|  |  |
| --- | --- |
| **Joint Collaborative Team on Video Coding (JCT-VC)**  **of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11**  7th Meeting: Geneva, CH, 21-30 November, 2011 | Document: JCTVC-G355 |

|  |  |  |  |
| --- | --- | --- | --- |
| *Title:* | **Joint coding of splitting flag and inter modes in CAVLC** | | |
| *Status:* | Input Document to JCT-VC | | |
| *Purpose:* | Proposal | | |
| *Author(s) or Contact(s):* | Wen Zhang  Ping Wu | Tel: Email: | +86-13528803560 [zhang.wen4@zte.com.cn](mailto:zhang.wen4@zte.com.cn) |
| *Source:* | ZTE Corporation | | |

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

# Abstract

This contribution was successor to JCTVC-F418 and was focused on CAVLC only. It had resolved parsing dependency problem in JCTVC-F418 meanwhile maintained coding gain. It provided an alternative joint coding scheme of splitting flag and inter modes in CAVLC rather than what adopted in HM 4.0. According test results, -0.1%/-0.4%/-0.6%/-0.6% BD-rate gain was achieved comparing to HM 4.0 for AI-LC/RA-LC/LB-LC/LP-LC configurations.

# Introduction

In HM 4.0, the joint coding scheme of splitting flag and inter modes in CAVLC was characterized as depth-dependent (JCTVC-D370) and counter adaptive (JCTVC-E143). In this proposal, the representing method of CU splitting flag and inter modes was however context-dependent and slice adaptive.

Firstly, the coding CU splitting flag and inter modes was classified into 4 types according to contexts of current CU. It’s specifically classified according to CU splitting and prediction mode of left and above CUs, along with current CU-depth. Please refer to Table-1 for detailed coding types of CU splitting flag and inter modes.

Table-1. The 4 coding types (contexts) defined for CU splitting flag and inter modes

|  |  |  |  |
| --- | --- | --- | --- |
| CU-depth | Prediction mode-top CU | Prediction mode-left CU | Coding Type |
| CU size > 8x8 (min) | SKIP | SKIP | 0 |
| INTER | SKIP |
| SKIP | INTER |
| INTRA | SKIP |
| SKIP | INTRA |
| CU Splitting | SKIP |
| SKIP | CU Splitting |
| INTER | INTER |
| INTRA | INTER | 1 |
| INTER | INTRA |
| CU Splitting | INTER |
| INTER | CU Splitting |
| INTRA | INTRA |
| CU Splitting | INTRA |
| INTRA | CU Splitting |
| CU Splitting | CU Splitting |
| CU size==8x8(min) | SKIP | SKIP | 2 |
| INTER | SKIP |
| SKIP | INTER |
| INTRA | SKIP |
| SKIP | INTRA |
| INTER | INTER |
| INTRA | INTER | 3 |
| INTER | INTRA |
| INTRA | INTRA |

Secondly, since AMP had been added in HM 4.0 meanwhile inter 4x4 was disabled, CU splitting flag and inter modes supported in HM 4.0 were summarized in Table-2 below. A new syntax element “split\_mode” was thus defined to represent CU splitting flag and inter modes.

From Table-2, the value of “split\_mode” was ranged from 0(SKIP) to 7(AMP) (details are indicated in Table-2). When “split\_mode” equals to INTER\_AMP, it included four PartMode of 2NxnU, 2NxnD, nLx2N, nRx2N which could be further signaled by two flags (amp\_direct\_flag/amp\_pos\_flag). If “split\_mode” is INTER\_2NxN, INTER\_Nx2N or INTER\_AMP, two additional flags (pu\_merge\_flag) should be used to signal merge information of two PU included in current CU.

Table-2 The split\_mode defined for CU splitting flag and inter modes

|  |  |  |  |
| --- | --- | --- | --- |
| PredMode and  CU splitting | PartMode | CU size > 8x8 (min)  split\_mode | CU size == 8x8(min)  split\_mode |
| MODE\_SKIP | PART\_2Nx2N | 0 (SKIP\_2Nx2N) | 0 (SKIP\_2Nx2N) |
| MODE\_MRG | PART\_2Nx2N | 5 (MRG\_2Nx2N) | 5 (MRG\_2Nx2N) |
| MODE\_INTER | PART\_2Nx2N | 1(INTER\_2Nx2N) | 1 (INTER\_2Nx2N) |
| PART\_2NxN | 3 (INTER\_2NxN) | 3 (INTER\_2NxN) |
| PART\_Nx2N | 4 (INTER\_Nx2N) | 4 (INTER\_Nx2N) |
| PART\_2NxnU | 7 (INTER\_AMP) | -- |
| PART\_2NxnD |
| PART\_nLx2N |
| PART\_nRx2N |
| PART\_NxN | -- | 7 (INTER\_NxN) disabled |
| MODE\_INTRA | PART\_2Nx2N | 2 (INTRA\_2Nx2N) | 2(INTRA\_2Nx2N) |
| PART\_NxN | -- | 6(INTRA\_NxN) |
| SPLIT |  | 6(SPLIT) | -- |
| PCM | PART\_2Nx2N | disabled | Disabled |

Thirdly, 31 codeword depending on distribution of each coding types for “split\_mode” were defined and explicitly signaled per slice. For each codeword, it represented the coding order of unary flags in “split\_mode”. And, the codeword was highly context dependent and basically slice adaptive. Please refer to below g\_auiCtxDepthInterSplitMode for detailed codeword information and this predefined codeword table should be maintained in both encoder and decoder.

#define ENCTYPE(a,b,c,d,e,f,g,h) (((h)<<21)+((g)<<18)+((f)<<15)+((e)<<12)+((d)<<9)+((c)<<6)+ ((b)<<3)+((a)<<0))

#define CODEWORD\_LEN 31

Const UInt g\_auiCtxDepthInterSplitMode[4][CODEWORD\_LEN] = {

{ENCTYPE(0,6,7,1,4,5,3,2),ENCTYPE(0,6,1,7,4,3,2,5),ENCTYPE(0,6,5,7,1,2,4,3),ENCTYPE(6,0,7,5,1,2,4,3),ENCTYPE(0,6,7,4,1,5,3,2),ENCTYPE(0,6,4,7,1,3,5,2),ENCTYPE(0,6,7,5,1,4,3,2),ENCTYPE(0,6,1,4,7,3,2,5),ENCTYPE(6,7,0,5,1,2,4,3),ENCTYPE(0,6,4,1,7,3,5,2),ENCTYPE(0,5,6,1,7,4,3,2),ENCTYPE(0,1,6,2,7,3,4,5),ENCTYPE(5,6,0,7,2,1,3,4),ENCTYPE(7,6,5,0,1,2,4,3),ENCTYPE(6,5,0,7,1,2,4,3),ENCTYPE(6,5,7,0,1,3,4,2),ENCTYPE(6,7,5,1,4,3,2,0),ENCTYPE(6,1,0,7,5,2,3,4),ENCTYPE(7,5,6,0,1,2,4,3),ENCTYPE(7,0,6,5,1,2,4,3),ENCTYPE(5,0,6,7,2,1,3,4),ENCTYPE(5,2,6,7,0,1,3,4),ENCTYPE(5,7,6,0,2,1,3,4),ENCTYPE(2,5,6,0,1,7,3,4),ENCTYPE(1,5,6,0,7,2,3,4),ENCTYPE(0,2,6,1,5,7,3,4),ENCTYPE(0,3,6,1,5,7,4,2),ENCTYPE(0,4,6,1,5,7,3,2),ENCTYPE(0,7,6,1,5,4,3,2),ENCTYPE(0,6,2,7,1,4,3,5),ENCTYPE(0,6,3,7,1,4,2,5)},

{ENCTYPE(6,0,7,1,4,3,5,2),ENCTYPE(6,7,0,4,1,5,3,2),ENCTYPE(0,6,1,7,4,3,5,2),ENCTYPE(6,0,1,7,4,3,2,5),ENCTYPE(6,2,7,5,0,1,4,3),ENCTYPE(6,5,7,2,0,1,4,3),ENCTYPE(6,0,4,7,1,5,2,3),ENCTYPE(6,7,5,0,2,4,1,3),ENCTYPE(2,6,0,1,5,7,3,4),ENCTYPE(6,7,2,0,5,4,1,3),ENCTYPE(6,7,4,0,5,2,1,3),ENCTYPE(6,2,5,7,0,1,4,3),ENCTYPE(0,1,6,4,3,7,2,5),ENCTYPE(0,6,7,1,4,3,5,2),ENCTYPE(6,0,5,7,1,4,2,3),ENCTYPE(1,0,6,4,3,2,5,7),ENCTYPE(2,0,6,1,5,7,3,4),ENCTYPE(0,6,4,1,7,3,5,2),ENCTYPE(0,2,6,1,4,3,7,5),ENCTYPE(0,3,6,1,4,2,7,5),ENCTYPE(0,4,6,1,3,7,2,5),ENCTYPE(0,7,6,1,4,3,2,5),ENCTYPE(6,1,0,7,2,5,4,3),ENCTYPE(6,4,1,0,7,2,5,3),ENCTYPE(6,7,1,0,5,2,4,3),ENCTYPE(6,7,3,1,0,5,2,4),ENCTYPE(6,0,2,7,1,4,5,3),ENCTYPE(6,2,1,7,5,0,4,3),ENCTYPE(6,2,0,7,5,1,4,3),ENCTYPE(6,5,0,7,2,1,4,3),ENCTYPE(6,5,2,7,0,1,4,3)},

{ENCTYPE(0,1,4,2,3,5,6,7),ENCTYPE(0,5,1,4,2,3,6,7),ENCTYPE(0,4,1,3,5,2,6,7),ENCTYPE(0,1,2,4,3,5,6,7),ENCTYPE(5,0,1,4,2,3,6,7),ENCTYPE(0,4,3,1,5,2,6,7),ENCTYPE(0,5,4,1,2,3,6,7),ENCTYPE(0,5,2,1,4,3,6,7),ENCTYPE(0,2,1,5,6,4,3,7),ENCTYPE(0,1,3,4,2,5,6,7),ENCTYPE(0,1,5,4,2,3,6,7),ENCTYPE(0,2,5,1,6,4,3,7),ENCTYPE(0,4,5,1,3,2,6,7),ENCTYPE(0,3,4,1,5,2,6,7),ENCTYPE(5,1,0,4,3,2,6,7),ENCTYPE(5,0,4,1,2,3,6,7),ENCTYPE(0,6,1,5,4,2,3,7),ENCTYPE(0,3,1,4,5,2,6,7),ENCTYPE(0,3,5,4,1,2,6,7),ENCTYPE(0,2,4,1,5,6,3,7),ENCTYPE(0,2,6,1,5,4,3,7),ENCTYPE(0,4,2,1,3,5,6,7),ENCTYPE(1,5,4,0,2,3,6,7),ENCTYPE(2,0,6,5,1,4,3,7),ENCTYPE(4,0,1,5,3,2,6,7),ENCTYPE(0,5,3,1,4,2,6,7),ENCTYPE(5,2,0,1,4,3,6,7),ENCTYPE(5,3,0,1,4,2,6,7),ENCTYPE(5,4,0,1,3,2,6,7),ENCTYPE(5,6,0,1,4,2,3,7),ENCTYPE(5,0,2,1,4,6,3,7)},

{ENCTYPE(2,6,0,5,1,4,3,7),ENCTYPE(0,1,2,4,3,5,6,7),ENCTYPE(2,0,6,5,1,4,3,7),ENCTYPE(0,2,1,4,3,5,6,7),ENCTYPE(2,5,6,0,1,4,3,7),ENCTYPE(5,2,0,1,4,6,3,7),ENCTYPE(6,2,5,1,4,0,3,7),ENCTYPE(2,1,6,0,5,4,3,7),ENCTYPE(1,0,2,4,6,5,3,7),ENCTYPE(0,4,1,2,3,5,6,7),ENCTYPE(5,0,2,1,4,6,3,7),ENCTYPE(1,2,0,4,6,5,3,7),ENCTYPE(3,0,4,1,5,2,6,7),ENCTYPE(0,3,1,2,4,5,6,7),ENCTYPE(4,0,1,3,5,6,2,7),ENCTYPE(0,6,1,2,4,5,3,7),ENCTYPE(0,5,1,2,4,3,6,7),ENCTYPE(1,4,0,2,6,5,3,7),ENCTYPE(1,5,0,2,4,6,3,7),ENCTYPE(1,6,0,2,4,5,3,7),ENCTYPE(3,4,0,1,5,2,6,7),ENCTYPE(4,1,0,3,5,6,2,7),ENCTYPE(4,3,0,1,5,6,2,7),ENCTYPE(4,5,0,1,3,6,2,7),ENCTYPE(5,1,2,0,4,6,3,7),ENCTYPE(5,3,2,0,1,4,6,7),ENCTYPE(5,4,2,0,1,6,3,7),ENCTYPE(5,6,2,0,1,4,3,7),ENCTYPE(6,1,2,5,4,0,3,7),ENCTYPE(6,4,2,5,1,0,3,7),ENCTYPE(6,5,2,1,4,0,3,7)}

};

In each inter slice, when the first time it has one of the four coding types (bFirstCodingTypeInSlice) before coding split\_mode. A codeword\_index which ranged from 0 to CODEWORD\_LEN-1 to represent codeword for that coding type was signaled and transmitted using UVLC.

Since codeword was explicitly signaled and transmitted, the decoder didn't need to gather statistics of previous slices like in JCTVC-F418. Thus there is no parsing dependency problem between slices anymore. But such information was still helpful and could be used at encoder side in RDO progress.

The fourth, in addition to above context dependent and slice adaptive coding method, swapping adaption process in HM 4.0 was also incorporated as complementary part to explicit codeword signaling.

Finally, in order to further remove redundancy existed in signaling of CU splitting flag both in inter and intra slice, A new syntax element “pred\_split\_flag” was defined while CU-depth equals to 0 or 1. If this flag equals to 1, it means not only CU of current depth with “split\_flag” equals to 1 but also all CUs here of current depth plus one with “split\_flag” equal to 1. The coding gain of “pred\_split\_flag” was roughly 0.1% BD-rate for AI-LC and RA-LC configurations.

Meanwhile, the presence of pred\_split\_flag was adaptively controlled at two levels. At slice level, a controlling flag (pred\_split\_coding\_flag) was added into slice header, it could switch on/off the presence of pred\_split\_flag in current slice. At CU level, it was decided by bHasPredSplitFlag to code pred\_split\_flag in current CU or not to code according to above and left CU-depth information. The detailed constrained conditions were defined as below to deduce bHasPredSplitFlag of current CU. When this bHasPredSplitFlag was false, the value of pred\_split\_flag should be deduced as false too.

If ((getSlice()->isIntra() && (uiDepth == 0)) ||

((getSlice()->isIntra() || (uiDepth == 0)) && (getSlice()->getPredSplitCodingFlag())))

{

bHasPredSplitFlag = above A CUs with smaller CU-depth **OR** left B CUs with smaller CU-depth;

}

Else if (!getSlice()->isIntra() && (uiDepth > 0) && getSlice()->getPredSplitCodingFlag())

{

bHasPredSplitFlag = above A CUs with smaller CU-depth **AND** left B CUs with smaller CU-depth;

}

Else

{

bHasPredSplitFlag = false;

}

# Proposed Changes to WD Text

According to syntax elements defined above and their explanation, the WD text of slice\_header(), coding\_tree(), coding\_unit() and prediction\_unit() should be revised (from JCTVC-F803-v7). All the necessary changes to WD are presented in this section.

1. Insert pred\_split\_coding\_flag into slice\_header().
2. Substitute cu\_split\_pred\_part\_mode[x0][y0] with pred\_split\_flag [x0][y0] (bHasPredSplitFlag), codeword\_index [i] (bFirstCodingTypeInSlice) and split\_mode [x0][y0] (PredMode, PartMode) in coding\_tree().
3. Insert amp\_direct\_flag/amp\_pos\_flag and pu\_merge\_flag[i] in coding\_unit().
4. Remove unnecessary merge\_flag in prediction\_unit().

The change to current WD draft in slice\_header() was shown in Table-3.

Table-3 Insert pred\_split\_coding\_flag to slice\_header()

|  |  |
| --- | --- |
| slice\_header( ) { | Descriptor |
| **lightweight\_slice\_flag** | u(1) |
| if( !lightweight\_slice\_flag ) { |  |
| **slice\_type** | ue(v) |
| **pic\_parameter\_set\_id** | ue(v) |
| **frame\_num** | u(v) |
| if( IdrPicFlag ) |  |
| **idr\_pic\_id** | ue(v) |
| if( pic\_order\_cnt\_type = = 0 ) |  |
| **pic\_order\_cnt\_lsb /\*** | u(v) |
| if( slice\_type = = P | | slice\_type = = B ) { |  |
| **num\_ref\_idx\_active\_override\_flag** | u(1) |
| if( num\_ref\_idx\_active\_override\_flag ) { |  |
| **num\_ref\_idx\_l0\_active\_minus1** | ue(v) |
| if( slice\_type = = B ) |  |
| **num\_ref\_idx\_l1\_active\_minus1** | ue(v) |
| } |  |
| } |  |
| ref\_pic\_list\_modification( ) |  |
| ref\_pic\_list\_combination( ) |  |
| if( nal\_ref\_idc != 0 ) |  |
| dec\_ref\_pic\_marking( ) |  |
| } |  |
| if( entropy\_coding\_mode\_flag && slice\_type != I) |  |
| **cabac\_init\_idc** | ue(v) |
| else if (!entropy\_coding\_mode\_flag) |  |
| **pred\_split\_coding\_flag** | u(1) |
| **first\_slice\_in\_pic\_flag** | u(1) |
| if( first\_slice\_in\_pic\_flag == 0 ) |  |
| **slice\_address** | u(v) |
| if( !lightweight\_slice\_flag ) { |  |
| **slice\_qp\_delta** | se(v) |
| if( sample\_adaptive\_offset\_enabled\_flag ) |  |
| sao\_param() |  |
| if( deblocking\_filter\_control\_present\_flag ) { |  |
| **disable\_deblocking\_filter\_idc** |  |
| if( disable\_deblocking\_filter\_idc != 1 ) { |  |
| **slice\_alpha\_c0\_offset\_div2** |  |
| **slice\_beta\_offset\_div2** |  |
| } |  |
| } |  |
| if( slice\_type = = B ) |  |
| **collocated\_from\_l0\_flag** | u(1) |
| if( adaptive\_loop\_filter\_enabled\_flag ) { |  |
| if( !shared\_pps\_info\_enabled\_flag ) |  |
| alf\_param( ) |  |
| alf\_cu\_control\_param( ) |  |
| } |  |
| } |  |
| } |  |

The change to current WD draft in coding\_tree() was shown in Table-4.

Table-4 Add pred\_split\_flag and codeword\_index/split\_mode in coding\_tree()

|  |  |
| --- | --- |
| coding\_tree( x0, y0, log2CUSize, cuDepth ) { | Descriptor |
| if( x0 + ( 1 << log2CUSize ) <= PicWidthInSamplesL &&  y0 + ( 1 << log2CUSize ) <= PicHeightInSamplesL &&  cuAddress( x0, y0 ) >= SliceAddress ) { |  |
| if( !entropy\_coding\_mode\_flag ) { |  |
|  |  |
| if (bHasPredSplitFlag) |  |
| **pred\_split\_flag[x0][y0]** | u(1) |
| if (!pred\_split\_flag) { |  |
| if (slice\_type != I) { |  |
| if (bFirstCodingTypeInSlice) |  |
| **codeword\_index[i]** | ce(v) |
| **split\_mode[x0][y0]** | ce(v) |
| } |  |
| else |  |
| **split\_coding\_unit\_flag[x0][y0]** | u(1) |
| } |  |
| } |  |
| else if( log2CUSize > Log2MinCUSize ) |  |
| **split\_coding\_unit\_flag[** x0 **][** y0 **]** | u(1) | ae(v) |
| } |  |
| if( adaptive\_loop\_filter\_flag && alf\_cu\_control\_flag ) { |  |
| if( cuDepth <= alf\_cu\_control\_max\_depth ) |  |
| if( cuDepth == alf\_cu\_control\_max\_depth || |  |
| split\_coding\_unit\_flag[ x0 ][ y0 ] == 0 ) |  |
| AlfCuFlagIdx++ |  |
| } |  |
| if( cu\_qp\_delta\_enabled\_flag &&   log2CUSize >= log2MinCUDQPSize ) |  |
| IsCuQpDeltaCoded = 0 |  |
| 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, cuDepth + 1 ) |  |
| if( cuAddress( x0, y1 ) > SliceAddress && moreDataFlag &&  x1 < PicWidthInSamplesL ) |  |
| moreDataFlag = coding\_tree( x1, y0, log2CUSize − 1, cuDepth + 1 ) |  |
| if( cuAddress( x1, y1 ) > SliceAddress && moreDataFlag &&  y1 < PicHeightInSamplesL ) |  |
| moreDataFlag = coding\_tree( x0, y1, log2CUSize − 1, cuDepth + 1 ) |  |
| if( moreDataFlag &&   x1 < PicWidthInSamplesL && y1 < PicHeightInSamplesL ) |  |
| moreDataFlag = coding\_tree( x1, y1, log2CUSize − 1, cuDepth + 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( granularity\_block\_boundary( x0, y0, log2CUSize ) ) { |  |
| **end\_of\_slice\_flag** | ae(v) |
| moreDataFlag = !end\_of\_slice\_flag |  |
| } else |  |
| moreDataFlag = 1 |  |
| } |  |
| } |  |
| return moreDataFlag |  |
| } |  |

The change to current WD draft in coding\_unit() was shown in Table-5.

Table-5 Insert amp\_direct\_flag/amp\_pos\_flag and pu\_merge\_flag in coding\_unit()

|  |  |
| --- | --- |
| coding\_unit( x0, y0, log2CUSize ) { | Descriptor |
| if( entropy\_coding\_mode\_flag && slice\_type != I ) |  |
| **skip\_flag[** x0 **][** y0 **]** | u(1) | ae(v) |
| if( skip\_flag[ x0 ][ y0 ] ) |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| else { |  |
| if( !entropy\_coding\_mode\_flag ) { |  |
| if( slice\_type == I && log2CUSize == Log2MinCUSize ) |  |
| **intra\_part\_mode** | u(1) |
| else if (split\_mode[x0][y0] == INTER\_AMP){ |  |
| **amp\_direct\_flag** | u(1) |
| **amp\_pos\_flag** | u(1) |
| } |  |
| if (split\_mode[x0][y0] == INTER\_AMP || split\_mode[x0][y0] == INTER\_2NxN || split\_mode[x0][y0] == INTER\_Nx2N) { |  |
| **pu\_merge\_flag[0]** | u(1) |
| **pu\_merge\_flag[1]** | u(1) |
| } |  |
| } else if( slice\_type != I | | log2CUSize = = Log2MinCUSize ) |  |
| **pred\_type** | u(v) | ae(v) |
| x1 = x0 + ( ( 1 << log2CUSize ) >> 1 ) |  |
| y1 = y0 + ( ( 1 << log2CUSize ) >> 1 ) |  |
| x2 = x1 − ( ( 1 << log2CUSize ) >> 2 ) |  |
| y2 = y1 − ( ( 1 << log2CUSize ) >> 2 ) |  |
| x3 = x1 + ( ( 1 << log2CUSize ) >> 2 ) |  |
| y3 = y1 + ( ( 1 << log2CUSize ) >> 2 ) |  |
| if( PartMode == PART\_2Nx2N ) { |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| } else if( PartMode == PART\_2NxN ) { |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| prediction\_unit( x0, y1 , log2CUSize ) |  |
| } else if( PartMode == PART\_Nx2N ) { |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| prediction\_unit( x1, y0 , log2CUSize ) |  |
| } else if( PartMode == PART\_2NxnU ) { |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| prediction\_unit( x0, y2 , log2CUSize ) |  |
| } else if( PartMode == PART\_2NxnD ) { |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| prediction\_unit( x0, y3 , log2CUSize ) |  |
| } else if( PartMode == PART\_nLx2N ) { |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| prediction\_unit( x2, y0 , log2CUSize ) |  |
| } else if( PartMode == PART\_nRx2N ) { |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| prediction\_unit( x3, y0 , log2CUSize ) |  |
| } else { /\* PART\_NxN \*/ |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| prediction\_unit( x1, y0 , log2CUSize ) |  |
| prediction\_unit( x0, y1 , log2CUSize ) |  |
| prediction\_unit( x1, y1 , log2CUSize ) |  |
| } |  |
| } |  |
| if( !pcm\_flag ) { |  |
| transform\_tree( x0, y0, log2CUSize, log2CUSize, 0, 0 ) |  |
| transform\_coeff( x0, y0, log2CUSize, log2CUSize, 0, 0 ) |  |
| transform\_coeff( x0, y0, log2CUSize, log2CUSize, 0, 1 ) |  |
| transform\_coeff( x0, y0, log2CUSize, log2CUSize, 0, 2 ) |  |
| } |  |
| } |  |

The change to current WD draft in prediction\_unit () was shown in Table-6.

Table-6 Remove merge\_flag in prediction\_unit ()

|  |  |
| --- | --- |
| prediction\_unit( x0, y0, log2CUSize ) { | Descriptor |
| if( skip\_flag[ x0 ][ y0 ] ) { |  |
| **merge\_idx[** x0 **][** y0 **]** | ue(v) | ae(v) |
| } else if( PredMode = = MODE\_INTRA ) { |  |
| if( PartMode == PART\_2Nx2N &&  log2CUSize >= Log2MinIPCMCUSize ) |  |
| **pcm\_flag** | u(1) | ae(v) |
| if( pcm\_flag ) { |  |
| while ( !byte\_aligned( ) ) |  |
| **pcm\_alignment\_zero\_bit** | u(v) |
| for( i = 0; i < 1 << ( log2CUSize << 1 ); i++ ) |  |
| **pcm\_sample\_luma**[ i ] | u(v) |
| for( i = 0; i < ( 1 << ( log2CUSize << 1 ) ) >> 1; i++ ) |  |
| **pcm\_sample\_chroma**[ i ] | u(v) |
| } else { |  |
| **prev\_intra\_luma\_pred\_flag[** x0 **][** y0 **]** | u(1) | ae(v) |
| if( prev\_intra\_luma\_pred\_flag[ x0 ][ y0 ] ) |  |
| **mpm\_idx[** x0 **][** y0 **]** | u(1) | ae(v) |
| else |  |
| **rem\_intra\_luma\_pred\_mode[** x0 **][**y0 **]** | ce(v) | ae(v) |
| **intra\_chroma\_pred\_mode**[ x0 ][ y0 ] | ue(v) | ae(v) |
| SignaledAsChromaDC =   ( chroma\_pred\_from\_luma\_enabled\_flag ?  intra\_chroma\_pred\_mode[ x0 ][ y0 ] == 3 :  intra\_chroma\_pred\_mode[ x0 ][ y0 ] == 2 ) |  |
| } |  |
| } else { /\* MODE\_INTER \*/ |  |
| if( entropy\_coding\_mode\_flag ) |  |
| **merge\_flag[** x0 **][** y0 **]** | u(1) | ae(v) |
| if( merge\_flag[ x0 ][ y0 ] ) { |  |
| **merge\_idx[** x0 **][** y0 **]** | ue(v) | ae(v) |
| } else { |  |
| if( slice\_type = = B ) { |  |
| if( !entropy\_coding\_mode\_flag ) { |  |
| **combined\_inter\_pred\_ref\_idx** | ue(v) |
| if( combined\_inter\_pred\_ref\_idx == MaxPredRef ) |  |
| **inter\_pred\_flag**[ x0 ][ y0 ] | ue(v) |
| } else |  |
| **inter\_pred\_flag[** x0 **][** y0 **]** | ue(v) | ae(v) |
| } |  |
| if( inter\_pred\_flag[ x0 ][ y0 ] = = Pred\_LC ) { |  |
| if( num\_ref\_idx\_lc\_active\_minus1 > 0 ) { |  |
| if( !entropy\_coding\_mode\_flag ) { |  |
| if( combined\_inter\_pred\_ref\_idx == MaxPredRef ) |  |
| **ref\_idx\_lc\_minus4**[ x0 ][ y0 ] | ue(v) |
| } else |  |
| **ref\_idx\_lc[** x0 **][** y0 **]** | ae(v) |
| } |  |
| if( entropy\_coding\_mode\_flag ) { |  |
| mvd\_coding\_cabac(mvd\_lc[ x0 ][ y0 ][ 0 ],   mvd\_lc[ x0 ][ y0 ][ 1 ]) |  |
| } else { |  |
| **mvd\_lc[** x0 **][** y0 **][** 0 **]** | se(v) |
| **mvd\_lc[** x0 **][** y0 **][** 1 **]** | se(v) |
| } |  |
| **mvp\_idx\_lc[ x0 ][ y0 ]** | ue(v) | ae(v) |
| } |  |
| else { /\* Pred\_L0 or Pred\_BI \*/ |  |
| if( num\_ref\_idx\_l0\_active\_minus1 > 0 ) { |  |
| if( !entropy\_coding\_mode\_flag ) { |  |
| if( combined\_inter\_pred\_ref\_idx == MaxPredRef ) |  |
| **ref\_idx\_l0\_minusX**[ x0 ][ y0 ] | ue(v) |
| } else |  |
| **ref\_idx\_l0\_minusX**[ x0 ][ y0 ] | ue(v) | ae(v) |
| } |  |
| if( entropy\_coding\_mode\_flag ) { |  |
| mvd\_coding\_cabac(mvd\_l0[ x0 ][ y0 ][ 0 ],   mvd\_l0[ x0 ][ y0 ][ 1 ]) |  |
| } else { |  |
| **mvd\_l0[** x0 **][** y0 **][** 0 **]** | se(v) |
| **mvd\_l0[** x0 **][** y0 **][** 1 **]** | se(v) |
|  |  |
| **mvp\_idx\_l0[ x0 ][ y0 ]** | ue(v) | ae(v) |
| } |  |
| if( inter\_pred\_flag[ x0 ][ y0 ] = = Pred\_BI ) { |  |
| if( num\_ref\_idx\_l1\_active\_minus1 > 0 ) { |  |
| if( !entropy\_coding\_mode\_flag ) { |  |
| if( combined\_inter\_pred\_ref\_idx == MaxPredRef ) |  |
| **ref\_idx\_l1\_minusX**[ x0 ][ y0 ] | ue(v) |
| } else |  |
| **ref\_idx\_l1[** x0 **][** y0 **]** | ue(v) | ae(v) |
| } |  |
| if( entropy\_coding\_mode\_flag ) { |  |
| mvd\_coding\_cabac(mvd\_l1[ x0 ][ y0 ][ 0 ],   mvd\_l1[ x0 ][ y0 ][ 1 ]) |  |
| } else { |  |
| **mvd\_l1[** x0 **][** y0 **][** 0 **]** | se(v) |
| **mvd\_l1[** x0 **][** y0 **][** 1 **]** | se(v) |
| } |  |
| **mvp\_idx\_l1[ x0 ][ y0 ]** | ue(v) | ae(v) |
| } |  |
| } |  |
| } |  |
| } |  |

# Simulation Results and Conclusion

The proposed algorithm was implemented based on HM 4.0 and tested on NT server 2003 32 bits OS. So, please be noted that the anchor used was a little bit different to BBC’s as a known problem of ticket #203 (slideshow LDPLC QP==32). For cross check results, please refer to JCTVC-G741 by Samsung and JCTVC-G905 by Qualcomm.

Table-7 Simulation Results with CODEWORD\_LEN equals to 31.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **All Intra LC** | | | |
|  | Y | U | V | |
| Class A | 0.0% | 0.0% | 0.0% | |
| Class B | 0.0% | -0.1% | 0.0% | |
| Class C | -0.1% | -0.1% | -0.1% | |
| Class D | -0.1% | -0.1% | -0.1% | |
| Class E | 0.0% | 0.0% | 0.0% | |
| Class F | -0.1% | -0.1% | -0.1% | |
| **Overall** | -0.1% | -0.1% | -0.1% | |
|  | -0.1% | -0.1% | -0.1% | |
| Enc Time[%] | 100% | | | |
| Dec Time[%] | 100% | | | |
|  | **Random Access LC** | | | | |
|  | Y | U | | V | |
| Class A | -0.2% | 0.0% | | 0.1% | |
| Class B | -0.3% | -0.2% | | -0.2% | |
| Class C | -0.7% | -0.7% | | -0.6% | |
| Class D | -0.6% | -0.7% | | -0.6% | |
| Class E |  |  | |  | |
| Class F | -0.3% | -0.4% | | -0.4% | |
| **Overall** | -0.4% | -0.4% | | -0.3% | |
|  | -0.4% | -0.4% | | -0.3% | |
| Enc Time[%] | 99% | | | | |
| Dec Time[%] | 99% | | | | |
|  | **Low delay B LC** | | | |
|  | Y | U | V | |
| Class A |  |  |  | |
| Class B | -0.3% | -0.2% | -0.2% | |
| Class C | -0.7% | -0.6% | -0.9% | |
| Class D | -0.8% | -1.1% | -1.0% | |
| Class E | -1.2% | -0.7% | -1.3% | |
| Class F | -0.3% | -0.6% | -0.5% | |
| **Overall** | -0.6% | -0.6% | -0.7% | |
|  | -0.6% | -0.6% | -0.8% | |
| Enc Time[%] | 99% | | | |
| Dec Time[%] | 100% | | | |
|  | **Low delay P LC** | | | |
|  | Y | U | V | |
| Class A |  |  |  | |
| Class B | -0.3% | -0.1% | -0.3% | |
| Class C | -0.7% | -0.7% | -0.7% | |
| Class D | -0.8% | -1.0% | -1.3% | |
| Class E | -1.0% | -0.9% | -0.8% | |
| Class F | -0.4% | -0.5% | -0.4% | |
| **Overall** | -0.6% | -0.6% | -0.7% | |
|  | -0.6% | -0.6% | -0.7% | |
| Enc Time[%] | 99% | | | |
| Dec Time[%] | 101% | | | |

Additional test results with shorten CODEWORD\_LEN equals to 15 and 7 were shown in table-7.

Table-8 Coding gain comparison of different CODEWORD\_LEN.



Since the simulation results showed that the modified scheme can further enhance CAVLC performance with a reasonable margin, it is recommended to adopt this design into WD and HM.

# Scheme On Top JCTVC-G312

According to the grouping method described in JCTVC-G312, detailed scheme on top JCTVC-G312 please refer to below description.

1. The above Table-2 was modified, The value of “split\_mode” was ranged from 0(SKIP) to 5(MRG) as indicated in Table-9.
2. The codeword table was changed and CODEWORD\_LEN was set to be 15.
3. The swapping process in HM 4.0 was removed.

Table-9 The split\_mode defined on top JCTVC-G312

|  |  |  |  |
| --- | --- | --- | --- |
| PredMode and  CU splitting | PartMode | CU size > 8x8 (min)  split\_mode | CU size == 8x8(min)  split\_mode |
| MODE\_SKIP | PART\_2Nx2N | 0 (SKIP\_2Nx2N) | 0 (SKIP\_2Nx2N) |
| MODE\_MRG | PART\_2Nx2N | 5 (MRG\_2Nx2N) | 5 (MRG\_2Nx2N) |
| MODE\_INTER | PART\_2Nx2N | 1(INTER\_2Nx2N) | 1 (INTER\_2Nx2N) |
| PART\_2NxN | 3(INTER\_2PU) | 3(INTER\_2PU) |
| PART\_Nx2N |
| PART\_2NxnU | -- |
| PART\_2NxnD |
| PART\_nLx2N |
| PART\_nRx2N |
| PART\_NxN | -- | disabled |
| MODE\_INTRA | PART\_2Nx2N | 2 (INTRA\_2Nx2N) | 2(INTRA\_2Nx2N) |
| PART\_NxN | -- | 4(INTRA\_NxN) |
| SPLIT |  | 4(SPLIT) | -- |
| PCM | PART\_2Nx2N | disabled | Disabled |

When “split\_mode” equals to INTER\_2PU, it included six PartMode of 2NxnU, 2NxnD, nLx2N, nRx2N, 2NxN and Nx2N for CU size greater than 8x8 (min) and two PartMode of 2NxN and Nx2N for CU size equals to 8x8 (min).

In this situation, additional syntax elements “horz\_part\_flag” and “rem\_part\_mode” were signaled to select one PartMode from the group. The definition of these two syntax elements was shown in below Table-3. The value of “rem\_part\_mode” was encoded with truncated unary code but without decoding table. And in 8x8 CU, only “horz\_part\_flag” was additionally signaled since there is no AMP.

Table-3 The definition of horz\_part\_flag and rem\_part\_mode

|  |  |  |  |
| --- | --- | --- | --- |
| split\_mode | PartMode | horz\_part\_flag | rem\_part\_mode |
| INTER\_2PU | PART\_2NxN | 1 | 0 |
| PART\_2NxnU |  | 1 |
| PART\_2NxnD |  | 2 |
| PART\_Nx2N | 0 | 0 |
| PART\_nLx2N |  | 1 |
| PART\_nRx2N |  | 2 |

If “split\_mode” is INTER\_2PU, two additional flags (pu\_merge\_flag) should also be used to signal merge information of two PU included in current CU.

It was 15 codeword depending on distribution of each coding types for “split\_mode” were defined and explicitly signaled per slice. For each codeword, it represented the coding order of unary flags in “split\_mode”. And, the codeword was highly context dependent and basically slice adaptive. Please refer to below g\_auiCtxDepthInterSplitMode for detailed codeword information and this predefined codeword table should be maintained in both encoder and decoder.

#define ENCTYPE(a,b,c,d,e,f) ( ((f)<<15)+((e)<<12)+((d)<<9)+((c)<<6)+ ((b)<<3)+((a)<<0))

#define CODEWORD\_LEN 15

Const UInt g\_auiCtxDepthInterSplitMode[4][CODEWORD\_LEN] = {

{ENCTYPE(0,4,3,1,5,2),ENCTYPE(0,3,4,1,5,2),ENCTYPE(0,4,5,3,1,2),ENCTYPE(0,4,3,5,1,2),ENCTYPE(0,4,1,3,2,5),ENCTYPE(0,3,4,5,1,2),ENCTYPE(0,3,1,4,2,5),ENCTYPE(0,4,1,5,3,2),ENCTYPE(4,0,3,5,1,2),ENCTYPE(0,1,4,3,2,5),ENCTYPE(0,4,5,1,3,2),ENCTYPE(0,5,4,1,3,2), ENCTYPE(4,3,0,5,1,2),ENCTYPE(5,4,0,3,2,1),ENCTYPE(0,1,4,5,3,2)},

{ENCTYPE(4,0,3,1,5,2),ENCTYPE(4,3,0,1,5,2),ENCTYPE(4,0,1,3,2,5),ENCTYPE(4,3,5,0,2,1),ENCTYPE(4,0,3,5,1,2),ENCTYPE(0,4,1,3,2,5),ENCTYPE(4,5,3,2,0,1),ENCTYPE(4,2,3,5,0,1), ENCTYPE(4,3,2,0,5,1),ENCTYPE(4,3,0,5,2,1),ENCTYPE(4,2,5,3,0,1),ENCTYPE(4,0,5,3,1,2), ENCTYPE(4,0,2,3,1,5),ENCTYPE(0,1,4,3,2,5),ENCTYPE(2,4,1,0,5,3)},

{ENCTYPE(0,1,3,2,5,4),ENCTYPE(0,5,1,3,2,4),ENCTYPE(0,3,1,5,2,4),ENCTYPE(0,1,2,3,5,4),ENCTYPE(0,1,3,5,2,4),ENCTYPE(0,5,3,1,2,4),ENCTYPE(0,5,2,3,1,4),ENCTYPE(0,2,1,5,4,3),ENCTYPE(0,5,1,2,3,4),ENCTYPE(0,1,5,3,2,4),ENCTYPE(3,0,5,1,2,4),ENCTYPE(0,2,5,1,4,3), ENCTYPE(0,3,5,1,2,4),ENCTYPE(3,0,1,5,2,4),ENCTYPE(5,3,0,1,2,4)},

{ENCTYPE(2,4,0,3,5,1),ENCTYPE(0,1,2,3,5,4),ENCTYPE(2,0,4,5,1,3),ENCTYPE(0,2,1,3,5,4),ENCTYPE(2,5,4,0,1,3),ENCTYPE(3,0,1,5,4,2),ENCTYPE(4,2,5,1,0,3),ENCTYPE(2,1,4,0,5,3),ENCTYPE(1,0,2,3,4,5),ENCTYPE(5,0,2,1,3,4),ENCTYPE(0,3,1,2,5,4),ENCTYPE(0,5,1,2,3,4), ENCTYPE(0,1,3,2,5,4),ENCTYPE(5,2,0,1,3,4),ENCTYPE(3,1,0,5,4,2)}

};

The change to WD text was also involved in by inserting horz\_part\_flag/rem\_part\_mode and pu\_merge\_flag[i] into coding\_unit(). It’s shown in Table-10.

Table-10 Insert horz\_part\_flag/rem\_part\_mode and pu\_merge\_flag in coding\_unit()

|  |  |
| --- | --- |
| coding\_unit( x0, y0, log2CUSize ) { | Descriptor |
| if( entropy\_coding\_mode\_flag && slice\_type != I ) |  |
| **skip\_flag[** x0 **][** y0 **]** | u(1) | ae(v) |
| if( skip\_flag[ x0 ][ y0 ] ) |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| else { |  |
| if( !entropy\_coding\_mode\_flag ) { |  |
| if( slice\_type == I && log2CUSize == Log2MinCUSize ) |  |
| **intra\_part\_mode** | u(1) |
| else if (split\_mode[x0][y0] == INTER\_2PU){ |  |
| **horz\_part\_flag** | u(1) |
| if (log2CUSize != Log2MinCUSize) |  |
| **rem\_part\_mode** | ce(v) |
| **pu\_merge\_flag[0]** | u(1) |
| **pu\_merge\_flag[1]** | u(1) |
| } |  |
| } else if( slice\_type != I | | log2CUSize = = Log2MinCUSize ) |  |
| **pred\_type** | u(v) | ae(v) |
| x1 = x0 + ( ( 1 << log2CUSize ) >> 1 ) |  |
| y1 = y0 + ( ( 1 << log2CUSize ) >> 1 ) |  |
| x2 = x1 − ( ( 1 << log2CUSize ) >> 2 ) |  |
| y2 = y1 − ( ( 1 << log2CUSize ) >> 2 ) |  |
| x3 = x1 + ( ( 1 << log2CUSize ) >> 2 ) |  |
| y3 = y1 + ( ( 1 << log2CUSize ) >> 2 ) |  |
| if( PartMode == PART\_2Nx2N ) { |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| } else if( PartMode == PART\_2NxN ) { |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| prediction\_unit( x0, y1 , log2CUSize ) |  |
| } else if( PartMode == PART\_Nx2N ) { |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| prediction\_unit( x1, y0 , log2CUSize ) |  |
| } else if( PartMode == PART\_2NxnU ) { |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| prediction\_unit( x0, y2 , log2CUSize ) |  |
| } else if( PartMode == PART\_2NxnD ) { |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| prediction\_unit( x0, y3 , log2CUSize ) |  |
| } else if( PartMode == PART\_nLx2N ) { |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| prediction\_unit( x2, y0 , log2CUSize ) |  |
| } else if( PartMode == PART\_nRx2N ) { |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| prediction\_unit( x3, y0 , log2CUSize ) |  |
| } else { /\* PART\_NxN \*/ |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| prediction\_unit( x1, y0 , log2CUSize ) |  |
| prediction\_unit( x0, y1 , log2CUSize ) |  |
| prediction\_unit( x1, y1 , log2CUSize ) |  |
| } |  |
| } |  |
| if( !pcm\_flag ) { |  |
| transform\_tree( x0, y0, log2CUSize, log2CUSize, 0, 0 ) |  |
| transform\_coeff( x0, y0, log2CUSize, log2CUSize, 0, 0 ) |  |
| transform\_coeff( x0, y0, log2CUSize, log2CUSize, 0, 1 ) |  |
| transform\_coeff( x0, y0, log2CUSize, log2CUSize, 0, 2 ) |  |
| } |  |
| } |  |

Simulation results on top JCTVC-312 with CODEWORD\_LEN equals to 15 was shown in Table-10.

Table-10 The Simulation Results with CODEWORD\_LEN equals to 15

|  |  |  |  |
| --- | --- | --- | --- |
|  | **All Intra LC** | | |
| Y | U | V |
| Class A | 0.0% | 0.0% | 0.0% |
| Class B | 0.0% | -0.1% | 0.0% |
| Class C | -0.1% | -0.1% | -0.1% |
| Class D | -0.1% | -0.1% | -0.1% |
| Class E | 0.0% | 0.0% | 0.0% |
| Class F | -0.1% | -0.1% | -0.1% |
| **Overall** | -0.1% | -0.1% | -0.1% |
|  | -0.1% | -0.1% | -0.1% |
| Enc Time[%] | 100% | | |
| Dec Time[%] | 100% | | |
|  | **Random Access LC** | | |
| Y | U | V |
| Class A | -0.3% | 0.0% | 0.0% |
| Class B | -0.4% | -0.3% | -0.2% |
| Class C | -0.7% | -0.7% | -0.7% |
| Class D | -0.7% | -0.8% | -0.7% |
| Class E |  |  |  |
| Class F | -0.4% | -0.4% | -0.4% |
| **Overall** | -0.5% | -0.4% | -0.4% |
|  | -0.5% | -0.4% | -0.4% |
| Enc Time[%] | 99% | | |
| Dec Time[%] | 99% | | |
|  | **Low delay B LC** | | |
| Y | U | V |
| Class A |  |  |  |
| Class B | -0.5% | -0.6% | -0.4% |
| Class C | -1.0% | -0.9% | -1.2% |
| Class D | -1.0% | -1.1% | -1.3% |
| Class E | -1.3% | -1.2% | -1.6% |
| Class F | -0.4% | -0.5% | -0.8% |
| **Overall** | -0.8% | -0.8% | -1.0% |
|  | -0.8% | -0.8% | -1.0% |
| Enc Time[%] | 99% | | |
| Dec Time[%] | 100% | | |
|  | **Low delay P LC** | | |
| Y | U | V |
| Class A |  |  |  |
| Class B | -0.6% | -0.4% | -0.8% |
| Class C | -0.9% | -1.0% | -0.9% |
| Class D | -1.0% | -1.1% | -1.2% |
| Class E | -1.3% | -1.8% | -1.6% |
| Class F | -0.5% | -0.1% | -0.2% |
| **Overall** | -0.8% | -0.8% | -0.9% |
|  | -0.8% | -0.8% | -0.9% |
| Enc Time[%] | 99% | | |
| Dec Time[%] | 100% | | |

# References

1. JCTVC-F418 Joint Coding of CU splitting flag and inter modes based on HM 3.1 [W. Zhang (ZTE)].
2. F. Bossen, “Common test conditions and software reference configurations”, Document no JVCVC-F900, Torino, July 2011.
3. Benjamin Bross, Woo-Jin Han, Jens-Rainer Ohm, Gary J. Sullivan, Thomas Wiegand, “WD4: Working Draft 4 of High-Efficiency Video Coding”, Document no JVCVC-F803, Torino, July 2011.
4. T. Yamamoto, “Non-Square Partition Mode Grouping for CAVLC,” Document of Joint Collaborative Team on Video Coding, JCTVC-G312, Nov. 2011.

# Patent rights declaration(s)

**ZTE Corporation 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).**