1[ xB ][ yB ] and edge\_code[ xB ][ yB ][ k ], for k = 0.. num\_edge\_codes\_minus1[ xB ][ yB ]. The direction of the edge is derived from edge\_code as shown in .
  2. The region boundary generated by connecting each edge separates the predicted samples predSamples[ x ][ y ], with x, y = 0..nT−1 into two regions: the region rA that covers the top-left pixel (x0, y0) and the region rB that covers the remaining region.

For x, y = 0..nT−1 the binary pattern edgePattern[ x ][ y ] is derived as follows:

* If (x, y) is covered by rA, edgePattern[ x ][ y ] = 0.
* Otherwise, edgePattern[ x ][ y ] = 1.
  1. The depth partition value derivation and assignment process as specified in subclause H.8.4.4.2.12 is invoked with the neighbouring samples p[ x ][ y ], the binary pattern edgePattern[ x ][ y ], the transform size nT, the dcOffsetAvailFlag set equal to ( intraPredMode = = Intra\_DepthPartition(44) ), intraChainFlag set equal to 1, and the DC offsets DcOffset[ xB ][ yB ][ 0 ], and DcOffset[ xB ][ yB ][ 1 ] as inputs and the output is assigned to predSamples[ x ][ y ].

Table H‑12 – Association of ctxIdx and syntax elements for each initializationType in the initialization process

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Syntax element** | **ctxIdxTable** | **initType** | | |
| **0** | **1** | **2** |
| coding\_unit( ) depth\_mode\_parameters( ) | depth\_intra\_mode |  | 0..7 | 8..15 | 16..23 |
| wedge\_full\_tab\_idx |  | 0 | 1 | 2 |
|  |  |  |  |  |
| dmm\_delta\_end\_flag dmm\_delta\_end\_abs\_minus1 |  | 0 | 1 | 2 |
| dmm\_dc\_flag |  | 0 | 1 | 2 |
| dmm\_dc\_abs |  | 0 | 1 | 2 |
| edge\_code |  | 0 | 1 | 2 |
| edge\_dc\_flag |  | 0 | 1 | 2 |
| edge\_dc\_abs |  | 0 | 1 | 2 |

Table H‑13 – Values of variable initValue for wedge\_full\_tab\_idx ctxIdx

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **wedge\_full\_tab\_idx** | | |
| **0** | **1** | **2** |
| **initValue** | 154 | 154 | 154 |



Table H‑15– Values of variable initValue for dmm\_delta\_end\_flag and dmm\_delta\_end\_abs\_minus1 ctxIdx

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **dmm\_delta\_end\_flag dmm\_delta\_end\_abs\_minus1** | | |
| **0** | **1** | **2** |
| **initValue** | 154 | 154 | 154 |

Table H‑16 – Values of variable initValue for dmm\_dc\_abs ctxIdx

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **dmm\_dc\_abs** | | |
| **0** | **1** | **2** |
| **initValue** | 154 | 154 | 154 |

Table H‑17 – Values of variable initValue for edge\_code ctxIdx

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **edge\_code** | | |
| **0** | **1** | **2** |
| **initValue** | 154 | 154 | 154 |

Table H‑22 – Values of variable initValue for depth\_intra\_mode ctxIdx

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **depth\_intra\_mode** | | | | | | | |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **initValue** | 0 | 0 | 64 | 0 | 154 | 0 | 154 | 0 |
|  | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| **initValue** | 0 | 64 | 0 | 154 | 0 | 154 | 0 | 0 |
|  | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** |
| **initValue** | 64 | 0 | 154 | 0 | 154 | 0 | 0 | 0 |

Table H‑23 – Values of variable initValue for dmm\_dc\_flag ctxIdx

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **dmm\_dc\_flag** | | |
| **0** | **1** | **2** |
| **initValue** | 0 | 0 | 64 |

Table H‑24 – Values of variable initValue for edge\_dc\_flag ctxIdx

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **edge\_dc\_flag** | | |
| **0** | **1** | **2** |
| **initValue** | 154 | 154 | 154 |

Table H‑25 – Values of variable initValue for edge\_dc\_abs ctxIdx

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **edge\_dc\_abs** | | |
| **0** | **1** | **2** |
| **initValue** | 154 | 154 | 154 |

Table H‑26 – Syntax elements and associated types of binarization, maxBinIdxCtx, ctxIdxTable, and ctxIdxOffset

| **Syntax element** | **initType** | **Type of binarization** | **maxBinIdxCtx** | **ctxIdxTable** | **ctxIdxOffset** |
| --- | --- | --- | --- | --- | --- |
| wedge\_full\_tab\_idx | 0 | FL, cMax = wedgeFullTabIdxBits[ log2PbSize ] (defined in ) | 0 |  | 0 |
| 1 | 0 |  | 1 |
| 2 | 0 |  | 2 |
|  |  |  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| dmm\_delta\_end\_flag | 0 | FL, cMax = 1 | 0 |  | 0 |
| 1 | 0 |  | 1 |
| 2 | 0 |  | 2 |
| dmm\_delta\_end\_abs\_minus1 | 0 | FL, cMax = 3 | 0 |  | 0 |
| 1 | 0 |  | 1 |
| 2 | 0 |  | 2 |
| dmm\_delta\_end\_sign\_flag | na | FL, cMax = 1 | na | na | na, (Bypass) |
| dmm\_dc\_abs | 0 | UEG0 [Ed. (GT) To be specified] | 0 |  | 0 |
| 1 | 0 |  | 1 |
| 2 | 0 |  | 2 |
| dmm\_dc\_sign\_flag | na | FL, cMax = 1 | na | na | na, (Bypass) |
| edge\_code | 0 | TU, cMax = 6 | 0 |  | 0 |
| 1 | 0 |  | 1 |
| 2 | 0 |  | 2 |
| edge\_dc\_flag | 0 | FL, cMax = 1 |  |  | 0 |
| 1 |  |  | 1 |
| 2 |  |  | 2 |
| edge\_dc\_abs | 0 | UEG0 [Ed. (GT) To be specified] | 0 |  | 0 |
| 1 | 0 |  | 1 |
| 2 | 0 |  | 2 |
| edge\_dc\_sign\_flag | na | FL, cMax = 1 | na | na | na, (Bypass) |
| depth\_intra\_mode | 0 | ( specified in subclause ) | 0 |  | 0 |
| 1 | 2 |  | 8 |
| 2 | 2 |  | 16 |
| dmm\_dc\_flag | 0 | FL, cMax = 1 | 0 |  | 0 |
| 1 | 0 |  | 1 |
| 2 | 0 |  | 2 |

Table H‑27 –Values of wedgeFullTabIdxBits[ log2PUSize ]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Initialization variable** | wedgeFullTabIdxBits | | | | |
| **log2PbSize** | 2 | 3 | 4 | 5 | 6 |
| **Value** | 7 | 10 | 11 | 11 | 13 |



Table ‑29 –Binarization for depth\_intra\_mode

|  |  |  |  |
| --- | --- | --- | --- |
| **Name of depth\_intra\_mode** | **Bin String** | | |
| ( cLog2CbSize = =  3 &&  PartMode[ xC ][ yC ] = = PART\_2Nx2N ) | |   ( cLog2CbSize > 3 && cLog2CbSize < 6 ) | cLog2CbSize = = 3 &&  PartMode[ xC ][ yC ] = = PART\_NxN | cLog2CbSize = = 6 |
| INTRA\_DEP\_SDC\_PLANAR | 00 | - | 0 |
| INTRA\_DEP\_NONE | 010 | 0 | 10 |
| INTRA\_DEP\_SDC\_DMM\_WFULL | 011 | - | - |
| INTRA\_DEP\_DMM\_WFULL | 100 | 10 | - |
|  |  |  |  |
|  |  |  |  |
| INTRA\_DEP\_SDC\_DC | 101 | - | 11 |
| INTRA\_DEP\_DMM\_WPREDDIR) | 110 | - | - |
| INTRA\_DEP\_CHAIN | 111 | 11 | - |

Table H‑30 –Assignment of ctxIdxInc to syntax elements with context coded bins

| **Syntax element** | **binIdx** | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **>=5** |
| wedge\_full\_tab\_idx | 0 | 0 | 0 | 0 | 0 | 0 |
| dmm\_delta\_end\_flag | 0 | na | na | na | na | na |
| dmm\_delta\_end\_abs\_minus1 | 0 | 0 | 0 | 0 | 0 | 0 |
| dmm\_dc\_flag | 0 | na | na | na | na | na |
| dmm\_dc\_abs | 0 | 0 | 0 | 0 | 0 | 0 |
| dmm\_dc\_sign\_flag | bypass | 0 | 0 | 0 | 0 | 0 |
| edge\_dc\_flag | 0 | na | na | na | na | na |
| edge\_dc\_abs | 0 | 0 | 0 | 0 | 0 | 0 |
| edge\_dc\_sign\_flag | bypass | 0 | 0 | 0 | 0 | 0 |
| edge\_code | 0 | 0 | 0 | 0 | 0 | 0 |