*In 7.3.2.2.1 (General sequence parameter set RBSP syntax), replace the syntax table with the following:*

|  |  |
| --- | --- |
| seq\_parameter\_set\_rbsp( ) { | Descriptor |
| **sps\_video\_parameter\_set\_id** | u(4) |
| **sps\_max\_sub\_layers\_minus1** | u(3) |
| **sps\_temporal\_id\_nesting\_flag** | u(1) |
| profile\_tier\_level( 1, sps\_max\_sub\_layers\_minus1 ) |  |
| **sps\_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) |
| **conformance\_window\_flag** | u(1) |
| if( conformance\_window\_flag ) { |  |
| **conf\_win\_left\_offset** | ue(v) |
| **conf\_win\_right\_offset** | ue(v) |
| **conf\_win\_top\_offset** | ue(v) |
| **conf\_win\_bottom\_offset** | ue(v) |
| } |  |
| **bit\_depth\_luma\_minus8** | ue(v) |
| **bit\_depth\_chroma\_minus8** | ue(v) |
| **log2\_max\_pic\_order\_cnt\_lsb\_minus4** | ue(v) |
| **sps\_sub\_layer\_ordering\_info\_present\_flag** | u(1) |
| for( i = ( sps\_sub\_layer\_ordering\_info\_present\_flag ? 0 : sps\_max\_sub\_layers\_minus1 );  i <= sps\_max\_sub\_layers\_minus1; i++ ) { |  |
| **sps\_max\_dec\_pic\_buffering\_minus1**[ i ] | ue(v) |
| **sps\_max\_num\_reorder\_pics**[ i ] | ue(v) |
| **sps\_max\_latency\_increase\_plus1**[ i ] | ue(v) |
| } |  |
| **log2\_min\_luma\_coding\_block\_size\_minus3** | ue(v) |
| **log2\_diff\_max\_min\_luma\_coding\_block\_size** | ue(v) |
| **log2\_min\_luma\_transform\_block\_size\_minus2** | ue(v) |
| **log2\_diff\_max\_min\_luma\_transform\_block\_size** | ue(v) |
| **max\_transform\_hierarchy\_depth\_inter** | ue(v) |
| **max\_transform\_hierarchy\_depth\_intra** | ue(v) |
| **scaling\_list\_enabled\_flag** | u(1) |
| if( scaling\_list\_enabled\_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\_disabled\_flag** | u(1) |
| } |  |
| **num\_short\_term\_ref\_pic\_sets** | ue(v) |
| for( i = 0; i < num\_short\_term\_ref\_pic\_sets; i++) |  |
| st\_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\_enabled\_flag** | u(1) |
| **strong\_intra\_smoothing\_enabled\_flag** | u(1) |
| **vui\_parameters\_present\_flag** | u(1) |
| if( vui\_parameters\_present\_flag ) |  |
| vui\_parameters( ) |  |
| **sps\_extension\_present\_flag** | u(1) |
| if( sps\_extension\_present\_flag ) { |  |
| **sps\_range\_extension\_flag** | u(1) |
| **sps\_multilayer\_extension\_flag** | u(1) |
| **sps\_3d\_extension\_flag** | u(1) |
| **sps\_scc\_extension\_flag** | u(1) |
| **sps\_hdrwcg\_extension\_flag** | u(1) |
| **sps\_extension\_3bits** | u(3) |
| } |  |
| if( sps\_range\_extension\_flag ) |  |
| sps\_range\_extension( ) |  |
| if( sps\_multilayer\_extension\_flag ) |  |
| sps\_multilayer\_extension( ) /\* specified in Annex F \*/ |  |
| if( sps\_3d\_extension\_flag ) |  |
| sps\_3d\_extension( ) /\* specified in I\*/ |  |
| if( sps\_scc\_extension\_flag ) |  |
| sps\_scc\_extension( ) |  |
| if( sps\_extension\_3bits ) |  |
| while( more\_rbsp\_data( ) ) |  |
| **sps\_extension\_data\_flag** | u(1) |
| rbsp\_trailing\_bits( ) |  |
| } |  |

*Add a new 7.3.2.2.4 syntax table (sps\_hdrwcg\_extension):*

|  |  |
| --- | --- |
| sps\_hdrwcg\_extension( ) { | Descriptor |
| **hdr\_output\_transfer\_characteristics\_present\_flag** | u(1) |
| if( hdr\_output\_transfer\_characteristics\_present\_flag ) |  |
| **hdr\_output\_transfer\_characteristics** | u(8) |
| **reshaper\_range\_id** | u(2) |
| } |  |

*In 7.3.2.3.1 (General picture parameter set RBSP syntax), replace the syntax table with the following:*

|  |  |
| --- | --- |
| pic\_parameter\_set\_rbsp( ) { | Descriptor |
| **pps\_pic\_parameter\_set\_id** | ue(v) |
| **pps\_seq\_parameter\_set\_id** | ue(v) |
| **dependent\_slice\_segments\_enabled\_flag** | u(1) |
| **output\_flag\_present\_flag** | u(1) |
| **num\_extra\_slice\_header\_bits** | u(3) |
| **sign\_data\_hiding\_enabled\_flag** | u(1) |
| **cabac\_init\_present\_flag** | u(1) |
| **num\_ref\_idx\_l0\_default\_active\_minus1** | ue(v) |
| **num\_ref\_idx\_l1\_default\_active\_minus1** | ue(v) |
| **init\_qp\_minus26** | se(v) |
| **constrained\_intra\_pred\_flag** | u(1) |
| **transform\_skip\_enabled\_flag** | u(1) |
| **cu\_qp\_delta\_enabled\_flag** | u(1) |
| if( cu\_qp\_delta\_enabled\_flag ) |  |
| **diff\_cu\_qp\_delta\_depth** | ue(v) |
| **pps\_cb\_qp\_offset** | se(v) |
| **pps\_cr\_qp\_offset** | se(v) |
| **pps\_slice\_chroma\_qp\_offsets\_present\_flag** | u(1) |
| **weighted\_pred\_flag** | u(1) |
| **weighted\_bipred\_flag** | u(1) |
| **transquant\_bypass\_enabled\_flag** | u(1) |
| **tiles\_enabled\_flag** | u(1) |
| **entropy\_coding\_sync\_enabled\_flag** | u(1) |
| if( tiles\_enabled\_flag ) { |  |
| **num\_tile\_columns\_minus1** | ue(v) |
| **num\_tile\_rows\_minus1** | ue(v) |
| **uniform\_spacing\_flag** | u(1) |
| if( !uniform\_spacing\_flag ) { |  |
| for( i = 0; i < num\_tile\_columns\_minus1; i++ ) |  |
| **column\_width\_minus1**[ i ] | ue(v) |
| for( i = 0; i < num\_tile\_rows\_minus1; i++ ) |  |
| **row\_height\_minus1**[ i ] | ue(v) |
| } |  |
| **loop\_filter\_across\_tiles\_enabled\_flag** | u(1) |
| } |  |
| **pps\_loop\_filter\_across\_slices\_enabled\_flag** | u(1) |
| **deblocking\_filter\_control\_present\_flag** | u(1) |
| if( deblocking\_filter\_control\_present\_flag ) { |  |
| **deblocking\_filter\_override\_enabled\_flag** | u(1) |
| **pps\_deblocking\_filter\_disabled\_flag** | u(1) |
| if( !pps\_deblocking\_filter\_disabled\_flag ) { |  |
| **pps\_beta\_offset\_div2** | se(v) |
| **pps\_tc\_offset\_div2** | se(v) |
| } |  |
| } |  |
| **pps\_scaling\_list\_data\_present\_flag** | u(1) |
| if( pps\_scaling\_list\_data\_present\_flag ) |  |
| scaling\_list\_data( ) |  |
| **lists\_modification\_present\_flag** | u(1) |
| **log2\_parallel\_merge\_level\_minus2** | ue(v) |
| **slice\_segment\_header\_extension\_present\_flag** | u(1) |
| **pps\_extension\_present\_flag** | u(1) |
| if( pps\_extension\_present\_flag ) { |  |
| **pps\_range\_extension\_flag** | u(1) |
| **pps\_multilayer\_extension\_flag** | u(1) |
| **pps\_3d\_extension\_flag** | u(1) |
| **pps\_scc\_extension\_flag** | u(1) |
| **pps\_hdrwcg\_extension\_flag** | u(1) |
| **pps\_extension\_3bits** | u(3) |
| } |  |
| if( pps\_range\_extension\_flag ) |  |
| pps\_range\_extension( ) |  |
| if( pps\_multilayer\_extension\_flag ) |  |
| pps\_multilayer\_extension( ) /\* specified in Annex F \*/ |  |
| if( pps\_3d\_extension\_flag ) |  |
| pps\_3d\_extension( ) /\* specified in Annex I \*/ |  |
| if( pps\_scc\_extension\_flag ) |  |
| pps\_scc\_extension( ) |  |
| if( pps\_hdrwcg\_extension\_flag ) |  |
| pps\_hdrwcg\_extension( ) |  |
| if( pps\_extension\_3bits ) |  |
| while( more\_rbsp\_data( ) ) |  |
| **pps\_extension\_data\_flag** | u(1) |
| rbsp\_trailing\_bits( ) |  |
| } |  |

*Add a new 7.3.2.3.4 syntax table (pps\_hdrwcg\_extension):*

|  |  |
| --- | --- |
| pps\_hdrwcg\_extension( ) { | Descriptor |
| hdr\_reshaping\_pps\_table( ) |  |
| } |  |

*Add a new 7.3.2.3.5 syntax table (hdr\_reshaping\_pps\_table):*

|  |  |
| --- | --- |
| hdr\_reshaping\_pps\_table( ) { | Descriptor |
| **reshape\_present\_flag** | ue(v) |
| if( reshape\_present\_flag ) { |  |
| **reshape\_chroma\_crosschannel\_flag** | u(1) |
| **reshape\_input\_luma\_bit\_depth\_minus8** | ue(v) |
| **reshape\_input\_chroma\_bit\_depth\_minus8** | ue(v) |
| **reshape\_output\_luma\_bit\_depth\_minus8** | ue(v) |
| **reshape\_output\_chroma\_bit\_depth\_minus8** | ue(v) |
| **reshape\_default\_flag** | u(1) |
| if( !reshape\_default\_flag ) { |  |
| **reshape\_scale\_int\_bit\_depth** | u(4) |
| **reshape\_offset\_int\_bit\_depth** | u(4) |
| **reshape\_scale\_frac\_bit\_depth** | u(4) |
| **reshape\_offset\_frac\_bit\_depth** | u(4) |
| **reshape\_negative\_scales\_present\_flag** | u(1) |
| } |  |
| **reshape\_num\_comps\_minus2** | ue(v) |
| for( c = 0; c < reshape\_num\_comps\_minus2 + 2; c++ ) { |  |
| **reshape\_num\_ranges\_minus1**[ c ] | ue(v) |
| **reshape\_equal\_ranges\_flag**[ c ] | u(1) |
| **reshape\_global\_offset\_val**[ c ] | u(v) |
| if( !reshape\_equal\_ranges\_flag[ c ] ) |  |
| for ( i = 0; i < reshape\_num\_ranges\_minus1[ c ] + 1; i++ ) |  |
| **reshape\_range\_val**[ c ][ i ] | u(v) |
| if ( c = = 0 ) { |  |
| **coeff\_log2\_offset\_minus2** | ue(v) |
| **reshape\_continuity\_flag** | u(1) |
| for( i = 0; i < reshape\_num\_ranges\_minus1[ c ] + 2; i++ ) { |  |
| **poly\_coef\_int**[ i ][ 0 ] | ue(v) |
| **poly\_coef\_frac**[ i ][ 0 ] | u(v) |
| } |  |
| if( reshape\_continuity\_flag = = 1 ) { |  |
| **poly\_coef\_int**[ 0 ][ 1 ] | se(v) |
| **poly\_coef\_frac**[ 0 ][ 1 ] | u(v) |
| } |  |
| } else { |  |
| for( i = 0; i < reshape\_num\_ranges\_minus1[ c ] + 1; i++ ) { |  |
| **reshape\_scale\_int\_val**[ c – 1 ][ i ] | u(v) |
| **reshape\_scale\_fract\_val**[ c – 1 ][ i ] | u(v) |
| } |  |
| if ( reshape\_chroma\_crosschannel\_flag ) { |  |
| **reshape\_offset1**[ c – 1 ] | u(16) |
| **reshape\_offset2**[ c – 1 ] | u(16) |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |

*In 7.4.3.2.1 (General sequence parameter set RBSP semantics), add the following semantics:*

**sps\_hdrwcg\_extension\_flag** equal to 1 specifies that the sps\_hdrwcg\_extension( ) syntax structure is present in the SPS RBSP syntax structure. sps\_hdrwcg\_extension\_flag equal to 0 specifies that this syntax structure is not present. When not present, the value of sps\_hdrwcg\_extension\_flag is inferred to be equal to 0.

**sps\_extension\_3bits** equal to 0 specifies that no sps\_extension\_data\_flag syntax elements are present in the SPS RBSP syntax structure. When present, sps\_extension\_3bits shall be equal to 0 in bitstreams conforming to this version of this Specification. Values of sps\_extension\_3bits not equal to 0 are reserved for future use by ITU-T | ISO/IEC. Decoders shall allow the value of sps\_extension\_3bits to be not equal to 0 and shall ignore all sps\_extension\_data\_flag syntax elements in an SPS NAL unit. When not present, the value of sps\_extension\_3bits is inferred to be equal to 0.

*Add a new semantics section* *7.4.3.2.4 (Sequence parameter set HDR and WCG extension semantic)*

**hdr\_output\_transfer\_characteristics\_present\_flag** equal to 1 specifies that the syntax element hdr\_output\_transfer\_characteristics is present in the SPS. hdr\_output\_transfer\_chracteristics\_present\_flag equal to 0 specifies that the syntax element hdr\_output\_transfer\_characteristics is not present in the SPS. When not present, the value of hdr\_output\_transfer\_characteristics\_present\_flag is inferred to be equal to 0.

**hdr\_output\_transfer\_characteristics** has the same semantics as specified in clause E.3.1 for the transfer\_characteristics syntax element, except that hdr\_output\_transfer\_characteristics specifies the colour space of the reconstructed HDR picture, rather than the colour space used for the CLVS. When not present, the value of hdr\_output\_transfer\_characteristics is inferred to be equal to the value of transfer\_characteristics.

**reshaper\_range\_id** indicates the black level and range of the luma and chroma signals.

The variable MinSampleVal[ c ] and MaxSampleVal[ c ] are derived as follows:

bitDepth = ( c = = 0 ) ? BitDepthY : BitDepthC  
if( reshaper\_range\_id = = 0 ) {  
 MinSampleVal[ c ] = 0  
 MaxSampleVal[ c] = (1 << bitDepth ) − 1  
} else if ( reshaper\_range\_id = = 1 ) {  
 MinSampleVal[ c ] = 16 << (bitDepth − 8)  
 if( c = = 0)   
 MaxSampleVal[ c ] = MinSampleVal[ c ] + 219 << (bitDepth − 8)  
 else  
 MaxSampleVal[ c ] = MinSampleVal[ c ] + 224 << (bitDepth − 8)  
}

The variable DeltaSampleVal[ c ] for c in the range of 0 to 2, inclusive, are derived as follows:

DeltaSampleVal[ c ] = MaxSampleVal[ c ] – MinSampleVal[ c ] (7-xx)

*Add a new semantics section* *7.4.3.3.5 (General HDR reshape table semantics)*

**reshape\_present\_flag** equal to 1 specifies that syntax elements reshape\_input\_luma\_bit\_depth\_minus8, reshape\_input\_chroma\_bit\_depth\_minus8, reshape\_output\_luma\_bit\_depth\_minus8, reshape\_output\_chroma\_bit\_depth\_minus8, coef\_log2\_denom\_minus14, reshape\_num\_pivots\_minus2, reshape\_coded\_pivot\_value, poly\_order, poly\_coef\_int, poly\_coef\_frac and reshape\_pps\_id are present. reshape\_present\_flag equal to 0 specifies that syntax elements reshape\_input\_luma\_bit\_depth\_minus8, reshape\_input\_chroma\_bit\_depth\_minus8, reshape\_output\_luma\_bit\_depth\_minus8, reshape\_output\_chroma\_bit\_depth\_minus8, coef\_log2\_denom\_minus14, reshape\_num\_pivots\_minus2, reshape\_coded\_pivot\_value, poly\_order, poly\_coef\_int, poly\_coef\_frac and reshape\_pps\_id are not present.

**reshape\_chroma\_crosschannel\_flag** equal to 0 specifies that reshaping for chroma sample values does not depend on luma sample values. reshape\_chroma\_crosschannel\_flag equal to 1 specifies that reshaping for chroma sample values depends on luma sample values.

**reshape\_input\_luma\_bit\_depth\_minus8** specifies the sample bit depth of the input luma component of the HDR reshaping process. The variable BitDepthReshapeInputL is derived as follows:

BitDepthReshapeInputL = 8 + reshape\_input\_luma\_bit\_depth\_minus8 (7‑xx)

**reshape\_input\_chroma\_bit\_depth\_minus8** specifies the sample bit depth of the input chroma component of the HDR reshaping process. The variable BitDepthReshapeInputC is derived as follows:

BitDepthReshapeInputC = 8 + reshape\_input\_chroma\_bit\_depth\_minus8 (7‑xx)

**reshape\_output\_luma\_bit\_depth\_minus8** specifies the sample bit depth of the output luma component of the HDR reshaping process. The variable BitDepthReshapeOutputL is derived as follows:

BitDepthReshapeOutputL = 8 + reshape\_output\_luma\_bit\_depth\_minus8 (7‑xx)

**reshape\_output\_chroma\_bit\_depth\_minus8** specifies the sample bit depth of the output chroma component of the HDR reshaping process. The variable BitDepthReshapeOutputC is derived as follows:

BitDepthReshapeOutputC = 8 + reshape\_output\_chroma\_bit\_depth\_minus8 (7‑xx)

**reshape\_default\_flag** equal to 0 specifies that the syntax elements reshape\_scale\_int\_bit\_depth, reshape\_offset\_int\_bit\_depth, reshape\_scale\_frac\_bit\_depth, reshape\_offset\_frac\_bit\_depth, and reshape\_negative\_scales\_present\_flag are signalled. reshape\_default\_flag equal to 1 specifies that the syntax elements reshape\_scale\_int\_bit\_depth, reshape\_offset\_int\_bit\_depth, reshape\_scale\_frac\_bit\_depth, reshape\_offset\_frac\_bit\_depth, and reshape\_negative\_scales\_present\_flag are not signalled and the values are inferred as specified by the semantics of the syntax elements..

**reshape\_scale\_int\_bit\_depth** specifies the number of bits used to represent the integer part of the scale factors of the chroma piece-wise linear models. The value of reshape\_scale\_int\_bit\_depth shall be in the range of 0 to 15, inclusive. When not present, the value of reshape\_scale\_int\_bit\_depth is set to 2.

**reshape\_offset\_int\_bit\_depth** specifies the number of bits used to represent the integer part of the scale factors of the chroma piece-wise linear models. The value of reshape\_offset\_int\_bit\_depth shall be in the range of 0 to 15, inclusive. When not present, the value of reshape\_offset\_int\_bit\_depth is set to 0.

**reshape\_scale\_frac\_bit\_depth** specifies the number of bits used to represent the fractional part of the scale factors of the chroma piece-wise linear models. The value of reshape\_scale\_frac\_bit\_depth shall be in the range of 0 to 15, inclusive. When not present, the value of reshape\_scale\_frac\_bit\_depth is set to 6.

**reshape\_offset\_frac\_bit\_depth** specifies the number of bits used to represent the offset values of the chroma piece-wise linear models. The value reshape\_offset\_frac\_bit\_depth shall be in the range of 0 to 15, inclusive. When not present, the value of reshape\_offset\_frac\_bit\_depth is set to 8.

The variable ReshapeOffsetBitDepth is derived as follows:

ReshapeOffsetBitDepth = reshape\_offset\_bit\_int\_depth + reshape\_offset\_bit\_frac\_bit\_depth (7-xx)

**reshape\_negative\_scales\_present\_flag** equal to 0 indicates that the MSB bit of hdr\_recon\_scale\_int\_val does not specify the sign of the value of reshape\_scale\_int\_val. reshape\_negative\_scales\_present\_flag equal to 1 indicates that the MSB bit of reshape\_scale\_int\_val specifies the sign of the value of reshape\_scale\_int\_val. When not present, the value of reshape\_negative\_scales\_present\_flag is set to 0.

**reshape\_num\_comps\_minus2** plus 2 specifies the number of components for which the reshape parameters are signalled. The value of reshape\_num\_comps\_minus2 shall be in the range of 0 to 1, inclusive. When reshape\_num\_comps\_minus2 equal to 0, the values of the syntax elements reshape\_num\_ranges\_minus[ 2 ], reshape\_equal\_ranges\_flag[ 2 ], reshape\_global\_offset\_val[ 2 ], reshape\_range\_val[ 2 ][ i ], reshape\_scale\_int\_val[ 1 ][ i ], and reshape\_scale\_fract\_val[ 1 ][ i ] are inferred to be equal to reshape\_num\_ranges\_minus[ 1 ], reshape\_equal\_ranges\_flag[ 1 ], reshape\_global\_offset\_val[ 1 ], reshape\_range\_val[ 1 ][ i ], reshape\_scale\_int\_val[ 0 ][ i ], and reshape\_scale\_fract\_val[ 0 ][ i ] for i in the range of 0 to reshape\_num\_ranges\_minus[ 1 ], inclusive.

**reshape\_num\_ranges\_minus1**[ c ] plus 1 specifies the number of ranges in the piece-wise reshaping function. When not present, the value of reshape\_num\_ranges\_minus1[ c ] is inferred to be 0. reshape\_num\_ranges \_minus1[ c ] shall be in the range of 0 to 7, inclusive for luma component and in the range of 0 to 31, inclusive for chroma component.

The variable NumberRanges[ c ] is derived as following:

NumberRanges[ c ] = reshape\_num\_ranges\_minus1[ c ] + 1 (7‑xx)

**reshape\_equal\_ranges\_flag**[ c ] equal to 1 specifies that piece-wise reshaping function is partitioned into NumberRanges[ c ] pieces with nearly equal length and the length of each range is not explicitly signalled. reshape\_equal\_ranges\_flag[ c ] equal to 0 specifies that the length of each range is explicitly signalled.

**reshape\_global\_offset\_val**[ c ] is used to derive the offset value that is used to specify the starting point of 0th range for the c-th component. The number of bits used for the representation of the reshape\_global\_offset\_val[ c ] is BitDepthReshapeInputL for luma component and ReshapeOffsetBitDepth for chroma component.

**reshape\_range\_val**[ c ][ i ] is used to derive the length of the i-th range of the c-th component. The number of bits used for the representation of the reshape\_range\_val[ c ][ i ] is BitDepthReshapeInputL for luma component and ReshapeOffsetBitDepth for chroma component.

The variables RangeValue[ c ][ i ] with i in the range of 0 to NumberRanges[ c ] – 1, inclusive, are derived as follows:

– If reshape\_equal\_ranges\_flag[ c ] is equal to 0, the value of RangeValue[ c ][ i ] is specified as follows:

RangeValue[ c ][ i ] = reshape\_range\_val[ i ] (7‑xx)

– Otherwise (reshape\_equal\_ranges\_flag[ c ] is equal to 1), the value of RangeValue[ c ][ i ] is specified as follows:

– The value InputDynamicRangeValue equal to 1.

– The value RangeValue[ c ][ i ] is derived as follows:

RangeValue[ c ][ i ] = ( ( InputDynamicRangeValue << reshape\_offset\_frac\_bit\_depth ) +  
 ( ( NumberRanges[ c ] + 1 ) >> 1 ) ) / NumberRanges[ c ] (7‑xx)

if (c = = 0)  
 RangeValue[ c ][ i ] = (RangeValue[ c ][ i ] \* DeltaSampleVal[ c ] + (  1 << (BitDepthL – 1)) ) >> BitDepthL

The variable ReshapeCodedPivotValue[ i ] with i in the range of 0 to NumberRanges[ 0 ] is derived as follows:

–

ReshapeCodedPivotValue[ 0 ] = reshape\_global\_offset\_val[ 0 ] (7‑xx)

– The value ReshapeCodedPivotValue[ i ], for i=1..NumberRanges[ 0 ], is derived as follows:

ReshapeCodedPivotValue[ i ] = RangeValue[ 0 ][ i − 1  ] + ReshapeCodedPivotValue[ i − 1 ] (7‑xx)

**coeff\_log2\_offset\_minus2** specifies the number of fractional bits for HDR reshaping related coefficients calculations for the luma component. The value of coeff\_log2\_offset\_minus2 shall be in the range of 0 to 3, inclusive. The variables ReshapeOrder0Log2Denom, ReshapeOrder1Log2Denom and ReshapeOrder2Log2Denom for the luma component are derived as follows:

ReshapeOrder0Log2Denom = 2 + coeff\_log2\_offset\_minus2 (7‑xx)

ReshapeOrder1Log2Denom = 2 + BitDepthReshapeInputL + coeff\_log2\_offset\_minus2 (7‑xx)

ReshapeOrder2Log2Denom = 2 + 2\* BitDepthReshapeInputL + coeff\_log2\_offset\_minus2  (7‑xx)

**reshape\_continuity\_flag** specifies the continuity properties of the reshaping function for the luma component. If reshape\_continuity\_flag is equal to 0, zeroth order continuity is applied to the piecewise linear inverse reshaping functions between consecutive pivot points. If reshape\_continuity\_flag is equal to 1, first order smoothness is used to derive the full second order polynomial inverse reshaping functions between consecutive pivot points.

**poly\_coef\_int**[ i ][ j ] specifies the integer value of the i-th piece j-th order polynomial coefficient for colour component 0. The value of poly\_coef\_int[i][0] is in the range of 0 to ( 1  << BitDepthReshapeInputL − 1 ). If reshape\_continuity\_flag is equal to 1, the value of poly\_coef\_int[ 0 ][ 1 ] shall be in the range of -64 to 63, inclusively.

**poly\_coef\_frac**[ i ][ j ] specifies the fractional value of the i-th piece j-th order polynomial coefficient for colour component 0. The number of bits used for the representation of poly\_coef\_frac[ i ][ 0 ] is ReshapeOrder0Log2Denom. The number of bits used for the representation of poly\_coef\_frac[ i ][ 1 ] is ReshapeOrder1Log2Denom.

The variables PolyCoef[ i ][ 0 ] with i in the range of 0 to NumberRanges[ 0 ] inclusive, are derived as follows:

PolyCoef[ i ][ 0 ] = ( ( poly\_coef\_int[ i ][ 0 ] << ReshapeOrder0Log2Denom ) + poly\_coef\_frac[ i ][ 0 ] ) (7‑xx)

The variables PolyCoef[ i ][ 1 ] and PolyCoef[ i ][ 2 ] are derived as follows:

– If reshape\_continuity\_flag is equal to 0, the variables PolyCoef[ i ][ 1 ] and PolyCoef[ i ][ 2 ], and i in the range of 0 to NumberRanges[ 0 ] – 1, inclusive, are derived as follows:

PolyCoef[ i ][ 1 ] = ( ( PolyCoef[ i+1][ 0 ] – PolyCoef[ i][ 0 ] ) <<   
 ( ReshapeOrder1Log2Denom – ReshapeOrder0Log2Denom ) +   
 ( ( ReshapeCodedPivotValue [ i+1 ] – ReshapeCodedPivotValue [ i ] ) >> 1 ) ) / (ReshapeCodedPivotValue [ i+1 ] – ReshapeCodedPivotValue [ i ])   (7-xx)

PolyCoef[ i ][ 2 ]  = 0 (7-xx)

– Otherwise ( reshape\_continuity\_flag is equal to 1 ),

* The variable PolyCoef[ 0 ][ 1 ], is derived as follows:

PolyCoef[ 0 ][ 1 ] =( (poly\_coef\_int[ 0 ][ 1 ] << ReshapeOrder1Log2Denom) + poly\_coef\_frac[ 0 ][ 1 ]) (7‑xx)

* The variables PolyCoef[ i ][ 1] with i in the range of 1 to NumberRanges[ 0 ], inclusive, are derived as follows:

for(i =1; i < NumberRanges[ 0] + 1; i ++) {  
 PolyCoef[ i ][ 1 ] = ( ( PolyCoef[ i ][ 0 ] – PolyCoef[ i – 1][ 0 ] )  
  << ( 1 + ReshapeOrder1Log2Denom – ReshapeOrder0Log2Denom )   
  + ( ( ReshapeCodedPivotValue[ i ] – ReshapeCodedPivotValue [ i –1] ) >> 1 ) )  
  / ( ReshapeCodedPivotValue [ i ] – ReshapeCodedPivotValue [ i – 1 ] )    
 – PolyCoef[ i – 1 ][ 1 ] (7‑xx)   
}

* The variables PolyCoef[ i ][ 2 ] with i in the range of 0 to NumberRanges[ 0 ] – 1, inclusive, are derived as follows:

PolyCoef[ i ][ 2] = ( ( PolyCoef[ i + 1 ][ 1 ] – PolyCoef[ i][ 1 ] )  
 << ( ReshapeOrder2Log2Denom – ReshapeOrder1Log2Denom – 1 )  
 + ( ( ReshapeCodedPivotValue[ i + 1 ] – ReshapeCodedPivotValue[ i ] ) >> 1 ) )  
  / ( ReshapeCodedPivotValue[ i + 1] – ReshapeCodedPivotValue [ i ] )   (7‑xx)

The variable MaxReshapeOrder is set equal to 2. The variable ReshapeLog2DenomL is set equal to ReshapeOrder2Log2Denom – MaxReshapeOrder \* BitdepthReshapeInputL + BitdepthReshapeOutputL.

The variables PolyCoef[ i ][ 0 ] with i in the range of 0 to NumberRanges[ 0 ] – 1, inclusive, are adjusted as follows:

PolyCoef[ i ][ 0 ] = ( PolyCoef[ i ][ 0 ] << ( ReshapeOrder2Log2Denom – ReshapeOrder0Log2Denom )  
  – ( PolyCoef[ i ][ 1 ] \* ReshapeCodedPivotValue [ i ] ) <<   
 ( ReshapeOrder2Log2Denom –  ReshapeOrder1Log2Denom ) +   
 PolyCoef[ i ][ 2 ] \* ReshapeCodedPivotValue[ i ] \* ReshapeCodedPivotValue [ i ]  
  + ( 1 << ( ReshapeOrder2Log2Denom – ReshapeOrder0Log2Denom – 1 ) ) )  
  >> ( ReshapeOrder2Log2Denom – ReshapeOrder0Log2Denom ) (7‑xx)

The variables PolyCoef[ i ][ 1 ] with i in the range of 0 to NumberRanges[ 0 ] – 1, inclusive, are adjusted as follows:

PolyCoef[ i ][ 1 ] = ( PolyCoef[ i ][ 1 ] << (ReshapeOrder2Log2Denom – ReshapeOrder1Log2Denom)   
  – 2 \* ReshapeCodedPivotValue [ i ] \* PolyCoef[ i ][ 2 ] +   
 ( 1 << ( ReshapeOrder2Log2Denom – ReshapeOrder1Log2Denom – 1 ) ) ) >>   
 ( ReshapeOrder2Log2Denom – ReshapeOrder1Log2Denom ) (7‑xx)

**reshape\_scale\_int\_val**[ c ][ i ] is used to derive the offset value that is used to derive the width of the of the i-th partition of the piece-wise linear model of the c-th color component. The number of bits used for the representation of reshape\_scale\_int\_val[ c ][ i ] is reshape\_offset\_int\_bit\_depth.

**reshape\_scale\_fract\_val**[ c ][ i ] is used to derive the offset value that is used to derive the width of the of the i-th partition of the the piece-wise linear model of the c-th component. The number of bits used for the representation of reshape\_scale\_fract\_val[ c ][ i ] is reshape\_offset\_frac\_bit\_depth .

When component c index is larger than 0, the following applies:

– The variable ScaleValue[ c ][ i ], for i in the range of 0 to NumberRanges[ c ] – 1, inclusive, is derived as follows:

– The value SignValue[ c ][ i ] is set equal to 1.

– The value hdrReconScaleBitDepth is set equal to ( reshape\_scale\_bit\_int\_depth + reshape\_scale\_bit\_frac\_depth – reshape\_negative\_scales\_present\_flag ).

– If reshape\_negative\_scales\_present\_flag is equal to 1, the following applies:

hdrScaleIntVal = ( ( reshape\_scale\_int\_val[ c ][ i ] & ( ( 1 << reshape\_scale\_int\_bit\_depth ) – 1 ) )

ScaleValue[ c ][ i ] = ( hdrScaleIntVal << reshape\_scale\_frac\_bit\_depth ) + reshape\_scale\_frac\_val[ c ][ i ] (7-xx)

SignValue[ c ][ i ] = 1 – 2 \* ( ( reshape\_scale\_int\_val[ c ][ i ] >>   
 ( reshape\_scale\_int\_bit\_depth – 1 ) ) & 1 ) (7-xx)

– Otherwise (rehshape\_negative\_scales\_present\_flag is not equal to 1), the following applies:

ScaleValue[ c ][ i ] = ( reshape\_scale\_int\_val[ c ][ i ] << reshape\_scale\_frac\_bit\_depth ) + reshape\_scale\_fract\_val[ c ][ i ] (7-xx)

The variables InputRanges[ c ][ i ] and OutputRanges[ c ][ i ], for i in the range of 0 to NumberRanges[ c ], inclusive, and c in the range of 0 to 1, inclusive, are derived as follows:

– The value DefaultPrecShift is derived as follows:

– If reshape\_chroma\_crosschannel\_flag is equal to 1, DefaultPrecShift is set equal to 4

– Otherwise (reshape\_chroma\_crosschannel\_flag is equal to 0), DefaultPrecShift is set equal to 9

– The value InvScaleValue[ c ][ i ] is derived as follows:

shiftInvScale = 1 << hdrReconScaleBitDepth  
InvScaleValue[ c ][ i ] = ( 1 << ( DefaultPrecShift + hdrReconScaleBitDepth ) + shiftInvScale ) /   
 ScaleValue[ c ][ i ] (7-xx)

– If i is equal to 0, the following applies:

multValue = SignValue[ c ][ 0 ] \* ( ( reshape\_chroma\_crosschannel\_flag ) ? ScaleValue[ c ][ i ] : InvScaleValue[ c ][ 0 ] )  
shVal = BitDepthC − reshape\_offset\_frac\_bit\_depth

if( shVal > 0 )  
 smallestValInp = ( MinSampleVal[ c ] + ( 1 << shVal – 1 ) ) >> shVal  
else  
 smallestValInp = MinSampleVal[ c ] << shVal  
shVal = BitDepthC – ( reshape\_offset\_frac\_bit\_depth + DefaultPrecShift )  
if( shVal > 0 )  
 smallestValOut = ( MinSampleVal[ c ] + ( 1 << shVal – 1 ) ) >> shVal  
else  
 smallestValOut = MinSampleVal[ c ] << shVal  
if( reshape\_chroma\_crosschannel\_flag ) {  
 OutputRanges[ c ][ 0 ] = reshape\_global\_offset\_val[ c ] <<   
 ( 4 + reshape\_scale\_frac\_bit\_depth + DefaultPrecShift )  
 InputRanges[ c ][ 0 ] = 0

} else {  
 OutputRanges[ c ][ 0 ] = − ( ( reshape\_global\_offset\_val[ c ] \* DeltaSampleVal[ c ] +   
 ( 1 << BitDepthC – 1 ) ) >> BitDepthC ) \* multValue + smallestValOut  
 InputRanges[ c ][ 0 ] = smallestValInp (7-xx)  
}

– Otherwise (i is not equal to 0), the following applies:

multValue = SignValue[ c ][ i −1 ] \* ( ( reshape\_chroma\_crosschannel\_flag ) ? ScaleValue[ c ][ i – 1 ] : InvScaleValue[ c ][ i – 1 ] )  
rangeVal = ( ( RangeValue[ c ][ i – 1 ] \* DeltaSampleVal[ c ] + (  1 << BitDepthC – 1) ) >> BitDepthC  
if( reshape\_chroma\_crosschannel\_flag ) {  
 InputRanges[ c ][ i ] = InputRanges[ c ][ i − 1 ] + RangeValue[ c ][ i – 1 ] << DefaultPrecShift (7-xx)  
 OutputRanges[ c ][ i ] = OutputRanges[ c ][ i − 1 ] + RangeValue[ c ][ i – 1 ] \* multValue (7-xx)  
} else {  
 InputRanges[ c ][ i ] = InputRanges[ c ][ i − 1 ] + rangeVal (7-xx)  
 OutputRanges[ c ][ i ] = OutputRanges[ c ][ i − 1 ] + rangeVal \* multValue (7-xx)  
}

The variables OffsetValue[ c ][ i ] for i in the range of 0 to NumberRanges[ c ] – 1, inclusive, are derived as follows:

– The value OffsetValue[ c ][ i ] is derived as follows:

precOffsetDeltaBits = DefaultPrecShift + reshape\_scale\_frac\_bit\_depth

if( reshape\_chroma\_crosschannel\_flag )  
 OffsetValue[ c ][ i ] = OutputRanges[ c ][ i + 1 ] −  
 InputRanges[ c ][ i + 1 ] \* ScaleValue[ c ][ i ] \* SignValue [ c ][ i ] (7-xx)

else  
 OffsetValue[ c ][ i ] = InputRanges[ c ][ i + 1 ] \* (1 << precOffsetDeltaBits ) −  
 OutputRanges[ c ][ i + 1 ] \* ScaleValue[ c ][ i ] \* SignValue [ c ][ i ] (7-xx)

**reshape\_offset1**[ c ] specifies an input offset value for the reshaping process. The value of reshape\_offset1[ k ] shall be in the range of 0 to ( 1 << BitDepthReshapeInputC ) – 1. If reshape\_offset1[ 2 ] is not present, reshape\_offset1[ 2 ] is equal to reshape\_offset1[ 1 ].

**reshape\_offset2**[ c ] specifies an output offset value for the reshaping process. The value of reshape\_offset2[ k ] shall be in the range of 0 to ( 1 << BitDepthReshapeInputC ) – 1. If reshape\_offset2[ 2 ] is not present, reshape\_offset2[ 2 ] is equal to reshape\_offset2[ 1 ].

*Add the following section at the end of 8.1.3 (Decoding process for a coded picture with nuh\_layer\_id equal to 0)*

1. Additionally, the HDR reconstruction process is specified in clause 8.8.

*Add the following new section 8.8 (HDR reconstruction process)*

## 8.8 HDR reconstruction process

* + 1. **General**

This clause specifies the application of HDR reconstruction process using reshaping.

For the HDR reshaping process, the following applies:

– The HDR reshaping process as specified in clause 8.8.2 is invoked with the reconstructed picture sample array SL and, when ChromaArrayType is not equal to 0, the arrays SC0 and SC1 as inputs, and the modified reconstructed picture sample array S′L and, when ChromaArrayType is not equal to 0, the arrays S′C0 and S′C1 after HDR reshaping as outputs.

– The array S′L and, when ChromaArrayType is not equal to 0, the arrays S′C0 and S′C1 are assigned to the array SL and, when ChromaArrayType is not equal to 0, the arrays SC0 and SC1 (which represent the decoded picture), respectively.

* + 1. **HDR reshaping process**
       1. **General**

Inputs to this process are the reconstructed picture sample array prior to HDR reshaping process recPictureL and, when ChromaArrayType is not equal to 0, the arrays recPictureC0 and recPictureC1.

Outputs of this process are the modified reconstructed picture sample array after HDR reshaping process reshapePictureL and, when ChromaArrayType is not equal to 0, the arrays reshapePictureC0 and reshapePictureC1.

The sample values in the modified reconstructed picture sample array reshapePictureL and, when ChromaArrayType is not equal to 0, the arrays reshapePictureC0 and reshapePictureC1 are initially set equal to the sample values in the reconstructed picture sample array recPictureL and, when ChromaArrayType is not equal to 0, the arrays recPictureC0 and recPictureC1, respectively.

The reshapped luma sample reshapePictureL[ xP ][ yP ] with xP = 0..PicWidthInSamplesY − 1, yP = 0..PicHeightInSamplesY − 1 is derived by invoking the hdr reshaping process of luma sample values as specified in clause 8.8.2.2 with luma sample location ( xP, yP ), sample arrays recPictureL as inputs.

The reshapped chroma sample reshapePictureC0[ xC ][ yC ] with xC = 0..PicWidthInSamplesC − 1, yC = 0..PicHeightInSamplesC − 1 is derived by invoking the hdr reshaping process of chroma sample values as specified in clause 8.8.2.3 with chroma sample location ( xC, yC ), and a variable cIdx set to 0 as inputs.

The reshapped chroma sample reshapePictureC1[ xC ][ yC ] with xC = 0..PicWidthInSamplesC − 1, yC = 0..PicHeightInSamplesC − 1 is derived by invoking the hdr reshaping process of chroma sample values as specified in clause 8.8.2.3 with chroma sample location ( xC, yC ), and a variable cIdx set to 1 as inputs.

* + - 1. **HDR reshaping process of luma sample values**

Inputs to this process is a luma location ( xP, yP ) specifying the luma sample location relative to the top-left luma sample of the current picture.

Outputs of this process is a hdr reshaped luma sample value reshapeLumaSample.

The value of reshapeLumaSample is derived by applying the following ordered steps:

1. The variables idxL is derived by invoking the identification of piece-wise function index as specified in clause 8.8.2.3 with the input of sample value recPictureL[ xP ][ yP ] for colour component 0.
2. The value of reshapeLumaSample is derived as follows:

reshapeLumaSample = ( ( PolyCoef[ idxL ][ 0 ]  <<  ( MaxReshapeOrder \* BitDepthReshapeInputL )  
  + ( PolyCoef[ idxL ][ 1 ] \* ( recPictureL[ xP ][ yP ]  <<  BitDepthReshapeInputL ) ) (7‑xx)   
  + ( PolyCoef[ idxL ][ 2 ] \* recPictureL[ xP ][ yP ] \* recPictureL[ xP ][ yP ]]  )  
>>  ( ReshapeLog2DenomL +  MaxReshapeOrder \* BitdepthReshapeInputL –  BitdepthReshapeOutputL )

reshapeLumaSample = Clip3( 0, (1  <<  BitDepthReshapeOutputL ) − 1, reshapeLumaSample ) (7‑xx)

* + - 1. **Identification of piecewise function index for luma components**

Inputs to this process are a luma sample value S.

Output of this process is an index idxS identifing the segment to which the sample S belongs. The variable idxS is derived as follows:

if( ( S < ReshapeCodedPivotValue[ 0 ] )   
 idxS = ReshapeCodedPivotValue [ 0 ] (7‑xx)  
else if(S > ReshapeCodedPivotValue [ NumberRanges[0] ] )  
 idxS = ReshapeCodedPivotValue [ NumberRanges[0] ]  
  
for( idxS = 0, idxFound = 0; idxS < NumberRanges[0]; idxS++ ) {  
 if( S < ReshapeCodedPivotValue [idxS + 1] ) {  
 idxFound = 1  
 }  
}  
if ( ! idxFound)   
 idxS = idxS − 1

* + - 1. **HDR reshaping process of chroma sample values**

Inputs to this process is a luma location ( xPC, yPC ) specifying the chroma sample location relative to the top-left chroma sample of the current picture and a variable cIdx specifying the chroma component index.

Outputs of this process is a hdr reshaped chroma sample value reshapeChromaSample.

recPictureC is set equal to recPictureC0 if cIdx is equal to 0 and set equal to recPictureC1 otherwise.

The value of reshapeChromaSample is derived as follows:

– If reshape\_chroma\_crosschannel\_flag is equal to 1 the following applies:

* The value of signChroma is set equal to 1.
* the value zeroMeanChromaSample is derived as follows:
* zeroMeanChromaSample = recPictureC[ xC ][ yC ] − reshape\_offset1[ c – 1 ]

if ( zeroMeanChromaSample <0 ) {  
 the value signChroma is set equal to – 1  
 zeroMeanChromaSample = zeroMeanChromaSample \* signChroma  
}

– If reshape\_chroma\_crosschannel\_flag is equal to 0, the following applies:

– The value of bitDepthDeltaC is set equal to ( DefaultPrecShift + reshape\_offset\_frac\_bit\_depth − BitDepthC ).

– the values idxS and fracBitDepthShiftVal are derived by invoking the identification of piece-wise function index as specified in clause 8.8.2.5 with sample value recPictureC[ xPC ][ yPC ], shift value bitDepthDeltaC and range array OutputRanges for component cIdx + 1 as inputs.

– The value of reshapeChromaSample is derived as follows:

if( idxS = = 0 || idxS = = NumberRanges[ cIdx + 1 ] )  
 reshapeChromaSample = InputRanges[ cIdx + 1 ][ idxS ]   
else {

reshapeChromaSample = ( recPictureC[ xC ][  yC  ]) ) \* ScaleValue[ cIdx ][ idxS ] \* SignValue[ cIdx ][ idxS ] \* ( 1 << ( DefaultPrecShift + reshape\_offset\_frac\_bit\_depth – BitDepthC ) ) +  
 OffsetValue[ cIdx ][ idxS ] (7-xx)

reshapeChromaSample = ( reshapeChromaSample +   
 (1 << (fracBitDepthShiftVal - 1)) ) >> fracBitDepthShiftVal (7-xx)

resapeChromaSample = Clip3( MinSampleVal[ cIdx ], MaxSampleVal[ cIdx ],   
 reshapeChromaSample )

}

– Otherwise ( reshape\_chroma\_crosschannel\_flag is equal to 1 ), the following applies:

– The value of bitDepthDeltaL is set equal to ( DefaultPrecShift + reshape\_offset\_frac\_bit\_depth – BitDepthY ).

– The variable lumaVal is derived as follows:

lumaVal = ( recPictureL[ xPC << 1 ][ yPC << 1 ] +   
 recPicureL[ ( xPC << 1 ) + 1 ][ yPC << 1 ] +  
 recPicureL[ xPC << 1 ][ ( yPC << 1 ) + 1 ] +   
 recPicureL[ ( xPC << 1 ) + 1 ][ ( yPC << 1 ) + 1 ] +2 ) >> 2 (7-xx)

– The values idxC and fracBitDepthShiftVal are derived by invoking the identification of piece-wise function index as specified in clause 8.8.2.5 with sample value lumaVal, shift value bitDepthDeltaL and range array InputRanges for component cIdx + 1 as inputs.

– The value of scaleVal is derived as follows:

scaleVal = lumaVal \* ScaleValue[ cIdx ][ idxS ] \* SignValue[ cIdx ][ idxS ] \*  
 ( 1 << bitDepthDeltaL ) + OffsetValue[ cIdx ][ idxS ] (7-xx)

– The value of crossPlaneshift is set equal to ( DefaultPrecShift + reshape\_scale\_frac\_bit\_depth + reshape\_offset\_frac\_bit\_depth )

– The value of reshapeChromaSample is derived as follows:

reshapeChromaSample = ( scaleVal \* zeroMeanChromaSample +   
 ( 1 << (crossChannelShift – 1 ) ) ) >> crossChannelShift (7-xx)

reshapeChromaSample = Clip3( MinSampleVal[ cIdx ], MaxSampleVal[ cIdx ],   
 reshapeChromaSample \* signChroma + reshape\_offset2[ c – 1 ] ) ) (7-xx)

* + - 1. **Identification of piecewise function index for chroma components**

Inputs to this process are a sample value S, bit shift value shiftVal and range array RangeArray and the component index c.

Output of this process is an index idxS identifing the segment to which the sample S belongs and the value fracBitDepthShiftVal.

The variable idxS and fracBitDepthShiftVal are derived as follows:

if( ( ( S << shiftVal ) <= RangeArray[ c ][ 0 ] )  {  
 idxS = 0   
 fracBitDepthShiftVal = reshape\_offset\_frac\_bit\_depth  
} else if( ( S  << shiftVal ) > RangeArray[ c ][  NumberRanges [ c ] ] ) {  
 idxS = NumberRanges[ c ]  
 fracBitDepthShiftVal = reshape\_offset\_frac\_bit\_depth  
} else {   
 for( idxS = 1, idxFound = 0; idxS <= NumberRanges[ c ] && !idxFound; idxS++ ) {  
 if(( S << shiftVal ) > RangeArray[ c ][ idxS − 1 ] && ( S << shiftVal ) <= RangeArray[ c ][idxS ] )  
 idxFound = 1  
 fracBitDepthShiftVal = DefaultPrecShift + reshape\_scale\_frac\_bit\_depth + reshape\_offset\_frac\_bit\_depth – BitDepth  
}