|  |  |
| --- | --- |
| **Joint Collaborative Team on Video Coding (JCT-VC)**  **of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11**  8th Meeting: San José, CA, USA, 1–10 February, 2012 | Document: JCTVC-H0521r1 |

|  |  |  |  |
| --- | --- | --- | --- |
| *Title:* | **Flexible tile dependency and loop filtering control** | | |
| *Status:* | Input Document to JCT-VC | | |
| *Purpose:* | Proposal | | |
| *Author(s) or Contact(s):* | Ye-Kui Wang Jaehong Chon Ying Chen In Suk Chong Muhammed Coban Marta Karczewicz  5775 Morehouse Drive San Diego, CA 92121 USA | Tel: Email: | 1-858-651-8345 [yekuiw@qualcomm.com](mailto:yekuiw@qualcomm.com) 1-858-651-1605 [jchon@qualcomm.com](mailto:jchon@qualcomm.com) 1-858-845-6589 [cheny@qualcomm.com](mailto:cheny@qualcomm.com) 1-858-845-5477 [ichong@qualcomm.com](mailto:ichong@qualcomm.com) 1-858-658-3973 [mcoban@qualcomm.com](mailto:mcoban@qualcomm.com) 1-858-658-5673 [martak@qualcomm.com](mailto:martak@qualcomm.com) |
| *Source:* | Qualcomm Incorporated | | |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Abstract

This document reproposes the proposal in JCTVC-G317, with some changes. It is proposed to enable the control of in-picture prediction and loop filtering between tiles individually for each vertical or horizontal tile boundary. In-picture prediction includes pixel value prediction, motion prediction, coding mode prediction, and entropy coding context prediction. Loop filtering operations include deblocking filtering, adaptive loop filtering, and sample adaptive offset.

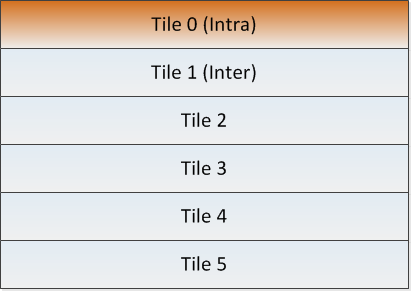
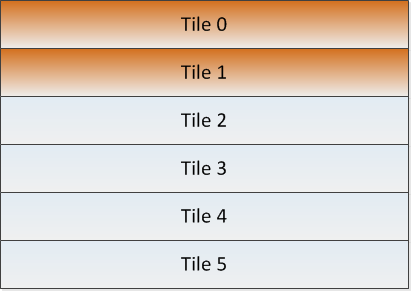
# Introduction

The decoding order of tiles is the tile raster scan order. Each tile contains a rectangle region covered by an integer number of LCUs. Inside each tile, the decoding order of LCUs is the LCU raster scan order.

HEVC WD5 includes three types of loop filtering operations, namely deblocking filtering, adaptive loop filtering (ALF) and sample adaptive offset (SAO). First, deblocking filtering removes blocking artifacts around 8x8 blocks of reconstructed pixels. Then sample adaptive offset is applied on top of deblocked pixels, which compensates DC offset of each pixel depending on different types of pixel classifications. Finally adaptive loop filter is applied to each pixel which is linear filtering of pixels.

In the existing tiles design, in-picture prediction, including pixel value prediction, motion prediction, coding mode prediction, and entropy coding context prediction, across all tile boundaries is controlled by the flag tile\_boundary\_independence\_idc, while all three types of loop filtering operations are performed across tile boundaries. However, in some scenarios, it is ideal to code one or more regions covered by different tiles completely independently, while tiles inside each region may be predicted from each other. One such scenario is described below.

In this scenario, as shown in Fig.1, a sequence of pictures is each evenly partitioned into 6 tiles by 7 horizontal tile boundaries (including the 2 horizontal picture boundaries), with the top tile being tile 0, and the tile immediately below tile 0 being tile 1, and so on. Each of these pictures contain at least one P slice (so there are pictures before the sequence of pictures in decoding order in the entire coded bitstream), and the decoding order is the same as the output order. In picture 0 (i.e., the first picture) in the sequence of pictures, all LCUs in tile 0 are intra coded, and all LCUs in other tiles are inter coded. In picture 1 in the sequence of pictures, all LCUs in tile 1 are intra coded, and all LCUs in other tiles are inter coded. And so on. In other words, in picture N in the sequence of pictures, all LCUs in tile N%6 are intra coded, and all LCUs in other tiles are inter coded, for any value of N in the range of 0 to the number of pictures in the sequence of pictures minus one, inclusive. Therefore, each picture with an index value N for which N%6 is equal to 0 can be used as a random access point, in the sense that if the decoding starts from the picture, except for the initial 5 pictures that cannot be fully correctly decoded, all pictures afterwards can be correctly decoded.

**Figure 1 An example of gradual decoding refresh using tiles**

In the above scenario, in picture 2 and any picture with an index value N for which N%6 is equal to 2 (as shown in the top-right picture in Fig. 1), it is ideal to disallow in-picture prediction as well as loop filtering across the tile boundary between tile 2 and tile 3, i.e., the boundary between the refreshed area (the area covered by tiles 0 and 1) and the un-refreshed area (the area covered by tiles 2 to 5, incusive). Generally, in picture N, it is idea to disallow in-picture prediction as well as loop filtering across the tile boundary between tile N%6 and tile N%6 + 1, and to allow in-picture prediction as well as loop filtering across other tile boundaries. This way, a clean and efficient gradual decoding refresh or gradual random access functionality can be provided.

# Proposal

## Syntax and semantics

The changed syntax and semantics are provided below. Syntax changes are highlighted, and only semantics for new syntax elements or existing syntax elements for which the semantics have been changed are provided.

|  |  |
| --- | --- |
| seq\_parameter\_set\_rbsp( ) { | Descriptor |
| **...** |  |
| tile\_info( ) |  |
| rbsp\_trailing\_bits( ) |  |
| } |  |

|  |  |
| --- | --- |
| pic\_parameter\_set\_rbsp( ) { | Descriptor |
| **...** |  |
| **tile\_info\_present\_flag** | u(1) |
| if( tile\_info\_present\_flag ) |  |
| tile\_info( ) |  |
| rbsp\_trailing\_bits( ) |  |
| } |  |

|  |  |
| --- | --- |
| tile\_info( ) { | Descriptor |
| **num\_tile\_columns\_minus1** | ue(v) |
| **num\_tile\_rows\_minus1** | ue(v) |
| if( num\_tile\_columns\_minus1 != 0 | | num\_tile\_rows\_minus1 != 0 ) { |  |
| **tile\_boundary\_independence\_idc** | ue(v) |
| **tile\_boundary\_loop\_filtering\_idc** | ue(v) |
| **uniform\_spacing\_flag** | u(1) |
| if( !uniform\_spacing\_flag ) { |  |
| for( i = 0; i < num\_tile\_columns\_minus1; i++ ) |  |
| **column\_width**[ i ] | ue(v) |
| for( i = 0; i < num\_tile\_rows\_minus1; i++ ) |  |
| **row\_height**[ i ] | ue(v) |
| } |  |
| for( i = 0; i < num\_tile\_columns\_minus1; i++ ) { |  |
| if( tile\_boundary\_independence\_idc = = 2 ) |  |
| **vertical\_tile\_boundary\_independence\_flag[** i **]** | u(1) |
| if( tile\_boundary\_loop\_filtering\_idc = = 2 ) |  |
| **vertical\_tile\_boundary\_loop\_filering\_flag[** i **]** | u(1) |
| } |  |
| for( i = 0; i < num\_tile\_rows\_minus1; i++ ) { |  |
| if( tile\_boundary\_independence\_idc = = 2 ) |  |
| **horizontal\_tile\_boundary\_independence\_flag[** i **]** | u(1) |
| if( tile\_boundary\_loop\_filtering\_idc = = 2 ) |  |
| **horizontal\_tile\_boundary\_loop\_filering\_flag[** i **]** | u(1) |
| } |  |
| } |  |
| } |  |

**tile\_boundary\_independence\_idc** equal to 0 specifies that in-picture prediction, including pixel value prediction, motion prediction, coding mode prediction, and entropy coding context prediction, is allowed across all tile boundaries. tile\_boundary\_independence\_idc equal to 1 specifies that in-picture prediction is disallowed across all tile boundaries. tile\_boundary\_independence\_idc equal to 2 indicates that in-picture prediction across tile boundaries is specified by the syntax elements vertical\_tile\_boundary\_independence\_flag[ i ] and horizontal\_tile\_boundary\_independence\_flag[ i ].

**tile\_boundary\_loop\_filtering\_idc** equal to 0 specifies that loop filtering operations, including deblocking loop filtering, ALF, and SAO, are allowed across all tile boundaries. tile\_boundary\_loop\_filtering\_idc equal to 1 specifies that loop filtering operations, including deblocking loop filtering, ALF, and SAO, are disallowed across all tile boundaries. tile\_boundary\_loop\_filtering\_idc equal to 2 indicates that the allowance of loop filtering operations, including deblocking loop filtering, ALF, and SAO, is specified by the syntax elements vertical\_tile\_boundary\_loop\_filtering\_flag[ i ] and horizontal\_tile\_boundary\_loop\_filtering\_flag[ i ].

**uniform\_spacing\_flag** has the same semantics as uniform\_spacing\_idc in JCTVC-F335, and replaces uniform\_spacing\_idc in JCTVC-F335.

**vertical\_tile\_boundary\_independence\_flag[** i **]** equal to 0 specifies that in-picture prediction is allowed across the vertical tile boundary with index value equal to i plus 1. The vertical tile boundary index is 0 for the left vertical picture boundary and counted from left to right, increased by 1 for each vertical tile boundary. vertical\_tile\_boundary\_independence\_flag[ i ] equal to 1 specifies that in-picture prediction is disallowed across the vertical tile boundary with index value equal to i plus 1.

**vertical\_tile\_boundary\_loop\_filtering\_flag[** i **]** equal to 0 specifies that loop filtering operations, including deblocking loop filtering, ALF, and SAO, are allowed across the vertical tile boundary with index value equal to i plus 1. The vertical tile boundary index is 0 for the left vertical picture boundary and counted from left to right, increased by 1 for each vertical tile boundary. vertical\_tile\_boundary\_loop\_filtering\_flag[ i ] equal to 1 specifies that loop filtering operations, including deblocking loop filtering, ALF, and SAO, are disallowed across the vertical tile boundary with index value equal to i plus 1.

**horizontal\_tile\_boundary\_independence\_flag[** i **]** equal to 0 specifies that in-picture prediction is allowed across the horizontal tile boundary with index value equal to i plus 1. The horizontal tile boundary index is 0 for the upper horizontal picture boundary and counted from top to bottom, increased by 1 for each horizontal tile boundary. horizontal\_tile\_boundary\_independence\_flag[ i ] equal to 1 specifies that in-picture prediction is disallowed across the horizontal tile boundary with index value equal to i plus 1.

**horizontal\_tile\_boundary\_loop\_filtering\_flag[** i **]** equal to 0 specifies that loop filtering operations, including deblocking loop filtering, ALF, and SAO, are allowed across the horizontal tile boundary with index value equal to i plus 1. horizontal\_tile\_boundary\_loop\_filtering\_flag[ i ] equal to 1 specifies that loop filtering operations, including deblocking loop filtering, ALF, and SAO, are disallowed across the horizontal tile boundary with index value equal to i plus 1.

## Derivation process for the availability of treeblock addresses

Input to this process is a treeblock address tbAddr.

Output of this process is the availability of the treeblock tbAddr.

NOTE – The meaning of availability is determined when this process is invoked.

The treeblock is marked as available, unless one of the following conditions is true in which case the treeblock shall be marked as not available:

– tbAddr < 0

– tbAddr > CurrTbAddr

– the treeblock with address tbAddr belongs to a different slice than the slice containing the treeblock with address CurrTbAddr

– the treeblock with address tbAddr belongs to a different tile than the treeblock with address CurrTbAddr, and tile\_boundary\_independence\_idc is equal to 1

– the treeblock with address tbAddr belongs to a tile to the left of the tile containing the treeblock with address CurrTbAddr, and tile\_boundary\_independence\_idc is equal to 2, and the vertical\_tile\_boundary\_independence\_flag[ i ] associated with the left-side tile boundary of the tile containing the treeblock with address CurrTbAddr is equal to 1

– the treeblock with address tbAddr belongs to a tile above the tile containing the treeblock with address CurrTbAddr, and tile\_boundary\_independence\_idc is equal to 2, and the horizontal\_tile\_boundary\_independence\_flag[ i ] associated with the upper tile boundary of the tile containing the treeblock with address CurrTbAddr is equal to 1

## Loop filtering

When horizontal\_tile\_boundary\_loop\_filtering\_flag and vertical\_tile\_boundary\_loop\_filtering\_flag are equal to 1, normal filtering operations are done.

If horizontal\_tile\_boundary\_loop\_filtering\_flag or vertical\_tile\_boundary\_loop\_filtering\_flag is equal to 0, deblocking, SAO, and ALF is disabled across the horizontal or vertical boundary. For ALF operation near the boundary, access to the pixels across the boundary is needed, which is substituted with padded pixels. In this case, the visual quality degradation is expected across the boundary pixels. Therefore alternative ways of ALF filtering operation across the boundary are needed.

In this case, we propose to use partial filtering in those horizontal or vertical boundaries (including slice boundaries). Partial filtering is already proposed to be used in the virtual boundaries and shown to remove possible visual artifacts around boundaries (see JCTVC-G212). It uses only subset of filter coefficients whose corresponding pixels are available as explained in Fig. 2 and Fig. 3.



**Figure 2 Partial filters for horizontal boundaries**



**Figure 3 Partial filters for vertical boundaries**

Note that symmetry of filter is considered and unused filter coefficients are added to the center position filter coefficients for renormalization. Also for different filter shapes, different strategy can be used adaptively (see JCTVC-G212).

### Related WD text changes

#### 8.6.3.4 Filtering process for luma samples

Inputs of this process are:

– a luma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture,

– a variable log2CUSize specifying the size of the current coding unit,

– a filter index array of (nS)x(nS), fIdx.

Output of this process is the filtered reconstruction of luma picture.

The boundary padding process specified in subclause 8.6.3.1 is invoked with the luma location ( xC, yC ), the size of coding unit log2CUSize and the chroma component index cIdx set equal to 0, and the output is assigned to the luma sample array s’’. [Ed. (WJ): s’’ is now a picture-size array, but actually CU size + appropriate margin is enough]

A variable nS is set equal to ( 1 << log2CUSize ) and a variable alfTap is set equal to ( alf\_length\_luma\_minus\_5\_div2 << 1 ) + 5.

If part of the position (xC+x+horPos[i], yC+y+verPos[i]) or (xC+x-horPos[i], yC+y-verPos[i]) with i=0,...,N-1 are outside of current slice or tile, cc[i] is modified as follows.

1. A set R(k) with k=0,...,K-1, is defined as the set of indexes where (xC+x+horPos[i], yC+y+verPos[i]) or (xC+x-horPos[i], yC+y-verPos[i]) with i=0,...,N-1, are outside of current slice or tile.
2. With R(k) with k=0,...,K-1, cL[i] with i=0,...,N-1, is modified as follows.

where

N = AlfNumCoeffLuma – 1, (8‑478)

horPos[i] and verPos[i] are specified in Table 8‑14 and Table 8‑15, respectively.

Each sample of luma picture recFiltPictureL[ xC + x ][ yC + y ] with x, y = 0..(nS)-1, is derived as follows.

 (8‑476)

**8.6.3.5 Filtering process for chroma samples**

Inputs of this process are:

– a chroma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top left chroma sample of the current picture,

– a variable log2CUSize specifying the size of the current coding unit.

– a variable cIdx specifying the chroma component index.

Output of this process is the filtered reconstruction of chroma picture.

The boundary padding process specified in subclause 8.6.3.1 is invoked with the chroma location ( xC, yC ), the size of coding unit log2CUSize and the chroma component index cIdx, and the output is assigned to the luma sample array s’’. [Ed. (WJ): s’’ is now a picture-size array, but actually CU size + appropriate margin is enough]

A variable nS is set equal to ( 1 << log2CUSize ) and a variable alfTapChroma is set equal to ( alf\_length\_chroma\_minus\_5\_div2 << 1 ) + 5.

If part of the position (xC+x+horPos[i], yC+y+verPos[i]) or (xC+x-horPos[i], yC+y-verPos[i]) with i=0,...,N-1 are outside of current slice or tile, cc[i] is modified as follows.

1. A set R(k) with k=0,...,K-1, is defined as the set of indexes where (xC+x+horPos[i], yC+y+verPos[i]) or (xC+x-horPos[i], yC+y-verPos[i]) with i=0,...,N-1, are outside of current slice or tile.
2. With R(k) with k=0,...,K-1, cc[i] with i=0,...,N-1, is modified as follows.

where

N = AlfNumCoeffChroma – 1, (8‑478)

horPos[ i ] = ( i %  alfTapChroma ) – ( alfTapChroma >> 1 ), and (8‑479)

verPos[ i ] = ( i /  alfTapChroma ) – ( alfTapChroma >> 1 ) (8‑480)

Filtered samples of chroma picture recFiltPicture[ xC + x ][ yC + y ] with x, y = 0..(nS)-1, are derived as follows.

 (8‑477)

# Patent rights declaration(s)

**Qualcomm Incorporated may have current or pending patent rights relating to the technology described in this contribution and, conditioned on reciprocity, is prepared to grant licenses under reasonable and non-discriminatory terms as necessary for implementation of the resulting ITU-T Recommendation | ISO/IEC International Standard (per box 2 of the ITU-T/ITU-R/ISO/IEC patent statement and licensing declaration form).**