|  |  |
| --- | --- |
| **Joint Collaborative Team on Video Coding (JCT-VC)**  **of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11**  6th Meeting: Torino, 14-22 July, 2011 | Document: JCTVC-F469 |

|  |  |  |  |
| --- | --- | --- | --- |
| *Title:* | **Syntax improvement for fine granularity slices** | | |
| *Status:* | Input Document to JCT-VC | | |
| *Purpose:* | Proposal | | |
| *Author(s) or Contact(s):* | Chanyul Kim Youngo Park Kwang Pyo Choi | Email: [dionism@samsung.com](mailto:dionism@samsung.com)  [youngo.park@samsung.com](mailto:youngo.park@samsung.com)  [kp5.choi@samsung.com](mailto:kp5.choi@samsung.com) |  |
| *Source:* | Samsung Electronics Co., Ltd. | | |

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

# Abstract

The fine granularity slices was adopted at the last meeting through the BoG report of JCTVC-E483. The adopted syntax structure looks a little not suitable for recursive coding\_tree() representation way. In this proposal the changed fine granularity slices syntax and process is proposed to make the syntax fit for quard tree structure.

# Introduction

The fine granularity slices which removes the restrictions of slice boundary on LCU level and enable the slice boundary to be CU level was adopted by the BoG report of JCTVC-E483. The syntax of **slice\_granularity** (PPS) and **slice\_address** (Slice Header) was included and cuAddress(x, y) function is newly defined and called in coding\_tree() function before entering into itself.

# Problems

In the E483 the current CU position should be compared with SliceAddress which represents the absolute position of slice start and is derived from slice\_address before entering every recursive coding\_tree() function to check whether current CU is proceeding the SliceAddress or not even though current LCU is not equal to LCUAddress. We think that this process is not fully suitable for recursive coding\_tree() concept. For that reason, we propose new process and syntax for fine granularity slices which is simpler and fit to the concept of recursive coding\_tree() structure.

# Syntax and semantics described in JCTVC-E483

This syantax and semantics is from JCTVC-E483 and the changes relative to WD2 are highlighted in yellow.

## Picture parameter set

|  |  |
| --- | --- |
| pic\_parameter\_set\_rbsp( ) { | Descriptor |
| **pic\_parameter\_set\_id** | ue(v) |
| **seq\_parameter\_set\_id** | ue(v) |
| **entropy\_coding\_mode\_flag** | u(1) |
| **num\_ref\_idx\_l0\_default\_active\_minus1** | ue(v) |
| **num\_ref\_idx\_l1\_default\_active\_minus1** | ue(v) |
| **pic\_init\_qp\_minus26** **/**\* relative to 26 \*/ | se(v) |
| **constrained\_intra\_pred\_flag** | u(1) |
| **slice\_granularity** | u(2) |
| rbsp\_trailing\_bits( ) |  |
| } |  |

**slice\_granularity** indicates the slice granularity within a picture. The value of slice\_granularity shall not be larger than Min(Log2MaxCUSize-4, log2\_diff\_max\_min\_coding\_block\_size). The variable SliceGranularity is set to the value of (slice\_granularity << 1).

## Slice header

|  |  |  |
| --- | --- | --- |
| slice\_header( ) { | C | Descriptor |
| **first\_slice\_in\_pic\_flag** | 2 | u(1) |
| if(first\_slice\_in\_pic\_flag == 0) { |  |  |
| **slice\_address** | 2 | u(v) |
| } |  |  |
| } |  |  |

**first\_slice\_in\_pic\_flag** indicates whether the slice is the first slice of the picture. If first\_slice\_in\_pic\_flag is equal to 1, the variables SliceAddress and LCUAddress are both set to 0 and the decoding starts with the first LCU in the picture.

**slice\_address** specifies the address in slice granularity resolution in which the slice starts and shall be represented by (Ceil(Log2(NumLCUsInPicture))+SliceGranularity) bits in the bitstream where NumLCUsInPicture is the number of LCU:s in a picture.

The variable LCUAddress is set to slice\_address>>SliceGranularity and represents the LCU part of the slice address in raster scan order. The variable GranularityAddress is set to slice\_address - (LCUAddress << SliceGranularity) and represents the sub-LCU part of the slice address expressed in z-scan order.

The variable SliceAddress is then set to (LCUAddress << ( log2\_diff\_max\_min\_coding\_block\_size << 1)) + (GranularityAddress << ((log2\_diff\_max\_min\_coding\_block\_size << 1) - SliceGranularity) and the slice decoding starts with the largest coding unit possible at the slice starting coordinate.

## Slice data

|  |  |
| --- | --- |
| slice\_data( ) { | Descriptor |
| CurrTbAddr = LCUAddress |  |
| moreDataFlag = 1 |  |
| if( adaptive\_loop\_filter\_flag && alf\_cu\_control\_flag ) |  |
| AlfCuFlagIdx = -1 |  |
| do { |  |
| xCU = HorLumaLocation( CurrTbAddr ) |  |
| yCU = VerLumaLocation( CurrTbAddr ) |  |
| IsCuQpDeltaCoded = 0 |  |
| moreDataFlag = coding\_tree( xCU, yCU, Log2TbSize ) |  |
| CurrTbAddr = NextTbAddress( CurrTbAddr ) |  |
| } while( moreDataFlag ) |  |
| } |  |

## Coding tree structure

|  |  |
| --- | --- |
| coding\_tree( x0, y0, log2CUSize ) { | Descriptor |
| if( x0 + ( 1 << log2CUSize ) <= PicWidthInSamplesL &&  y0 + ( 1 << log2CUSize ) <= PicHeightInSamplesL &&  log2CUSize > Log2MinCUSize &&  cuAddress(x0 ,y0) >= SliceAddress ) |  |
| **split\_coding\_unit\_flag[** x0 **][** y0 **]** | u(1) | ae(v) |
| if( adaptive\_loop\_filter\_flag && alf\_cu\_control\_flag ) { |  |
| cuDepth = Log2MaxCUSize – log2CUSize |  |
| if( cuDepth <= alf\_cu\_control\_max\_depth ) |  |
| if( cuDepth == alf\_cu\_control\_max\_depth || |  |
| split\_coding\_unit\_flag[ x0 ][ y0 ] == 0 ) |  |
| AlfCuFlagIdx++ |  |
| } |  |
| if( split\_coding\_unit\_flag[ x0 ][ y0 ] ) { |  |
| x1 = x0 + ( ( 1 << log2CUSize ) >> 1 ) |  |
| y1 = y0 + ( ( 1 << log2CUSize ) >> 1 ) |  |
| if( cuAddress(x1,y0) > SliceAddress ) |  |
| moreDataFlag = coding\_tree( x0, y0, log2CUSize – 1 ) |  |
| if(cuAddress(x0,y1) > SliceAddress && moreDataFlag && x1 < PicWidthInSamplesL ) |  |
| moreDataFlag = coding\_tree( x1, y0, log2CUSize − 1 ) |  |
| if(cuAddress(x1,y1) > SliceAddress && moreDataFlag && y1 < PicHeightInSamplesL ) |  |
| moreDataFlag = coding\_tree( x0, y1, log2CUSize − 1 ) |  |
| if( moreDataFlag && x1 < PicWidthInSamplesL && y1 < PicHeightInSamplesL ) |  |
| moreDataFlag = coding\_tree( x1, y1, log2CUSize − 1 ) |  |
| } else { |  |
| if(adaptive\_loop\_filter\_flag && alf\_cu\_control\_flag ) |  |
| AlfCuFlag[ x0 ][ y0 ] = alf\_cu\_flag[ AlfCuFlagIdx ] |  |
| coding\_unit( x0, y0, log2CUSize ) |  |
| if( !entropy\_coding\_mode\_flag ) |  |
| moreDataFlag = more\_rbsp\_data( ) |  |
| else { |  |
| if( get\_bit\_left( ) > 0 ) |  |
| moreDataFlag = 1 |  |
| else { |  |
| **end\_of\_slice\_flag** | ae(v) |
| moreDataFlag = !end\_of\_slice\_flag |  |
| } |  |
| } |  |
|  |  |
| return moreDataFlag |  |
| } |  |

# Proposed syntax and semantics

The proposed syantax and semantics are described below. The changes from JCTVC-E483 are highlighted in yellow and the changes relative to WD3 of proposed algorithm is highlighted in green.

## Slice header syntax

|  |  |
| --- | --- |
| slice\_header( ) { | Descriptor |
| **first\_tb\_in\_slice** | ue(v) |
| if(log2\_diff\_max\_min\_coding\_block\_size > 0) |  |
| **preroll\_cnt** | u(v) |
| **entropy\_slice\_flag** | u(1) |
| <snip> |  |
| } |  |

**preroll\_cnt** specifies the value of the variable PrerollCnt that is used to find the CU start position of fine granularity slice. The value of preroll\_cnt shall be in the range of 0 to 63, inclusive. When preroll\_cnt is not present the value of PrerollCnt is inferred to be equal to 0.

– The maximum value of preroll\_cnt is the number of SCU in one LCU minus one

– The number of SCU in one LCU and required bits for the representation of the preroll\_cnt that is dependent on log2\_diff\_max\_min\_coding\_block\_size are specified in Table X-X

Table X-X – The number of bits for the representation of the preroll\_cnt

|  |  |  |
| --- | --- | --- |
| **log2\_diff\_max\_min\_coding\_block\_size** | **The number of SCU in one LCU** | **The number of bits for the representation of the preroll\_cnt** |
| 1 | 4 | 2 |
| 2 | 16 | 4 |
| 3 | 64 | 6 |

## Slice data syntax

|  |  |
| --- | --- |
| slice\_data( ) { | Descriptor |
| CurrTbAddr = first\_tb\_in\_slice |  |
| moreDataFlag = 1/\* CHECK-ME\*/ |  |
| if( adaptive\_loop\_filter\_flag && alf\_cu\_control\_flag ) |  |
| AlfCuFlagIdx = -1 |  |
| do { |  |
| xCU = HorLumaLocation( CurrTbAddr ) |  |
| yCU = VerLumaLocation( CurrTbAddr ) |  |
| moreDataFlag = coding\_tree( xCU, yCU, Log2TbSize ) |  |
| CurrTbAddr = NextTbAddress( CurrTbAddr ) |  |
| } while( moreDataFlag ) |  |
| } |  |

## Coding tree syntax

|  |  |
| --- | --- |
| coding\_tree( x0, y0, log2CUSize ) { | Descriptor |
| if( x0 + ( 1 << log2CUSize ) <= PicWidthInSamplesL &&  y0 + ( 1 << log2CUSize ) <= PicHeightInSamplesL ) { |  |
| if( !entropy\_coding\_mode\_flag && slice\_type != I && PrerollCnt==0) |  |
| **cu\_split\_pred\_part\_mode**[ x0 ][ y0 ] | ce(v) |
| else if( log2CUSize > Log2MinCUSize) |  |
| **split\_coding\_unit\_flag[** x0 **][** y0 **]** | u(1) | ae(v) |
| } |  |
| if( adaptive\_loop\_filter\_flag && alf\_cu\_control\_flag ) { |  |
| cuDepth = Log2MaxCUSize – log2CUSize |  |
| if( cuDepth <= alf\_cu\_control\_max\_depth ) |  |
| if( cuDepth == alf\_cu\_control\_max\_depth || |  |
| split\_coding\_unit\_flag[ x0 ][ y0 ] == 0 ) |  |
| AlfCuFlagIdx++ |  |
| } |  |
| if( split\_coding\_unit\_flag[ x0 ][ y0 ] ) { |  |
| if( cu\_qp\_depta\_enabled\_flag &&   log2CUSize == log2MinCUDQPSize ) |  |
| IsCuQpDeltaCoded = 0 |  |
| x1 = x0 + ( ( 1 << log2CUSize ) >> 1 ) |  |
| y1 = y0 + ( ( 1 << log2CUSize ) >> 1 ) |  |
| moreDataFlag = coding\_tree( x0, y0, log2CUSize – 1 ) |  |
| if( moreDataFlag && x1 < PicWidthInSamplesL ) |  |
| moreDataFlag = coding\_tree( x1, y0, log2CUSize − 1 ) |  |
| if( moreDataFlag && y1 < PicHeightInSamplesL ) |  |
| moreDataFlag = coding\_tree( x0, y1, log2CUSize − 1 ) |  |
| if(moreDataFalg && x1 < PicWidthInSamplesL && y1 < PicHeightInSamplesL ) |  |
| moreDataFlag = coding\_tree( x1, y1, log2CUSize − 1 ) |  |
| } else { |  |
| if(PrerollCnt > 0) { |  |
| PrerollCnt-- |  |
| return 1 /\* moreDataFlag \*/ |  |
| } |  |
| if(adaptive\_loop\_filter\_flag && alf\_cu\_control\_flag ) |  |
| AlfCuFlag[ x0 ][ y0 ] = alf\_cu\_flag[ AlfCuFlagIdx ] |  |
| coding\_unit( x0, y0, log2CUSize ) |  |
| if( !entropy\_coding\_mode\_flag ) |  |
| moreDataFlag = more\_rbsp\_data( ) |  |
| else { |  |
| if( get\_bit\_left( ) > 0 ) |  |
| moreDataFlag = 1 |  |
| else { |  |
| **end\_of\_slice\_flag** | ae(v) |
| moreDataFlag = !end\_of\_slice\_flag |  |
| } |  |
| } |  |
| } |  |
| return moreDataFlag |  |
| } |  |

# Algorithm Details

First the encoder sets the **preroll\_cnt** value to slice header. Second the so called pseudo split flag is generated and located in front of a slice data. The decoder only decodes the bitstream without checking the current CU position even though a current decoding CU is located in the slice start LCU.

The way to generate preroll\_cnt and pseudo split flat in the side of an encoder is explained in the Figure 1. The decoding process is shown in the Figure 2.

**Figure 1. Encoder PrerollCnt and pseudo split flag generation example**



**Figure 1. Decoder PrerollCnt and pseudo split flag parsing example**



# Conclusion

In this contribution the new syntax and process for fine granularity slices is proposed. It is thought that proposed syntax and process is more fit to the recursive coding\_tree() concept and can increase synax readability. Furthermore slice granularity specifying syntax (**slice\_granularity**) is not needed any more and the also need not to limit the SCU boundary to any level.

# Patent rights declaration(s)

**Samsung Electronics Co., Ltd. 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).**