|  |  |
| --- | --- |
| **Joint Collaborative Team on Video Coding (JCT-VC)**  **of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11**  11th Meeting: Shanghai, CN, 10–19 October 2012 | Document: JCTVC-K0217 |

|  |  |  |  |
| --- | --- | --- | --- |
| *Title:* | **Proposed Changes on Coding Tree Unit Syntax and Sequence parameter set RBSP syntax** | | |
| *Status:* | Input Document to JCT-VC | | |
| *Purpose:* | Proposal | | |
| *Author(s) or Contact(s):* | Xue Fang, Limin Wang  6450 Sequence Drive  San Diego, CA 92121 USA | Tel: Email: | +1-858-404-3618  [x.fang@motorola.com](mailto:x.fang@motorola.com) |
| *Source:* | Motorola Mobility Inc. | | |

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

# Abstract

This contribution proposes high level syntax modifications in two sections. The first section proposes modification of Coding Tree Unit Syntax change regarding the InverseRasterScan function. The second section addresses the clean up as well as simplification related to PCM parameters in Sequence parameter set RBSP syntax.

.

# Coding Tree Unit Syntax

## Possible Issues

According to HEVC spec [1], Coding Tree Unit (CTU) contains a luma location ( xCtb, yCtb ) specifying the top-left sample of a current tree unit (CU) relative to the top‑left luma sample of a current picture, as shown in Table 1 and highlighted in red.

**Table 1** Coding Tree Unit Syntax

|  |  |
| --- | --- |
| coding\_tree\_unit( xCtb, yCtb ) { | Descriptor |
| NumPCMBlock = 0 |  |
| xCtb = InverseRasterScan( CtbAddrRS, CtbSize, CtbSize, pic\_width\_in\_luma\_samples, 0 ) |  |
| yCtb = InverseRasterScan( CtbAddrRS, CtbSize, CtbSize, pic\_width\_in\_luma\_samples, 1 ) |  |
| CtbAddrInSlice = CtbAddrRS − slice\_address |  |
| if( slice\_sao\_luma\_flag | | slice\_sao\_chroma\_flag ) |  |
| sao( xCtb >> Log2CtbSizeY, yCtb >> Log2CtbSizeY ) |  |
| coding\_quadtree( xCtb, yCtb, Log2CtbSizeY, 0 ) |  |
| } |  |

where InverseRasterScan() is given as

InverseRasterScan( a, b, c, d, e ) =  (5‑7)

When *pic\_width\_in\_luma\_samples* is not dividable by the tree unit size *CtbSize* which is equal to 1 << Log2CtbSizeY, xCtb and yCtb will not be calculated correctly in some cases.

For example, given a picture size of 416x240 and CtbSize of 64x64, there will be 7x4 CTU in total. Each CTU row has 7 CTUs according to the PicWidthInCtbsY defined in equation (7-20). For CTU of CtbAddrRS = 6, the calculated location of the up right most unit, (xCtb, yCtb), will be (0, 64) while the actual position should be (384, 0).

PicWidthInCtbsY = Ceil( pic\_width\_in\_luma\_samples ÷ CtbSizeY ) (7‑20)

PicHeightInCtbsY = Ceil( pic\_height\_in\_luma\_samples ÷ CtbSizeY ) (7‑22)

## Proposal Editoral Changes

|  |  |
| --- | --- |
| coding\_tree\_unit( xCtb, yCtb ) { | Descriptor |
| NumPCMBlock = 0 |  |
| xCtb = (CtbAddrRS %PicWidthInCtbsY)<< Log2CtbSizeY |  |
| yCtb = (CtbAddrRS / PicHeightInCtbsY) << Log2CtbSizeY |  |
| CtbAddrInSlice = CtbAddrRS − slice\_address |  |
| if( slice\_sao\_luma\_flag | | slice\_sao\_chroma\_flag ) |  |
| sao( xCtb >> Log2CtbSizeY, yCtb >> Log2CtbSizeY ) |  |
| coding\_quadtree( xCtb, yCtb, Log2CtbSizeY, 0 ) |  |
| } |  |

Here (CtbAddrRS/ PicHeightInCtbsY) and (CtbAddrRS/PicWidthInCtbsY) are corresponding to the row and column coding tree unit address of a picture. PicHeightInCtbsY is defined in equation (7-22).

# PCM in Sequence parameter set RBSP syntax

## PCM parameter Issues in Sequence parameter set RBSP syntax

|  |  |
| --- | --- |
| seq\_parameter\_set\_rbsp( ) { | Descriptor |
| **video\_parameter\_set\_id** | u(4) |
| **sps\_max\_sub\_layers\_minus1** | u(3) |
| **sps\_reserved\_zero\_bit** | u(1) |
| profile\_tier\_level( 1, sps\_max\_sub\_layers\_minus1 ) |  |
| **seq\_parameter\_set\_id** | ue(v) |
| **chroma\_format\_idc** | ue(v) |
| if( chroma\_format\_idc = = 3 ) |  |
| **separate\_colour\_plane\_flag** | u(1) |
| **pic\_width\_in\_luma\_samples** | ue(v) |
| **pic\_height\_in\_luma\_samples** | ue(v) |
| **pic\_cropping\_flag** | u(1) |
| if( pic\_cropping\_flag ) { |  |
| **pic\_crop\_left\_offset** | ue(v) |
| **pic\_crop\_right\_offset** | ue(v) |
| **pic\_crop\_top\_offset** | ue(v) |
| **pic\_crop\_bottom\_offset** | ue(v) |
| } |  |
| **bit\_depth\_luma\_minus8** | ue(v) |
| **bit\_depth\_chroma\_minus8** | ue(v) |
| [Ed. (BB): chroma bit depth present in HM software but not used further ] |  |
| **pcm\_enabled\_flag** | u(1) |
| if( pcm\_enabled\_flag ) { |  |
| **pcm\_sample\_bit\_depth\_luma\_minus1** | u(4) |
| **pcm\_sample\_bit\_depth\_chroma\_minus1** | u(4) |
| } |  |
| **log2\_max\_pic\_order\_cnt\_lsb\_minus4** | ue(v) |
| for( i = 0; i <= sps\_max\_sub\_layers\_minus1; i++ ) { |  |
| **sps\_max\_dec\_pic\_buffering**[ i ] | ue(v) |
| **sps\_max\_num\_reorder\_pics**[ i ] | ue(v) |
| **sps\_max\_latency\_increase**[ i ] | ue(v) |
| } |  |
| **restricted\_ref\_pic\_lists\_flag** | u(1) |
| if( restricted\_ref\_pic\_lists\_flag ) |  |
| **lists\_modification\_present\_flag** | u(1) |
| **log2\_min\_luma\_coding\_block\_size\_minus3** | ue(v) |
| **log2\_diff\_max\_min\_luma\_coding\_block\_size** | ue(v) |
| **log2\_min\_transform\_block\_size\_minus2** | ue(v) |
| **log2\_diff\_max\_min\_transform\_block\_size** | ue(v) |
| if( pcm\_enabled\_flag ) { |  |
| **log2\_min\_pcm\_luma\_coding\_block\_size\_minus3** | ue(v) |
| **log2\_diff\_max\_min\_pcm\_luma\_coding\_block\_size** | ue(v) |
| } |  |
| **max\_transform\_hierarchy\_depth\_inter** | ue(v) |
| **max\_transform\_hierarchy\_depth\_intra** | ue(v) |
| **scaling\_list\_enable\_flag** | u(1) |
| if( scaling\_list\_enable\_flag ) { |  |
| **sps\_scaling\_list\_data\_present\_flag** | u(1) |
| if( sps\_scaling\_list\_data\_present\_flag ) |  |
| scaling\_list\_data( ) |  |
| } |  |
| **amp\_enabled\_flag** | u(1) |
| **sample\_adaptive\_offset\_enabled\_flag** | u(1) |
| if( pcm\_enabled\_flag ) |  |
| **pcm\_loop\_filter\_disable\_flag** | u(1) |
| **sps\_temporal\_id\_nesting\_flag** | u(1) |
| **num\_short\_term\_ref\_pic\_sets** | ue(v) |
| for( i = 0; i < num\_short\_term\_ref\_pic\_sets; i++) |  |
| short\_term\_ref\_pic\_set( i ) |  |
| **long\_term\_ref\_pics\_present\_flag** | u(1) |
| if( long\_term\_ref\_pics\_present\_flag ) { |  |
| **num\_long\_term\_ref\_pics\_sps** | ue(v) |
| for( i = 0; i < num\_long\_term\_ref\_pics\_sps; i++ ) { |  |
| **lt\_ref\_pic\_poc\_lsb\_sps**[ i ] | u(v) |
| **used\_by\_curr\_pic\_lt\_sps\_flag**[ i ] | u(1) |
| } |  |
| } |  |
| **sps\_temporal\_mvp\_enable\_flag** | u(1) |
| **vui\_parameters\_present\_flag** | u(1) |
| if( vui\_parameters\_present\_flag ) |  |
| vui\_parameters( ) |  |
| **sps\_extension\_flag** | u(1) |
| if( sps\_extension\_flag ) |  |
| while( more\_rbsp\_data( ) ) |  |
| **sps\_extension\_data\_flag** | u(1) |
| rbsp\_trailing\_bits( ) |  |
| } |  |

Issue 1: PCM parameters are spread in three locations of the syntax, each with a condition check if pcm\_enabled\_flag is on.

Issue 2: **log2\_min\_pcm\_luma\_coding\_block\_size\_minus3** is already limited within the range set by **log2\_min\_luma\_coding\_block\_size\_minus3** and **log2\_diff\_max\_min\_luma\_coding\_block\_size**. Coding the delta from **log2\_min\_luma\_coding\_block\_size\_minus3** highlited in cyancan save bits and define more clearly without the sematic “The variable Log2MinIpcmCbSizeY shall be in the range of Log2MinCbSizeY to Min( Log2CtbSizeY, 5 ), inclusive”.

## Proposal PCM parater cleanup

### Group all the PCM related parameters together

This part also includes the suggested changes adopted in [2].

|  |  |
| --- | --- |
| seq\_parameter\_set\_rbsp( ) { | **Descriptor** |
| **video\_parameter\_set\_id** | **u(4)** |
| **sps\_max\_sub\_layers\_minus1** | **u(3)** |
| **sps\_reserved\_zero\_bit** | **u(1)** |
| profile\_tier\_level( 1, sps\_max\_sub\_layers\_minus1 ) |  |
| **seq\_parameter\_set\_id** | **ue(v)** |
| **chroma\_format\_idc** | **ue(v)** |
| if( chroma\_format\_idc = = 3 ) |  |
| **separate\_colour\_plane\_flag** | **u(1)** |
| **pic\_width\_in\_luma\_samples** | **ue(v)** |
| **pic\_height\_in\_luma\_samples** | **ue(v)** |
| **pic\_cropping\_flag** | **u(1)** |
| if( pic\_cropping\_flag ) { |  |
| **pic\_crop\_left\_offset** | **ue(v)** |
| **pic\_crop\_right\_offset** | **ue(v)** |
| **pic\_crop\_top\_offset** | **ue(v)** |
| **pic\_crop\_bottom\_offset** | **ue(v)** |
| } |  |
| **bit\_depth\_luma\_minus8** | **ue(v)** |
| **bit\_depth\_chroma\_minus8** | **ue(v)** |
| [Ed. (BB): chroma bit depth present in HM software but not used further ] |  |
| ~~pcm\_enabled\_flag~~ | **~~u(1)~~** |
| ~~if( pcm\_enabled\_flag ) {~~ |  |
| ~~pcm\_sample\_bit\_depth\_luma\_minus1~~ | **~~u(4)~~** |
| ~~pcm\_sample\_bit\_depth\_chroma\_minus1~~ | **~~u(4)~~** |
| ~~}~~ |  |
| **log2\_max\_pic\_order\_cnt\_lsb\_minus4** | **ue(v)** |
| for( i = 0; i <= sps\_max\_sub\_layers\_minus1; i++ ) { |  |
| **sps\_max\_dec\_pic\_buffering[ i ]** | **ue(v)** |
| **sps\_max\_num\_reorder\_pics[ i ]** | **ue(v)** |
| **sps\_max\_latency\_increase[ i ]** | **ue(v)** |
| } |  |
| **restricted\_ref\_pic\_lists\_flag** | **u(1)** |
| if( restricted\_ref\_pic\_lists\_flag ) |  |
| **lists\_modification\_present\_flag** | **u(1)** |
| **log2\_min\_luma\_coding\_block\_size\_minus3** | **ue(v)** |
| **log2\_diff\_max\_min\_luma\_coding\_block\_size** | **ue(v)** |
| **log2\_min\_transform\_block\_size\_minus2** | **ue(v)** |
| **log2\_diff\_max\_min\_transform\_block\_size** | **ue(v)** |
| ~~if( pcm\_enabled\_flag ) {~~ |  |
| ~~log2\_min\_pcm\_luma\_coding\_block\_size\_minus3~~ | **~~ue(v)~~** |
| ~~log2\_diff\_max\_min\_pcm\_luma\_coding\_block\_size~~ | **~~ue(v)~~** |
| ~~}~~ |  |
| **max\_transform\_hierarchy\_depth\_inter** | **ue(v)** |
| **max\_transform\_hierarchy\_depth\_intra** | **ue(v)** |
| **scaling\_list\_enable\_flag** | **u(1)** |
| if( scaling\_list\_enable\_flag ) { |  |
| **sps\_scaling\_list\_data\_present\_flag** | **u(1)** |
| if( sps\_scaling\_list\_data\_present\_flag ) |  |
| scaling\_list\_data( ) |  |
| } |  |
| **amp\_enabled\_flag** | **u(1)** |
| **sample\_adaptive\_offset\_enabled\_flag** | **u(1)** |
| **pcm\_enabled\_flag** | **u(1)** |
| if( pcm\_enabled\_flag ) { |  |
| **pcm\_sample\_bit\_depth\_luma\_minus1** | **u(4)** |
| **pcm\_sample\_bit\_depth\_chroma\_minus1** | **u(4)** |
| **log2\_min\_pcm\_luma\_coding\_block\_size\_minus3** | **ue(v)** |
| **log2\_diff\_max\_min\_pcm\_luma\_coding\_block\_size** | **ue(v)** |
| **pcm\_loop\_filter\_disable\_flag** | **u(1)** |
| } |  |
| **sps\_temporal\_id\_nesting\_flag** | **u(1)** |
| **num\_short\_term\_ref\_pic\_sets** | **ue(v)** |
| for( i = 0; i < num\_short\_term\_ref\_pic\_sets; i++) |  |
| short\_term\_ref\_pic\_set( i ) |  |
| **long\_term\_ref\_pics\_present\_flag** | **u(1)** |
| if( long\_term\_ref\_pics\_present\_flag ) { |  |
| **num\_long\_term\_ref\_pics\_sps** | **ue(v)** |
| for( i = 0; i < num\_long\_term\_ref\_pics\_sps; i++ ) { |  |
| **lt\_ref\_pic\_poc\_lsb\_sps[ i ]** | **u(v)** |
| **used\_by\_curr\_pic\_lt\_sps\_flag[ i ]** | **u(1)** |
| } |  |
| } |  |
| **sps\_temporal\_mvp\_enable\_flag** | **u(1)** |
| **vui\_parameters\_present\_flag** | **u(1)** |
| if( vui\_parameters\_present\_flag ) |  |
| vui\_parameters( ) |  |
| **sps\_extension\_flag** | **u(1)** |
| if( sps\_extension\_flag ) |  |
| while( more\_rbsp\_data( ) ) |  |
| **sps\_extension\_data\_flag** | **u(1)** |
| rbsp\_trailing\_bits( ) |  |
| } |  |

**bit\_depth\_chroma\_minus8** + 8specifies the bit depth of the samples of the chroma arrays and the value of the chroma quantization parameter range offset QpBdOffsetC, as specified by

BitDepthC = 8 + bit\_depth\_chroma\_minus8 (7‑10)  
QpBdOffsetC = 6 \* bit\_depth\_chroma\_minus8 (7‑11)

bit\_depth\_chroma\_minus8 shall be in the range of 0 to 6, inclusive.

**~~pcm\_enabled\_flag~~** ~~equal to 0 specifies that PCM data shall not be present in the video sequence.~~

**~~pcm\_sample\_bit\_depth\_luma\_minus1~~**~~+ 1 specifies the number of bits used to represent each of PCM sample values of luma component. The value of pcm\_sample\_bit\_depth\_luma\_minus1 + 1 shall be less than or equal to the value of BitDepth~~~~Y~~~~.~~

~~PCMBitDepth~~~~Y~~ ~~= 1 + pcm\_sample\_bit\_depth\_luma\_minus1 (7‑12)~~

**~~pcm\_sample\_bit\_depth\_chroma\_minus1~~**~~+ 1 specifies the number of bits used to represent each of PCM sample values of chroma components. The value of pcm\_sample\_bit\_depth\_chroma\_minus1 + 1 shall be less than or equal to the value of BitDepth~~~~C~~~~.~~

~~PCMBitDepth~~~~C~~ ~~= 1 + pcm\_sample\_bit\_depth\_chroma\_minus1 (7‑13)~~

**~~log2\_min\_pcm\_luma\_coding\_block\_size\_minus3~~**~~+ 3 specifies the minimum size of coding blocks with pcm\_flag equal to 1.~~

~~The variable Log2MinIpcmCbSizeY is set equal to log2\_min\_pcm\_luma\_coding\_block\_size\_minus3 + 3. The variable Log2MinIpcmCbSizeY shall be in the range of Log2MinCbSizeY to Min( Log2CtbSizeY, 5 ), inclusive.~~

**~~log2\_diff\_max\_min\_pcm\_luma\_coding\_block\_size~~** ~~specifies the difference between the maximum and minimum size of coding blocks with pcm\_flag equal to 1.~~

~~The variable Log2MaxIpcmCbSizeY is set equal to log2\_min\_pcm\_luma\_coding\_block\_size\_minus3 + 3 + log2\_diff\_max\_min\_pcm\_luma\_coding\_block\_size. The variable Log2MaxIpcmCbSizeY shall be equal or less than Min( Log2CtbSizeY, 5 ).~~

**max\_transform\_hierarchy\_depth\_inter** specifies the maximum hierarchy depth for transform units of coding units coded in inter prediction mode. The value of max\_transform\_hierarchy\_depth\_inter shall be in the range of 0 to Log2CtbSizeY − Log2MinTrafoSize, inclusive.

**sample\_adaptive\_offset\_enabled\_flag** equal to 1 specifies that the sample adaptive offset process is applied to the reconstruced picture after the deblocking filter process. sample\_adaptive\_offset\_enabled\_flag equal to 0 specifies that the sample adaptive offset process is not applied to the reconstruced picture after the deblocking filter process.

**pcm\_enabled\_flag** equal to 0 specifies that PCM data shall not be present in the video sequence.

NOTE X – When Log2MinCbSizeY is equal to 6, PCM data is not present in the video sequence even if pcm\_enabled\_flag is equal to 1. The maximum size of coding block with pcm\_enabled\_flag equal to 1 is restricted to be equal to or less than Min( Log2CtbSizeY,  5 ). Encoders are encouraged to use an appropriate combination of log2\_min\_luma\_coding\_block\_size\_minus3, log2\_min\_pcm\_luma\_coding\_block\_size\_minus3, and log2\_diff\_max\_min\_pcm\_luma\_coding\_block\_size values when they send PCM data in the video sequence.

**pcm\_sample\_bit\_depth\_luma\_minus1** + 1 specifies the number of bits used to represent each of PCM sample values of luma component. The value of pcm\_sample\_bit\_depth\_luma\_minus1 + 1 shall be less than or equal to the value of BitDepthY.

PCMBitDepthY = 1 + pcm\_sample\_bit\_depth\_luma\_minus1 (7‑12)

**pcm\_sample\_bit\_depth\_chroma\_minus1** + 1 specifies the number of bits used to represent each of PCM sample values of chroma components. The value of pcm\_sample\_bit\_depth\_chroma\_minus1 + 1 shall be less than or equal to the value of BitDepthC.

PCMBitDepthC = 1 + pcm\_sample\_bit\_depth\_chroma\_minus1 (7‑13)

**log2\_min\_pcm\_luma\_coding\_block\_size\_minus3** + 3 specifies the minimum size of coding blocks with pcm\_flag equal to 1.

The variable Log2MinIpcmCbSizeY is set equal to log2\_min\_pcm\_luma\_coding\_block\_size\_minus3 + 3. The variable Log2MinIpcmCbSizeY shall be in the range of Log2MinCbSizeY to Min( Log2CtbSizeY, 5 ), inclusive.

**log2\_diff\_max\_min\_pcm\_luma\_coding\_block\_size** specifies the difference between the maximum and minimum size of coding blocks with pcm\_flag equal to 1.

The variable Log2MaxIpcmCbSizeY is set equal to log2\_min\_pcm\_luma\_coding\_block\_size\_minus3 + 3 + log2\_diff\_max\_min\_pcm\_luma\_coding\_block\_size. The variable Log2MaxIpcmCbSizeY shall be equal or less than Min( Log2CtbSizeY, 5 ).

**pcm\_loop\_filter\_disable\_flag** specifies whether the loop filter process is disabled on reconstructed samples in a coding unit with pcm\_flag equal to 1. If the pcm\_loop\_filter\_disable\_flag value is equal to 1, deblocking filter and sample adaptive offset filter processes on the reconstructed samples in a coding unit with pcm\_flag equal to 1 are disabled; otherwise if the pcm\_loop\_filter\_disable\_flag value is equal to 0, deblocking filter and sample adaptive offset filter processes on the reconstructed samples in a coding unit with pcm\_flag equal to 1 are not disabled. When pcm\_loop\_filter\_disable\_flag is not present, it is inferred to be equal to 0.

[Ed. (WJ): select one expression – enabled\_flag or disable\_flag]

### Change log2\_min\_pcm\_luma\_coding\_block\_size\_minus3 to log2\_diff\_min\_pcm\_luma\_coding\_block\_size.

**log2\_diff\_min\_pcm\_luma\_coding\_block\_size** +  log2\_min\_luma\_coding\_block\_size\_minus3 + 3 specifies the minimum size of coding blocks with pcm\_enabled\_flag equal to 1.

|  |  |
| --- | --- |
| seq\_parameter\_set\_rbsp( ) { | **Descriptor** |
| **…..** |  |
| **log2\_min\_luma\_coding\_block\_size\_minus3** | ue(v) |
| **log2\_diff\_max\_min\_luma\_coding\_block\_size** | ue(v) |
| **log2\_min\_transform\_block\_size\_minus2** | ue(v) |
| **log2\_diff\_max\_min\_transform\_block\_size** | ue(v) |
| if( pcm\_enabled\_flag ) { |  |
| **log2\_diff\_min\_pcm\_luma\_coding\_block\_size** | ue(v) |
| **log2\_diff\_max\_min\_pcm\_luma\_coding\_block\_size** | ue(v) |
| } |  |
| **max\_transform\_hierarchy\_depth\_inter** | ue(v) |
| **max\_transform\_hierarchy\_depth\_intra** | ue(v) |
| …….. |  |

Or if proposal 2.2.1 is adopted,

|  |  |
| --- | --- |
| seq\_parameter\_set\_rbsp( ) { | **Descriptor** |
| **…..** |  |
| **pcm\_enabled\_flag** | u(1) |
| if( pcm\_enabled\_flag ) { |  |
| **pcm\_sample\_bit\_depth\_luma\_minus1** | u(4) |
| **pcm\_sample\_bit\_depth\_chroma\_minus1** | u(4) |
| **log2\_diff\_min\_pcm\_luma\_coding\_block\_size** | ue(v) |
| **log2\_diff\_max\_min\_pcm\_luma\_coding\_block\_size** | ue(v) |
| **pcm\_loop\_filter\_disable\_flag** | u(1) |
| } |  |
| …….. |  |

The following is one of the possible syntax examples of proposals above.

|  |  |
| --- | --- |
| seq\_parameter\_set\_rbsp( ) { | Descriptor |
| **video\_parameter\_set\_id** | u(4) |
| **sps\_max\_sub\_layers\_minus1** | u(3) |
| **sps\_reserved\_zero\_bit** | u(1) |
| profile\_tier\_level( 1, sps\_max\_sub\_layers\_minus1 ) |  |
| **seq\_parameter\_set\_id** | ue(v) |
| **chroma\_format\_idc** | ue(v) |
| if( chroma\_format\_idc = = 3 ) |  |
| **separate\_colour\_plane\_flag** | u(1) |
| **pic\_width\_in\_luma\_samples** | ue(v) |
| **pic\_height\_in\_luma\_samples** | ue(v) |
| **pic\_cropping\_flag** | u(1) |
| if( pic\_cropping\_flag ) { |  |
| **pic\_crop\_left\_offset** | ue(v) |
| **pic\_crop\_right\_offset** | ue(v) |
| **pic\_crop\_top\_offset** | ue(v) |
| **pic\_crop\_bottom\_offset** | ue(v) |
| } |  |
| **bit\_depth\_luma\_minus8** | ue(v) |
| **bit\_depth\_chroma\_minus8** | ue(v) |
| [Ed. (BB): chroma bit depth present in HM software but not used further ] |  |
| **~~pcm\_enabled\_flag~~** | ~~u(1)~~ |
| ~~if( pcm\_enabled\_flag ) {~~ |  |
| **~~pcm\_sample\_bit\_depth\_luma\_minus1~~** | ~~u(4)~~ |
| **~~pcm\_sample\_bit\_depth\_chroma\_minus1~~** | ~~u(4)~~ |
| ~~}~~ |  |
| **log2\_max\_pic\_order\_cnt\_lsb\_minus4** | ue(v) |
| for( i = 0; i <= sps\_max\_sub\_layers\_minus1; i++ ) { |  |
| **sps\_max\_dec\_pic\_buffering**[ i ] | ue(v) |
| **sps\_max\_num\_reorder\_pics**[ i ] | ue(v) |
| **sps\_max\_latency\_increase**[ i ] | ue(v) |
| } |  |
| **restricted\_ref\_pic\_lists\_flag** | u(1) |
| if( restricted\_ref\_pic\_lists\_flag ) |  |
| **lists\_modification\_present\_flag** | u(1) |
| **log2\_min\_luma\_coding\_block\_size\_minus3** | ue(v) |
| **log2\_diff\_max\_min\_luma\_coding\_block\_size** | ue(v) |
| **log2\_min\_transform\_block\_size\_minus2** | ue(v) |
| **log2\_diff\_max\_min\_transform\_block\_size** | ue(v) |
| ~~if( pcm\_enabled\_flag ) {~~ |  |
| **~~log2\_min\_pcm\_luma\_coding\_block\_size\_minus3~~** | ~~ue(v)~~ |
| **~~log2\_diff\_max\_min\_pcm\_luma\_coding\_block\_size~~** | ~~ue(v)~~ |
| ~~}~~ |  |
| **max\_transform\_hierarchy\_depth\_inter** | ue(v) |
| **max\_transform\_hierarchy\_depth\_intra** | ue(v) |
| **scaling\_list\_enable\_flag** | u(1) |
| if( scaling\_list\_enable\_flag ) { |  |
| **sps\_scaling\_list\_data\_present\_flag** | u(1) |
| if( sps\_scaling\_list\_data\_present\_flag ) |  |
| scaling\_list\_data( ) |  |
| } |  |
| **amp\_enabled\_flag** | u(1) |
| **sample\_adaptive\_offset\_enabled\_flag** | u(1) |
| ~~if( pcm\_enabled\_flag )~~ |  |
| **~~pcm\_loop\_filter\_disable\_flag~~** | ~~u(1)~~ |
| **sps\_temporal\_id\_nesting\_flag** | u(1) |
| **num\_short\_term\_ref\_pic\_sets** | ue(v) |
| for( i = 0; i < num\_short\_term\_ref\_pic\_sets; i++) |  |
| short\_term\_ref\_pic\_set( i ) |  |
| **long\_term\_ref\_pics\_present\_flag** | u(1) |
| if( long\_term\_ref\_pics\_present\_flag ) { |  |
| **num\_long\_term\_ref\_pics\_sps** | ue(v) |
| for( i = 0; i < num\_long\_term\_ref\_pics\_sps; i++ ) { |  |
| **lt\_ref\_pic\_poc\_lsb\_sps**[ i ] | u(v) |
| **used\_by\_curr\_pic\_lt\_sps\_flag**[ i ] | u(1) |
| } |  |
| } |  |
| **sps\_temporal\_mvp\_enable\_flag** | u(1) |
| **pcm\_enabled\_flag** | u(1) |
| if( pcm\_enabled\_flag ) { |  |
| **pcm\_sample\_bit\_depth\_luma\_minus1** | u(4) |
| **pcm\_sample\_bit\_depth\_chroma\_minus1** | u(4) |
| **log2\_diff\_min\_pcm\_luma\_coding\_block\_size** | ue(v) |
| **log2\_diff\_max\_min\_pcm\_luma\_coding\_block\_size** | ue(v) |
| **pcm\_loop\_filter\_disable\_flag** | u(1) |
| } |  |
| **vui\_parameters\_present\_flag** | u(1) |
| if( vui\_parameters\_present\_flag ) |  |
| vui\_parameters( ) |  |
| **sps\_extension\_flag** | u(1) |
| if( sps\_extension\_flag ) |  |
| while( more\_rbsp\_data( ) ) |  |
| **sps\_extension\_data\_flag** | u(1) |
| rbsp\_trailing\_bits( ) |  |
| } |  |

**Motorola Mobility Inc. 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).**

# Reference

[1] B. Bross, W.-J. Han, G. J. Sullivan, J.-R. Ohm, T. Wiegand, “High Efficiency Video Coding (HEVC) text specification draft 7,” JCTVC-K0030.

[2] K. Chono, “AHG9: On Tickets 637, 763 and, 777,” JCTVC-K0213