**7.3.4 Slice data syntax**

|  |  |
| --- | --- |
| slice\_data( ) { | Descriptor |
| CtbAddrRS = SliceCtbAddrRS |  |
| CtbAddrTS = CtbAddrRStoTS[ CtbAddrRS ] |  |
| moreDataFlag = 1 |  |
| if( adaptive\_loop\_filter\_flag && alf\_cu\_control\_flag ) |  |
| AlfCuFlagIdx = −1 |  |
| do { |  |
| xCtb = InverseRasterScan( CtbAddrRS, CtbSize, CtbSize, pic\_width\_in\_luma\_samples, 0 ) |  |
| yCtb = InverseRasterScan( CtbAddrRS, CtbSize, CtbSize, pic\_width\_in\_luma\_samples, 1 ) |  |
| NumPCMBlock = 0 |  |
| CtbAddrInSlice = CtbAddrRS − ( slice\_address >> SliceGranularity ) |  |
| ~~for( cIdx = 0; cIdx < 3; cIdx++ ) {~~ |  |
| ~~if( slice\_sample\_adaptive\_offset\_flag[ cIdx ] )~~ |  |
| ~~sao\_param( xCtb, yCtb, cIdx )~~ |  |
| all\_sao\_merge\_flags(xCtb, yCtb) |  |
| all\_sao\_type\_idx(xCtb, yCtb) |  |
| all\_sao\_offset(xCtb, yCtb) |  |
| ~~}~~ |  |
| moreDataFlag = coding\_tree( xCtb, yCtb, Log2CtbSize, 0 ) |  |
| CtbAddrTS++ |  |
| CtbAddrRS = CtbAddrTStoRS[ CtbAddrTS ] |  |
| if( moreDataFlag && ( ( tiles\_or\_entropy\_coding\_sync\_idc = = 1 &&  TileId[ CtbAddrTS ] != TileId[ CtbAddrTS − 1 ] ) | |  ( tiles\_or\_entropy\_coding\_sync\_idc = = 2 && CtbAddrTS % PicWidthInCtbs = = 0 ) ) ) |  |
| rbsp\_alignment( ) |  |
| } while( moreDataFlag ) |  |
| } |  |

**~~7.3.4.1 Sample adaptive offset parameter syntax~~**(remove the table)

**7.3.4.1 Sample adaptive offset merge flags syntax**

|  |  |
| --- | --- |
| all\_sao\_merge\_flags( rx, ry, cIdx ){ | Descriptor |
| if( rx > 0 ) { |  |
| leftCtbInSlice = CtbAddrInSlice > 0 |  |
| leftCtbInTile = TileId[ CtbAddrTS ] = = TileId[ CtbAddrRStoTS[ CtbAddrRS − 1 ] ] |  |
| if( leftCtbInSlice && leftCtbInTile ) { |  |
| for( cIdx = 0; cIdx < 3; cIdx++ ) { |  |
| if( slice\_sample\_adaptive\_offset\_flag[ cIdx ] ) |  |
| **sao\_merge\_left\_flag[cIdx]** | ae(v) |
| } |  |
| } |  |
| } |  |
| if( ry > 0) { |  |
| upCtbInSlice = (CtbAddrTS – CtbAddrRStoTS[CtbAddrRS − PicWidthInCtbs]) <= CtbAddrInSlice |  |
| upCtbInTile = TileId[ CtbAddrTS ] = = TileId[ CtbAddrRStoTS[ CtbAddrRS − PicWidthInCtbs ] ] |  |
| if( upCtbInSlice && upCtbInTile ) { |  |
| for( cIdx = 0; cIdx < 3; cIdx++ ) { |  |
| if( slice\_sample\_adaptive\_offset\_flag[ cIdx ] && !sao\_merge\_left\_flag[cIdx]) |  |
| **sao\_merge\_up\_flag[cIdx]** | ae(v) |
| } |  |
| } |  |
| } |  |
| } |  |

**7.3.4.2 Sample adaptive offset type index syntax**

|  |  |
| --- | --- |
| all\_sao\_type\_idx(rx, ry){ | Descriptor |
| for( cIdx = 0; cIdx < 3; cIdx++ ) { |  |
| if( !sao\_merge\_up\_flag[cIdx] && !sao\_merge\_left\_flag[cIdx] ) { |  |
| **sao\_type\_idx\_prefix**[ cIdx ][ rx ][ ry ] | ae(v) |
| } |  |
| } |  |
| for( cIdx = 0; cIdx < 3; cIdx++ ) { |  |
| if(sao\_type\_idx\_prefix[ cIdx ][ rx ][ ry ]  && !sao\_merge\_up\_flag[cIdx]  && !sao\_merge\_left\_flag[cIdx] )  { |  |
| **sao\_type\_idx\_suffix**[ cIdx ][ rx ][ ry ] | ae(v) |
| } |  |
| } |  |
| } |  |

**7.3.4.3 Sample adaptive offset syntax**

|  |  |
| --- | --- |
| all\_sao\_offset( rx, ry){ | Descriptor |
| for( cIdx = 0; cIdx < 3; cIdx++ ) { |  |
| if( !sao\_merge\_up\_flag[cIdx] && !sao\_merge\_left\_flag[cIdx] ) { |  |
| if( SaoTypeIdx[ cIdx ][ rx ][ ry ] = =5 ) |  |
| **sao\_band\_position**[ cIdx ][ rx ][ ry ] | ae(v) |
| if(SaoTypeIdx[ cIdx ][ rx ][ ry ] != 0 ) |  |
| for( i = 0; i < 4; i++ ) |  |
| **sao\_offset\_abs**[ cIdx ][ rx][ ry ][ i ] | ae(v) |
| if(SaoTypeIdx[ cIdx ][ rx ][ ry ] = = 5 ) { |  |
| for( i = 0; i < 4; i++ ) { |  |
| if( sao\_offset\_abs[ cIdx ][ rx ][ ry ] != 0 ) |  |
| **sao\_offset\_sign**[ cIdx ][ rx ][ ry ][ i ] | ae(v) |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |

#### 7.4.4.1 Sample adaptive offset parameter semantics

**sao\_merge\_left\_flag** equal to 1 specifies that the syntax elements sao\_type\_idx\_prefix, sao\_type\_idx\_suffix, sao\_band\_position, sao\_offset\_abs and sao\_offset\_sign are derived from the corresponding syntax elements of the left coding tree block; equal to 0 specifies that these syntax elements are not derived from the corresponding syntax elements of the left coding tree block. When sao\_merge\_left\_flag is not present, it is inferred to be equal to 0.

**sao\_merge\_up\_flag** equal to 1 specifies that the syntax elements sao\_type\_idx\_prefix, sao\_type\_idx\_suffix, sao\_band\_position, sao\_offset\_abs and sao\_offset\_sign are derived from the corresponding syntax elements of the above coding tree block; equal to 0 specifies that these syntax elements are not derived from the corresponding syntax elements of the above coding tree block. When sao\_merge\_up\_flag is not present, it is inferred to be equal to 0.

**~~sao\_type\_idx~~**~~[ asdascIdx ][ rx ][ ry ] indicates the offset type as specified in Table 7 9Error! Reference source not found. of current coding tree block 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\_left\_flag is equal to 1, sao\_type\_idx[ cIdx ][ rx ][ ry ] is set equal to sao\_type\_idx[ cIdx ][ rx − 1 ][ ry ].~~

~~– Otherwise, if sao\_merge\_up\_flag is equal to 1, sao\_type\_idx[ cIdx ][ rx ][ ry ] is set equal to sao\_type\_idx[ cIdx ][ rx ][ ry − 1 ].~~

~~– Otherwise, sao\_type\_idx[ cIdx ][ rx ][ ry ] is set equal to 0.~~

**sao\_type\_idx\_prefix**[ cIdx ][ rx ][ ry ] indicates the prefix of the offset type of current coding tree block at position rx and ry for the colour component cIdx.

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

* If sao\_merge\_left\_flag is equal to 1, sao\_type\_idx\_prefix[ cIdx ][ rx ][ ry ] is set equal to sao\_type\_idx\_prefix[ cIdx ][ rx − 1 ][ ry  ].
* Otherwise, if sao\_merge\_up\_flag is equal to 1, sao\_type\_idx\_prefix[ cIdx ][ rx ][ ry ] is set equal to sao\_type\_idx\_prefix[ cIdx ][ rx ][ ry − 1 ].

Otherwise, sao\_type\_idx\_prefix[ cIdx ][ rx ][ ry ] is set equal to 0.

**sao\_type\_idx\_suffix**[ cIdx ][ rx ][ ry ] indicates the suffix of the offset type of current coding tree block at position rx and ry for the colour component cIdx.

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

* If sao\_merge\_left\_flag is equal to 1, sao\_type\_idx\_suffix[ cIdx ][ rx ][ ry ] is set equal to sao\_type\_idx\_suffix[ cIdx ][ rx − 1 ][ ry  ].
* Otherwise, if sao\_merge\_up\_flag is equal to 1, sao\_type\_idx\_suffix[ cIdx ][ rx ][ ry ] is set equal to sao\_type\_idx\_suffix[ cIdx ][ rx ][ ry − 1 ].

Otherwise, sao\_type\_idx\_suffix[ cIdx ][ rx ][ ry ] is set equal to 0.

The offset type of current coding tree block at position rx and ry for the colour component cIdx SaoTypeIdx is derived as follows:

* If sao\_type\_idx\_prefix is equal to 0, SaoTypeIdx is set equal to 0.
* Otherwise SaoTypeIdx is set equal to sao\_type\_idx\_suffix plus 1.

**Table 9 7 – Values of variable initValue for sao\_type\_idx\_prefix ctxIdx**

|  |  |
| --- | --- |
| **Initialization value** | **Sao\_type\_idx\_prefix ctxIdx** |
| **0** |
| **initValue** | 61 |

**Table 9 34 – Syntax elements and associated types of binarization, maxBinIdxCtx, ctxIdxTable, and ctxIdxOffset**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| sao\_type\_idx\_prefix | 0 | FL, cMax = 1 | 0 | Table 9-7 | 0 |
| 1 | 0 | Table 9-7 | 1 |
| 2 | 0 | Table 9-7 | 2 |
| sao\_type\_idx\_suffix | 0 | FL, cMax = 3 | na | na | na, (uses Decode Bypass) |
| 1 | na | na | na, (uses Decode Bypass) |
| 2 | na | na | na, (uses Decode Bypass) |

**Table 9 39 – Assignment of ctxIdxInc to binIdx for all ctxIdxTable and ctxIdxOffset values**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| sao\_type\_idx\_prefix | Table 9-7 | 0 | 0 | na | na | na | na |
| 1 | 0 | na | na | na | na |
| 2 | 0 | na | na | na | na |

# Bypass concatenation when merge flags are combined:

**7.3.4 Slice data syntax**

|  |  |
| --- | --- |
| slice\_data( ) { | Descriptor |
| CtbAddrRS = SliceCtbAddrRS |  |
| CtbAddrTS = CtbAddrRStoTS[ CtbAddrRS ] |  |
| moreDataFlag = 1 |  |
| if( adaptive\_loop\_filter\_flag && alf\_cu\_control\_flag ) |  |
| AlfCuFlagIdx = −1 |  |
| do { |  |
| xCtb = InverseRasterScan( CtbAddrRS, CtbSize, CtbSize, pic\_width\_in\_luma\_samples, 0 ) |  |
| yCtb = InverseRasterScan( CtbAddrRS, CtbSize, CtbSize, pic\_width\_in\_luma\_samples, 1 ) |  |
| NumPCMBlock = 0 |  |
| CtbAddrInSlice = CtbAddrRS − ( slice\_address >> SliceGranularity ) |  |
| ~~for( cIdx = 0; cIdx < 3; cIdx++ ) {~~ |  |
| ~~if( slice\_sample\_adaptive\_offset\_flag[ cIdx ] )~~ |  |
| sao\_param( xCtb, yCtb~~, cIdx~~ ) |  |
| moreDataFlag = coding\_tree( xCtb, yCtb, Log2CtbSize, 0 ) |  |
| CtbAddrTS++ |  |
| CtbAddrRS = CtbAddrTStoRS[ CtbAddrTS ] |  |
| if( moreDataFlag && ( ( tiles\_or\_entropy\_coding\_sync\_idc = = 1 &&  TileId[ CtbAddrTS ] != TileId[ CtbAddrTS − 1 ] ) | |  ( tiles\_or\_entropy\_coding\_sync\_idc = = 2 && CtbAddrTS % PicWidthInCtbs = = 0 ) ) ) |  |
| rbsp\_alignment( ) |  |
| } while( moreDataFlag ) |  |
| } |  |

**7.3.4.1 Sample adaptive offset parameter syntax**

|  |  |
| --- | --- |
| sao\_param( rx, ry~~, cIdx~~ ){ | Descriptor |
| if( rx > 0 ) { |  |
| leftCtbInSlice = CtbAddrInSlice > 0 |  |
| leftCtbInTile = TileId[ CtbAddrTS ] = = TileId[ CtbAddrRStoTS[ CtbAddrRS − 1 ] ] |  |
| if( leftCtbInSlice && leftCtbInTile ) |  |
| **sao\_merge\_left\_flag** | ae(v) |
| } |  |
| if( ry > 0 && !sao\_merge\_left\_flag ) { |  |
| upCtbInSlice = (CtbAddrTS – CtbAddrRStoTS[CtbAddrRS − PicWidthInCtbs]) <= CtbAddrInSlice |  |
| upCtbInTile = TileId[ CtbAddrTS ] = = TileId[ CtbAddrRStoTS[ CtbAddrRS − PicWidthInCtbs ] ] |  |
| if( upCtbInSlice && upCtbInTile ) |  |
| **sao\_merge\_up\_flag** | ae(v) |
| } |  |
| if( !sao\_merge\_up\_flag && !sao\_merge\_left\_flag ) { |  |
| for( cIdx = 0; cIdx < 3; cIdx++ ) { |  |
| if( slice\_sample\_adaptive\_offset\_flag[ cIdx ] ) |  |
| **sao\_type\_idx\_prefix**[ cIdx ][ rx ][ ry ] | ae(v) |
| for( cIdx = 0; cIdx < 3; cIdx++ ) { |  |
| if( slice\_sample\_adaptive\_offset\_flag[ cIdx ] && sao\_type\_idx\_prefix[ cIdx ][ rx ][ ry ] ) |  |
| **sao\_type\_idx\_suffix**[ cIdx ][ rx ][ ry ] | ae(v) |
| } |  |
| for( cIdx = 0; cIdx < 3; cIdx++ ) { |  |
| if( slice\_sample\_adaptive\_offset\_flag[ cIdx ] ) |  |
| if( sao\_type\_idx[ cIdx ][ rx ][ ry ] = =5 ) |  |
| **sao\_band\_position**[ cIdx ][ rx ][ ry ] | ae(v) |
| if( sao\_type\_idx[ cIdx ][ rx ][ ry ] != 0 ) |  |
| for( i = 0; i < 4; i++ ) |  |
| **sao\_offset\_abs**[ cIdx ][ rx][ ry ][ i ] | ae(v) |
| if( sao\_type\_idx[ cIdx ][ rx ][ ry ] = = 5 ) { |  |
| for( i = 0; i < 4; i++ ) { |  |
| if( sao\_offset\_abs[ cIdx ][ rx ][ ry ][ i ] != 0 ) |  |
| **sao\_offset\_sign**[ cIdx ][ rx ][ ry ][ i ] | ae(v) |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |