#### Adaptive loop filter info syntax

|  |  |
| --- | --- |
| alf\_info( rx, ry, cIdx ) { | **Descriptor** |
| if(NumALFFiltersInStoredBuffer[cIdx] > 0) |  |
| **alf\_new\_filter\_set\_flag** | u(1) |
| if( alf\_new\_filter\_set\_flag = = 0 && NumALFFiltersInStoredBuffer[cIdx] > 0) |  |
| **alf\_stored\_filter\_set\_idx**[ cIdx ] | u(v) |
| else { |  |
| if( cIdx = = 0 ) { |  |
| **alf\_no\_filters\_minus1** | ue(v) |
| if( alf\_no\_filters\_minus1 = = 1 ) |  |
| **alf\_start\_second\_filter** | ue(v) |
| else if( alf\_no\_filters\_minus1 > 1 ) |  |
| for( i = 1; i < 15; i++ ) |  |
| **alf\_filter\_pattern\_flag[ cIdx ][ ry ][ rx ][ i ]** | u(1) |
| if( alf\_no\_filters\_minus1 > 0 ) |  |
| **alf\_pred\_flag[ cIdx ][ ry ][ rx ]** | u(1) |
| for( i = 0; i < AlfNumFilters; i++ ) |  |
| **alf\_nb\_pred\_luma\_flag[ cIdx ][ ry ][ rx ][ i ]** | u(1) |
| if( AlfNumFilters > 1 ) { |  |
| **alf\_min\_kstart\_minus1** | ue(v) |
| for( i = 1; i < 4; i++ ) |  |
| **alf\_golomb\_index\_flag[ i ]** | u(1) |
| } |  |
| for( i = 0; i < AlfNumFilters; i++ ) |  |
| for( j = 0; j < AlfCodedLength; j++ ) |  |
| **alf\_filt\_coeff[ cIdx ][ ry ][ rx ][ i ][ j ]** | ge(v) |
| } else |  |
| **alf\_chroma\_coeff\_from\_luma\_flag[ cIdx ][ ry ][ rx ]** | u(1) |
| if( alf\_chroma\_coeff\_from\_luma\_flag[ cIdx ][ ry ][ rx ] = = 0 ) { |  |
| for( j = 0; j < AlfCodedLength; j++ ) |  |
| **alf\_filt\_coeff[ cIdx ][ ry ][ rx ][ 0 ][ j ]** | se(v) |
| } |  |
| } |  |
| } |  |

#### 

#### Adaptive loop filter info semantics

**alf\_new\_filter\_set\_flag** equal to 1 specifies that the current coding treeblock uses a new filter set; equal to 0 specifies that the current coding treeblock uses the stored filter set with the buffer index equal to alf\_stored\_filter\_set\_idx[ cIdx ] of the colour component cIdx.

When alf\_new\_filter\_set\_flag is not present, it is inferred to be 1.

When alf\_new\_filter\_set\_flag is equal to 1, NumALFFiltersInStoredBuffer[cIdx] is increased by 1.

**… …**

**alf\_filt\_coeff**[ cIdx ][ ry ][ rx ][ i ][ j ] specifies the j-th filter coefficient of i-th filter used in the adaptive loop filtering process for the current coding treeblock at location ( rx, ry ) for the colour component cIdx.

**alf\_chroma\_coeff\_from\_luma\_flag**[ cIdx ][ ry ][ rx ] specifies whether the filter coefficients of the first filter of the luma component shall be used for the colour component cIdx at treeblock location ( rx, ry ). When alf\_lcu\_enable\_flag[ 0 ][ ry ][ rx ] is equal to zero, alf\_chroma\_coeff\_from\_luma[ cIdx ][ ry ][ rx ] shall be equal to zero.



##### Derivation process for chroma filter coefficients

Inputs to this process are

* + a variable cIdx specifying colour component index,
  + a variable alfPrecisionBit specifying the ALF coefficient precision

Outputs of this process are filter coefficients cc for the samples for the colour component cIdx.

The chroma filter coefficients cC with elements cC[ i ], i = 0..9 is derived as follows:

* If alf\_chroma\_coeff\_from\_luma[ cIdx ][ ry ][ rx ] is equal to 1,

cC[ i ] = cL[ 0 ][ i ] with j = 0..18

* Otherwise (alf\_chroma\_coeff\_from\_luma[ cIdx ][ ry ][ rx ] is equal to zero),
  + If i is equal to 9, the coefficient cC[i] is derived as

sum = Σj( alf\_filt\_coeff [ cIdx ][ ry ][ rx ][ 0 ] [ j ] << 1 ) (8‑370)  
 with j = 0..8

cC[ i ] = ( 1 << alfPrecisionBit ) − ( sum – alf\_filt\_coeff [ cIdx ][ ry ][ rx ][ 0 ][ i ] ) (8‑371)

* + Otherwise (i is less than 9), considering the symmetry of the filter, the chroma filter coefficients cC with elements cC[ i ], i = 0..18 are derived as follows:

cC[ i ] = alf\_filt\_coeff [ cIdx ][ ry ][ rx ][ 0 ] [ i ] (8‑372)

cC[ i ][ 18 ] = cC[ i ][9 ] (8‑373)

cC[ i ][ 17 – j ] = cC[ i ][ j ] (8‑374)  
 with j = 0..8