#### **Sample adaptive offset parameter syntax**

|  |  |
| --- | --- |
| aps\_sao\_param( ) { | Descriptor |
| **log2\_sao\_unit\_width\_minus6** | ue(v) |
| **log2\_sao\_unit\_height\_minus6** | ue(v) |
| **sao\_cb\_enable\_flag** | u(1) |
| **sao\_cr\_enable\_flag** | u(1) |
| **sao\_num\_unit\_in\_width\_minus1** | ue(v) |
| **sao\_num\_unit\_in\_height\_minus1** | ue(v) |
| **sao\_one\_luma\_unit\_flag** | u(1) |
| if(sao\_one\_luma\_unit\_flag) |  |
| sao\_offset\_vlc( 0, 0, 0 ) |  |
| if( sao\_cb\_enable\_flag ){ |  |
| **sao\_one\_cb\_unit\_flag** | u(1) |
| if( sao\_one\_cb\_unit\_flag ) |  |
| sao\_offset\_vlc( 0, 0, 1 ) |  |
| } |  |
| if( sao\_cr\_enable\_flag ){ |  |
| **sao\_one\_cr\_unit\_flag** | u(1) |
| if( sao\_one\_cr\_unit\_flag ) |  |
| sao\_offset\_vlc( 0, 0, 2 ) |  |
| } |  |
| for( ry = 0; ry <= sao\_num\_unit\_in\_height\_minus1; ry++ ) { |  |
| for( rx = 0; rx <= sao\_num\_unit\_in\_width\_minus1; rx++ ) { |  |
| if( aps\_sample\_adaptive\_offset\_flag && ! sao\_one\_luma\_unit\_flag ) { |  |
| if( ry > 0 && rx = = 0 ) |  |
| **sao\_repeat\_row\_flag**[ 0 ] | u(1) |
| sao\_unit\_vlc(rx, ry, 0 ) |  |
| } |  |
| if( sao\_cb\_enable\_flag && ! sao\_one\_cb\_unit\_flag ) { |  |
| if( ry > 0 & & rx = = 0 ) |  |
| **sao\_repeat\_row\_flag**[ 1 ] | u(1) |
| sao\_unit\_vlc(rx, ry, 1 ) |  |
| } |  |
| if( sao\_cr\_enable\_flag & & !sao\_one\_cr\_unit\_flag ) { |  |
| if( ry > 0 & & rx = = 0 ) |  |
| **sao\_repeat\_row\_flag**[ 2 ] | u(1) |
| sao\_unit\_vlc( rx, ry, 2 ) |  |
| } |  |
| } |  |
| } |  |
| } |  |

#### **7.3.2.8 Sample adaptive offset VLC syntax**

|  |  |
| --- | --- |
| sao\_offset\_vlc( rx, ry, cIdx ) { | Descriptor |
| **sao\_type\_idx**[ cIdx ][ rx ][ ry ] | ue(v) |
| if( sao\_type\_idx[ cIdx ][ rx ][ ry ] = =5 ) { |  |
| **sao\_band\_position**[ cIdx ][ rx ][ ry ] | u(5) |
| for( i = 0; i < 4; i++ ) |  |
| **sao\_offset**[ cIdx ][ rx][ ry ][ i ] | se(v) |
| } else if( sao\_type\_idx[ cIdx ][ rx ][ ry ] != 0 ) |  |
| for( i = 0; i < 4; i++ ) |  |
| **sao\_offset**[ cIdx ][ rx][ ry ][ i ] | ue(v) |
| } |  |

#### **7.4.2.6 Sample adaptive offset parameter semantics**

**sao\_cb\_enable\_flag** equal to 1 specifies that the sample adaptive offset process for Cb is applied to the current picture.

**sao\_cr\_enable\_flag** equal to 1 specifies that the sample adaptive offset process for Cr is applied to the current picture.

**log2\_sao\_unit\_width\_minus6** is used to specify the sample width of SAO unit.

**log2\_sao\_unit\_height\_minus6** is used to specify the sample height of SAO unit.

If aps\_sao\_interleaving\_flag is equal to 0,

Log2NumCtbInSaoUnitWidth = log2\_sao\_unit\_width\_minus6 + (6 - Log2CtbSize ),

Log2NumCtbInSaoUnitHeight = log2\_sao\_unit\_height\_minus6 + (6 - Log2CtbSize ).

Otherwise,

Log2NumCtbInSaoUnitWidth = 0

Log2NumCtbInSaoUnitHeight = 0

**sao\_num\_unit\_in\_width\_minus1** plus 1 specifies the number of SAO units in picture width.

**sao\_num\_unit\_in\_height\_minus1** plus 1 specifies the number of SAO units in picture height.

**sao\_one\_luma\_unit\_flag** equal to 1 specifies that all of SAO units in luma component of current slice are processed using the same SAO parameters; equal to 0 specifies that the SAO parameters for the luma component are signalled on a SAO unit basis.

**sao\_one\_cb\_unit\_flag** equal to 1 specifies that all of SAO units in Cb component of current slice are processed using the same SAO parameters; equal to 0 specifies that the SAO parameters for the Cb component are signalled on a SAO unit basis.

**sao\_one\_cr\_unit\_flag** equal to 1 specifies that all of SAO units in Cr component of current slice are processed using the same SAO parameters; equal to 0 specifies that the SAO parameters for the Cr component are signalled on a SAO unit basis.

**sao\_repeat\_row\_flag**[ cIdx ] equal to 1 specifies that the SAO parameters of SAO unit in the current SAO unit row are the same as those of above SAO unit row for the colour component cIdx; equal to 0 specifies that the current SAO unit row has different SAO parameters. When sao\_repeat\_row\_flag is not present, it is inferred to be equal to 0.

#### **7.4.2.7 Sample adaptive offset unit VLC semantics**

The number of times the SAO parameters corresponding to a SAO unit are repeated for subsequent SAO units in the same row is represented by saoRun[ cIdx ][ rx ][ ry ]. The array index cIdx specifies the colour component; cIdx is equal to 0 for luma, equal to 1 for Cb, and equal to 2 for Cr. The array indices rx and ry specify the location ( rx, ry ) of the considered SAO unit relative to the top-left SAO unit of the picture.

**sao\_run\_diff** specifies the saoRun of current SAO unit if the current row is the first row, otherwise specifies the difference between the run of current SAO unit and the run of the above SAO unit. When saoRun is greater than or equal to 0, the syntax elements in sao\_offset\_vlc() are derived from the corresponding syntax elements of the left SAO unit. When ry is equal to 0, the length of the sao\_run\_diff syntax element is Ceil( Log2(sao\_num\_lcu\_in\_width\_minus1 − rx + 2) ) bits.

**sao\_merge\_up\_flag** equal to 1 specifies that the syntax elements in sao\_offset\_vlc() are derived from the corresponding syntax elements of the above SAO unit; equal to 0 specifies that the syntax elements in sao\_offset\_vlc() are not derived from the corresponding syntax elements of the above SAO unit. When sao\_merge\_up\_flag is not present, sao\_merge\_up\_flag is inferred to be equal to 0.

#### **7.4.2.8** **Sample adaptive offset VLC semantics**

**sao\_type\_idx**[ cIdx ][ rx ][ ry ] indicates the offset type as specified in of current SAO unit at position rx and ry for the colour component cIdx.

When sao\_type\_idx[ cIdx ][ rx ][ ry ] is not present, it is inferred as follows.

* If sao\_merge\_up\_flag is equal to 1, sao\_type\_idx[ cIdx ][ rx ][ ry ] is set equal to sao\_unit**\_**type\_idx[ cIdx ][ rx ][ ry − 1 ].
* Otherwise, sao\_type\_idx[ cIdx ][ rx ][ ry ] is set equal to sao\_type\_idx[ cIdx ][ rx − 1 ][ ry ].

Table 7‑6 – Specification of the edge type for SAO

|  |  |
| --- | --- |
| **sao\_type\_idx[ cIdx ][ rx ][ ry ]** | **Edge type (informative)** |
| 0 | Not applied |
| 1 | 1D 0-degree edge |
| 2 | 1D 90-degree edge |
| 3 | 1D 135-degree edge |
| 4 | 1D 45-degree edge |
| 5 | Band |

**sao\_band\_position**[ cIdx ][ rx ][ ry ] indicates the displacement of the band offset of the pixel range when sao\_ type\_idx[ cIdx ][ rx ][ ry ] is equal to 5.

When sao\_band\_position[ cIdx ][ rx ][ ry ] is not present it is inferred as follows.

* If sao\_merge\_up\_flag is equal to 1, sao\_band\_position[ cIdx ][ rx ][ ry ] is set equal to sao\_band\_position[ cIdx ][ rx ][ ry − 1 ].
* Otherwise, sao\_band\_position[ cIdx ][ rx ][ ry ] is set equal to sao\_band\_position[ cIdx ][ rx − 1 ][ ry ].

**sao**\_**offset**[ cIdx ][ rx ][ ry ][ i ] indicates the offset value of i-th category of current SAO unit at position rx and ry for the colour component cIdx.

The variable bitDepth is derived as follows.

* If cIdx is equal to 0, bitDepth is set equal to BitDepthY..
* Otherwise (cIdx is equal to1 or 2), bitDepth is set equal to BitDepthC.

It is a requirement of bitstream conformance that when sao\_type\_idx[ cIdx ][ rx ][ ry ] is not equal to 5, the values of sao\_ offset[ cIdx ][ rx ][ ry ][ i ] shall be in the range of 0 to ( 1<< ( Min( bitDepth, 10 ) − 5 ) ) − 1, inclusive and otherwise, the values of sao\_offset[ cIdx ][ rx ][ ry ][ i ] shall be in the range of −( 1<< ( Min( bitDepth, 10 ) − 5 ) ) to ( 1<< ( Min( bitDepth, 10 ) − 5 ) ) − 1, inclusive.

When sao\_offset[ cIdx ][ rx ][ ry ][ i ] is not present, it is inferred as follows.

* If sao\_merge\_up\_flag is equal to 1, sao\_offset[ cIdx ][ rx ][ ry ][ i ] is set equal to sao\_offset[ cIdx ][ rx ][ ry − 1 ][ i ].
* Otherwise, sao\_offset[ cIdx ][ rx ][ ry ][ i ] is set equal to sao\_offset[ cIdx ][ rx − 1 ][ ry ][ i ].

The variable offsetSign is derived as follows.

* If sao\_type\_idx[ cIdx ][ rx ][ ry ] is less than 5 and i is larger than 1,offsetSign is set to −1.
* Otherwise (sao\_type\_idx[ cIdx ][ rx ][ ry ] is equal to 5 or i is smaller than 2), offsetSign is set to 1.

The array SaoOffsetVal is derived as follows.

SaoOffsetVal[ cIdx ][ rx ][ ry ][ 0 ] = 0 (7‑36)

SaoOffsetVal[ cIdx ][ rx ][ ry ][ i + 1 ] =   
 offsetSign\*sao\_offset[ cIdx ][ rx ][ ry ][ i ] << ( bitDepth – Min( bitDepth, 10 ) ) (7‑37)

#### **Sample adaptive offset unit CABAC semantics**

**sao\_merge\_left\_flag** equal to 1 specifies that the syntax elements sao\_type\_idx, sao\_band\_position and sao\_offset are derived from the corresponding syntax elements of the left SAO unit. sao\_merge\_left\_flag equal to 0 specifies that these syntax elements are not derived from the left SAO unit syntax elements. When sao\_merge\_left\_flag is not present, is inferred to be equal to 0; If slice\_sao\_interleaving\_flagequal to 1, the SAO unit equals to coding treeblock.

### 8.7.2 Sample adaptive offset process

A sample adaptive offset process shall be conditionally performed after the completion of the deblocking filter process for the decoded picture. This process is invoked when both sample\_adaptive\_offset\_enabled\_flag and aps\_sample\_adaptive\_offset\_flag are equal to 1.

This process is performed on a region basis which is aligned with the coding treeblock boundaries after the completion of the picture construction process prior to sample adaptive offset process for the entire decoded process.

When slice\_sao\_interleaving\_flag is equal to 1,

Log2NumCtbInSaoUnitWidth = 0

Log2NumCtbInSaoUnitHeight = 0.

The sample adaptive offset process is invoked with the region equal to entire picture here.

Depending on slice\_sample\_adaptive\_offset\_flag, sao\_cb\_enable\_flag and sao\_cr\_enable\_flag, the following applies:

– When slice\_sample\_adaptive\_offset\_flag is equal to 1 the following applies:

* + The sample adaptive offset process for luma region as specified in subclause is invoked with the component index set equal to 0, the coding treeblock location ( 0, 0 ), the largest CU location ( 0, 0 ), the variable saoWidthInCtbs set equal to PicWidthInCtbs, and the variable saoHeightInCtbs set equal to PicHeightInCtbs, and the modified luma picture buffer of current region as output.
  + When sao\_cb\_enable\_flag is equal to 1, the sample adaptive offset process for colour component Cb as specified in subclause is invoked with the component index set equal to 1, the coding treeblock location ( 0, 0 ), the largest CU location ( 0, 0 ), the variable saoWidthInCtbs set equal to PicWidthInCtbs, and the variable saoHeightInCtbs set equal to PicHeightInCtbs, and the modified chroma picture buffer for Cb of current region as output.
  + If sao\_cb\_enable\_flag is equal to 1, the sample adaptive offset process for colour component Cr as specified in subclause is invoked with the component index set equal to 2, the coding treeblock location ( 0, 0 ), the largest CU location ( 0, 0 ), the variable saoWidthInCtbs set equal to PicWidthInCtbs, and the variable saoHeightInCtbs set equal to PicHeightInCtbs, and the modified chroma picture buffer for Cr of current region as output.

#### **8.7.2.1 Modification process for sample adaptive offset**

Inputs to this process are:

– a variable cIdx specifying colour component index,

– a pair of variables ( rx, ry ) specifying the coding treeblock location,

– a sample location ( x0, y0 ) specifying the top-left sample of the current largest coding unit,

– a variable saoWidthInCtbs specifying the number of largest CUs in width of the current region

– a variable saoHeightInCtbs specifying the number of largest CUs in height of the current region.

Output of this process is a modified picture buffer for the colour component cIdx.

The modified picture buffer for the colour component cIdx is derived as follows.

1. Variable nS is set equal to ( 1 << Log2CtbSize ).
2. Modification process for samples in a largest coding unit covering ( iC, jC ) with iC = x0 + k\*nS for k = 0..saoWidthInCtb − 1 and jC = y0 + j\*nS for j = 0..saoHeightInCtb − 1 specified in subclause is invoked with the colour component index cIdx, the top-left sample of current largest coding unit ( iC, jC ), the coding treeblock location ( rx, ry ), the offset array saoValueArray set equal to SaoOffsetVal[ cIdx ][ rx >> Log2NumCtbInSaoUnitWidth ][ ry >> Log2NumCtbInSaoUnitHeight ] and the largest coding unit size nS as inputs and the output is the modified picture buffer of colour component cIdx.

##### 8.7.2.1.1 Modification process for luma and chroma samples

Inputs to this process are:

– a variable cIdx specifying colour component index,

– a sample position ( xC, yC ),

– a pair of variables ( rx, ry ) specifying the coding treeblock location,

– an array saoValueArray specifying offset values,

– a block size nS.

Output of this process is a modified picture buffer for the colour component cIdx.

Let recSaoPicture represents the processed sample array of the current picture of colour component cIdx and saoTypeIdx is set equal to sao\_type\_idx[ cIdx ][ rx >> Log2NumCtbInSaoUnitWidth ][ ry >> Log2NumCtbInSaoUnitHeight ].

Variable bitDepth is set equal to BitDepthY if cIdx is equal to 0, otherwise, set equal to BitDepthC.

Depending on the value of saoTypeIdx, the following applies:

– If saoTypeIdx is equal to one of the values of 1, 2, 3 or 4, the following ordered steps apply:

1. Arrays hPos[2] and vPos[2] are specified in .
2. A variable edgeIdx is derived as follows.

edgeIdx = 2 + ∑k( Sign( recPicture[ xC + i, yC + j ] –

recPicture[ xC + i + hPos[ k ], yC + j + vPos[ k ] ] ) ) with k = 0..1 (8‑351)

1. When any of the following conditions is true, edgeIdx is set equal to 2.

* The pixel at position (xC + i + hPos[ k ], yC + j + vPos[ k ]) is outside picture boundary
* slice\_loop\_filter\_across\_slices\_enabled\_flag is equal to 0 and the pixel at position (xC + i + hPos[ k ], yC + j + vPos[ k ]) belongs to a different slice.
* loop\_filter\_across\_tiles\_enabled\_flag is equal to 0 and the pixel at position (xC + i + hPos[ k ], yC + j + vPos[ k ]) belongs to a different tile.

1. The reconstructed picture buffer is modified as follows:

* If pcm\_loop\_filter\_disable\_flag is equal to 1 and the recPicture[ xC + i, yC + j ] is a reconstructed sample of an I\_PCM block or qpprime\_y\_zero\_transquant\_bypass\_flag is equal to 1 and QP’Y is equal to 0, the recSaoPicture[ xC + I yC + j ] is derived as

recSaoPicture[ xC + i, yC + j ] = recPicture[ xC + i, yC + j ] (8‑352)

with i = 0..nS−1 and j = 0..nS−1.

* Otherwise,

recSaoPicture[ xC + i, yC + j ] = recPicture[ xC + i, yC + j ] + saoValueArray[ edgeTable[ edgeIdx ] ] (8‑353)

with i = 0..nS−1 and j = 0..nS−1 where edgeTable[5] = { 1, 2, 0, 3, 4}.

* Otherwise, if saoTypeIdx is equal to 5, the following ordered steps apply:

1. If cIdx is equal to 0, the variable bandShift is set equal to BitDepthY − 5
2. Otherwise (cIdx is not equal to 0), bandShift is set equal to BitDepthC − 5.
3. The variable saoLeftClass is set equal to sao\_band\_position[ cIdx ][ rx >> Log2NumCtbInSaoUnitWidth  ][ ry >> Log2NumCtbInSaoUnitHeight].
4. The vector bandTable is defined with 32 elements and all elements are initially set to 0. Then, 4 of its elements (indicating the position of bands for which an offset is explicitly transmitted) are modified as follows:

for( i = 0; i < 4; i++ )  
 bandTable[ (i + saoLeftClass) & 0x1F ] = i + 1 (8‑354)

1. The reconstructed picture buffer recSaoPicture is modified as follows.

bandIdx = ( recPicture[ xC + i, yC + j ] >> bandShift )  
recSaoPicture[ xC + i, yC + j ] = recPicture[ xC + i, yC + j ] + saoValueArray[ bandTable[ bandIdx ] ] (8‑355)  
 with i = 0..nS−1 and j = 0..nS−1

– Otherwise (sao\_type\_idx[ cIdx ][ rx >> Log2NumCtbInSaoUnitWidth ][ ry >> Log2NumCtbInSaoUnitHeight ] is equal to 0), the following applies:

recSaoPicture[ xC + i, yC + j ] = recPicture[ xC + i, yC + j ] with i = 0..nS−1 and j = 0..nS−1 (8‑356)

Table 8‑13 – Specification of hPos[2] and vPos[2] according to the type of sample adaptive offset process

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| sao\_type\_idx[ cIdx ][ rx ][ ry ] | 1 | 2 | 3 | 4 |
| hPos[0] | −1 | 0 | −1 | 1 |
| hPos[1] | 1 | 0 | 1 | −1 |
| vPos[0] | 0 | −1 | −1 | −1 |
| vPos[1] | 0 | 1 | 1 | 1 |